Preprint ANL/MCS-P1857-0311

A MATRIX-FREE APPROACH FOR SOLVING THE GAUSSIAN PROCESS MAXIMUM LIKELIHOOD PROBLEM

MIHAI ANITESCU*, JIE CHEN*, AND LEI WANG*

Abstract. Gaussian processes are the cornerstone of statistical analysis in many application areas. Nevertheless, most of the applications are limited by their need to use the Cholesky factorization in the computation of the likelihood. In this work, we present a matrix-free approach for computing the solution of the maximum likelihood problem involving Gaussian processes. The approach is based on a stochastic programming reformulation followed by sample average approximation applied to either the maximization problem or its optimality conditions. We provide statistical estimates of the approximate solution. The method is illustrated on several examples where the data is provided on a regular or irregular grid. In the latter case, the action of a covariance matrix on a vector is computed by means of fast multipole methods. For each of the examples presented, we demonstrate that the approach scales linearly with an increase in the number of sites.

Key words. Gaussian process, maximum likelihood estimation, sample average approximation, preconditioned conjugate gradient, Toeplitz system, circulant preconditioner, fast multipole method

AMS subject classifications. 65C60, 90C15, 65F10, 65F30

1. Introduction.

Gaussian processes (GPs) have been widely used throughout the statistical and machine learning communities for modeling natural processes [6, 7] for regression and classification problems [26], and for interpolation in parameter space for computer model output [18, 16]. In addition, the Gaussian processes web site (http://www.gaussianprocess.org) provides a good overview of the reach of these models. Their widespread use is in part due to the conceptual and computational advantages for processes that are a function of a continuous index, but also because they can be used as a building block for modeling non-Gaussian processes. For example, Diggle et al. [8] combine generalized linear models and GP models to model integer or binary-valued spatial processes. A similar approach is used for multiclass classification problems by attaching a latent GP to each class [26]. An important property of GPs is that they can be fully characterized by only their means and covariance functions.

Unfortunately, several factors limit their application to very large data problems. In this work we address one of the factors: the reliance of most algorithms involving GPs on the Cholesky factorization of the covariance matrix, which many times is dense. For this reason, such algorithms cannot scale to the size of problem currently confronting us. We discuss this difficulty in §1.1.

1.1. Calculation of the hypeparameters.

We are interested here in manipulating the multivariate Gaussian with mean $\mathbf{0}$ and covariance matrix $K(\boldsymbol{\theta})$. While the approach we present is general, the focus case is the one where $K(\boldsymbol{\theta})$ is attached to a stationary Gaussian process over a d dimensional space, defined by the covariance function $\phi(\boldsymbol{r};\boldsymbol{\theta})$, where ϕ is a scalar, symmetric, positive-definite function defined over $\mathbb{R}^d$ and $\boldsymbol{\theta}$ is its hyperparameter (examples of such functions are given in §4). Given the sites $\boldsymbol{x}_i$, $i = 1, 2, \ldots, n_d$, the i, j entry in $K(\boldsymbol{\theta})$ is $\phi(\boldsymbol{x}_i - \boldsymbol{x}_j;\boldsymbol{\theta})$. In this work, we are interested in the case where the number n_d of sites is large, with the aim of treating cases with $n_d = 10^6$ and beyond.

*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439. Emails: (anitescu, jiechen, lwang)@mcs.anl.gov

Denote by $\boldsymbol{y}$ the observed data vector of length n_d. The log-likelihood function assumes the form

$$\log p(\boldsymbol{y} \mid \boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{y}^T K(\boldsymbol{\theta})^{-1}\boldsymbol{y} - \frac{1}{2}\log|K(\boldsymbol{\theta})| - \frac{n_d}{2}\log 2\pi, \tag{1.1}$$

where $|A|$ is the determinant of A. For spatiotemporal processes, the hyperparameter $\boldsymbol{\theta}$ describes dependency as a function of the spatiotemporal coordinates of the observations. The gradients of the likelihood function can be computed analytically [26]:

$$\frac{\partial}{\partial\theta_j}\log p(\boldsymbol{y} \mid \boldsymbol{\theta}) = \frac{1}{2}\mathrm{tr}\left(\left((K(\boldsymbol{\theta})^{-1}\boldsymbol{y})(K(\boldsymbol{\theta})^{-1}\boldsymbol{y})^T - K(\boldsymbol{\theta})^{-1}\right)\frac{\partial K(\boldsymbol{\theta})}{\partial\theta_j}\right), \tag{1.2}$$

for $j = 1, 2, \ldots, n_\theta$. Here, $\mathrm{tr}(A)$ denotes the trace of a square matrix A, the sum of its diagonal entries.

Our aim is to find the parameter $\boldsymbol{\theta}$ that solves the maximum likelihood problem:

$$\max_{\boldsymbol{\theta}}\left[\log p(\boldsymbol{y} \mid \boldsymbol{\theta})\right] = \max_{\boldsymbol{\theta}}\left\{-\frac{1}{2}\boldsymbol{y}^T K(\boldsymbol{\theta})^{-1}\boldsymbol{y} - \frac{1}{2}\log|K(\boldsymbol{\theta})| - \frac{n_d}{2}\log 2\pi\right\}. \tag{1.3}$$

Such a parameter satisfies the optimality condition [24]

$$0 = \frac{\partial}{\partial\theta_j}\log p(\boldsymbol{y} \mid \boldsymbol{\theta}) = \frac{1}{2}\mathrm{tr}\left(\left((K(\boldsymbol{\theta})^{-1}\boldsymbol{y})(K(\boldsymbol{\theta})^{-1}\boldsymbol{y})^T - K(\boldsymbol{\theta})^{-1}\right)\frac{\partial K(\boldsymbol{\theta})}{\partial\theta_j}\right) \tag{1.4}$$

for $j = 1, 2, \ldots, n_\theta$. The equations (1.4) are sometimes called the score equations. The parameter $\boldsymbol{\theta}$ can thus be obtained by either the optimization approach (1.3) or the nonlinear equation approach (1.4). The problem can be readily modified to accommodate a mean function as well [26], with few changes to the principal challenges that we will encounter in this work. We thus confine the treatment to the mean $\mathbf{0}$ case, though all the results are readily extendable to the nonzero mean case.

