smashGP: Large-scale Spatial Modeling via Matrix-free Gaussian Processes

Lucas Erlandson
School of Computational Science and Engineering
Georgia Institute of Technology

Ana María Estrada Gómez
School of Industrial Engineering
Purdue University

Edmond Chow
School of Computational Science and Engineering
Georgia Institute of Technology

Kamran Paynabar
School of Industrial and Systems Engineering
Georgia Institute of Technology

Abstract

Gaussian processes are essential for spatial data analysis. Not only do they allow the prediction of unknown values, but they also allow for uncertainty quantification. However, in the era of big data, directly using Gaussian processes has become computationally infeasible as cubic run times are required for dense matrix decomposition and inversion. Various alternatives have been proposed to reduce the computational burden of directly fitting Gaussian processes. These alternatives rely on assumptions on the underlying structure of the covariance or precision matrices, such as sparsity or low-rank. In contrast, this article uses hierarchical matrices and matrix-free methods to enable the computation of Gaussian processes for large spatial datasets by exploiting the underlying kernel properties. The proposed framework, smashGP, represents the covariance matrix as an $\mathcal{H}^2$ matrix in $\mathcal{O}(n)$ time and is able to estimate the unknown parameters of the model and predict the values of spatial observations at unobserved locations in $\mathcal{O}(n \log n)$ time thanks to fast matrix-vector products. Additionally, it can be parallelized to take full advantage of shared-memory computing environments. With simulations and case studies, we illustrate the advantage of smashGP to model large-scale spatial datasets. Supplementary materials are provided online.

Keywords: Gaussian processes, hierarchical matrices, matrix-free methods, spatial data analysis

1 Introduction

Gaussian processes (GPs) are a powerful machine learning tool (Rasmussen and Williams 2005) and have a dominant role in spatial statistics (Stein 2012; Banerjee et al. 2014; Cressie 2015) where they provide insights into many geophysical and environmental problems. For example, they enable the study of greenhouse gas concentrations for climate change (Fang et al. 2018), of soil properties for precision agriculture (Andugula et al. 2017), and of different atmospheric states for weather forecasting (Heaton et al. 2019). Three main features explain the usefulness of GPs in the analysis of spatial data. (1) Unknown parameters of the model can be estimated; (2) GPs predict the values of spatial responses at unobserved locations; and (3) GPs quantify the uncertainty in the predictions and parameters (Cressie 2015).

A spatial process $Y(\boldsymbol{s})$ for $\boldsymbol{s} \in \mathcal{D} \subset \mathbb{R}^2$ is said to follow a GP if any realization $\boldsymbol{Y} = (Y(\boldsymbol{s}_1), \ldots, Y(\boldsymbol{s}_n))^\top$ at the finite number of locations $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$ follows an n-variate Gaussian distribution. More specifically, let $\mu(\boldsymbol{s}) : \mathcal{D} \to \mathbb{R}$ denote the mean function and $\Sigma(\boldsymbol{s}_1, \boldsymbol{s}_2) : \mathcal{D}^2 \to \mathbb{R}$ denote the positive-definite covariance function. Then, $\boldsymbol{Y}$ is distributed as $N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu} = (\mu(\boldsymbol{s}_1), \ldots, \mu(\boldsymbol{s}_n))^\top$ is the mean vector and $\boldsymbol{\Sigma} = \{\Sigma(\boldsymbol{s}_i, \boldsymbol{s}_j)\}_{i,j=1}^n$ is the $n \times n$ covariance matrix. The Gaussian structure of the spatial process allows for a large degree of analytical capability, enabling out-of-sample predictions and uncertainty quantification. In this paper, we use GPs for large-scale spatial modeling. However, notice that GPs are not restricted to $\mathbb{R}^2$.

Even though GPs are a great tool for modeling spatial processes, and allow the prediction and uncertainty quantification of out-of-sample data points, they have one big limitation. When the number of spatial locations, n, in the training dataset is large, they become computationally intractable. The prediction and uncertainty quantification for an out-of-sample data point require solving a linear system, typically done by inverting (i.e., Cholesky factorization and triangular solves) the covariance matrix $\boldsymbol{\Sigma}$ which involves $\mathcal{O}(n^3)$ operations and $\mathcal{O}(n^2)$ memory. Furthermore, in order to estimate the model's optimal parameters, the inversion computations need to be carried out many times. Therefore, direct GPs, with Cholesky factorization and triangular solves, cannot be used when n is larger than around twenty thousand data points (Hensman et al. 2013), which is a common setting in modern spatial datasets. The development of advanced sensing technologies, mounted on satellites and aircraft, collecting massive amounts of spatial data, limits the use of

direct GPs in many applications (Katzfuss 2017; Heaton et al. 2019).

Various alternatives have been proposed to overcome the computational limitations of direct GPs for the analysis of spatial datasets. The vast majority of these methods rely on simplifying assumptions or approximations and can be grouped into four categories. (1) Low-rank methods aim at reducing the rank of the covariance matrix $\mathbf{\Sigma}$, examples of such methods include fixed rank kriging (Cressie and Johannesson 2008; Zammit-Mangion et al. 2018) and predictive processes (Finley et al. 2009)). (2) Sparse covariance methods introduce zeros into the covariance matrix $\mathbf{\Sigma}$ to allow for sparse computations, examples of such methods include spatial partitioning (Knorr-Held and Rasser 2000; Kim et al. 2005; Sang et al. 2011; Anderson et al. 2014; Heaton et al. 2017) and covariance tapering (Furrer et al. 2006; Furrer and Sain 2010). (3) Sparse precision methods introduce zeros into the precision matrix $\mathbf{\Sigma}^{-1}$ to speed up the computations required, examples of such methods include stochastic partial differential equations (Lindgren et al. 2011), latticeKrig (Nychka et al. 2015), multiresolution approximations (Katzfuss 2017; Jurek and Katzfuss 2021), nearest neighbor processes (Datta et al. 2016; Finley et al. 2019), and periodic embedding (Guinness 2019). (4) Algorithmic methods with new fitting schemes have been developed to reduce the computational cost, examples of such methods include laGP (Gramacy 2016), metakriging (Guhaniyogi and Banerjee 2018), and gapfill (Gerber et al. 2018). For a comprehensive overview of these methods, see the work by Heaton et al. (2019). The main drawback of these alternatives is that, if the assumptions on the structure of the covariance/precision matrix are not satisfied, the approximations can hinder the out-of-sample predictions and the uncertainty quantification.

Recently, there has been a push, led mostly by applied mathematicians and computer scientists, to relax these assumptions. Abdulah et al. (2018) and Salvaña et al. (2021) proposed to use state-of-the-art high-performance dense linear algebra libraries associated with edge parallel architectures to directly solve GPs. Others represent the covariance matrix $\mathbf{\Sigma}$ as a hierarchical matrix (Hackbusch 2015), where the relationships between different low-rank blocks and nested bases are exploited. The structure of the hierarchical matrix allows for efficient computations and reduces the storage requirements. In 2007, Börm and Garcke used hierarchical matrices to represent the GP covariance matrix for the first time. They were able to estimate the matrix using only $\mathcal{O}(nm)$ units of storage, where m is a parameter controlling the accuracy of the approximation.

The computation of the hierarchical matrix scales with $\mathcal{O}(nm \log n)$, which allows the evaluation of matrix-vector products in $\mathcal{O}(nm \log n)$ operations. Other operations like multiplication or inversion can be accomplished in almost linear complexity. The proposed representation was used for prediction and uncertainty quantification. However, the authors did not provide a solution for learning the GP and thus estimating the optimal model parameters. Since then, others have exploited the structure of hierarchical matrices to fit GPs.

On the one hand, some authors propose to exploit a particular representation of the covariance matrix to reduce the computational complexity of estimating a GP (Anitescu et al. 2012; Minden et al. 2017; Geoga et al. 2020; Keshavarzzadeh et al. 2021; Majumder et al. 2022). For example, Minden et al. (2017) propose to use recursive skeletonization factorization to represent the covariance matrix, and use an adaptation of the matrix peeling algorithm to learn GPs in $\mathcal{O}(n^{3/2})$ time under certain conditions. Geoga et al. (2020) exploit hierarchical off-diagonal low-rank matrices to represent the covariance matrix, and use the Hutchinson stochastic trace estimator to learn GPs in quasilinear $\mathcal{O}(n \log^2 n)$ time. Majumder et al. (2022) exploit Krylov subspaces, using Golub-Kahan bidiagonalization for the solution of linear systems, and use the Krylov subspace to estimate the objective function and its gradients in $\mathcal{O}(n \log n)$ time. This method uses Fast Fourier Transforms for the matrix-vector products, which requires the spatial data to be on a grid, which requires additional approximations for use on ungridded spatial datasets. On the other hand, some works focus on reducing the number of operations required to tune a GP given any hierarchical representation of the covariance matrix. In particular, Gardner et al. (2018) propose a blackbox matrix-matrix multiplication framework to estimate the optimal parameters of a given GP. The proposed framework uses a modified batched version of the conjugate gradient algorithm, reducing the asymptotic complexity of GP inference from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$. However, none of the existing frameworks propose a hierarchical matrix representation of the covariance matrix, which allows for customized matrix-free operations to efficiently learn a GP. By "matrix-free" operations, we mean that no dense matrices of size $n \times n$ are formed when learning a GP. By exploiting both the hierarchical representation of the covariance matrix and the matrix-free operations, the computational complexity of fitting a GP can be further reduced to $\mathcal{O}(n \log n)$ without requiring the data to be on a grid.

The goal of this paper is to develop a framework to learn GPs without assuming a specific structure of the covariance matrix in quasilinear time, to improve the out-of-sample predictions and uncertainty quantification. Instead of making strong assumptions about the structure of the covariance matrix, we capitalize on the properties of the underlying covariance function and approximate the covariance matrix, where the accuracy of the approximation can be controlled. We use Structured Matrix Approximation by Separation and Hierarchy (SMASH, Cai et al. (2018)) to represent the covariance matrix. We exploit the special structure of this approximation to efficiently estimate the model parameters in a matrix-free manner (i.e., without using dense matrices of size $n \times n$). Specifically, given the spatial points $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$, a tree structure is first constructed based on an adaptive partitioning of the domain $\mathcal{D}$ to facilitate approximation procedures of the covariance matrix $\boldsymbol{\Sigma}$. A rank-revealing factorization is applied to an initial analytic approximation so that a special structure is incorporated into the nested bases. As a consequence, the storage is significantly reduced, and a hierarchy of the spatial points is constructed. Operations associated with each level can be performed in parallel, which greatly reduces the computational time. Using SMASH to represent the covariance matrix and perform its associated matrix-free operations allows us to develop an efficient framework to find the optimal modeling parameters of a GP and perform out-of-sample predictions. The framework exploits the use of preconditioners to minimize the number of iterations required in the matrix-free operations and considers the numerical instability often encountered when dealing with GPs.

The main contributions of the paper are: (1) we develop a framework, smashGP, for matrix-free GP optimization via SMASH, which allows us to represent the covariance matrix $\boldsymbol{\Sigma}$ as an $\mathcal{H}^2$ matrix in $\mathcal{O}(n)$ time; (2) smashGP is able to estimate the unknown parameters of the model and predict the values of spatial observations at unobserved locations in $\mathcal{O}(n \log n)$ time thanks to fast matrix-vector products; (3) smashGP is able to overcome the numerical instability that often arises when dealing with GPs ; (4) smashGP is able to perform prediction and uncertainty quantification for a dataset with a million data points. Finally, as far as the authors are aware, we are the first to present a rigorous comparison between GPs methods imposing a special structure to the covariance matrix and GPs methods relying on hierarchical matrices and matrix-free operations. Methods developed to learn GPs using hierarchical matrices have been evaluated on their ability to

estimate the model parameters accurately. However, the out-of-sample predictions and uncertainty quantification capabilities have not been fully studied. Comparing these two approaches is critical as practitioners need guidelines to decide which framework should be preferred when learning GPs for spatial data.

The rest of the article is organized as follows. Section 2 provides a review on GPs for spatial data. Section 3 introduces SMASH and its matrix-free operations. In Section 4, we present the proposed methodology: smashGP. Simulation studies and real data analysis are conducted in Sections 5 and 6, respectively. Finally, we conclude in Section 7. The codes written for this paper can be found at `https://gitlab.com/libsmash_public/smashgp`. We provide supplementary materials with additional details on smashGP.

2 GPs for Spatial Data

As defined in the introduction, a spatial process $Y(\boldsymbol{s})$ for $\boldsymbol{s} \in \mathcal{D} \subset \mathbb{R}^2$ is said to follow a GP if any realization $\boldsymbol{Y} = (Y(\boldsymbol{s}_1), \ldots, Y(\boldsymbol{s}_n))^\top$ at the finite number of locations $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$ follows an n-variate Gaussian distribution, i.e., $\boldsymbol{Y} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. In what follows, without loss of generality, we assume that the mean function of the spatial process $Y(\boldsymbol{s})$ is constant. Therefore, $\boldsymbol{\mu} = \mu \mathbb{1}_n$ (derivations for a non-constant mean can be found in the supplementary materials). Additionally, we assume that $\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{R}$, where σ^2 is the process variance and $\boldsymbol{R} = \{R(\boldsymbol{s}_i, \boldsymbol{s}_j)\}_{i,j=1}^n$ is the $n \times n$ correlation matrix. The correlation function $R(\boldsymbol{s}_1, \boldsymbol{s}_2) : \mathcal{D}^2 \to [-1, 1]$ is a positive-definite kernel function. Popular kernel choices include the Gaussian kernel, the power exponential family of kernels, and the Matérn family of kernels (Roustant et al. 2012). All of these kernels depend on characteristic length-scale parameters $\boldsymbol{\theta}$ (Rasmussen and Williams 2005).

In almost all geophysical and environmental situations, sensors collecting spatial data only provide noisy observations. Therefore, instead of observing a realization $\boldsymbol{Y}$ of the spatial process $Y(\boldsymbol{s})$, we observe $\tilde{Y}_i = Y(\boldsymbol{s}_i) + \epsilon_i$, for $i = 1, \ldots, n$, where $\epsilon_i \sim N(0, \tau^2)$ is a realization of a noise random variable. If we assume the spatial process $Y(\boldsymbol{s})$ follows a GP and is stochastically independent from the Gaussian measurement errors ϵ_i, we have that the observed vector $\tilde{\boldsymbol{Y}} = \left(\tilde{Y}_1, \ldots, \tilde{Y}_n\right)^\top$ follows an n-variate Gaussian distribution with mean $\boldsymbol{\mu} = \mu \mathbb{1}_n$ and covariance

matrix $\tilde{\boldsymbol{\Sigma}} = \sigma^2 \boldsymbol{R} + \tau^2 \boldsymbol{I}_n$, where $\boldsymbol{I}_n$ is the $n \times n$ identity matrix. The likelihood of $\tilde{\boldsymbol{Y}}$ is

$$L(\mu, \sigma^2, \tau^2, \boldsymbol{\theta}; \tilde{\boldsymbol{Y}}) = \frac{1}{(2\pi)^{(n/2)} |\sigma^2 \boldsymbol{R} + \tau^2 \boldsymbol{I}_n|^{(1/2)}} \exp\left(-\frac{1}{2} (\tilde{\boldsymbol{Y}} - \mu \mathbb{1}_n)^\top (\sigma^2 \boldsymbol{R} + \tau^2 \boldsymbol{I}_n)^{-1} (\tilde{\boldsymbol{Y}} - \mu \mathbb{1}_n) \right) \quad (1)$$

Training a GP means finding the optimal values for the parameters, $\mu, \sigma^2, \tau^2, \boldsymbol{\theta}$, given the vector of noisy observations $\tilde{\boldsymbol{Y}}$. These parameters are commonly learned by minimizing the negative log-likelihood of the observed data. To reduce the optimization dimensionality, we define $v = \sigma^2 + \tau^2$, the total variance, and $\alpha = \sigma^2/(\sigma^2 + \tau^2)$, the proportion of variance explained by $Y(\boldsymbol{s})$. We rewrite $\tilde{\boldsymbol{\Sigma}} = v \boldsymbol{R}_\alpha$ with $\boldsymbol{R}_\alpha = \alpha \boldsymbol{R} + (1 - \alpha) \boldsymbol{I}_n$, note that $\boldsymbol{R}_\alpha$ is also symmetric positive-definite since $\alpha \in [0, 1]$. When minimizing the negative log-likelihood, the first-order conditions provide analytical solutions for μ and v:

$$\hat{\mu} = \frac{\mathbb{1}_n^\top \boldsymbol{R}_\alpha^{-1} \tilde{\boldsymbol{Y}}}{\mathbb{1}_n^\top \boldsymbol{R}_\alpha^{-1} \mathbb{1}_n} \qquad \hat{v} = \frac{1}{n} (\tilde{\boldsymbol{Y}} - \hat{\mu} \mathbb{1}_n)^\top \boldsymbol{R}_\alpha^{-1} (\tilde{\boldsymbol{Y}} - \hat{\mu} \mathbb{1}_n) \quad (2)$$

Thus, the concentrated log-likelihood depends only on α and $\boldsymbol{\theta}$:

$$-2 \log L(\hat{\mu}, \hat{v}, \alpha, \boldsymbol{\theta}; \tilde{\boldsymbol{Y}}) = n \log(2\pi) + n \log \hat{v} + \log |\boldsymbol{R}_\alpha| + n \quad (3)$$

The fact that α is bounded is convenient for optimization. Let $\boldsymbol{\phi} = (\alpha, \boldsymbol{\theta})^\top$, then the kth partial derivative of the log-likelihood is given by:

$$-2 \frac{\partial \log L(\hat{\mu}, \hat{v}, \alpha, \boldsymbol{\theta}; \tilde{\boldsymbol{Y}})}{\partial \phi_k} = -(\tilde{\boldsymbol{Y}} - \hat{\mu} \mathbb{1}_n)^\top \boldsymbol{R}_\alpha^{-1} \frac{\partial \boldsymbol{R}_\alpha}{\partial \phi_k} \boldsymbol{R}_\alpha^{-1} (\tilde{\boldsymbol{Y}} - \hat{\mu} \mathbb{1}_n)/\hat{v} + \operatorname{Tr}\left(\boldsymbol{R}_\alpha^{-1} \frac{\partial \boldsymbol{R}_\alpha}{\partial \phi_k} \right) \quad (4)$$

Under this scenario, L-BFGS-B (Zhu et al. 1997), an optimization algorithm in the family of quasi-Newton methods, is used to find the optimal parameters, $\hat{\boldsymbol{\phi}}$, given the observed data. Once the model is trained and the optimal parameters are estimated, inference and prediction of a spatial process are made by utilizing the conditional distribution, $p\left(Y(\boldsymbol{s}) | \tilde{\boldsymbol{Y}}\right)$. The predictive mean and the predictive variance, for $\boldsymbol{s} \in \mathcal{D}$, are given by:

$$m(\boldsymbol{s}) = \hat{\mu} + \hat{\alpha} \hat{\boldsymbol{r}}(\boldsymbol{s})^\top \hat{\boldsymbol{R}}_\alpha^{-1} (\tilde{\boldsymbol{Y}} - \hat{\mu} \mathbb{1}_n) \quad (5)$$

$$s^2(\boldsymbol{s}) = \hat{\sigma}^2 (1 - \hat{\alpha} \hat{\boldsymbol{r}}(\boldsymbol{s})^\top \hat{\boldsymbol{R}}_\alpha^{-1} \hat{\boldsymbol{r}}(\boldsymbol{s})) + (1 - \hat{\alpha} \hat{\boldsymbol{r}}(\boldsymbol{s})^\top \hat{\boldsymbol{R}}_\alpha^{-1} \mathbb{1}_n)^2 / (\mathbb{1}_n^\top (\hat{v} \hat{\boldsymbol{R}}_\alpha)^{-1} \mathbb{1}_n) \quad (6)$$

where $\hat{\boldsymbol{R}}_\alpha$ is the estimated correlation matrix and $\hat{\boldsymbol{r}}(\boldsymbol{s}) = (\hat{R}(\boldsymbol{s}, \boldsymbol{s}_i))_{i=1,\ldots,n}$ is the vector of estimated correlations between $Y(\boldsymbol{s})$ and $\tilde{\boldsymbol{Y}}$.

3 SMASH and Matrix-free Operations

As already mentioned, the main limitation to directly learn GPs from large-scale spatial data is the inversion of the correlation matrix, $\boldsymbol{R}_\alpha$, which requires $\mathcal{O}(n^3)$ operations and $\mathcal{O}(n^2)$ memory. To overcome this limitation, hierarchical matrices and matrix-free solves can be used. Hierarchical matrices are data-sparse representations of dense kernel matrices that exploit the block low-rank properties arising from the underlying kernel functions. In the case of GPs, recall that $\boldsymbol{R}_\alpha = \{R_\alpha(\boldsymbol{s}_i, \boldsymbol{s}_j)\}_{i,j=1}^n$ where $R_\alpha(\boldsymbol{s}_i, \boldsymbol{s}_j)$ is a positive-definite kernel function.

In this paper, we use SMASH $\mathcal{H}^2$ matrices, which can be constructed in linear time, and provide linear scaling matrix-vector products (Cai et al. 2018; Erlandson et al. 2020). We provide a brief overview here for completeness purposes. Without loss of generality, we describe how the dense correlation matrix $\boldsymbol{R}_\alpha$ is represented as a SMASH $\mathcal{H}^2$ matrix. This is done by constructing a SMASH $\mathcal{H}^2$ representation of $\boldsymbol{R}$, and then using the identity $\boldsymbol{R}_\alpha = \alpha\boldsymbol{R} + (1-\alpha)\boldsymbol{I}_n$. We also explain how the matrix-vector products are carried out by exploiting the structure of the hierarchical matrix.

To construct the SMASH $\mathcal{H}^2$ matrix representation of $\boldsymbol{R}$, we start by recursively splitting the domain $\mathcal{D}$ into subdomains, creating a tree structure $\mathcal{T}$. The basic idea of this partitioning algorithm is to recursively divide the domain into several subdomains until the number of points included in each resulting subdomain is less than a prescribed constant δ (usually much smaller than the number of points in the domain). Specifically, at level 1 of the tree, we begin with a root node corresponding to the entire domain. Then, from level l ($l \geq 2$), each subdomain obtained at level $l-1$ that contains more than δ points is bisected along the dimension with the largest range. Let L be the maximum level where the recursion stops. Then the information about the partitioning can be represented by a tree $\mathcal{T}$ with L levels. Such a splitting can be seen in Figure 1 for a 1D example. The adaptive partitioning guarantees that each subdomain corresponding to a leaf node contains a small number of points less than the prescribed constant δ. The choice of δ depends on the tolerance specified for the matrix approximation. Details on how δ is determined are provided in the supplementary materials.

Consider the node pair (i, j) in $\mathcal{T}$. Let S_{i} be the set of data points in node i and S_{j}' be the set of points in node j. The key idea is that if S_{i} and S_{j}' are far away, their correlation, denoted by

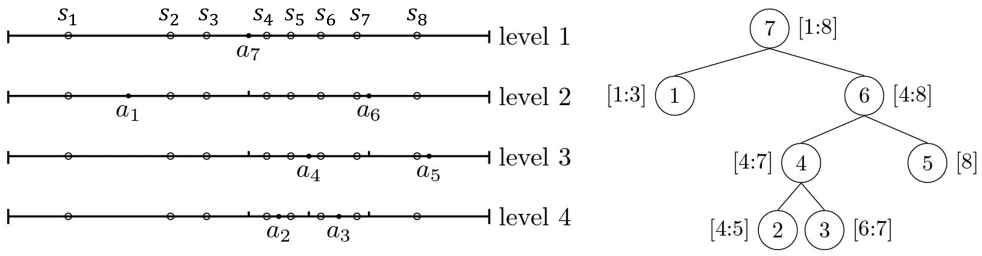


Figure 1: Illustration of an adaptive partitioning (adapted from (Cai et al. 2018)). Left: the domain $\mathcal{D}$ is recursively bisected until the number of points in each subdomain $\mathcal{D}_i$ centered at a_i is less than $\delta = 4$ (circled dots represent the data points s_i). Right: the corresponding tree $\mathcal{T}$ with indices of points stored at each node.

$\boldsymbol{R}|_{\mathrm{i}\times\mathrm{j}}$, can be represented as a low-rank matrix. Two questions need to be answered. (1) How is far away defined? (2) How to represent $\boldsymbol{R}|_{\mathrm{i}\times\mathrm{j}}$ as a low-rank matrix?

To define far away, we establish an admissibility condition (see Definitions S1 and S2 in the supplementary materials). If S_{i} and S'_{j} are well-separated in the sense of Definition S1, the submatrix $\boldsymbol{R}|_{\mathrm{i}\times\mathrm{j}}$ is called a farfield block, otherwise, it is called a nearfield block. The major difference between farfield and nearfield blocks is that each farfield block can be approximated by a low-rank matrix. In other words, if S_{i} and S'_{j} are well-separated, the farfield block $\boldsymbol{R}|_{\mathrm{i}\times\mathrm{j}}$ admits a low-rank approximation of the form

$$\boldsymbol{R}|_{\mathrm{i}\times\mathrm{j}} \approx \boldsymbol{U}_i \boldsymbol{A}_{i,j} \boldsymbol{V}_j^\top, \tag{7}$$

where $\boldsymbol{U}_i$ is the basis associated with points in S_{i}, $\boldsymbol{V}_j$ is the basis associated with points in S'_{j}, and $\boldsymbol{A}_{i,j}$ is the coupling matrix between S_{i} and S'_{j}. To define the bases, we use Lagrange polynomials (see the supplementary materials). Let n_i be the number of points in S_{i}, n_j be the number of points in S'_{j}, and r be the rank of the approximation. Then, the matrices $\boldsymbol{U}_i, \boldsymbol{A}_{i,j}$, and $\boldsymbol{V}_j$ have dimensions $n_i \times r$, $r \times r$, and $n_j \times r$, respectively. The choice of r depends on the tolerance specified for the matrix approximation. Details on how r is determined are provided in the supplementary materials.

To represent $\boldsymbol{R}|_{\mathrm{i}\times\mathrm{j}}$ as a low-rank matrix, we use the rank-revealing QR algorithm (see the supplementary materials). When $\boldsymbol{U}_i$ and $\boldsymbol{V}_j$ have more rows than columns, applying the rank-

revealing QR algorithm to $\boldsymbol{U}_i^\top$ and $\boldsymbol{V}_j^\top$ yields:

$$\boldsymbol{U}_i = \boldsymbol{P}_i \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{G}_i \end{bmatrix} \boldsymbol{U}_i|_{\hat{\mathrm{i}}}, \quad \boldsymbol{V}_j = \boldsymbol{Q}_j \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{H}_j \end{bmatrix} \boldsymbol{V}_j|_{\hat{\mathrm{j}}} \tag{8}$$

where $\boldsymbol{P}_i$ and $\boldsymbol{Q}_j$ are permutation matrices, $\boldsymbol{G}_i$ and $\boldsymbol{H}_j$ are dense matrices, $\boldsymbol{U}_i|_{\hat{\mathrm{i}}}$ is a matrix made up of selected rows of $\boldsymbol{U}_i$, and $\boldsymbol{V}_j|_{\hat{\mathrm{j}}}$ is a matrix made up of selected rows of $\boldsymbol{V}_j$. $\hat{\mathrm{i}}$ and $\hat{\mathrm{j}}$ represent subsets of i and j, respectively. Substituting the above equation in Equation (7) leads to another low-rank approximation for $\boldsymbol{R}|_{\mathrm{i}\times\mathrm{j}}$:

$$\boldsymbol{R}|_{\mathrm{i}\times\mathrm{j}} \approx \boldsymbol{P}_i \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{G}_i \end{bmatrix} \boldsymbol{U}_i|_{\hat{\mathrm{i}}} \boldsymbol{A}_{i,j} (\boldsymbol{V}_j|_{\hat{\mathrm{j}}})^\top \left(\boldsymbol{Q}_j \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{H}_j \end{bmatrix} \right)^\top \approx \boldsymbol{P}_i \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{G}_i \end{bmatrix} \boldsymbol{R}|_{\hat{\mathrm{i}}\times\hat{\mathrm{j}}} \left(\boldsymbol{Q}_j \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{H}_j \end{bmatrix} \right)^\top \tag{9}$$

A major advantage of this approximation is a storage reduction. Now, only four index sets $\{\boldsymbol{P}_i, \boldsymbol{Q}_j, \hat{\mathrm{i}}, \hat{\mathrm{j}}\}$ and two smaller dense matrices $\{\boldsymbol{G}_i, \boldsymbol{H}_j\}$ need to be stored rather than two dense matrices $\{\boldsymbol{U}_i, \boldsymbol{V}_j\}$.

Recall that our goal is to represent the correlation matrix $\boldsymbol{R}$ as a SMASH $\mathcal{H}^2$ matrix (see Definition S4 in the supplementary materials). A key property of $\mathcal{H}^2$ matrices is that the bases at one level of the tree $\mathcal{T}$ can be expressed using the bases of the children. For example, assume parent node p has children nodes $c_1, \ldots, c_k$. We can get the basis $\boldsymbol{U}_p$ for the parent node from the children's basis $\{\boldsymbol{U}_{c_1}, \ldots, \boldsymbol{U}_{c_k}\}$ and some transfer matrices $\{\boldsymbol{B}_{c_1}, \ldots, \boldsymbol{B}_{c_k}\}$. For more details, please refer to the supplementary materials. A similar process can be applied to obtain the row-basis $\boldsymbol{V}_p$. Therefore, we can write:

$$\boldsymbol{U}_p = \begin{bmatrix} \boldsymbol{U}_{c_1} \boldsymbol{B}_{c_1} \\ \vdots \\ \boldsymbol{U}_{c_k} \boldsymbol{B}_{c_k} \end{bmatrix}, \quad \boldsymbol{V}_p = \begin{bmatrix} \boldsymbol{V}_{c_1} \boldsymbol{C}_{c_1} \\ \vdots \\ \boldsymbol{V}_{c_k} \boldsymbol{C}_{c_k} \end{bmatrix}. \tag{10}$$

Hence, only the matrices $\boldsymbol{U}_i$ and $\boldsymbol{V}_i$ for all leaf nodes must be stored. Matrices $\boldsymbol{U}_p$ and $\boldsymbol{V}_p$ for a non-leaf node p can be obtained via transfer matrices which require much less storage.

In summary, the construction of SMASH $\mathcal{H}^2$ matrices involves creating a tree $\mathcal{T}$ using adaptive partitioning, computing the basis matrices $\boldsymbol{U}$, $\boldsymbol{V}$ at the leaf nodes, along with the transfer matrices $\boldsymbol{B}$, $\boldsymbol{C}$, and the coupling matrices $\boldsymbol{A}$. In particular, each leaf node i is assigned four matrices $\{\boldsymbol{U}_i, \boldsymbol{V}_i, \boldsymbol{B}_i, \boldsymbol{C}_i\}$ and each non-leaf node i is assigned two matrices $\{\boldsymbol{B}_i, \boldsymbol{C}_i\}$. For the leaf nodes,

$\boldsymbol{U}_i$ and $\boldsymbol{V}_i$ are obtained by using the rank-revealing QR algorithm. One of the advantages of using the SMASH $\mathcal{H}^2$ representation of the correlation matrix $\boldsymbol{R}$ is that the user can specify the level of approximation by using a tolerance parameter tol. The tol parameter controls the L2 norm of the reconstruction error, i.e., $||\boldsymbol{R} - \hat{\boldsymbol{R}}||_2^2 \leq tol$. Throughout the paper, we used $tol = 10^{-7}$ to ensure an accurate covariance matrix reconstruction.

A schematic representation of the construction of a SMASH $\mathcal{H}^2$ matrix is presented in Figure 2. Figure 2a represents the tree $\mathcal{T}$ for a 1D domain. Figure 2b represents the nearfield (black) and farfield (gray) blocks from the tree structure. Figure 2c shows the compression for leaf and parent nodes. Thanks to the nested basis structure of the SMASH $\mathcal{H}^2$ representation, parallel computing, and shared memory can be used to approximate the correlation matrix $\boldsymbol{R}$.