Since we are interested in situations where the number of sample points is very large, the resolution of the maximum likelihood problem (1.3) or of the score equations (1.4) requires the inversion of large, symmetric, positive-definite matrices (or rather, the solution of their associated linear systems) and, in the case of the likelihood function, the computation of the determinants of those matrices. Since the inversion scales as n_d^3 and since the matrix $K(\boldsymbol{\theta})$ is typicaly dense, direct approaches (such as Cholesky factorization, the typically recommended approach to carry out such calculations [26]) rapidly lose their usefulness in terms of both computing time and storage.

Several approaches have been adopted to circumvent this difficulty, including data reduction strategies—effectively introducing irreducible approximations [15]. In this work, we are interested in asymptotically exact methods, such as in [12]. That reference adapts methods due to Skilling [29] to perform approximations to terms like $K(\boldsymbol{\theta})^{-1}\boldsymbol{y}$ that involve only matrix-vector multiplications by using the conjugate gradient method. For circumventing the issue of the calculation of the determinant in maximum likelihood, Gibbs and MacKay [12] propose an approach using only a stochastic estimator that approximates the score equations based on Hutchinson's trace estimator [17]. The approach needs only linear system solves with the covariance matrix, which reduces again to a conjugate gradient approach.

For the matrix-vector multiplication, several ideas for scalable computation have been explored. One can work with compactly supported kernels [26] or combine them

in Schur products with more general kernels, a technique called "tapering" [11]—but such ideas nonetheless result in an irreducible approximation. Unfortunately, as we demonstrate in Figure 4.1, such problems can result in maximum likelihoods with a very "bumpy" likelihood surface, which is thus difficult to explore by local optimization methods. If the data is on a grid and if the covariance matrix is stationary, then one can use spectral methods [26, 27]. Separability of the covariance function in output space leads to further computational simplifications—the covariance matrix for a given $\boldsymbol{\theta}$ value has a Kronecker product form, so its inverse can be calculated separately in each subspace. For cases not on the grid, the situation is considerably more difficult. Using techniques from N-body problems, one can reduce the multiplication complexity to $n_d \log n_d$ by using efficient approximations based on kd-trees and fast multipole expansions [13]—though in that reference the focus was on specific tasks of statistical analysis as opposed to fast matrix-vector multiplication. We note, however, that these methods were used primarily for the conditional prediction calculation, which assumes the parameter $\boldsymbol{\theta}$ is known. None of the methods have been tested in the context of computing the maximum likelihood, which is the focus of this work.

1.2. Our work. Building on the work we started in [5], where we presented a matrix-free approach for sampling from very large scale stationary GPs, we propose a matrix-free approach for computing the solution of the maximum likelihood problem by solving either (1.3) or (1.4). The approach uses the Hutchinson trace estimator to convert these problems in their stochastic average approximation versions. In turn, this results in operations that need only matrix-vector multiplications with $K(\boldsymbol{\theta})$ in a way that does not store any matrix. If the matrix-vector multiplication can be carried out in $O(n_d \log n_d)$, then these approaches also become properly scalable.

As a result, the approach opens an avenue in solving GP problems for very large data sets. In this work, we demonstrate the approach with data sets of the order of 10^6, but there is no conceptual limit to apply this approach in efficiently to data sets whose size is proportional to the computer memory available.

Our contribution, compared with the other references cited, is to connect the Hutchinson trace estimator with a sample average approximation (SAA) approach (see §2.2). This provides a mechanism to produce confidence intervals for the solution of the approximation procedure (§2.1). Note that using Hutchinson's trace estimator as a stochastic estimator only—for example, as a stochastic gradient method [31]—in a procedure that is subjected to the noise in the estimator would create difficulties when coupled with deterministic optimization approaches. The SAA approach uses sampling to create a smooth optimization problem that can be then solved with many of the tools available. The statistical nature of the estimator appears only as an error in the final solution, which we can control by the use of confidence intervals. Moreover, the SAA approach for the maximization formulation (1.3) as opposed to only the score equations (1.4), although not central to this work, is entirely new, to our knowledge.

In addition, we discuss in detail two novel items: the benefit of using block preconditioned conjugate gradients (§3.1) and the use of circulant preconditioners to accelerate the resulting linear solves (§3.2). We explore the use of fast multipole methods to provide a maximum likelihood solution that is matrix-free for the out-of-regular-grid case even for kernels that are not compactly supported (§3.3.1), another novel development to our knowledge since the focus in [13] was on functions of GPs that assume the parameters known. Moreover, our approach provides specific coefficients for the Matern function class. Most important, we demonstrate our findings in

the 10^6 data size range (§5), much larger than these described in [12]. These developments result in a scalable approach for GP analysis, provided that the preconditioning strategy is successful.

2. Sample average approximation. Important for our work is the following relationship, which connects the trace of a matrix A with an expectation [17]:

$$\mathrm{tr}(A) = \mathbb{E}_{\boldsymbol{u}}[\boldsymbol{u}^T A \boldsymbol{u}]. \tag{2.1}$$

Here $\boldsymbol{u}$ is a random vector with independent components, each taking the values -1 and 1 with probability 0.5. Using the relationship $\log|A| = \mathrm{tr}(\log(A))$ for positive definite A, we obtain that

$$\log p(\boldsymbol{y} \mid \boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{y}^T K(\boldsymbol{\theta})^{-1}\boldsymbol{y} - \frac{1}{2}\mathrm{tr}(\log(K(\boldsymbol{\theta}))) - \frac{n_d}{2}\log 2\pi. \tag{2.2}$$

From (2.2) and (2.1) and with a sign change, the maximum likelihood problem (1.3) then becomes the stochastic programming problem

$$\min_{\boldsymbol{\theta}} \left\{ \frac{1}{2}\mathbb{E}_{\boldsymbol{u}}\left[\boldsymbol{u}^T \log(K(\boldsymbol{\theta}))\boldsymbol{u}\right] + \frac{1}{2}\boldsymbol{y}^T K(\boldsymbol{\theta})^{-1}\boldsymbol{y} + \frac{n_d}{2}\log 2\pi \right\}. \tag{2.3}$$