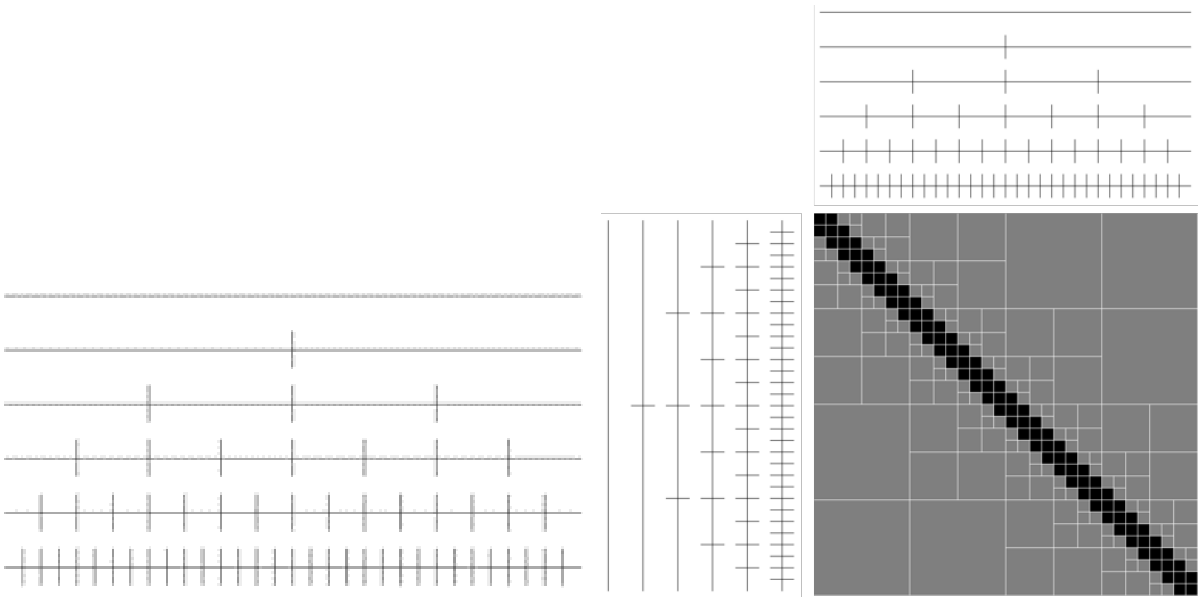
Once the SMASH $\mathcal{H}^2$ representation of the kernel matrix is constructed, the matrix-vector products can be performed matrix-free. Rather than performing a large matrix-vector product of size n resulting in a cost of $\mathcal{O}(n^2)$, multiplications by small low-rank matrices are performed and aggregated hierarchically. More details on the matrix-vector products are presented in the supplementary materials. As seen in (Erlandson et al. 2020), the time taken by matrix-free matrix-vector products scales linearly with the number of points. It is worth noting that the time for a matrix-free matrix-vector product depends on the approximation tolerance specified by the user. However, a 10^6 decrease in the L2 norm of the approximation can be achieved with just eight times more computational time.

In the next section, we describe how we use the SMASH $\mathcal{H}^2$ representation of $\boldsymbol{R}_\alpha$ and the corresponding matrix-free operations to learn the optimal parameters of a GP model.

4 Large-scale Spatial Modeling via smashGP

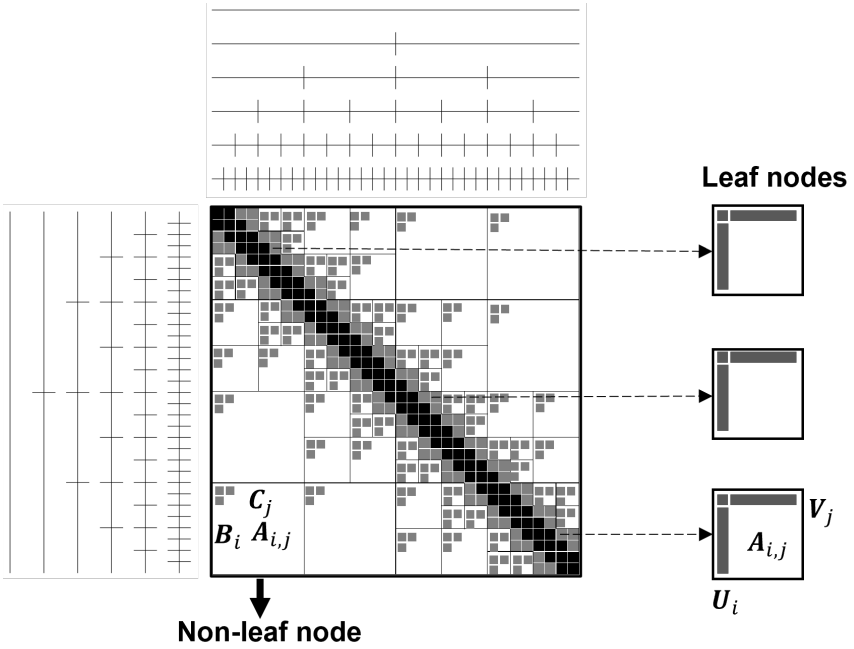
In this section, we present the smashGP framework for large-scale spatial modeling. Then, we introduce practical considerations to speed up the computations and avoid numerical instability.

We start by specifying the kernels implemented in the current version of smashGP. Recall that SMASH $\mathcal{H}^2$ matrices exploit the block low-rank properties arising from positive-definite kernel functions. One convenient way of getting positive definite kernel functions for two or more dimensions is to take tensor products of 1-dimensional positive definite kernel functions.



(a) Hierarchical partitioning of a 1D domain.

(b) Corresponding nearfield (black) and farfield (gray) correlations.



(c) Leaf node and non-leaf node compression associated with the farfield blocks.

Figure 2: Construction of SMASH $\mathcal{H}^2$ matrices.

Such kernels, called separable kernels, are the most commonly used in the computer experiments literature (Roustant et al. 2012). For the spatial setting, these kernels have the form:

$$R(\boldsymbol{s}_i, \boldsymbol{s}_j) := g(h_1; \theta) g(h_2; \theta) \tag{11}$$

where $\boldsymbol{s}_i, \boldsymbol{s}_j \in \mathcal{D} \subseteq \mathbb{R}^2$, $h_d = s_{id} - s_{jd}$, for $d = 1, 2$, g is a 1-dimensional positive definite kernel, and $\theta > 0$ is the length-scale parameter of the kernel. The current version of smashGP allows the user to select between three of the most commonly used kernels (Roustant et al. 2012):

$$\text{Gaussian:} \quad g(h; \theta) = e^{\frac{-h^2}{2\theta^2}}, \tag{12}$$

$$\text{Matérn } \nu = 5/2: \quad g(h; \theta) = \left(1 + \frac{\sqrt{5}|h|}{\theta} + \frac{5h^2}{3\theta^2}\right) e^{\frac{-\sqrt{5}|h|}{\theta}}, \tag{13}$$

$$\text{Matérn } \nu = 3/2: \quad g(h; \theta) = \left(1 + \frac{\sqrt{3}|h|}{\theta}\right) e^{\frac{-\sqrt{3}|h|}{\theta}}. \tag{14}$$

The above kernels will result in different levels of smoothness. With the Gaussian kernel, the sample paths of the associated GP have derivatives of all orders and, therefore, are very smooth. With the Matérn kernel with parameter ν, the GP is differentiable at order k if and only if $\nu > k$. Thus, with $\nu = 5/2$, the process is twice differentiable and, with $\nu = 3/2$, only once. When $\nu \to \infty$, the Matérn kernel coincides with the Gaussian kernel. The general Matérn covariance depends on the modified Bessel function and has not been implemented yet. We only consider at least one-time differentiable kernels as differentiability is needed for the smashGP framework. The three kernels considered correspond to commonly needed levels of smoothness encountered in practice (Roustant et al. 2012).

4.1 smashGP Framework

Learning a GP for large-scale spatial modeling involves estimating the unknown parameters of the model, μ, v, α, and θ, (Eq. 2, 3, and 4, Sec. 2), predicting the values of spatial observations at unobserved locations (Eq. 5, Sec. 2), and quantifying the uncertainty of the predictions (Eq. 6, Sec. 2).

In what follows, we first assume that the model parameters are known and propose a prediction and uncertainty quantification framework using matrix-free linear solvers and matrix-vector

products. Then, we present the framework to estimate the optimal model parameters, which involves computing a matrix-free log-likelihood with its partial derivatives, which rely upon log-determinants, traces, and matrix-vector products.

4.1.1 smashGP prediction and uncertainty quantification framework

For a sample $\boldsymbol{s} \in \mathcal{D}$, the predictive mean, $m(\boldsymbol{s})$, and the predictive variance, $s^2(\boldsymbol{s})$, are defined in Equations 5 and 6, in Section 2. Here, we assume that the values for μ, v, α, and θ are known. Our goal is to calculate $m(\boldsymbol{s})$ and $s^2(\boldsymbol{s})$ without forming the dense correlation matrix, $\boldsymbol{R}_\alpha$, which appears in three instances:

- $\boldsymbol{x}_1 = \boldsymbol{R}_\alpha^{-1}(\tilde{\boldsymbol{Y}} - \hat{\mu}\mathbb{1}_n)$ – (Eq. 5, Sec. 2)
- $\boldsymbol{x}_2 = \boldsymbol{R}_\alpha^{-1}\boldsymbol{r}(\boldsymbol{s})$ – (Eq. 6, Sec. 2)
- $\boldsymbol{x}_3 = \boldsymbol{R}_\alpha^{-1}\mathbb{1}_n$ – (Eq. 6, Sec. 2).

Notice that calculating $\boldsymbol{x}_1, \boldsymbol{x}_2$, and $\boldsymbol{x}_3$ is equivalent to solving the following systems of linear equations:

- $\boldsymbol{R}_\alpha \boldsymbol{x}_1 = (\tilde{\boldsymbol{Y}} - \hat{\mu}\mathbb{1}_n)$
- $\boldsymbol{R}_\alpha \boldsymbol{x}_2 = \boldsymbol{r}(\boldsymbol{s})$
- $\boldsymbol{R}_\alpha \boldsymbol{x}_3 = \mathbb{1}_n$.

The Preconditioned Conjugate Gradient (PCG) algorithm is an iterative method, able to solve such systems without requiring direct access to entries of $\boldsymbol{R}_\alpha$ (Saad 2003). This algorithm is one of the best methods for this purpose for symmetric positive-definite matrices. Thus, we use the SMASH $\mathcal{H}^2$ representation of $\boldsymbol{R}_\alpha$, and its matrix-free operations, presented in Section 3, together with the PCG algorithm to solve the systems of linear equations and compute $\boldsymbol{x}_1, \boldsymbol{x}_2$, and $\boldsymbol{x}_3$. Algorithm 1 presents the computations of the predictive mean, $m(\boldsymbol{s})$, and the predictive variance, $s^2(\boldsymbol{s})$, using smashGP.

Algorithm 1 smashGP prediction and uncertainty quantification framework.

INPUT: $\boldsymbol{s}$, $\tilde{\boldsymbol{Y}}$, μ, v, α, θ

OUTPUT: predictive mean $m(\boldsymbol{s})$, predictive variance $s^2(\boldsymbol{s})$

Step 1: Using the SMASH $\mathcal{H}^2$ representation of $\boldsymbol{R}_\alpha$, its matrix-free operations, and the PCG algorithm compute:

- $\boldsymbol{x}_1 = \boldsymbol{R}_\alpha^{-1}(\tilde{\boldsymbol{Y}} - \hat{\mu}\mathbb{1}_n)$
- $\boldsymbol{x}_2 = \boldsymbol{R}_\alpha^{-1}\boldsymbol{r}(\boldsymbol{s})$
- $\boldsymbol{x}_3 = \boldsymbol{R}_\alpha^{-1}\mathbb{1}_n$

Step 2: Using Equation 5, in Section 2, compute the predictive mean.

- $m(\boldsymbol{s}) = \mu + \alpha\boldsymbol{r}(\boldsymbol{s})^\top \boldsymbol{x}_1$

Step 3: Using Equation 6, in Section 2, compute the predictive variance.

- $\sigma^2 = \alpha v$
- $s^2(\boldsymbol{s}) = \sigma^2(1 - \alpha\boldsymbol{r}(\boldsymbol{s})^\top \boldsymbol{x}_2) + v(1 - \alpha\boldsymbol{r}(\boldsymbol{s})^\top \boldsymbol{x}_3)^2/(\mathbb{1}_n^\top \boldsymbol{x}_3)$

4.1.2 smashGP parameter estimation framework

Training a GP means finding the optimal values for the parameters μ, v, α, and θ. This is achieved by minimizing the negative log-likelihood of the vector of observations $\tilde{\boldsymbol{Y}}$, i.e.,

$$(\hat{\mu}, \hat{v}, \hat{\alpha}, \hat{\theta}) = \arg\min_{\mu,v,\alpha,\theta} \frac{n}{2}\log 2\pi + \frac{n}{2}\log v + \frac{1}{2}\log|\boldsymbol{R}_\alpha| + \frac{1}{2v}(\tilde{\boldsymbol{Y}} - \mu\mathbb{1}_n)^\top \boldsymbol{R}_\alpha^{-1}(\tilde{\boldsymbol{Y}} - \mu\mathbb{1}_n). \quad (15)$$

Next, we explain how to estimate each one of these parameters.

- **Process mean, μ, and total variance, v**

  Writing the first order conditions in terms of μ and v results in the following analytical expressions:

  $$\hat{\mu} = \frac{\mathbb{1}_n^\top \boldsymbol{R}_\alpha^{-1}\tilde{\boldsymbol{Y}}}{\mathbb{1}_n^\top \boldsymbol{R}_\alpha^{-1}\mathbb{1}_n} \text{ and } \hat{v} = \frac{1}{n}(\tilde{\boldsymbol{Y}} - \hat{\mu}\mathbb{1}_n)^\top \boldsymbol{R}_\alpha^{-1}(\tilde{\boldsymbol{Y}} - \hat{\mu}\mathbb{1}_n).$$

  Therefore, to estimate μ and v, we need to compute $\boldsymbol{R}_\alpha^{-1}\tilde{\boldsymbol{Y}}$, $\boldsymbol{R}_\alpha^{-1}\mathbb{1}_n$, and $\boldsymbol{R}_\alpha^{-1}(\tilde{\boldsymbol{Y}} - \hat{\mu}\mathbb{1}_n)$, without forming the dense correlation matrix $\boldsymbol{R}_\alpha$. We achieve this by using the SMASH $\mathcal{H}^2$ representation of $\boldsymbol{R}_\alpha$, its matrix-free operations, and the PCG algorithm in a similar fashion as we did in the smashGP prediction and uncertainty quantification framework.

- **Proportion of variance explained by $Y(\boldsymbol{s})$, α, and length scale, θ**

  With the estimates for μ and v, we estimate α and θ by minimizing the concentrated negative log-likelihood, i.e.,

$$(\hat{\alpha}, \hat{\theta}) = \arg\min_{\alpha,\theta} n \log \hat{v} + \log |\boldsymbol{R}_\alpha| = \arg\min_{\alpha,\theta} l(\alpha, \theta). \tag{16}$$

  Recall that α is bounded between $[0, 1]$, therefore, to solve the optimization problem, we can use the L-BFGS-B algorithm (Zhu et al. 1997). To use this algorithm, we need to compute the partial derivatives of the optimization function with respect to α and θ, i.e,

$$\frac{\partial l(\alpha,\theta)}{\partial \alpha} = -(\tilde{\boldsymbol{Y}} - \hat{\mu}\mathbb{1}_n)^\top \boldsymbol{R}_\alpha^{-1} \frac{\partial \boldsymbol{R}_\alpha}{\partial \alpha} \boldsymbol{R}_\alpha^{-1} (\tilde{\boldsymbol{Y}} - \hat{\mu}\mathbb{1}_n)/\hat{v} + \operatorname{Tr}\left(\boldsymbol{R}_\alpha^{-1} \frac{\partial \boldsymbol{R}_\alpha}{\partial \alpha}\right), \tag{17}$$

$$\frac{\partial l(\alpha,\theta)}{\partial \theta} = -(\tilde{\boldsymbol{Y}} - \hat{\mu}\mathbb{1}_n)^\top \boldsymbol{R}_\alpha^{-1} \frac{\partial \boldsymbol{R}_\alpha}{\partial \theta} \boldsymbol{R}_\alpha^{-1} (\tilde{\boldsymbol{Y}} - \hat{\mu}\mathbb{1}_n)/\hat{v} + \operatorname{Tr}\left(\boldsymbol{R}_\alpha^{-1} \frac{\partial \boldsymbol{R}_\alpha}{\partial \theta}\right). \tag{18}$$

  First, notice that to compute the derivatives, we need to calculate $\boldsymbol{x}_1 = \boldsymbol{R}_\alpha^{-1}(\tilde{\boldsymbol{Y}} - \hat{\mu}\mathbb{1}_n)$. We achieve this by using the SMASH $\mathcal{H}^2$ representation of $\boldsymbol{R}_\alpha$, its matrix-free operations, and the PCG algorithm as we have explained before. Additionally, to compute $l(\alpha, \theta)$ and its derivatives, we need to calculate $\log |\boldsymbol{R}_\alpha|$, $\boldsymbol{x}_1^\top \frac{\partial \boldsymbol{R}_\alpha}{\partial \alpha} \boldsymbol{x}_1$, $\boldsymbol{x}_1^\top \frac{\partial \boldsymbol{R}_\alpha}{\partial \theta} \boldsymbol{x}_1$, $\operatorname{Tr}(\boldsymbol{R}_\alpha^{-1} \frac{\partial \boldsymbol{R}_\alpha}{\partial \alpha})$, and $\operatorname{Tr}(\boldsymbol{R}_\alpha^{-1} \frac{\partial \boldsymbol{R}_\alpha}{\partial \theta})$. Next, we explain how to compute these.

  - $\log |\boldsymbol{R}_\alpha|$

    From linear algebra, we have that $\log |\boldsymbol{R}_\alpha| = \operatorname{Tr}(\log(\boldsymbol{R}_\alpha)) = \sum_{i=1}^n \log(\lambda_i)$, where λ_i, $i = 1, \ldots, n$, are the eigenvalues of $\boldsymbol{R}_\alpha$. As the correlation matrix $\boldsymbol{R}_\alpha$ is symmetric and positive-definite, one method for computing $\log |\boldsymbol{R}_\alpha|$ is using the Stochastic Lanczos Quadrature (SLQ) (Ubaru et al. 2017). This method utilizes the matrix-free Lanczos algorithm to provide eigenvalue estimates, which can then be used to estimate the trace of matrix functions of a symmetric and positive-definite matrix. In this case, we have that $\log |\boldsymbol{R}_\alpha| = \sum_{i=1}^n \log(\hat{\lambda}_i)$, where $\hat{\lambda}_i$, $i = 1, \ldots, n$, are the eigenvalue estimates by the SLQ algorithm without directly computing $\boldsymbol{R}_\alpha$.

  - $\operatorname{Tr}(\boldsymbol{R}_\alpha^{-1} \frac{\partial \boldsymbol{R}_\alpha}{\partial \alpha})$

    Recall that $\boldsymbol{R}_\alpha = \alpha \boldsymbol{R} + (1 - \alpha)\boldsymbol{I}_n$. Therefore, $\frac{\partial \boldsymbol{R}_\alpha}{\partial \alpha} = \boldsymbol{R} - \boldsymbol{I}_n$. In consequence, $\operatorname{Tr}(\boldsymbol{R}_\alpha^{-1} \frac{\partial \boldsymbol{R}_\alpha}{\partial \alpha}) = \operatorname{Tr}((\alpha \boldsymbol{R} + (1 - \alpha)\boldsymbol{I}_n)^{-1}(\boldsymbol{R} - \boldsymbol{I}_n))$. Denote by γ_i, $i = 1, \ldots, n$, the

eigenvalues of $\boldsymbol{R}$. Since $\boldsymbol{R}$ is symmetric and positive-definite, we can use the matrix-free SLQ algorithm to provide estimates for these eigenvalues, i.e., $\hat{\gamma}_i$, $i = 1, \ldots, n$. Therefore, $\text{Tr}(\boldsymbol{R}_\alpha^{-1} \frac{\partial \boldsymbol{R}_\alpha}{\partial \alpha}) = \sum_{i=1}^{n} (\alpha\hat{\gamma}_i + (1-\alpha))^{-1}(\hat{\gamma}_i - 1)$.

- $\boldsymbol{x}_1^\top \frac{\partial \boldsymbol{R}_\alpha}{\partial \alpha} \boldsymbol{x}_1$

  We have that $\boldsymbol{x}_1^\top \frac{\partial \boldsymbol{R}_\alpha}{\partial \alpha} \boldsymbol{x}_1 = \boldsymbol{x}_1^\top (\boldsymbol{R} - \boldsymbol{I}_n)\boldsymbol{x}_1 = \boldsymbol{x}_1^\top \boldsymbol{R}\boldsymbol{x}_1 - \boldsymbol{x}_1^\top \boldsymbol{x}_1$. We can calculate $\boldsymbol{R}\boldsymbol{x}_1$ without directly computing $\boldsymbol{R}$ by using its SMASH $\mathcal{H}^2$ representation and its matrix-free operations.

Notice that the computations of $\text{Tr}(\boldsymbol{R}_\alpha^{-1} \frac{\partial \boldsymbol{R}_\alpha}{\partial \theta})$ and $\boldsymbol{x}_1^\top \frac{\partial \boldsymbol{R}_\alpha}{\partial \theta} \boldsymbol{x}_1$ depend on the kernel used. Here, without loss of generality, we present the details for the computations when the kernel is Gaussian. These can be easily adapted for the Matérn $\nu = 5/2$ and Matérn $\nu = 3/2$ kernels.

- $\text{Tr}(\boldsymbol{R}_\alpha^{-1} \frac{\partial \boldsymbol{R}_\alpha}{\partial \theta})$

  We have that $\frac{\partial \boldsymbol{R}_\alpha}{\partial \theta} = \frac{\boldsymbol{E}}{\theta^3} \circ \boldsymbol{R}$ where $\boldsymbol{E}_{i,j} = ||\boldsymbol{s}_i - \boldsymbol{s}_j||_2^2$ is the squared euclidean distance between two training data points, $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$ in $\mathcal{D}$, and $\circ$ is the elementwise product between the two matrices. To compute $\text{Tr}(\boldsymbol{R}_\alpha^{-1} \frac{\partial \boldsymbol{R}_\alpha}{\partial \theta})$, we need to introduce the Hutchinson estimator. For a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, we have that $\text{Tr}(\boldsymbol{A}) \approx \frac{1}{K} \sum_{k=1}^{K} \boldsymbol{z}_k^\top \boldsymbol{A} \boldsymbol{z}_k$, for a sufficiently large K, where the n entries of each $\boldsymbol{z}_k$, $k = 1, \ldots, K$, are chosen between -1 and 1 with probability 0.5 (Hutchinson 1989). Let $\boldsymbol{A} = \boldsymbol{R}_\alpha^{-1} \frac{\partial \boldsymbol{R}_\alpha}{\partial \theta} = \boldsymbol{R}_\alpha^{-1} \left(\frac{\boldsymbol{E}}{\theta^3} \circ \boldsymbol{R}\right)$. For each $k = 1, \ldots, K$, we first compute $\tilde{\boldsymbol{z}}_k = \left(\frac{\boldsymbol{E}}{\theta^3} \circ \boldsymbol{R}\right) \boldsymbol{z}_k$, by using the SMASH $\mathcal{H}^2$ representation of $\left(\frac{\boldsymbol{E}}{\theta^3} \circ \boldsymbol{R}\right)$ and its matrix-free operations. Then, we compute $\boldsymbol{R}_\alpha^{-1} \tilde{\boldsymbol{z}}_k$ by using the SMASH $\mathcal{H}^2$ representation of $\boldsymbol{R}_\alpha$, its matrix-free operations, and the PCG algorithm.

- $\boldsymbol{x}_1^\top \frac{\partial \boldsymbol{R}_\alpha}{\partial \theta} \boldsymbol{x}_1$

  We have that $\boldsymbol{x}_1^\top \frac{\partial \boldsymbol{R}_\alpha}{\partial \theta} \boldsymbol{x}_1 = \boldsymbol{x}_1^\top \left(\frac{\boldsymbol{E}}{\theta^3} \circ \boldsymbol{R}\right) \boldsymbol{x}_1$. Once again, to compute $\left(\frac{\boldsymbol{E}}{\theta^3} \circ \boldsymbol{R}\right) \boldsymbol{x}_1$ we use the SMASH $\mathcal{H}^2$ representation of $\left(\frac{\boldsymbol{E}}{\theta^3} \circ \boldsymbol{R}\right)$ and its matrix-free operations.

We now have all of the building blocks required for calculating the log-likelihood, the partial derivatives of the log-likelihood, and the predictive mean and variance for a new sample. These are the components of the proposed smashGP framework. Next, we present some practical considerations that allow to speed up the computations and avoid numerical instability.

4.2 Practical Considerations

4.2.1 Blocked Calculations

First, we incorporate blocked computations to the smashGP framework to further reduce the computation time. Let us define blocking with a simple example. Suppose we want to perform a series of dot products. We want to calculate $w_k = \boldsymbol{u_k}^\top \boldsymbol{v}$, for vectors $\boldsymbol{u}_k$, $k = 1, \ldots, K$, and $\boldsymbol{v}$. This could be achieved with the following lines of code:

for $k = 1, \ldots, K$ **do**

$w_k \leftarrow \boldsymbol{u}_k^\top \boldsymbol{v}$.

end for

Alternatively, one could use the following blocked line of code:

$\boldsymbol{w} = \boldsymbol{U}^\top \boldsymbol{v}$,

where $\boldsymbol{w} = [w_1, w_2, \ldots, w_K]^\top$ and $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_K]$. Blocking can result in more efficient use of the hardware and increased parallelism, thus reducing the computation time (Dongarra et al. 1990).

In smashGP, blocking is implemented in three different instances: (1) computation of the predictive mean and variance, (2) use of the Hutchinson estimator, and (3) use of the SLQ algorithm. Next, we explain how blocking is achieved in each one of these instances.

- **Predictive mean and variance**

  Instead of predicting the mean and variance separately for unobserved locations, $\boldsymbol{s}'_1, \boldsymbol{s}'_2, \ldots, \boldsymbol{s}'_K \in \mathcal{D}$, blocking can be used to consider multiple predictions at once. The blocked version of the predictive mean computation (Eq. 5, Sec. 2) is:

  $$m(\boldsymbol{S}) = \hat{\mu}\mathbb{1}_K + \hat{\alpha}\hat{\boldsymbol{r}}(\boldsymbol{S})(\hat{\boldsymbol{R}}_\alpha^{-1}(\tilde{\boldsymbol{Y}} - \hat{\mu}\mathbb{1}_n)), \tag{19}$$

  where $\boldsymbol{S} = [\boldsymbol{s}'_1, \boldsymbol{s}'_2, \ldots, \boldsymbol{s}'_K]$ and $\hat{\boldsymbol{r}}(\boldsymbol{S})$ is a $K \times n$ matrix such that the kth row is equal to $\hat{\boldsymbol{r}}(\boldsymbol{s}'_k)$. Observe that we replace K dot products with a $K \times n$ matrix-vector product, which reduces to the setting we described earlier.

Similarly, the blocked version of the predictive variance computation (Eq. 6, Sec. 2) is:

$$s^2(\boldsymbol{S}) = \hat{\sigma}^2 \left(\mathbb{1}_K - \left(\hat{\alpha} \mathbb{1}_N^\top \left(\hat{\boldsymbol{r}}(\boldsymbol{S})^\top \circ \left(\hat{\boldsymbol{R}}_\alpha^{-1} \hat{\boldsymbol{r}}(\boldsymbol{S})^\top \right) \right) \right)^\top \right) + \frac{\left(\mathbb{1}_K - \hat{\alpha} \hat{\boldsymbol{r}}(\boldsymbol{S}) \hat{\boldsymbol{R}}_\alpha^{-1} \mathbb{1}_n \right)^{\circ 2}}{\mathbb{1}_n^\top (\hat{v} \hat{\boldsymbol{R}}_\alpha)^{-1} \mathbb{1}_n}, \quad (20)$$

where $^{\circ 2}$ denotes an elementwise square. Algorithm 2 summarizes the block computations of the predictive mean and variance using smashGP.

Algorithm 2 smashGP block prediction and uncertainty quantification framework.

INPUT: $\boldsymbol{S} = [\boldsymbol{s}'_1, \boldsymbol{s}'_2, \ldots, \boldsymbol{s}'_K]$, $\tilde{\boldsymbol{Y}}$, $\hat{\mu}, \hat{v}, \hat{\alpha}, \hat{\theta}$

OUTPUT: predictive mean $m(\boldsymbol{S}) \in \mathbb{R}^K$, predictive variance $s^2(\boldsymbol{S}) \in \mathbb{R}^K$

Step 1: Using the SMASH $\mathcal{H}^2$ representation of $\hat{\boldsymbol{R}}_\alpha$, its-matrix-free operations, and the PCG algorithm compute:

- $\boldsymbol{x}_1 = \hat{\boldsymbol{R}}_\alpha^{-1} (\tilde{\boldsymbol{Y}} - \hat{\mu} \mathbb{1}_n)$
- $\boldsymbol{x}_2 = \hat{\boldsymbol{R}}_\alpha^{-1} \hat{\boldsymbol{r}}(\boldsymbol{S})^\top$
- $\boldsymbol{x}_3 = \hat{\boldsymbol{R}}_\alpha^{-1} \mathbb{1}_n$

Step 2: Using Equation 19, compute the predictive mean.

- $m(\boldsymbol{S}) = \hat{\mu} \mathbb{1}_K + \alpha \hat{\boldsymbol{r}}(\boldsymbol{S}) \boldsymbol{x}_1$

Step 3: Using Equation 20, compute the predictive variance.

- $\hat{\sigma}^2 = \hat{\alpha} \hat{v}$
- $s^2(\boldsymbol{S}) = \hat{\sigma}^2 (\mathbb{1}_K - (\hat{\alpha} \mathbb{1}_N^\top (\hat{\boldsymbol{r}}(\boldsymbol{S})^\top \circ \boldsymbol{x}_2))^\top) + \hat{v} (\mathbb{1}_K - \alpha \hat{\boldsymbol{r}}(\boldsymbol{S}) \boldsymbol{x}_3)^{\circ 2} / (\mathbb{1}_n^\top \boldsymbol{x}_3)$

- **Hutchinson estimator**

  Recall that we use the Hutchinson estimator to estimate the trace of $\boldsymbol{A} = \boldsymbol{R}_\alpha^{-1} \frac{\partial \boldsymbol{R}_\alpha}{\partial \theta}$. We have that $\mathrm{Tr}(\boldsymbol{A}) \approx \frac{1}{K} \sum_{k=1}^K \boldsymbol{z}_k^\top \boldsymbol{A} \boldsymbol{z}_k$, for a sufficiently large K, where the n entries of each $\boldsymbol{z}_k$, $k = 1, \ldots, K$, are chosen between -1 and 1 with probability 0.5 (Hutchinson 1989). To accelerate the computations, we use a block operation. Let us construct the $n \times K$ matrix $\boldsymbol{Z} = [\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_K]$. To estimate $\mathrm{Tr}(\boldsymbol{A})$, we first compute the matrix-matrix product $\boldsymbol{AZ} \in \mathbb{R}^{n \times K}$, by using the SMASH $\mathcal{H}^2$ representation of $\boldsymbol{A}$ and its matrix-free operations. Then, we compute $\mathrm{Tr}(\boldsymbol{A}) = \sum_{k=1}^K \boldsymbol{z}_k^\top (\boldsymbol{AZ})_k$, where $(\boldsymbol{AZ})_k \in \mathbb{R}^n$ is the kth column of the matrix $\boldsymbol{AZ}$.

- **SLQ algorithm**

  We also use a blocked version of the Lanczos algorithm proposed by Gardner et al. (2018) for their batched PCG method. For our problem setting, the block version presents computational improvements in comparison with using the original Lanczos algorithm for SLQ presented by Ubaru et al. (2017).

4.2.2 Preconditioning

When solving linear systems with smashGP, we use the PCG algorithm, which is an iterative method. To solve the system to a desired accuracy, the number of iterations required depends on the length scale θ. To reduce the number of iterations and the amount of computational time, the Nyström preconditioner is used. The hope is to reduce the number of iterations to a constant number so that the number of iterations does not depend on the length scale.

Let us consider a system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where we want to solve for $\boldsymbol{x}$. A naive left preconditioner would be $\boldsymbol{M} = \boldsymbol{A}^{-1}$, because $\boldsymbol{M}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{x} = \boldsymbol{M}\boldsymbol{b}$. Thus, we could calculate $\boldsymbol{x}$ with a single application of the preconditioner. In general, we do not have an exact inverse to $\boldsymbol{A}$, but perhaps some approximation to the inverse of $\boldsymbol{A}$ that is easy to solve with.

In smashGP, we need to solve linear systems with $\boldsymbol{R}_\alpha$. Thus, as a preconditioner, we would like an approximation of $\boldsymbol{R}_\alpha^{-1}$. Based on the eigenspectrum of kernel matrices, we assume that $\boldsymbol{R}_\alpha$ can be approximated by a low-rank matrix plus a diagonal shift as $\boldsymbol{R}_\alpha \approx \boldsymbol{U}\boldsymbol{V}^\top + \Delta\boldsymbol{I}_n$, where $\boldsymbol{R}_\alpha \in \mathbb{R}^{n\times n}, \boldsymbol{U} \in \mathbb{R}^{n\times r}, \boldsymbol{V} \in \mathbb{R}^{n\times r}$, and $\Delta > 0$. If we have the factors $\boldsymbol{U}, \boldsymbol{V}$ and Δ, then $\boldsymbol{R}_\alpha^{-1}$ can then be approximated as

$$(\boldsymbol{U}\boldsymbol{V}^\top + \Delta\boldsymbol{I}_n)^{-1} = \frac{1}{\Delta}(\boldsymbol{I}_n - \boldsymbol{U}(\Delta\boldsymbol{I}_n + \boldsymbol{V}^\top\boldsymbol{U})^{-1}\boldsymbol{V}^\top) \tag{21}$$

(Woodbury 1950).