Similary, from (2.2) and (2.1), the score equations (1.4) become the stochastic nonlinear equations

$$\boldsymbol{0} = \mathbb{E}_{\boldsymbol{u}}\left[\boldsymbol{F}(\boldsymbol{\theta}, \boldsymbol{u})\right], \tag{2.4}$$

where for $j = 1, 2, \ldots, n_\theta$,

$$F_j(\boldsymbol{\theta}, \boldsymbol{u}) = \frac{1}{2}\boldsymbol{u}^T \left(\left((K(\boldsymbol{\theta})^{-1}\boldsymbol{y})(K(\boldsymbol{\theta})^{-1}\boldsymbol{y})^T - K(\boldsymbol{\theta})^{-1} \right) \frac{\partial K(\boldsymbol{\theta})}{\partial \theta_j} \right) \boldsymbol{u}.$$

2.1. Foundation of SAA methods. At the center of our approach sits stochastic programming and one of its approaches to solving it. Stochastic programming typically refers to the following problem:

$$\min_{\boldsymbol{x}} \mathbb{E}_\omega\left[f\left(\boldsymbol{x}, \boldsymbol{\zeta}(\omega)\right)\right]. \tag{2.5}$$

Here $\boldsymbol{\zeta}$ is a vector-valued random variable over a probability space $(\Omega, \Sigma, \mathcal{P})$ with values in $\mathbb{R}^m$. Also, the function f satisfies $f : \mathbb{R}^p \times \mathbb{R}^m \to \mathbb{R}$. We assume that $f(\boldsymbol{x}, \boldsymbol{\zeta})$ is continuous in both variables and twice continuously differentiable in the first variable. A related problem is the stochastic nonlinear equation

$$\mathbb{E}_\omega\left[\boldsymbol{F}\left(x, \zeta(\omega)\right)\right] = 0, \tag{2.6}$$

where $\boldsymbol{F} : \mathbb{R}^p \times \mathbb{R}^m \to \mathbb{R}^p$. We assume that $\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{\zeta})$ is continuous and continuously differentiable in the first variable.

One of the ways to solve such problems, which we pursue in this paper, is by using a sample average approximation (SAA) of the expectation. That is, we extract N independent, identically distributed samples $\boldsymbol{\zeta}^i(\omega)$, $i = 1, 2, \ldots, N$, and, for a function $\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\zeta}) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^p$, we write

$$\mathbb{E}_\omega\left[\boldsymbol{g}\left(\boldsymbol{x}, \boldsymbol{\zeta}(\omega)\right)\right] \approx \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{g}\left(\boldsymbol{x}, \boldsymbol{\zeta}^i(\omega)\right).$$

Note that for one experiment, $\boldsymbol{\zeta}^i(\omega)$ will be a fixed vector; the ω dependence simply denotes its random variable nature.

For the nonlinear equation formulation (2.6), the SAA approach becomes

$$\mathbf{0} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{F}\left(\boldsymbol{x}, \boldsymbol{\zeta}^i(\omega)\right), \tag{2.7}$$

whose solution we denote by $\boldsymbol{x}^N(\omega)$, a random variable. We assume that (2.6) has a solution $\boldsymbol{x}^*$ at which its Jacobian J, defined as:

$$J = \nabla_{\boldsymbol{x}} \mathbb{E}_\omega \left[\boldsymbol{F}\left(\boldsymbol{x}^*, \boldsymbol{\zeta}(\omega)\right)\right],$$

is nonsingular. It follows that $\boldsymbol{x}^N$ is locally unique in a neighborhood of $\boldsymbol{x}^*$ [28, Theorem 5.14] for N sufficiently large. We will assume from here on that it is unique. We define the empirical Jacobian and covariance matrix as

$$J^N = \frac{1}{N}\sum_{i=1}^{N} \nabla_{\boldsymbol{x}} \boldsymbol{F}\left(\boldsymbol{x}^N, \boldsymbol{\zeta}^i(\omega)\right), \quad \Sigma^N = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{F}\left(\boldsymbol{x}^N, \boldsymbol{\zeta}^i(\omega)\right) \boldsymbol{F}\left(\boldsymbol{x}^N, \boldsymbol{\zeta}^i(\omega)\right)^T. \tag{2.8}$$

We then obtain [28, (5.82)] the following convergence in distribution:

$$N^{\frac{1}{2}} \left[V^N\right]^{-\frac{1}{2}} \left(\boldsymbol{x}^N(\omega) - \boldsymbol{x}^*\right) \xrightarrow{\mathcal{D}} \mathcal{N}(\mathbf{0}, I), \tag{2.9}$$

where V^N is the empirical variance matrix

$$V^N = \left[J^N\right]^{-T} \Sigma^N \left[J^N\right]^{-1}. \tag{2.10}$$

Here we assume the variance of $\boldsymbol{F}\left(\boldsymbol{x}^*, \boldsymbol{\zeta}(\omega)\right)$ is positive definite, which means that, with probability 1, so will be Σ^N for N sufficiently large. In addition, to obtain (2.9) from [28, (5.82)], we use the fact that Σ^N and J^N converge with probability 1. The relationships (2.9) and (2.10) are used to obtain confidence intervals for the estimates $\boldsymbol{x}^N$ of $\boldsymbol{x}^*$.

For the optimization problem (2.5), the SAA approximation becomes

$$\min_{\boldsymbol{x}} \frac{1}{N}\sum_{i=1}^{N} f\left(\boldsymbol{x}, \boldsymbol{\zeta}^i(\omega)\right). \tag{2.11}$$

We denote the solution of this problem by $\boldsymbol{x}^{N,o}(\omega)$. Define the Hessian

$$H = \nabla^2_{\boldsymbol{x}\boldsymbol{x}} \mathbb{E}_\omega \left[f\left(\boldsymbol{x}^*, \boldsymbol{\zeta}(\omega)\right)\right]. \tag{2.12}$$