We estimate the factors $\boldsymbol{U}$ and $\boldsymbol{V}$ by using the Nyström approximation (Williams and Seeger 2000). With the Nyström approximation, a subset of m sample points $\boldsymbol{S}^{(m)} = \{\boldsymbol{s}_{(1)}, \ldots, \boldsymbol{s}_{(m)}\} \subset \{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n\}$ is selected. In smashGP, we use $m = 4\sqrt{n}$ with the points randomly sampled, following the recommendations provided in (Cutajar et al. 2016). Selecting too small of an m results in the preconditioner not being as effective as desired, but this should only result in more iterations being required compared to the ideal preconditioner. Selecting too large of an m will

result in an increase in the computation time required for forming the preconditioner. Let us define $\boldsymbol{R}^{(m)} = \{R(\boldsymbol{s}_{(i)}, \boldsymbol{s}_{(j)})\}_{i,j=1}^{m} \in \mathbb{R}^{m \times m}$ as the correlation matrix evaluated pairwise over the samples in $\boldsymbol{S}^{(m)}$. The eigendecomposition of $\boldsymbol{R}^{(m)}$ is $\boldsymbol{U}_m \boldsymbol{\Lambda}_m \boldsymbol{U}_m^\top$. We can then approximate $\boldsymbol{R}$ as

$$\boldsymbol{R} \approx \boldsymbol{r}(\boldsymbol{S}^{(m)})^\top (\boldsymbol{R}^{(m)})^{-1} \boldsymbol{r}(\boldsymbol{S}^{(m)}) = \tilde{\boldsymbol{U}} \tilde{\boldsymbol{\Lambda}} \tilde{\boldsymbol{U}}^\top.$$

where $\boldsymbol{r}(\boldsymbol{S}^{(m)})$ is an $m \times n$ matrix such that the ith row is equal to $\boldsymbol{r}(\boldsymbol{s}_{(i)})$, $\tilde{\boldsymbol{\Lambda}} = \frac{n}{m}\boldsymbol{\Lambda}_m, \tilde{\boldsymbol{U}} = (\boldsymbol{r}(\boldsymbol{S}^{(m)})^\top \boldsymbol{U}_m \sqrt{\frac{m}{n}} \boldsymbol{\Lambda}_m^{-1})$. In Equation 21, let us set $\boldsymbol{U} = \tilde{\boldsymbol{U}}, \boldsymbol{V}^\top = \tilde{\boldsymbol{\Lambda}} \tilde{\boldsymbol{U}}^\top$, and $\Delta = (1 - \alpha)$. This yields

$$(\boldsymbol{U}\boldsymbol{V}^\top + \Delta \boldsymbol{I}_n)^{-1} = (\tilde{\boldsymbol{U}} \tilde{\boldsymbol{\Lambda}} \tilde{\boldsymbol{U}}^\top + (1 - \alpha)\boldsymbol{I}_n)^{-1} = \frac{1}{1 - \alpha}((\boldsymbol{I}_n - \tilde{\boldsymbol{U}}((1 - \alpha)\boldsymbol{I}_n + (\tilde{\boldsymbol{\Lambda}} \tilde{\boldsymbol{U}}^\top)\tilde{\boldsymbol{U}})^{-1})\tilde{\boldsymbol{\Lambda}} \tilde{\boldsymbol{U}}^\top).$$

Thus, we can use this Nyström preconditioner to approximately solve $\boldsymbol{R}_\alpha \boldsymbol{x} = \boldsymbol{b}$. By using this preconditioner, the number of iterations required for a solve is drastically reduced.

4.2.3 Numerical Issues

An additional challenge when modeling large-scale spatial data is that there is no guarantee that the approximation of the correlation matrix $\boldsymbol{R}_\alpha$ will be positive definite. Samples that are close together can artificially reduce the numerical rank of the matrix resulting in negative eigenvalues, breaking the positive definiteness of $\boldsymbol{R}_\alpha$. However, PCG only has a convergence guarantee for positive definite matrices. Additionally, the log-determinant, $\log |\boldsymbol{R}_\alpha|$, in the log-likelihood function, depends largely on the approximation of the smallest eigenvalues. If these are negative, they can cause the log-determinant and log-likelihood to be undefined. To guarantee estimating a positive definite correlation matrix, we build the SMASH $\mathcal{H}^2$ representation such that the eigenvalues are accurate up to the approximation error *tol* defined by the user. Additionally, to control for the effect of small eigenvalues, we use a threshold. In the computation of the log-determinant, if we have eigenvalue estimates that are smaller than the estimated noise $\hat{\tau}^2$, we replace them by this value, as we know the true eigenvalues would never be lower than the noise.

With this, we have explained the smashGP framework. Next, we evaluate its performance with simulation experiments and case studies.

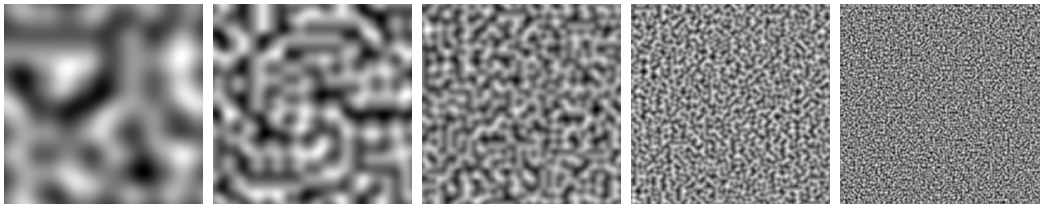
5 Performance Evaluation via Simulations

In this section, we evaluate the performance of smashGP with a range of synthetic data. First, we compare smashGP, which uses hierarchical matrix-free operations, and DiceKriging (Roustant et al. 2012), which uses dense matrix decomposition and inversion. DiceKriging is presently the fastest R package to directly learn a GP using CPUs, but still requires $\mathcal{O}(n^3)$ operations and $\mathcal{O}(n^2)$ memory. Then, we empirically study the computational complexity of smashGP. Finally, we compare the predictive accuracy and computational time of smashGP with state-of-art methods for large-scale spatial modeling.

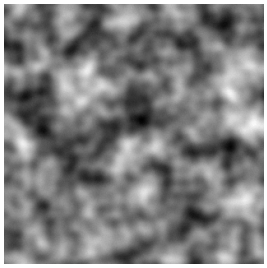
5.1 Data Generation and Evaluation Metrics

The data is generated by overlapping different layers of Perlin noise (Perlin 1985). Each layer is generated with a different frequency, which controls the smoothness of the data, as can be seen in Figure 3a. The smaller the frequency, the smoother the data. This procedural generation of noise allows creating natural appearing spatial data, while controlling for smoothness. In the experiments, we considered varying layers of Perlin noise. After the spatial data is created, we add a random noise to each data point with mean 0 and variance τ^2, to account for the measurement noise that one could encounter in real-life settings. Examples of the data generated are presented in Figures 3b and 3c.

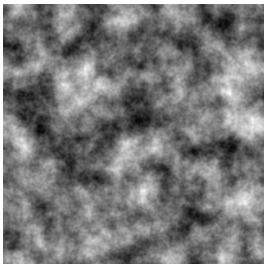
To evaluate the performance of each method, we split the spatial data into training and test datasets. To define the test dataset we considered two scenarios. In the first one, the test dataset is generated by randomly sampling a percentage of the data. This scenario is denoted by "random". In the second one, an additional Perlin noise layer is generated and the locations associated with the largest magnitudes are used as the test dataset. When collecting spatial datasets, it is common to have large sections of missing data. For example, when using satellite images, one could not have access to data due to cloud or tree coverage. The second scenario is denoted by "Perlin", and attempts to emulate the large sections of missing data. An example of the different test datasets is provided in Figure 4.



(a) Different layers of Perlin noise with varying frequencies.

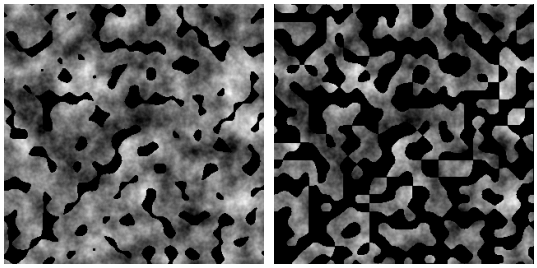


(b) Three layers of Perlin noise and random noise.

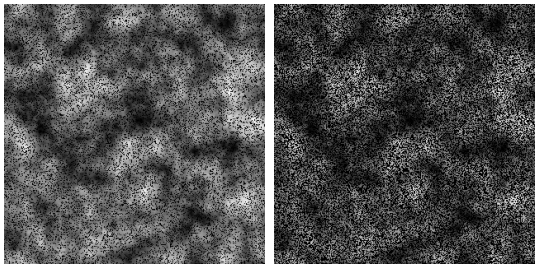


(c) Five layers of Perlin noise and random noise.

Figure 3: Data generation using Perlin noise.



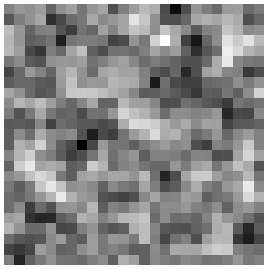
(a) Perlin test data, 20% (left), 50% (right).



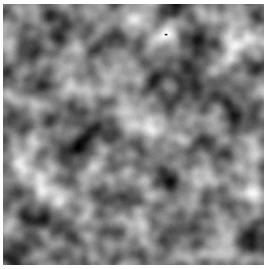
(b) Random test data, 20% (left), 50% (right).

Figure 4: Examples of training and test datasets using Perlin noise. The training data is observed in the image while the test data appears in solid black.

Our goal is to compare smashGP with other state-of-the-art methods on the ability to predict accurately as well as to quantify the uncertainty associated with the predictions. We do this by varying the smoothness and noise levels, as well as the amount and shape of test data, to ensure a wide coverage of scenarios. We compare smashGP with other methods for spatial modeling with GPs in terms of mean absolute error (MAE), root-mean-squared-error (RMSE), interval score (INT; see Gneiting and Raftery (2007)), and prediction interval coverage (CVG; the percent of intervals containing the true value). The definition for each of these metrics can be found in the supplementary materials.



(a) 25×25 (625) data points.



(b) 140×140 (19,600) data points.

Figure 5: Example of the data used to compare smashGP and DiceKriging.

5.2 Comparison with DiceKriging

We start by comparing smashGP with DiceKriging (Roustant et al. 2012), which uses dense matrix decomposition and inversion. Both smashGP and DiceKriging are configured to use a Gaussian kernel. We compare the two methods for different sample sizes ranging from a 25×25 grid (625 points) up to a 140×140 grid (19600 points), as can be seen in Figure 5. For larger datasets, the use of DiceKriging becomes restrictive. We will test smashGP performance for larger datasets in the following sections. For these tests, we withhold 20% of the data as testing data using a "Perlin" filter, and use a global noise standard deviation τ of 0.01, and three layers of Perlin noise for the dataset. The results are presented in Table 1. We see that the results obtained with smashGP are very close to those obtained when using the dense computations of DiceKriging. However, as can be seen in Figure 6, tuning smashGP by using hierarchical matrices and matrix-free operations presents considerable computational time savings for large datasets. DiceKriging scales with $\mathcal{O}(n^3)$ while smashGP scales with $\mathcal{O}(n \log n)$, achieving quasilinear computational time. Above 6,400 points, a small size problem for today's spatial datasets, smashGP outperforms DiceKriging in terms of computational time, while maintaining the same performance in terms of accuracy and uncertainty quantification. In next section, we investigate the computational time of the different operations required to fit a GP using smashGP.

Table 1: Performance comparison between smashGP and DiceKriging.

	MAE		RMSE		INT		CVG	
Points	**smashGP**	**Dice**	**smashGP**	**Dice**	**smashGP**	**Dice**	**smashGP**	**Dice**
625	0.0892	0.0886	0.1132	0.1127	0.5203	0.5235	0.9040	0.9040
900	0.0817	0.0812	0.1015	0.1006	0.4947	0.4496	0.8722	0.8944
1600	0.0568	0.0573	0.0740	0.0743	0.3382	0.3550	0.9375	0.9125
2500	0.0331	0.0331	0.0433	0.0433	0.1930	0.1930	0.9360	0.9360
3600	0.0263	0.0264	0.0373	0.0374	0.1612	0.1620	0.9264	0.9347
4900	0.0231	0.0231	0.0322	0.0321	0.1286	0.1282	0.9143	0.9153
6400	0.0186	0.0186	0.0251	0.0251	0.1105	0.1107	0.9383	0.9352
8100	0.0166	0.0166	0.0220	0.0220	0.1027	0.1028	0.9420	0.9414
10000	0.0178	0.0178	0.0233	0.0232	0.1006	0.1008	0.9260	0.9245
14400	0.0165	0.0165	0.0217	0.0217	0.0966	0.0966	0.9201	0.9201
19600	0.0143	0.0143	0.0188	0.0188	0.0884	0.0880	0.9227	0.9242

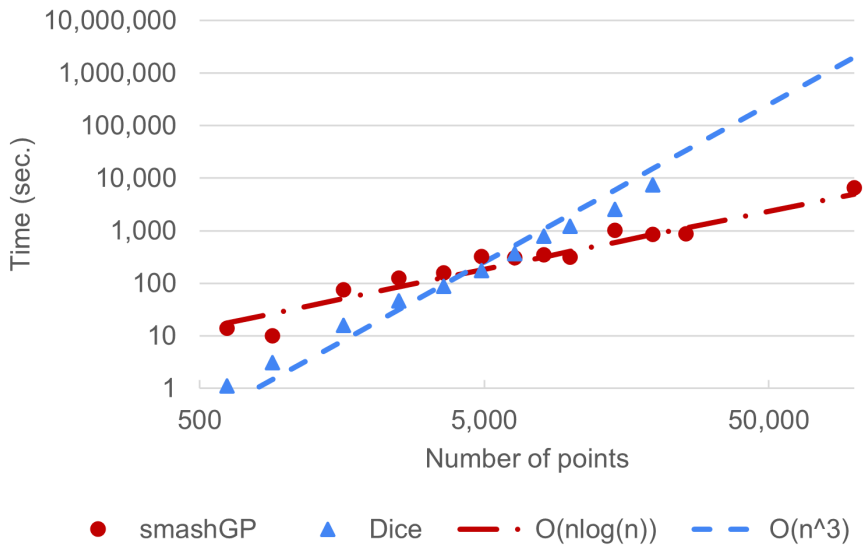


Figure 6: Comparison of the tuning time for smashGP and DiceKriging.

5.3 Empirical study on the computational complexity of smashGP

In this section, we study the computational complexity of smashGP for large spatial datasets. Without loss of generality, we consider the case when the kernel used is Gaussian. In the supplementary materials, we present the computational complexity of constructing the SMASH $\mathcal{H}^2$ matrix representation of the correlation matrix $\boldsymbol{R}_\alpha$ for different sample sizes. We also present the

computational complexity of SMASH matrix-free matrix-vector products. Here, we summarize the results by evaluating the computational time of calculating $\log|\boldsymbol{R}_\alpha|$ and the gradient of the log-likelihood function as these operations are critical for fitting a GP model using smashGP. For this purpose, we generate data using three layers of Perlin noise for different sample sizes ranging from 10,000 to 100,000, with a global noise standard deviation τ of 0.01, and use 80% of the data to train a GP using smashGP. The cumulative results over the required iterations to learn a GP are presented in Figure 7. We observe that the computation of the log-determinant scales with $\mathcal{O}(n)$ and the computation of the gradient scales with $\mathcal{O}(n \log n)$. The $\log n$ term in the gradient computation occurs because of the trace operations needed to compute the partial derivatives of the log-likelihood (See Section 4.1.2). Empirically, we can conclude that by using smashGP we are able to learn a GP in quasilinear time.

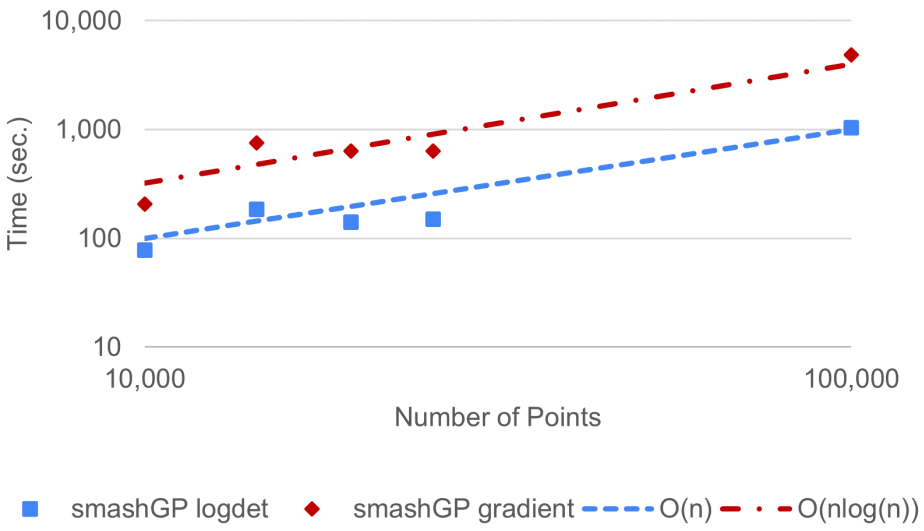


Figure 7: Comparison of the computational time for the different operations required to tune smashGP for different problem sizes.

5.4 Comparison with state-of-art methods for large datasets

In this section, we compare the predictive accuracy of smashGP with the predictive accuracy of state-of-art methods for large-scale spatial modeling with GPs. We consider four methods: spatial partitioning (SP; Heaton et al. (2017)), lattice Kriging (latticeKrig; Nychka et al. (2015)),

stochastic partial differential equations (SPDE; Lindgren et al. (2011)), and GPyTorch (Gardner et al. 2018).

SP, latticeKrig, and SPDE are GP methods that assume a particular structure of the covariance matrix to overcome the computational limitations of directly fitting a GP. SP is a sparse covariance method based on spatial partitioning of the domain. It assumes independence between observations across subregions, and therefore allows for parallel computations. LatticeKrig is a sparse precision method based on multiresolution radial basis functions. The main assumption is that the spatial process can be approximated by a linear combination of the basis functions. Finally, SPDE is a sparse precision method based on the equivalence between Matérn covariance fields and stochastic partial differential equations. The main assumption is that the spatial process can be approximated by using basis functions chosen to be piecewise linear on a triangulation of the domain. The sparse matrix coefficients are determined solely by the choice of triangulation. These three methods were chosen as benchmarks because in the case study competition paper by Heaton et al. (2019), they showed the best performance. Their codes are provided as part of the aforementioned paper and can be found at `https://github.com/finnlindgren/heatoncomparison`. For the competition, SP, latticeKrig, and SPDE considered a constant mean and approximated an exponential kernel function. The exponential kernel corresponds with the Matérn $\nu = 1/2$ kernel and, in one dimension, is defined as $g(h) = exp(-|h|/\theta)$.

GPyTorch is a popular package for GP regression. We selected this package as a benchmark because it is based on covariance matrix approximation and has open source code that allows to fit a GP and generate predictions for large data sets. The code for GPyTorch is available at `https://gpytorch.ai/`. GPyTorch has different settings that need to be specified by the user: the covariance representation, the method for log-likelihood calculation, and the optimization method. The settings used in this paper were set following the authors' recommendations: the Grid Kernel was used for the covariance matrix, the default black-box matrix-matrix multiplications (`fast_computations`) were used for log-likelihood calculation, and ADAM was used as the optimizer. The version of GPyTorch used is 1.3.1.

We tested sixteen scenarios, on a 316×316 grid (i.e., the grid has $99,856$ points), by modifying the different parameters introduced in Section 5.1 with the goal of understanding the limitations

and benefits of the different methods. We considered "high" and "low" levels of smoothness when generating the data using Perlin noise, which corresponds to three and four layers of noise, respectively. We changed the random noise added to each data point by setting $\tau = 0.005$ or $\tau = 0.01$. We considered different percentages of testing data, with $t = 20\%$ or $t = 50\%$, as well as different shapes of testing data by using a "random" mask or a "Perlin" mask. The results over 50 simulation replicates can be found in Tables 2 - 5. Boxplots comparing smashGP with the best benchmark identified, SPDE are presented in the supplementary materials.

When the data is smoother, we observe that smashGP with the three implemented kernels (i.e., Gaussian, Matérn $\nu = 5/2$, Matérn $\nu = 3/2$) outperforms the benchmarks in terms of prediction (MAE, RMSE) and uncertainty quantification (INT, CVG). In general, the Gaussian kernel provides the best results, followed by the Matérn $\nu = 5/2$ kernel. This is reasonable as with the Gaussian kernel, the sample paths are assumed to have derivatives of all orders and thus to be very smooth. When the data becomes less smooth, the prediction performance of smashGP with Gaussian kernel decreases, as is to be expected. In these scenarios, smashGP with the Matérn $\nu = 5/2$ kernel is, in general, the best-performing method in terms of prediction (MAE, RMSE) and uncertainty quantification (INT, CVG). Following the smashGP methods, SPDE has the best performance in terms of MAE, RMSE, and INT, while latticeKrig has the best performance in terms of CVG.

For GPyTorch, the prediction intervals are wide, limiting the uncertainty quantification capabilities of the method. Based on our experiments, this occurs because GPyTorch uses a different matrix approximation (i.e. Grid Kernel). With this approximation, the estimated parameters differ significantly from the parameters found by smashGP, affecting the prediction and uncertainty quantification capabilities.

Based on these results, we recommend using smashGP over state-of-art methods as it provides the smallest prediction error with the best uncertainty quantification. Depending on the smoothness of the underlying process, practitioners can choose the best kernel to be used. The Gaussian kernel is recommended for smooth processes, and Matérn kernels are recommended for less smooth scenarios. Furthermore, smashGP does not make strong assumptions about the structure of the covariance/precision matrix and allows the user to control the accuracy of the GP approximation.

Table 2: MAE over 50 simulation replicates. Results are in the form of mean (standard deviation). A * identifies the best-performing method. We have boldened the smashGP methods that outperform the benchmarks.

			MAE							
Smooth.	**Noise**	**Test %**	**Test mask**	**smashGP Gaussian**	**smashGP Matern 5/2**	**smashGP Matern 3/2**	**GPyTorch**	**SP**	**latticeKrig**	**SPDE**
Low	0.005	20%	Random	**0.0046(0.0000)**	***0.0045(0.0000)**	**0.0046(0.0000)**	0.0586(0.0018)	0.0048(0.0001)	0.0047(0.0000)	0.0050(0.0000)
			Perlin	0.0218(0.0021)	***0.0184(0.0014)**	**0.0194(0.0014)**	0.0694(0.0038)	0.0248(0.0016)	0.0205(0.0014)	0.0202(0.0015)
		50%	Random	**0.0049(0.0001)**	***0.0047(0.0000)**	**0.0048(0.0000)**	0.0405(0.0017)	0.0053(0.0001)	0.0050(0.0000)	0.0051(0.0000)
			Perlin	0.0410(0.0038)	***0.0311(0.0024)**	**0.0312(0.0024)**	0.0625(0.0086)	0.0393(0.0030)	0.0331(0.0025)	0.0315(0.0023)
	0.01	20%	Random	**0.0087(0.0001)**	***0.0086(0.0001)**	**0.0087(0.0001)**	0.0588(0.0017)	0.0092(0.0001)	0.0089(0.0001)	0.0090(0.0001)
			Perlin	0.0244(0.0015)	***0.0220(0.0014)**	**0.0226(0.0014)**	0.0696(0.0039)	0.0285(0.0019)	0.0238(0.0014)	0.0233(0.0014)
		50%	Random	**0.0089(0.0001)**	***0.0089(0.0000)**	**0.0091(0.0001)**	0.0410(0.0017)	0.0097(0.0001)	0.0093(0.0001)	0.0094(0.0001)
			Perlin	0.0419(0.0032)	**0.0339(0.0025)**	***0.0337(0.0023)**	0.0627(0.0086)	0.0424(0.0030)	0.0359(0.0025)	0.0340(0.0022)
High	0.005	20%	Random	***0.0042(0.0000)**	***0.0042(0.0000)**	**0.0043(0.0000)**	0.0564(0.0017)	0.0044(0.0000)	0.0045(0.0000)	0.0045(0.0000)
			Perlin	***0.0075(0.0004)**	**0.0082(0.0005)**	**0.0098(0.0007)**	0.0681(0.0044)	0.0175(0.0015)	0.0142(0.0012)	0.0109(0.0007)
		50%	Random	***0.0042(0.0000)**	**0.0043(0.0000)**	**0.0044(0.0000)**	0.0364(0.0013)	0.0046(0.0000)	0.0047(0.0000)	0.0047(0.0000)
			Perlin	***0.0154(0.0016)**	**0.0156(0.0015)**	**0.0186(0.0017)**	0.0611(0.0088)	0.0336(0.0034)	0.0270(0.0027)	0.0206(0.0019)
	0.01	20%	Random	***0.0082(0.0001)**	**0.0083(0.0001)**	**0.0083(0.0001)**	0.0565(0.0017)	0.0085(0.0001)	0.0087(0.0001)	0.0086(0.0004)
			Perlin	***0.0120(0.0004)**	**0.0121(0.0005)**	**0.0134(0.0006)**	0.0683(0.0044)	0.0209(0.0014)	0.0178(0.0011)	0.0148(0.0007)
		50%	Random	***0.0084(0.0000)**	***0.0084(0.0000)**	**0.0085(0.0000)**	0.0368(0.0013)	0.0088(0.0001)	0.0089(0.0001)	0.0087(0.0000)
			Perlin	***0.0192(0.0017)**	**0.0194(0.0015)**	**0.0218(0.0017)**	0.0612(0.0088)	0.0359(0.0032)	0.0300(0.0027)	0.0239(0.0019)

Table 3: RMSE over 50 simulation replicates. Results are in the form of mean (standard deviation). A * identifies the best-performing method. We have boldened the smashGP methods that outperform the benchmarks.

			RMSE							
Smooth.	**Noise**	**Test %**	**Test mask**	**smashGP Gaussian**	**smashGP Matern 5/2**	**smashGP Matern 3/2**	**GPyTorch**	**SP**	**latticeKrig**	**SPDE**
Low	0.005	20%	Random	**0.0057 (0.0000)**	***0.0056 (0.0000)**	**0.0057 (0.0001)**	0.0732 (0.0022)	0.0060 (0.0001)	0.0059 (0.0001)	0.0063 (0.0001)
			Perlin	0.0354 (0.0040)	***0.0275 (0.0025)**	**0.0283 (0.0024)**	0.0870 (0.0049)	0.0346 (0.0024)	0.0299 (0.0023)	0.0290 (0.0023)
		50%	Random	**0.0061 (0.0001)**	***0.0059 (0.0000)**	**0.0061 (0.0001)**	0.0504 (0.0022)	0.0068 (0.0001)	0.0063 (0.0001)	0.0064 (0.0000)
			Perlin	0.0634 (0.0063)	0.0458 (0.0040)	0.0447 (0.0037)	0.0797 (0.0106)	0.0549 (0.0045)	0.0473 (0.0038)	*0.0445 (0.0036)
	0.01	20%	Random	**0.0109 (0.0001)**	***0.0108 (0.0001)**	**0.0109 (0.0001)**	0.0735 (0.0021)	0.0116 (0.0001)	0.0111 (0.0001)	0.0113 (0.0001)
			Perlin	0.0360 (0.0029)	***0.0309 (0.0023)**	**0.0313 (0.0023)**	0.0871 (0.0050)	0.0384 (0.0027)	0.0331 (0.0022)	0.0320 (0.0022)
		50%	Random	**0.0112 (0.0001)**	***0.0112 (0.0001)**	**0.0114 (0.0001)**	0.0510 (0.0021)	0.0122 (0.0002)	0.0117 (0.0001)	0.0117 (0.0001)
			Perlin	0.0617 (0.0053)	0.0478 (0.0039)	***0.0467 (0.0036)**	0.0800 (0.0106)	0.0578 (0.0045)	0.0498 (0.0038)	0.0468 (0.0034)
High	0.005	20%	Random	***0.0052 (0.0000)**	**0.0053 (0.0000)**	**0.0054 (0.0001)**	0.0704 (0.0021)	0.0055 (0.0000)	0.0057 (0.0000)	0.0056 (0.0001)
			Perlin	***0.0109 (0.0010)**	**0.0116 (0.0009)**	**0.0141 (0.0012)**	0.0852 (0.0056)	0.0253 (0.0024)	0.0213 (0.0020)	0.0157 (0.0013)
		50%	Random	***0.0053 (0.0000)**	**0.0054 (0.0000)**	**0.0055 (0.0000)**	0.0450 (0.0016)	0.0059 (0.0000)	0.0059 (0.0000)	0.0059 (0.0001)
			Perlin	**0.0253 (0.0037)**	***0.0242 (0.0029)**	**0.0281 (0.0030)**	0.0782 (0.0111)	0.0486 (0.0054)	0.0400 (0.0044)	0.0307 (0.0033)
	0.01	20%	Random	***0.0103 (0.0001)**	***0.0103 (0.0001)**	**0.0105 (0.0001)**	0.0705 (0.0022)	0.0107 (0.0001)	0.0108 (0.0001)	0.0108 (0.0005)
			Perlin	**0.0160 (0.0009)**	***0.0160 (0.0008)**	**0.0180 (0.0011)**	0.0855 (0.0056)	0.0287 (0.0022)	0.0248 (0.0019)	0.0198 (0.0012)
		50%	Random	***0.0105 (0.0000)**	***0.0105 (0.0000)**	**0.0106 (0.0000)**	0.0455 (0.0016)	0.0110 (0.0001)	0.0112 (0.0001)	0.0109 (0.0001)
			Perlin	**0.0284 (0.0036)**	***0.0278 (0.0027)**	**0.0311 (0.0029)**	0.0783 (0.0111)	0.0507 (0.0052)	0.0428 (0.0044)	0.0339 (0.0032)

Table 4: INT over 50 simulation replicates. Results are in the form of mean (standard deviation). A * identifies the best-performing method. We have boldened the smashGP methods that outperform the benchmarks.

			INT							
				smashGP Gaussian	**smashGP Matern 5/2**	**smashGP Matern 3/2**	**GPyTorch**	**SP**	**latticeKrig**	**SPDE**
Low	0.005	20%	Random	**0.0268 (0.0002)**	***0.0262 (0.0002)**	**0.0269 (0.0002)**	3.2006 (0.0409)	0.0365 (0.0015)	0.0276 (0.0003)	0.0311 (0.0004)
			Perlin	**0.1323 (0.0174)**	***0.1156 (0.0146)**	**0.1448 (0.0194)**	3.1967 (0.0454)	0.3004 (0.0341)	0.1545 (0.0188)	0.1709 (0.0229)
		50%	Random	**0.0286 (0.0004)**	***0.0276 (0.0002)**	**0.0287 (0.0003)**	1.6359 (0.1236)	0.0450 (0.0020)	0.0298 (0.0003)	0.0304 (0.0002)
			Perlin	**0.2707 (0.0399)**	***0.2079 (0.0250)**	**0.2480 (0.0317)**	1.9220 (0.5879)	0.5832 (0.0668)	0.2974 (0.0372)	0.2850 (0.0321)
	0.01	20%	Random	**0.0509 (0.0004)**	***0.0506 (0.0004)**	**0.0512 (0.0004)**	3.2075 (0.0405)	0.0647 (0.0019)	0.0522 (0.0005)	0.0529 (0.0006)
			Perlin	**0.1421 (0.0093)**	***0.1280 (0.0103)**	**0.1406 (0.0143)**	3.2043 (0.0451)	0.2931 (0.0355)	0.1567 (0.0152)	0.1547 (0.0170)
		50%	Random	***0.0524 (0.0003)**	***0.0524 (0.0003)**	**0.0533 (0.0004)**	1.6437 (0.1203)	0.0680 (0.0023)	0.0548 (0.0004)	0.0552 (0.0005)
			Perlin	0.2449 (0.0209)	***0.2000 (0.0179)**	**0.2172 (0.0222)**	1.9398 (0.5820)	0.5559 (0.0629)	0.2716 (0.0313)	0.2365 (0.0229)
High	0.005	20%	Random	***0.0243 (0.0002)**	**0.0248 (0.0002)**	**0.0253 (0.0002)**	3.1087 (0.0420)	0.0348 (0.0015)	0.0265 (0.0003)	0.0279 (0.0005)
			Perlin	***0.0443 (0.0030)**	**0.0484 (0.0037)**	**0.0641 (0.0076)**	3.1084 (0.0476)	0.2069 (0.0323)	0.1068 (0.0153)	0.0775 (0.0095)
		50%	Random	***0.0247 (0.0001)**	**0.0252 (0.0001)**	**0.0259 (0.0002)**	1.4450 (0.1310)	0.0344 (0.0018)	0.0280 (0.0002)	0.0291 (0.0005)
			Perlin	***0.0937 (0.0124)**	**0.1046 (0.0159)**	**0.1587 (0.0257)**	1.6853 (0.6520)	0.5653 (0.0872)	0.2740 (0.0497)	0.1942 (0.0295)
	0.01	20%	Random	***0.0483 (0.0003)**	**0.0484 (0.0003)**	**0.0489 (0.0003)**	3.1170 (0.0420)	0.0562 (0.0010)	0.0508 (0.0004)	0.0512 (0.0047)
			Perlin	**0.0706 (0.0030)**	***0.0704 (0.0032)**	**0.0801 (0.0055)**	3.1165 (0.0485)	0.1970 (0.0271)	0.1169 (0.0124)	0.0941 (0.0116)
		50%	Random	***0.0491 (0.0002)**	***0.0491 (0.0002)**	**0.0498 (0.0002)**	1.4598 (0.1366)	0.0605 (0.0017)	0.0525 (0.0003)	0.0512 (0.0003)
			Perlin	***0.1143 (0.0115)**	**0.1161 (0.0132)**	**0.1459 (0.0190)**	1.7090 (0.6443)	0.4958 (0.0777)	0.2525 (0.0423)	0.1744 (0.0221)

Table 5: CVG over 50 simulation replicates. Results are in the form of mean (standard deviation). A * identifies the best-performing method. We have boldened the smashGP methods that outperform the benchmarks.