Then, if H is invertible (which follows if the optimization problem satisfies the second-order sufficient condition [24]), we obtain, similarly to the argument in the nonlinear equation case, that

$$N^{\frac{1}{2}} \left[V^{N,o}\right]^{-\frac{1}{2}} \left(\boldsymbol{x}^{N,o}(\omega) - \boldsymbol{x}^*\right) \xrightarrow{\mathcal{D}} \mathcal{N}(\mathbf{0}, I), \tag{2.13}$$

where $V^{N,o}$ is the empirical variance matrix

$$V^{N,o} = [H^{N,o}]^{-1} \Sigma^{N,o} [H^{N,o}]^{-1} \tag{2.14}$$

with

$$H^{N,o} = \frac{1}{N}\sum_{i=1}^{N} \nabla^2_{\boldsymbol{xx}} \left[f\left(\boldsymbol{x}^{N,o}, \boldsymbol{\zeta}^i(\omega)\right)\right], \quad \Sigma^{N,o} = \frac{1}{N}\sum_{i=1}^{N} \nabla_x f\left(\boldsymbol{x}^{N,o}, \boldsymbol{\zeta}^i(\omega)\right) \nabla_x f\left(\boldsymbol{x}^{N,o}, \boldsymbol{\zeta}^i(\omega)\right)^T.$$

The relationships (2.14) and (2.13) are important for establishing confidence intervals for the estimates $\boldsymbol{x}^{N,o}$ of the true vector $\boldsymbol{x}^*$. We note that statistical estimates can be indeed obtained from one sample set of length N as long as we assume local uniqueness for x^*; the latter is not the case if there are multiple solutions when replications may be needed [28, Theorem 5.7].

2.2. SAA-based matrix-free methods for maximum likelihood. The SAA approach described in §2.1 is thus applicable to either the stochastic optimization (2.3) or the stochastic nonlinear equation (2.4). Let $\boldsymbol{u}_1$, $\boldsymbol{u}_2$, ..., $\boldsymbol{u}_N$ be identically distributed, independent samples from variables whose entries are independently distributed with values ± 1, each with probability 0.5. Define $\boldsymbol{\theta}^{N,o}$ to be the solution of the following SAA approximation of the expectation form (2.3) of (1.3):

$$\min_{\boldsymbol{\theta}} \left\{ \frac{1}{2}\sum_{i=1}^{N} \boldsymbol{u}_i^T \log(K(\boldsymbol{\theta}))\boldsymbol{u}_i + \frac{1}{2}\boldsymbol{y}^T K(\boldsymbol{\theta})^{-1}\boldsymbol{y} + \frac{n_d}{2}\log 2\pi \right\}. \tag{2.15}$$

If $\boldsymbol{\theta}^*$ is the unique solution of the maximum likelihood problem (1.3) and if it satisfies the second-order sufficient condition, then based on (2.13), we obtain that $\boldsymbol{\theta}^{N,o}$ converges to $\boldsymbol{\theta}^*$ in the following distribution sense:

$$N^{\frac{1}{2}}\left[V^{N,o}\right]^{-\frac{1}{2}}\left(\boldsymbol{\theta}^{N,o}(\omega) - \boldsymbol{\theta}^{*,o}\right) \xrightarrow{\mathcal{D}} \mathcal{N}(\mathbf{0}, I), \tag{2.16}$$

where $V^{N,o}$ is the variance estimator obtained from (2.14) based on the Hessian of the objective (2.15) at $\boldsymbol{\theta}^{N,o}$. Under the same conditions, we define the following SAA nonlinear equation of the expectation form (2.4) of (1.4):

$$0 = \sum_{i=1}^{N} \boldsymbol{u}_i^T \left(\left((K(\boldsymbol{\theta})^{-1}\boldsymbol{y})(K(\boldsymbol{\theta})^{-1}\boldsymbol{y})^T - K(\boldsymbol{\theta})^{-1}\right)\frac{\partial K(\boldsymbol{\theta})}{\partial \theta_j}\right)\boldsymbol{u}_i \quad j = 1, 2, \ldots, n_\theta. \tag{2.17}$$

We define its solution as $\boldsymbol{\theta}^N$. Then, based on (2.9), we obtain the following convergence result in distribution of $\boldsymbol{\theta}^N$ to $\boldsymbol{\theta}^*$:

$$N^{\frac{1}{2}}\left[V^{N}\right]^{-\frac{1}{2}}\left(\boldsymbol{\theta}^{N}(\omega) - \boldsymbol{\theta}^{*}\right) \xrightarrow{\mathcal{D}} \mathcal{N}(\mathbf{0}, I), \tag{2.18}$$

where V^N is the estimate of the variance matrix of the solution, obtained from (2.10), where J^N is the Jacobian of (2.17) at $\boldsymbol{\theta}^N$.

Using the techniques from our previous work [5], we can compute $\log(A)\boldsymbol{u}$ using only matrix-vector multiplications with the matrix A. In turn, the objective function of the optimization problem (2.15) can be computed by using only matrix-vector multiplications by means of a Krylov method to compute $K(\boldsymbol{\theta})^{-1}\boldsymbol{y}$ and thus $\boldsymbol{y}^T K(\boldsymbol{\theta})^{-1}\boldsymbol{y}$. Subsequently, a derivative-free method can be used to carry out the optimization, or the gradients can be approximated with finite differences and used in a quasi-Newton method. Both approaches are matrix-free.

Similarly, we can solve (2.17) using a matrix-free approach. The terms $K(\boldsymbol{\theta})^{-1}\boldsymbol{y}$ can be evaluated matrix-free as above. The term $\frac{\partial K(\boldsymbol{\theta})}{\partial \theta_j}\boldsymbol{u}_i$ can be evaluated matrix-free

by generating the partial derivative on the fly. This can be used in a matrix-free quasi-Newton approach for the nonlinear equation, or by computing divided differences and using a Newton approach, or finally, by implementing their gradient analytically or by using automatic differentiation. In addition (2.18) and (2.16) can be used to generate confidence intervals for the solution of the maximum likelihood problem itself, (1.3) and (1.4).

The two approaches are closely related (note, however, that the SAA approximation of the nonlinear equation is not the same with the gradient of the SAA approximation of the optimization problem, as shown with a simple example in the Appendix), and our development provides statistical estimates for either. Nevertheless, the one involving the optimization problem has the additional complexity of evaluating the action of the logarithm of the covariance matrix on a vector. In the rest of the work we therefore concentrate on the nonlinear equation approach.