			CVG							
				smashGP Gaussian	**smashGP Matern 5/2**	**smashGP Matern 3/2**	**GPyTorch**	**SP**	**latticeKrig**	**SPDE**
Low	0.005	20%	Random	0.9519 (0.0026)	0.9566 (0.0017)	0.9609 (0.0018)	1.0000 (0.0000)	0.9972 (0.0008)	*0.9502 (0.0019)	0.9066 (0.0050)
			Perlin	***0.9314 (0.0099)**	**0.9116 (0.0116)**	0.8629 (0.0137)	1.0000 (0.0000)	0.7207 (0.0158)	0.8742 (0.0127)	0.8192 (0.0149)
		50%	Random	0.9520 (0.0030)	0.9581 (0.0015)	0.9614 (0.0016)	1.0000 (0.0000)	0.9984 (0.0003)	0.9517 (0.0015)	*0.9504 (0.0064)
			Perlin	***0.9053 (0.0167)**	**0.8880 (0.0114)**	**0.8318 (0.0138)**	1.0000 (0.0000)	0.6285 (0.0157)	0.8127 (0.0137)	0.7799 (0.0131)
	0.01	20%	Random	0.9489 (0.0022)	0.9524 (0.0019)	0.9548 (0.0019)	1.0000 (0.0000)	0.9939 (0.0012)	*0.9497 (0.0019)	0.9418 (0.0024)
			Perlin	***0.9469 (0.0062)**	**0.9361 (0.0079)**	**0.9098 (0.0109)**	1.0000 (0.0000)	0.7676 (0.0165)	0.9064 (0.0102)	0.8888 (0.0159)
		50%	Random	0.9489 (0.0015)	0.9536 (0.0015)	0.9548 (0.0026)	1.0000 (0.0000)	0.9937 (0.0012)	*0.9505 (0.0014)	0.9435 (0.0018)
			Perlin	***0.9437 (0.0087)**	**0.9289 (0.0084)**	**0.8919 (0.0123)**	1.0000 (0.0000)	0.6746 (0.0157)	0.8590 (0.0121)	0.8625 (0.0102)
High	0.005	20%	Random	0.9514 (0.0019)	0.9532 (0.0024)	0.9552 (0.0021)	1.0000 (0.0000)	0.9978 (0.0008)	*0.9498 (0.0018)	0.9042 (0.0034)
			Perlin	**0.9441 (0.0064)**	***0.9449 (0.0080)**	**0.9202 (0.0132)**	1.0000 (0.0000)	0.7738 (0.0217)	0.9046 (0.0139)	0.8906 (0.0129)
		50%	Random	0.9519 (0.0016)	0.9551 (0.0014)	0.9576 (0.0017)	1.0000 (0.0000)	0.9956 (0.0015)	*0.9508 (0.0015)	0.9038 (0.0051)
			Perlin	***0.9338 (0.0086)**	**0.9084 (0.0127)**	**0.8430 (0.0169)**	1.0000 (0.0000)	0.6164 (0.0210)	0.8088 (0.0199)	0.8056 (0.0174)
	0.01	20%	Random	0.9505 (0.0021)	0.9526 (0.0020)	0.9537 (0.0018)	1.0000 (0.0000)	0.9890 (0.0020)	*0.9499 (0.0018)	0.9368 (0.0206)
			Perlin	**0.9420 (0.0055)**	***0.9511 (0.0055)**	**0.9397 (0.0075)**	1.0000 (0.0000)	0.8246 (0.0183)	0.9265 (0.0107)	0.9366 (0.0238)
		50%	Random	0.9506 (0.0013)	0.9527 (0.0019)	0.9533 (0.0019)	1.0000 (0.0000)	0.9925 (0.0018)	*0.9503 (0.0014)	0.9423 (0.0016)
			Perlin	***0.9401 (0.0064)**	**0.9340 (0.0120)**	**0.9006 (0.0148)**	1.0000 (0.0000)	0.6881 (0.0207)	0.8533 (0.0175)	0.8735 (0.0212)

Next, we compare the computational time of smashGP and SPDE, as these were identified as the two best methods in terms of accuracy and uncertainty quantification. The results for learning the GP in each of the sixteen scenarios are presented in Table 6. We observe that in two scenarios, when the smoothness is low, the noise is 0.005, and the test percentage is 20%, SPDE outperforms smashGP. For the remaining fourteen scenarios, smashGP has a smaller tuning time. It is worth highlighting that, in general, smashGP with the Matérn $\nu = 3/2$ kernel is the fastest. With this kernel, when solving the systems of linear equations using the PCG algorithm, fewer iterations are needed, potentially due to the fact that the kernel is less smooth, making the problem better conditioned. Once the GP is learned, estimations need to be made for out-of-sample observations. Figures 8 and 9 present the computational times to predict mean and variance, respectively, for different sizes of the test dataset. For these figures, we withheld 50% of the data as testing data using a "Perlin" filter and used a global noise standard deviation τ of 0.01 and three layers of Perlin noise. We observe that the computational time to estimate the predicted mean is always smaller when using smashGP. However, the computational time to estimate the predicted variance is higher. Additionally, we observe that the computational time for SPDE is mostly constant while the computational time for smashGP increases with the number of out-of-sample observations. This behavior is expected. SPDE computes the correlation matrix once when estimating the predicted mean and then uses it to compute the predicted variance. The matrix computation does not increase with the number of out-of-sample observations. On the other hand, smashGP needs to solve systems of linear equations that increase with the number of out-of-sample observations to compute the predicted mean and variance (See Algorithm 2 in Section 4.2.1). The trade-off between accuracy, uncertainty quantification, and computational time is something that practitioners need to consider when deciding which method to use.

In the supplementary materials, we present an analysis of the parameter estimation performance of smashGP.

Table 6: Computational time in minutes for smashGP and SPDE. Results are in the form of mean (standard deviation). A * identifies the best-performing method. We have boldened the smashGP methods that outperform the benchmark.

				Time (minutes)			
Smooth.	**Noise**	**Test %**	**Test mask**	**smashGP Gaussian**	**smashGP Matern 5/2**	**smashGP Matern 3/2**	**SPDE**
Low	0.005	20%	Random	166.4 (47.2)	156.8 (44.6)	128.4 (34.0)	*99.0 (22.0)
			Perlin	154.7 (36.5)	151.0 (33.5)	112.7 (30.1)	*94.3 (23.2)
		50%	Random	**88.9 (14.6)**	***82.4 (16.2)**	**85.4 (12.7)**	115.3 (11.1)
			Perlin	**102.0 (18.7)**	**95.1 (22.8)**	***81.9 (17.7)**	104.7 (19.5)
	0.01	20%	Random	**113.1 (32.2)**	**82.1 (23.6)**	***79.2 (23.5)**	148.8 (25.0)
			Perlin	**98.2 (20.3)**	**80.2 (19.1)**	***73.4 (14.1)**	144.7 (36.1)
		50%	Random	**62.4 (11.3)**	**50.9 (16.2)**	***47.9 (12.4)**	121.5 (29.4)
			Perlin	**70.7 (15.1)**	**59.0 (13.9)**	***56.3 (14.2)**	134.7 (21.8)
High	0.005	20%	Random	153.9 (39.4)	**105.3 (22.5)**	***99.3 (23.0)**	138.2 (30.7)
			Perlin	147.0 (38.0)	**97.8 (17.5)**	***95.0 (19.9)**	139.6 (27.5)
		50%	Random	**89.6 (19.1)**	**62.2 (11.4)**	***55.3 (8.0)**	109.2 (41.1)
			Perlin	**96.1 (16.6)**	**66.2 (13.8)**	***60.4 (9.3)**	129.3 (26.3)
	0.01	20%	Random	**75.9 (21.3)**	**71.7 (17.6)**	***65.0 (14.9)**	168.7 (69.4)
			Perlin	**75.7 (23.6)**	**69.1 (19.3)**	***62.3 (16.0)**	179.9 (85.7)
		50%	Random	**47.2 (12.8)**	**44.2 (8.7)**	***42.2 (6.4)**	127.0 (28.1)
			Perlin	**55.9 (18.1)**	**54.4 (11.9)**	***52.5 (12.8)**	160.4 (57.7)

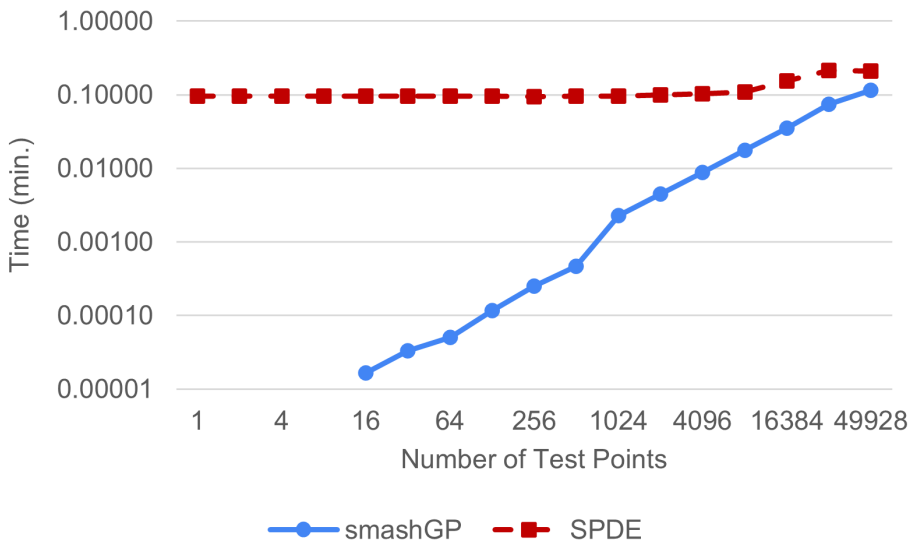


Figure 8: Computational times to estimate the predictive mean for out-of-sample observations for smashGP and SPDE.

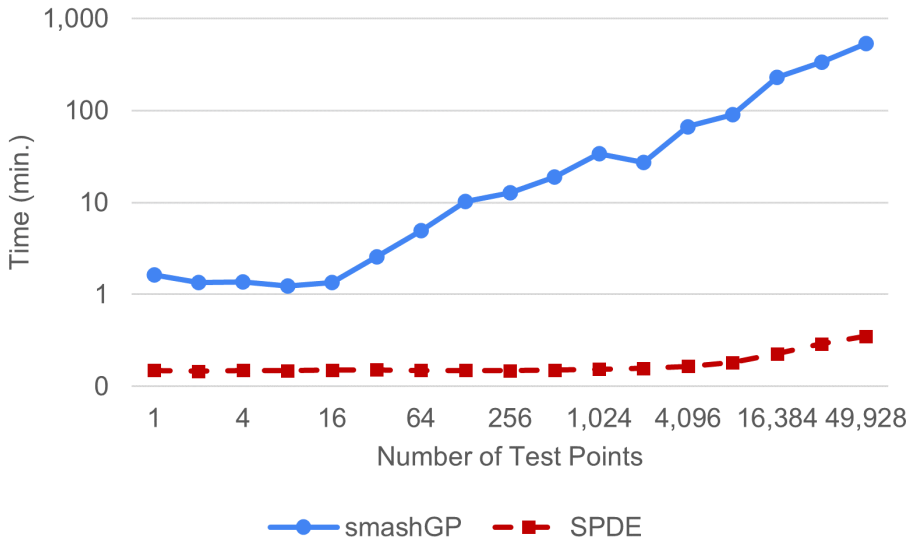


Figure 9: Computational times to estimate the predictive variance for out-of-sample observations for smashGP and SPDE.

6 Case Studies

In this section, smashGP and SPDE are applied to datasets from two sources. The first source contains daytime land surface temperatures and was used in the case study competition by Heaton et al. (2019). The second source contains elevation information for a 2D road network in North Jutland, Denmark (Kaul et al. 2013). In the supplementary materials, we present a third case study where smashGP is applied to a very large dataset of a million data points.

6.1 Daytime land surface temperatures

As a first case study, we use two datasets, one real and one simulated, provided by Heaton et al. (2019) to compare methods for modelling large spatial data. Both datasets contain observations on a 500×300 grid, with the longitude ranging from -95.91153 to -91.28381, and the latitude ranging from 34.29519 to 37.06811.

The real dataset consists of daytime land surface temperatures as measured by the Terra instrument onboard of the MODIS satellite on August 4, 2016. The longitude and latitude ranges,

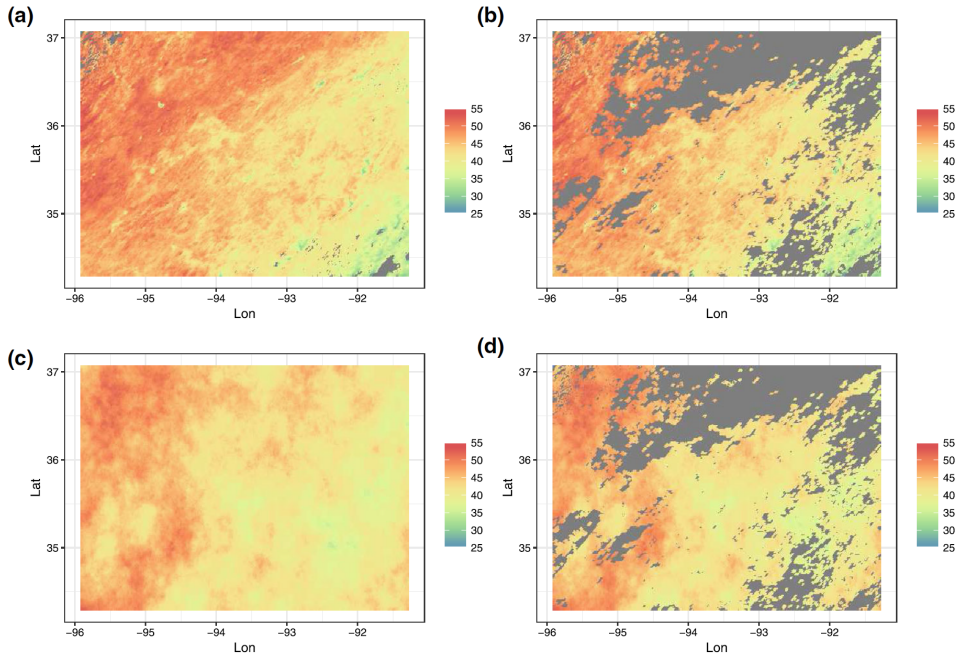


Figure 10: Borrowed from (Heaton et al. 2019). (a) Satellite data on August 4, 2016. (b) Training satellite dataset. (c) Simulated dataset. (d) Training simulated dataset.

as well as the date, were chosen because of the small area covered by clouds. Only 1.1% of the MODIS data was corrupted by cloud cover, leaving 148,309 of the 150,000 possible observed values.

To generate the simulated dataset, a random sample of 2,500 observations from the MODIS data on August 4, 2016 was considered. A GP with constant mean and exponential kernel was fitted to the sampled data. The resulting parameter estimates were 4/3, 16.40, 0.05, and 44.49 for the length scale, process variance, noise variance, and constant mean, respectively. These parameters were then used to simulate 150,000 observations on the same grid as the MODIS data.

To define the training and test datasets, the missing data pattern from the August 6, 2016 MODIS satellite data product was used. The training dataset consisted of 105,569 observations, leaving 42,740 observations in the test set. Figure 10 displays the datasets, along with the training datasets.

We compare smashGP with the most competitive benchmark of Section 5, SPDE, in terms of prediction and uncertainty quantification. The results are presented in Table 7.

Table 7: Comparison for simulated and satellite datasets. A * identifies the best-performing method. We have boldened the smashGP methods that outperform the benchmark.

	MAE				INT			
	smashGP Gaussian	**smashGP Matern 5/2**	**smashGP Matern 3/2**	**SPDE**	**smashGP Gaussian**	**smashGP Matern 5/2**	**smashGP Matern 3/2**	**SPDE**
Simulated data	0.91	0.75	0.69	*0.62	**5.39**	**4.48**	***4.26**	7.81
Satellite data	2.61	2.17	1.89	*1.10	13.72	10.36	9.35	*8.85
	RMSE				**CVG**			
	smashGP Gaussian	**smashGP Matern 5/2**	**smashGP Matern 3/2**	**SPDE**	**smashGP Gaussian**	**smashGP Matern 5/2**	**smashGP Matern 3/2**	**SPDE**
Simulated data	1.26	1.03	0.94	*0.86	***0.95**	**0.97**	**0.97**	1.00
Satellite data	3.32	2.88	2.56	*1.53	0.85	**0.93**	***0.94**	0.97

SPDE is the best method in terms of prediction accuracy (MAE and RMSE) for both the simulated and the satellite data. In terms of uncertainty quantification, for the simulated data, smashGP with the Matérn $\nu = 3/2$ kernel has the smallest INT, and smashGP with Gaussian kernel has the best CVG. SPDE has the smallest INT for the satellite data, and smashGP with the Matérn $\nu = 3/2$ kernel has the best CVG. For this case study, smashGP performs worse because the underlying process is not smooth, and the exponential kernel approximated by SPDE provides better results. It is worth noticing that from the three smashGP methods, the one with the Matérn $\nu = 3/2$ kernel performs better as this kernel is only one-time differentiable. These results are not surprising since SPDE was already known to work well on these datasets. Next, we present a case study where smashGP outperforms the benchmark.