3. Solving linear systems arising from covariance kernels. The stochastic nonlinear equation approach (2.17) requires solving $N+1$ linear systems ($K^{-1}\boldsymbol{y}$ and $K^{-1}\boldsymbol{u}_i$ for $i = 1, \ldots, N$) in each evaluation of the nonlinear system. The (i, j)-entry of the covariance matrix K is defined based on a covariance kernel, which is a positive-definite function. Therefore, the resulting K is always symmetric positive definite. When K is large, a natural choice of the solver is a Krylov-type iterative method, in particular the preconditioned conjugate gradient (PCG). Here, one can take advantage of the existence of the multiple right hand-sides and use the block version of PCG [25, 23]. In the literature, this method is usually called *block preconditioned conjugate gradient* (BPCG). We note here that the block form is applied to CG, rather than to the preconditioner (in which case it is called a "block preconditioner".) For the sake of completeness, the BPCG method is briefly introduced in §3.1; more details can be found in, e.g., [25, 23].

The problem presented here entails many similarities to the problem of radial basis function (RBF) interpolation [10, 14]. The interpolation problem finds an interpolant in the form $\sum_{i=1}^{n} a_i \psi(\|\boldsymbol{x} - \boldsymbol{x}_i\|) + c$, to interpolate some function whose values are known only at the points $\{\boldsymbol{x}_i\}$. Here, the unknowns of the interpolant are a_i and c, subject to the constraint $\sum a_i = 0$, and $\psi(r)$ is a radial basis function. The problem amounts to solving a linear system that is augmented from K with entries $K_{ij} = \psi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|)$, where the solution vector $\boldsymbol{a} = [a_i]$ is orthogonal to $\mathbf{1}$ (the vector of all ones). In contrast to the covariance kernel, which is decreasing as r increases, the generally used RBF kernels (such as biharmonic, polyharmonic and multiquadric) are increasing functions. One can show that the matrix K resulting from some RBF kernels is negative definite constrained to the subspace, which is the orthogonal complement of span$\{\mathbf{1}\}$ (see, e.g., [22]). A preconditioned Krylov iteration method was proposed in [10, 14]. Although it was mentioned that the method was analogous to PCG, we note that in each iteration the method entails a minimization property whereby the solution vector is chosen to minimize the (semi) K-norm of the residual vector, whereas in the standard PCG method the solution vector minimizes the K-norm of the error vector.

To simplfy notation, in what follows, we use n (instead of n_d as in §1) to denote the size of K. Since we use a BPCG solver, the key to a good performance relies on two factors: (a) how efficiently a matrix-vector multiplication can be performed and (b) how efficiently a preconditioner can be constructed and be applied to a vector. In addition to solving linear systems, matrix-vector multiplications are needed for the partial derivatives $\partial K/\partial \theta_j$. Since the matrix K is dense (essentially full), it is not

expected that the matrix-vector multiplication can be conducted in $O(n)$ time. However, considering the special structure of K, there exist $O(n \log n)$ time/$O(n)$ space algorithms that run asymptotically faster and require much less memory than does the brute-force matrix-vector multiplication. This is also true for some preconditioners. Details are in order starting from §3.2.

3.1. Block preconditioned conjugate gradient. The block PCG method is an extension of the standard PCG method for solving a symmetric positive definite system (with multiple right-hand sides)

$$AX = B,$$

where the $n \times s$ matrix B contains s right-hand vectors (sometimes called block vectors) and $X \in \mathbb{R}^{n\times s}$ contains the s unknown vectors. BPCG generates a sequence of A-conjugate search direction matrices $\{P_j\}$ and updates the solution matrix X_{j+1} and the residual matrix R_{j+1} for each j. With a symmetric positive-definite preconditioner M, the update formulas are as follows:

$$\begin{aligned} X_{j+1} &= X_j + P_j\alpha_j, \\ R_{j+1} &= R_j - AP_j\alpha_j, \\ P_{j+1} &= (M^{-1}R_{j+1} + P_j\beta_j)\gamma_{j+1}, \end{aligned}$$

where

$$\begin{aligned} \alpha_j &= (P_j^T AP_j)^{-1}\gamma_j^T(R_j^T M^{-1}R_j), \\ \beta_j &= \gamma_j^{-1}(R_j^T M^{-1}R_j)^{-1}(R_{j+1}^T M^{-1}R_{j+1}). \end{aligned}$$

Here, the series of $s \times s$ matrices $\{\gamma_j\}$ is unspecified, but a typical construction is to use them to orthogonalize the search direction matrices P_j.

One can show that the residual matrices R_j are M^{-1}-orthogonal. If we define the *block-span* of a set of matrices $Y_j \in \mathbb{R}^{n\times s}$, $j = 1 : k$ as

$$\text{block-span}\{Y_1, \ldots, Y_k\} := \left\{ \sum_{j=1}^{k} Y_j\gamma_j \;\middle|\; \gamma_j \in \mathbb{R}^{s\times s} \right\},$$

then one can see that R_{k+1} is orthogonal to

$$\mathcal{K}_k := \text{block-span}\{M^{-1}R_0, (M^{-1}A)M^{-1}R_0, \ldots, (M^{-1}A)^k M^{-1}R_0\}.$$

Thus, the estimated solution X_{k+1} minimizes $\text{tr}[(X - X_*)^T A(X - X_*)]$ among all $X \in X_0 + \mathcal{K}_k$, where $X_* = A^{-1}B$ is the exact solution. An immediate consequence is that in exact arithmetic, BPCG will terminate within $\lceil n/s \rceil$ iterations. However, two issues affect the performance of BPCG in practical situations. The first is that similar to PCG, BPCG has a linear rate convergence; thus it often requires far fewer than $\lceil n/s \rceil$ steps to converge when n/s is large. The second fact is that because of round-off errors, the orthogonality of R_{k+1} to $\mathcal{K}_k$ can be quickly lost. Consequently when n/s is not large enough, $\lceil n/s \rceil$ iterations often are not enough to decrease the residual matrices to sufficiently small.