6.2 Elevation for 2D road network

As a second case study, we use a dataset containing elevation information for a 2D road network in North Jutland, Denmark. The network covers a region of 185 x 135 km^2. Elevation values were extracted from a publicly available massive Laser Scan Point Cloud for Denmark (Kaul et al. 2013). The data can be accessed from: `https://archive.ics.uci.edu/ml/datasets/3D+Road+Network+\%28North+Jutland\%2C+Denmark\%29#`. The data is presented in Figure 11.

We randomly select 80,000 points as training data and 20,000 points as test data. We compare smashGP with SPDE in terms of prediction accuracy and uncertainty quantification. The results

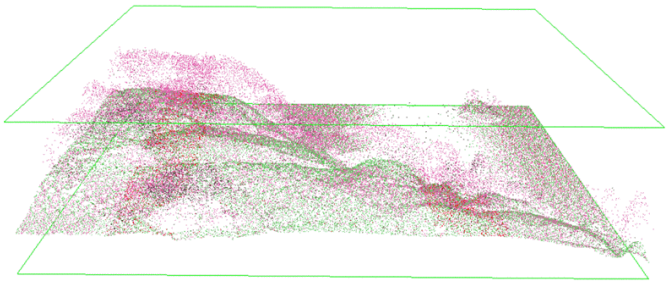


Figure 11: Laser Scan Point Cloud for 2D road network in North Jutland, Denmark. Borrowed from (Kaul et al. 2013).

are presented in Table 8. We observe that for this dataset, smashGP outperforms the benchmark in terms of predictive accuracy (MAE and RMSE) and uncertainty quantification (INT and CVG). smashGP with the Matérn $\nu = 3/2$ kernel is the best performing method. It is hard to assert exactly why SPDE performs poorly. We have two hypotheses for this behavior. (1) It is possible that the precision matrix, in this case, is far from being sparse (this is hard to verify in practice). (2) It is possible that we misspecified some of the hyperparameters required for SPDE, although we chose the best parameters we could find. In this case study, we can see the advantage of using smashGP over SPDE. smashGP does not make an assumption on the structure of the covariance matrix and thus will perform better in general settings. Additionally, no hyperparameters need to be specified by the user. The only information needed is the desired tolerance for the approximation.

Table 8: Prediction and uncertainty quantification comparison for 2D road network dataset. A * identifies the best-performing method. We have boldened the smashGP methods that outperform the benchmark.

MAE				**INT**			
smashGP Gaussian	**smashGP Matern 5/2**	**smashGP Matern 3/2**	**SPDE**	**smashGP Gaussian**	**smashGP Matern 5/2**	**smashGP Matern 3/2**	**SPDE**
1.49	**1.16**	***1.04**	14.46	**14.42**	**11.75**	***10.61**	97.27
RMSE				**CVG**			
smashGP Gaussian	**smashGP Matern 5/2**	**smashGP Matern 3/2**	**SPDE**	**smashGP Gaussian**	**smashGP Matern 5/2**	**smashGP Matern 3/2**	**SPDE**
2.75	**2.20**	***1.98**	18.35	***0.95**	0.96	0.96	0.95

7 Conclusion

GPs are essential for spatial data analysis as they allow to predict unknown values and quantify uncertainty. However, with advanced sensing technologies, the use of direct GPs is restricted as the sample size n is large and naively estimating a GP requires $\mathcal{O}(n^3)$ computations and $\mathcal{O}(n^2)$ memory. In this paper, we propose smashGP, a framework to estimate GPs for large-scale spatial modeling. smashGP represents the GP covariance matrix as an $\mathcal{H}^2$ hierarchical matrix and uses matrix-free operations for training and prediction. With smashGP, the asymptotic computational time is reduced to $\mathcal{O}(n \log n)$.

We compared smashGP with state-of-art methods for large-scale spacial modeling in terms of prediction accuracy, uncertainty quantification, and computational time. As illustrated by the simulations and case studies, no one method dominates the others. In general, we saw that smashGP and SPDE are the best methods overall. If there is no domain knowledge on the sparsity of the precision matrix, which is hard to verify in practice, we recommend using smashGP. The proposed framework can handle different levels of smoothness thanks to the different kernels implemented (Gaussian, Matérn $\nu = 5/2$, Matérn $\nu = 3/2$). Additionally, no hyperparameters need to be specified by the user, and the level of approximation can be easily controlled. If enough resources are available, we suggest running both smashGP and SPDE and using the one that provides the best results over a validation data set.

smashGP was developed to model spatial data in two dimensions. As is, it can be used for GPs in higher than two dimensions. However, it is expected to suffer from the curse of dimensionality. Its performance for high-dimensional GPs is of interest for future research. Additionally, in the future, we will consider extending smashGP to work with non-differentiable kernel functions such as the exponential kernel approximated by SPDE. Finally, we will extend smashGP to learn a GP in cases when the effect of covariates needs to be accounted for in the mean of the process.

Supplementary Materials

Code: Code used to run smashGP can be found at `https://gitlab.com/libsmash_public/smashgp`. See the Readme file for detailed instructions.

Supplementary Materials: Document containing: (A) derivations for GPs with non-constant

mean; (B) additional details on SMASH and its matrix-free operations; (C) definitions of the evaluation metrics used in the simulations and case studies; (D) additional details on the computational complexity of smashGP; (E) parameter estimation evaluation via simulations; (F) comparison with state-of-art methods for large datasets; and (G) case study for a million data points. (.pdf file)

References

Abdulah, S., Ltaief, H., Sun, Y., Genton, M. G., and Keyes, D. E. (2018), "ExaGeoStat: A High Performance Unified Software for Geostatistics on Manycore Systems," *IEEE Transactions on Parallel and Distributed Systems*, 29, 2771–2784.

Anderson, C., Lee, D., and Dean, N. (2014), "Identifying Clusters in Bayesian Disease Mapping," *Biostatistics*, 15, 457–469.

Andugula, P., Durbha, S. S., Lokhande, A., and Suradhaniwar, S. (2017), "Gaussian Process based Spatial Modeling of Soil Moisture for Dense Soil Moisture Sensing Network," in *2017 6th International Conference on Agro-Geoinformatics*, pp. 1–5.

Anitescu, M., Chen, J., and Wang, L. (2012), "A Matrix-free Approach for Solving the Parametric Gaussian Process Maximum Likelihood Problem," *SIAM Journal on Scientific Computing*, 34, A240–A262.

Banerjee, S., Carlin, B. P., and Gelfand, A. E. (2014), *Hierarchical Modeling and Analysis for Spatial Data*, CRC press.

Börm, S. and Garcke, J. (2007), "Approximating Gaussian Processes with H2 Matrices," in *European Conferenceon Machine Learning*, Springer, pp. 42–53.

Cai, D., Chow, E., Erlandson, L., Saad, Y., and Xi, Y. (2018), "SMASH: Structured Matrix Approximation by Separation and Hierarchy," *Numerical Linear Algebra with Applications*, 25, e2204.

Cressie, N. (2015), *Statistics for Spatial Data*, John Wiley & Sons.

Cressie, N. and Johannesson, G. (2008), "Fixed Rank Kriging for Very Large Spatial Data Sets," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70, 209–226.

Cutajar, K., Osborne, M., Cunningham, J., and Filippone, M. (2016), "Preconditioning Kernel Matrices," in *International Conference on Machine Learning*, pp. 2529–2538.

Datta, A., Banerjee, S., Finley, A. O., and Gelfand, A. E. (2016), "Hierarchical Nearest-Neighbor Gaussian Process Models for Large Geostatistical Datasets," *Journal of the American Statistical Association*, 111, 800–812.

Dongarra, J., Croz, J. D., Hammarling, S., and Duff, I. (1990), "A Set of Level 3 Basic Linear Algebra Subprograms," *ACM Transactions on Mathematical Software*, 16, 1–17.

Erlandson, L., Cai, D., Xi, Y., and Chow, E. (2020), "Accelerating Parallel Hierarchical Matrix-Vector Products via Data-Driven Sampling," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 749–758.

Fang, D., Zhang, X., Yu, Q., Jin, T. C., and Tian, L. (2018), "A Novel Method for Carbon Dioxide Emission Forecasting Based on Improved Gaussian Processes Regression," *Journal of Cleaner Production*, 173, 143–150.

Finley, A. O., Datta, A., Cook, B. D., Morton, D. C., Andersen, H. E., and Banerjee, S. (2019), "Efficient Algorithms for Bayesian Nearest Neighbor Gaussian Processes," *Journal of Computational and Graphical Statistics*, 28, 401–414.

Finley, A. O., Sang, H., Banerjee, S., and Gelfand, A. E. (2009), "Improving the Performance of Predictive Process Modeling for Large Datasets," *Computational statistics & data analysis*, 53, 2873–2884.

Furrer, R., Genton, M. G., and Nychka, D. (2006), "Covariance Tapering for Interpolation of Large Spatial Datasets," *Journal of Computational and Graphical Statistics*, 15, 502–523.

Furrer, R. and Sain, S. (2010), "spam: A Sparse Matrix R Package with Emphasis on MCMC Methods for Gaussian Markov Random Fields," *Journal of Statistical Software*, 36, 1–25.

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. (2018), "GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration," in *Advances in Neural Information Processing Systems*, eds. Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., Curran Associates, Inc., vol. 31.

Geoga, C. J., Anitescu, M., and Stein, M. L. (2020), "Scalable Gaussian Process Computations Using Hierarchical Matrices," *Journal of Computational and Graphical Statistics*, 29, 227–237.

Gerber, F., de Jong, R., Schaepman, M. E., Schaepman-Strub, G., and Furrer, R. (2018), "Predicting Missing Values in Spatio-Temporal Remote Sensing Data," *IEEE Transactions on Geoscience and Remote Sensing*, 56, 2841–2853.

Gneiting, T. and Raftery, A. E. (2007), "Strictly Proper Scoring Rules, Prediction, and Estimation," *Journal of the American Statistical Association*, 102, 359–378.

Gramacy, R. (2016), "laGP: Large-Scale Spatial Modeling via Local Approximate Gaussian Processes in R," *Journal of Statistical Software, Articles*, 72, 1–46.

Guhaniyogi, R. and Banerjee, S. (2018), "Meta-Kriging: Scalable Bayesian Modeling and Inference for Massive Spatial Datasets," *Technometrics*, 60, 430–444.

Guinness, J. (2019), "Spectral Density Estimation for Random Fields via Periodic Embeddings," *Biometrika*, 106, 267–286.

Hackbusch, W. (2015), *Hierarchical Matrices: Algorithms and Analysis*, vol. 49, Springer.

Heaton, M. J., Christensen, W. F., and Terres, M. A. (2017), "Nonstationary Gaussian Process Models Using Spatial Hierarchical Clustering from Finite Differences," *Technometrics*, 59, 93–101.

Heaton, M. J., Datta, A., Finley, A. O., Furrer, R., Guinness, J., Guhaniyogi, R., Gerber, F., Gramacy, R. B., Hammerling, D., Katzfuss, M., Lindgren, F., Nychka, D. W., Sun, F., and Zammit-Mangion, A. (2019), "A Case Study Competition Among Methods for Analyzing Large Spatial Data," *Journal of Agricultural, Biological and Environmental Statistics*, 24, 398–425.

Hensman, J., Fusi, N., and Lawrence, N. D. (2013), "Gaussian Processes for Big Data," in *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, Arlington, Virginia, USA: AUAI Press, UAI'13, p. 282–290.

Hutchinson, M. (1989), "A Stochastic Estimator of the Trace of the Influence Matrix for Laplacian Smoothing Splines," *Communication in Statistics- Simulation and Computation*, 18, 1059–1076.

Jurek, M. and Katzfuss, M. (2021), "Multi-resolution Filters for Massive Spatio-temporal Data," *Journal of Computational and Graphical Statistics*, 30, 1095–1110.

Katzfuss, M. (2017), "A Multi-Resolution Approximation for Massive Spatial Datasets," *Journal of the American Statistical Association*, 112, 201–214.

Kaul, M., Yang, B., and Jensen, C. S. (2013), "Building Accurate 3D Spatial Networks to Enable Next Generation Intelligent Transportation Systems," in *2013 IEEE 14th International Conference on Mobile Data Management*, vol. 1, pp. 137–146.

Keshavarzzadeh, V., Zhe, S., Kirby, R. M., and Narayan, A. (2021), "GP-HMAT: Scalable, ${O}(n\log(n))$ Gaussian Process Regression with Hierarchical Low-Rank Matrices," ArXiv:2201.00888 [cs, math].

Kim, H.-M., Mallick, B. K., and Holmes, C. C. (2005), "Analyzing Nonstationary Spatial Data Using Piecewise Gaussian Processes," *Journal of the American Statistical Association*, 100, 653–668.

Knorr-Held, L. and Rasser, G. (2000), "Bayesian Detection of Clusters and Discontinuities in Disease Maps," *Biometrics*, 56, 13–21.

Lindgren, F., Rue, H., and Lindström, J. (2011), "An Explicit Link Between Gaussian Fields and Gaussian Markov Random Fields: the Stochastic Partial Differential Equation Approach," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73, 423–498.

Majumder, S., Guan, Y., Reich, B. J., and Saibaba, A. K. (2022), "Kryging: Geostatistical Analysis of Large-Scale Datasets Using Krylov Subspace Methods," *Statistics and Computing*, 32, 74.

Minden, V., Damle, A., Ho, K. L., and Ying, L. (2017), "Fast Spatial Gaussian Process Maximum Likelihood Estimation via Skeletonization Factorizations," *Multiscale Modeling & Simulation*, 15, 1584–1611.

Nychka, D., Bandyopadhyay, S., Hammerling, D., Lindgren, F., and Sain, S. (2015), "A Multiresolution Gaussian Process Model for the Analysis of Large Spatial Datasets," *Journal of Computational and Graphical Statistics*, 24, 579–599.

Perlin, K. (1985), "An Image Synthesizer," *ACM SIGGRAPH Computer Graphics*, 19, 287–296.

Rasmussen, C. E. and Williams, C. K. I. (2005), *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*, The MIT Press.

Roustant, O., Ginsbourger, D., and Deville, Y. (2012), "DiceKriging, DiceOptim: Two R Packages for the Analysis of Computer Experiments by Kriging-Based Metamodeling and Optimization," *Journal of Statistical Software*, 51, 1–55.

Saad, Y. (2003), *Iterative Methods for Sparse Linear Systems*, SIAM.

Salvaña, M. L. O., Abdulah, S., Huang, H., Ltaief, H., Sun, Y., Genton, M. G., and Keyes, D. E. (2021), "High Performance Multivariate Geospatial Statistics on Manycore Systems," *IEEE Transactions on Parallel and Distributed Systems*, 32, 2719–2733.

Sang, H., Jun, M., and Huang, J. Z. (2011), "Covariance Approximation for Large Multivariate Spatial Data Sets with an Application to Multiple Climate Model Errors," *Annals of Applied Statistics*, 5, 2519–2548.

Stein, M. L. (2012), *Interpolation of Spatial Data: Some Theory for Kriging*, Springer Science & Business Media.

Ubaru, S., Chen, J., and Saad, Y. (2017), "Fast Estimation of Tr(f(A)) via Stochastic Lanczos Quadrature," *SIAM Journal on Matrix Analysis and Applications*, 38, 1075–1099.

Williams, C. and Seeger, M. (2000), "Using the Nyström Method to Speed Up Kernel Machines," in *Advances in Neural Information Processing Systems*, eds. Leen, T., Dietterich, T., and Tresp, V., MIT Press, vol. 13.

Woodbury, M. A. (1950), *Inverting Modified Matrices*, Statistical Research Group.

Zammit-Mangion, A., Cressie, N., and Shumack, C. (2018), "On Statistical Approaches to Generate Level 3 Products from Satellite Remote Sensing Retrievals," *Remote Sensing*, 10, 155.

Zhu, C., Byrd, R. H., Lu, P., and Nocedal, J. (1997), "Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization," *ACM Transactions on mathematical software (TOMS)*, 23, 550–560.