BPCG may break down when $P_j^T AP_j$ or $R_j^T M^{-1}R_j$ becomes numerically rank-deficient, which can be triggered by the convergence of one or several systems or

by round-off errors. A general practice in such a case is to drop the corresponding columns that cause rank-deficiency in all the involved iterates (X_j, P_j and R_j) and to continue the iteration. In addition, normalizing all the right-hand sides and initial solution vectors helps reducing the numerical rank deficiency.

3.2. Matrix-vector multiplication and preconditioner: regular grid case. We now investigate the two details of implementing BPCG: matrix-vector multiplication and the preconditioner. First consider a simple case, where the sites $\{\boldsymbol{x}_i\}$ are located on a regular $n_1 \times n_2$ grid with equally spacing grid points. Let the sites (grid points) be ordered from top to bottom and from left to right, that is, the top-left corner is $\boldsymbol{x}_1$, the point below it is $\boldsymbol{x}_2$, and the one to the right is $\boldsymbol{x}_{n_1+1}$. It is not hard to see that the resulting covariance matrix K (and the partial derivatives $\partial K/\partial\theta_j$) is an $n_2 \times n_2$ block Toeplitz matrix, where each block is also Toeplitz and has size $n_1 \times n_1$. Such matrices are called BTTB matrices (block Toeplitz with Toeplitz blocks).

3.2.1. Circulant matrix and BCCB matrix. A circulant matrix C_n of order n has the following form:

$$C_n = \begin{bmatrix} c_0 & c_{n-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{n-1} & & c_2 \\ \vdots & c_1 & c_0 & \ddots & \vdots \\ c_{n-2} & & \ddots & \ddots & c_{n-1} \\ c_{n-1} & c_{n-2} & \cdots & c_1 & c_0 \end{bmatrix}.$$

It is well known that C_n can be diagonalized by the FFT (fast Fourier transform) matrix F_n of the same order:

$$F_n C_n F_n^* = \Lambda_n = \operatorname{diag}(\lambda_1, \ldots, \lambda_n), \tag{3.1}$$

where

$$(F_n)_{jk} = \omega^{jk}/\sqrt{n}, \quad \omega = \exp(2\pi\mathbf{i}/n), \quad \mathbf{i} = \sqrt{-1},$$

and the λ_i's are the eigenvalues of C_n. Let $\boldsymbol{e}_i$ be the ith column of the identity matrix. From (3.1), it is not hard to verify the following:

$$\Lambda_n \mathbf{1}_n = \sqrt{n} F_n (C_n \boldsymbol{e}_1),$$

which indicates that the eigenvalues of C_n can be obtained by performing one FFT on the first column of C_n. Therefore, the time cost of finding the eigenvalues of a circulant matrix is $O(n \log n)$.

Multiplying C_n by a vector $\boldsymbol{q}$ or solving a linear system with respect to C_n also takes $O(n \log n)$ time, where in essence the process involves only one or a few FFTs or inverse FFTs. For example, consider

$$C_n^{-1}\boldsymbol{q} = F_n^* \Lambda_n^{-1} F_n \boldsymbol{q} = F_n^*(\Lambda_n^{-1}(F_n \boldsymbol{q})).$$

This means that $C_n^{-1}\boldsymbol{q}$ can be obtained by first computing an FFT on $\boldsymbol{q}$, then dividing the resulting vector by the eigenvalues of C_n element wise, and performing an inverse FFT on the resulting vector.

A BCCB matrix is a block circulant matrix with circulant blocks. In notation, we let

$$C_{n_1n_2} = \begin{bmatrix} C_{(0)} & C_{(n_2-1)} & \cdots & C_{(2)} & C_{(1)} \\ C_{(1)} & C_{(0)} & C_{(n_2-1)} & & C_{(2)} \\ \vdots & C_{(1)} & C_{(0)} & \ddots & \vdots \\ C_{(n_2-2)} & & \ddots & \ddots & C_{(n_2-1)} \\ C_{(n_2-1)} & C_{(n_2-2)} & \cdots & C_{(1)} & C_{(0)} \end{bmatrix}$$

denote an $n_2 \times n_2$ block circulant matrix, where each block $C_{(i)}$ is $n_1 \times n_1$ circulant. To be consistent, we assume $n = n_1n_2$ and, when necessary (avoiding tedious notation), drop the order indicator n in the subscript. The BCCB matrix $C_{n_1n_2}$ can be diagonalized by the Kronecker product of two FFT matrices of appropriate orders. Let each block $C_{(i)}$ be diagonalized as

$$F_{n_1} C_{(i)} F_{n_1}^* = \Lambda_{(i)}, \qquad i = 0, \ldots, n_2 - 1.$$

Then

$$(F_{n_2} \otimes F_{n_1}) C_{n_1n_2} (F_{n_2} \otimes F_{n_1})^* = \operatorname{diag}\left(\sum_{i=0}^{n_2-1} \Lambda_{(i)},\ \sum_{i=0}^{n_2-1} \omega^i \Lambda_{(i)},\ \ldots\ \sum_{i=0}^{n_2-1} \omega^{(n_2-1)i} \Lambda_{(i)} \right).$$

In other words, $C_{n_1n_2}$ is diagonalized by $F_{n_2} \otimes F_{n_1}$, which results in a block diagonal matrix whose jth diagonal block is a diagonal matrix $\sum_i \omega^{ji} \Lambda_{(i)}$. Therefore, the eigenvalues of a BCCB matrix can be obtained by a 2D FFT. Since the computational cost of 2D FFT is $O(n \log n)$, multiplying a BCCB matrix with a vector or solving a BCCB system thus also takes $O(n \log n)$ time.

3.2.2. Matrix-vector multiplication with a BTTB matrix. A Toeplitz matrix T_n of order n has the form

$$T_n = \begin{bmatrix} t_0 & t_{-1} & \cdots & t_{-n+2} & t_{-n+1} \\ t_1 & t_0 & t_{-1} & & t_{-n+2} \\ \vdots & t_1 & t_0 & \ddots & \vdots \\ t_{n-2} & & \ddots & \ddots & t_{-1} \\ t_{n-1} & t_{n-2} & \cdots & t_1 & t_0 \end{bmatrix}.$$

A matrix-vector multiplication with T_n can be performed efficiently by first embedding T_n in a circulant matrix:

$$C_{2n} = \begin{bmatrix} T_n & * \\ * & T_n \end{bmatrix},$$

where the two blocks denoted by $*$ are filled with elements so as to ensure that C_{2n} is circulant. Then, the matrix-vector multiplication $T_n \boldsymbol{q}$ can be augmented as

$$\begin{bmatrix} T_n \boldsymbol{q} \\ * \end{bmatrix} = \begin{bmatrix} T_n & * \\ * & T_n \end{bmatrix} \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{0} \end{bmatrix}.$$

In other words, one needs to pad $\boldsymbol{q}$ with n zeros and then multiply it by the circulant matrix C_{2n}. The vector $T_n \boldsymbol{q}$ is obtained from the first n components of the resulting vector. The computational cost is thus $O(n \log n)$.

With a similar notation as a BCCB matrix, a BTTB matrix has the form

$$T_{n_1 n_2} = \begin{bmatrix} T_{(0)} & T_{(-1)} & \cdots & T_{(-n_2+2)} & T_{(-n_2+1)} \\ T_{(1)} & T_{(0)} & T_{(-1)} & & T_{(-n_2+2)} \\ \vdots & T_{(1)} & T_{(0)} & \ddots & \vdots \\ T_{(n_2-2)} & & \ddots & \ddots & T_{(-1)} \\ T_{(n_2-1)} & T_{(n_2-2)} & \cdots & T_{(1)} & T_{(0)} \end{bmatrix},$$

where each $T_{(i)}$ is an $n_1 \times n_1$ block. To multiply $T_{n_1 n_2}$ with a vector $\boldsymbol{q}$, we need a double circulant embedding: first embed each block $T_{(i)}$ in a circulant form to obtain a block Toeplitz matrix with circulant blocks and then embed the resulting block Toeplitz matrix in a block circulant matrix, which results in a BCCB matrix. Using the same zero-padding trick as above, we can compute the matrix-vector product $T_{n_1 n_2}\boldsymbol{q}$ in $O(n \log n)$ time by using this BCCB embedding.

3.2.3. Preconditioner. There exist many circulant preconditioners for a Toeplitz system. One benefit of using a circulant preconditioner for a dense system is that it entails a $O(n \log n)$ multiplication cost. In what follows, we consider the T. Chan's circulant preconditioner [3, 4, 2, 32]. This preconditioner possesses several advantages: (a) it is a general preconditioner and does not only applied to Toeplitz systems; (b) when applied to a Toeplitz system, it does not require the knowledge of the generating function, if any, of the Toeplitz system; (c) it can be efficiently constructed; and (d) it can be easily generalized for a BTTB system, resulting in a BCCB preconditioner.

The T. Chan's circulant preconditioner, denoted $c^{(1)}(A_n)$, for a general symmetric matrix A_n of order n, minimizes

$$\|C_n - A_n\|_F$$

among all circulant matrices C_n. Since the objective is to minimize the Frobenius norm, each diagonal of the matrix A_n can be considered separately. Clearly, c_i, the ith entry of the first column of $c^{(1)}(A_n)$, should be the average of all the entries located at the ith lower diagonal and the $(n-i)$th upper diagonal of A_n, namely,

$$c_i = \frac{1}{n} \sum_{(k-\ell) \bmod n = i} (A_n)_{k\ell}. \tag{3.2}$$

Therefore, when $A_n = T_n$, a Toeplitz matrix, the first column of $c^{(1)}(T_n)$ is given by

$$c_i = \frac{1}{n}[(n-i)t_i + it_{-n+i}], \qquad i = 0, \ldots, n-1.$$

Clearly, $c^{(1)}(T_n)$ can be constructed in $O(n)$ time, whereas $c^{(1)}(A_n)$ for a general matrix A_n usually requires $O(n^2)$ time, unless special structures of A_n can be exploited to reduce the cost.

Using a similar notation as for block Toeplitz/circulant matrices, we let

$$A_{n_1 n_2} = \begin{bmatrix} A_{(0,0)} & \cdots & A_{(0,n_2-1)} \\ \vdots & \ddots & \vdots \\ A_{(n_2-1,0)} & \cdots & A_{(n_2-1,n_2-1)} \end{bmatrix}$$

denote an $n_2 \times n_2$ block matrix where each block $A_{(k,\ell)}$ has size $n_1 \times n_1$. The BCCB preconditioner for $A_{n_1 n_2}$, denoted by $c^{(2)}(A_{n_1 n_2})$, minimizes

$$\|C_{n_1 n_2} - A_{n_1 n_2}\|_F$$

among all BCCB matrices $C_{n_1 n_2}$. Let $c_j^{(i)}$ denote the jth entry of the first column of $C_{(i)}$. Then, using a similar argument as above, the optimal $c_j^{(i)}$ is given by the formula

$$c_j^{(i)} = \frac{1}{n_1 n_2} \sum_{(k-\ell) \bmod n_2 = i} \; \sum_{(p-q) \bmod n_1 = j} \left(A_{(k,\ell)}\right)_{p,q}. \tag{3.3}$$

In the special case when $A_{n_1 n_2} = T_{n_1 n_2}$, a BTTB matrix, $c_j^{(i)}$ admits the following expression:

$$c_j^{(i)} = \frac{1}{n_1 n_2} \left\{ (n_2 - i) \left[(n_1 - j) t_j^{(i)} + j t_{j-n_1}^{(i)} \right] + i \left[(n_1 - j) t_j^{(i-n_2)} + j t_{j-n_1}^{(i-n_2)} \right] \right\},$$

where $t_j^{(i)}$ for all j denotes the elements of the block $T_{(i)}$. Therefore, the construction cost of the BCCB preconditioner $c^{(2)}(T_{n_1 n_2})$ is also $O(n)$.

For both the circulant and the BCCB preconditioners, a superlinear convergence rate can be established for PCG if the Toeplitz/BTTB system is generated by a function in the Wiener class. These generating kernels are different from what we call here covariance kernels, which define the covariance matrix (although they are related by the spectral density of the Gaussian process). Generating kernels of a Toeplitz system are not discussed in this paper, and interested readers are referred to Reference [3].

3.2.4. Summary. Since the covariance matrix K resulting from a regular grid is a BTTB system, it can be multiplied by a vector in $O(n \log n)$ time by first being embedded in a BCCB matrix. It also entails a corresponding BCCB preconditioner that can be constructed in $O(n)$ time and be multiplied by a vector in $O(n \log n)$ time.

3.3. Matrix-vector multiplication and preconditioner: non-regular grid case. When the covariance matrix K is generated from a non-regular grid, the BTTB property of K (and the partial derivatives $\partial K / \partial \theta_j$) is lost. Therefore, the BCCB embedding trick for performing matrix-vector multiplication is no longer applicable. Since K is generated from a kernel function, however, the fast multipole method (FMM) is a suitable candidate for efficiently performing the multiplication. The time cost of FMM generally is considered $O(n \log n)$.

3.3.1. Fast multipole method. To compute the matrix-vector product $\boldsymbol{s} = K\boldsymbol{q}$, we write each entry in the following form:

$$s_i = \sum_{j=1}^{n} q_j \phi(\boldsymbol{x}_i - \boldsymbol{x}_j; \boldsymbol{\theta}). \tag{3.4}$$

FMM uses a far-field expansion to approximate the kernel ϕ for a given level of accuracy. The treecode algorithm for implementing FMM was introduced by Barnes and Hut in [1]; it used a monopole (first-order) approximation and a divide-and-conquer evaluation strategy. The treecode with higher-order approximations was

developed for different kernel functions in [9, 20, 21]. Our code was implemented based on [19].

The outline of the algorithm is as following. The first step is inputting data, such as the coefficients q_j and the particle positions $\boldsymbol{x}_i$ (here we treat the sites $\boldsymbol{x}_i$ as particles). Then we construct a hierarchical tree of particle clusters. The root cluster is a box (in 2D) containing all the particles. The root is bisected along the Cartesian axes, and the four children become subclusters of the root. The child clusters are then bisected, and the process continues until a cluster contains fewer than N_0 particles, where N_0 is a user-specified parameter. After the tree is constructed, the code cycles through every point. If the point and the cluster are well separated, then Talyor approximation is processed. The clusters that are not well separated are processed by direct summation. The code uses a multipole acceptance criterion to determine whether a given point and cluster are well separated. The criterion for being well separated is $r/R < c$, where r is the radius of the cluster, R is the distance between the point and the center of the cluster, and c is a user-specified parameter that, together with the order p of Taylor expansion, controls the series truncation error.

Assume that the particles have been divided into disjoint clusters, and write the sum (3.4) as a sum of particle-cluster interactions

$$s(\boldsymbol{x}_i) = \sum_{j=1}^{n} q_j \phi(\boldsymbol{x}_i - \boldsymbol{x}_j) = \sum_{C} s(\boldsymbol{x}_i, C),$$

where, for each cluster C,

$$s(\boldsymbol{x}_i, C) = \sum_{\boldsymbol{y}_j \in C} q_j \phi(\boldsymbol{x}_i - \boldsymbol{y}_j).$$

If particle $\boldsymbol{x}_i$ and cluster C are well separated, we expand ϕ in a Taylor series with respect to $\boldsymbol{y}$ about $\boldsymbol{y}_c$, where $\boldsymbol{y}_c$ is the center of the cluster C:

$$\begin{aligned} s(\boldsymbol{x}_i, C) &\approx \sum_{\boldsymbol{y}_j \in C} q_j \sum_{||\boldsymbol{k}||=0}^{p} \frac{1}{\boldsymbol{k}} D_{\boldsymbol{y}}^{\boldsymbol{k}} \phi(\boldsymbol{x}_i, \boldsymbol{y}_c)(\boldsymbol{y}_j - \boldsymbol{y}_c)^{\boldsymbol{k}} \\ &= \sum_{||\boldsymbol{k}||=0}^{p} \frac{1}{\boldsymbol{k}} D_{\boldsymbol{y}}^{\boldsymbol{k}} \phi(\boldsymbol{x}_i, \boldsymbol{y}_c) \sum_{\boldsymbol{y}_j \in C} q_j (\boldsymbol{y}_j - \boldsymbol{y}_c)^{\boldsymbol{k}}. \end{aligned}$$

Here, Cartesian multi-index notation has been used with $\boldsymbol{k} = (k_1, k_2), k_i \in \mathbb{N}$, $||\boldsymbol{k}|| = k_1 + k_2$, $\boldsymbol{k}! = k_1! k_2!$, $\boldsymbol{y} = (y_1, y_2), y_i \in \mathbb{R}$, $\boldsymbol{y}^k = y_1^{k_1} y_2^{k_2}$ and p is the order of the Taylor expansion. In the above approximation, the inner summation term

$$m_{\boldsymbol{k}}(C) = \sum_{\boldsymbol{y}_j \in C} q_j (\boldsymbol{y}_j - \boldsymbol{y}_c)^{\boldsymbol{k}} \tag{3.5}$$

is called the cluster moment. It needs to be calculated only once for each cluster. On the other hand, the Taylor coefficients

$$a_{\boldsymbol{k}}(\boldsymbol{x}_i, \boldsymbol{y}_c) = \frac{1}{\boldsymbol{k}!} D_{\boldsymbol{y}}^{k} \phi(\boldsymbol{x}_i, \boldsymbol{y}_c) \tag{3.6}$$

are dynamically generated on the fly since not all clusters are far away from $\boldsymbol{x}_i$. We will need to derive a recurrence relation to efficiently compute $a_{\boldsymbol{k}}(\boldsymbol{x}_i, \boldsymbol{y}_c)$ for a specific kernel ϕ we are interested in; this task is the subject of §4.

3.3.2. Preconditioner. The T. Chan's circulant preconditioner $c^{(1)}(A)$ and the BCCB preconditioner $c^{(2)}(A)$ are general preconditioners. They can be constructed by using (3.2) and (3.3) for a general symmetric matrix A that does not necessarily have a (block) Toeplitz structure. Empirically, the two preconditioners perform similarly, and they both work well when there is an underlying grid structure for the sites $\{\boldsymbol{x}_i\}$, even though the grid may not be a regular one. In §5, we show such results for a deformed grid.

3.3.3. Summary. When the sites are not on a regular grid, FMM with $O(n \log n)$ cost is used to perform matrix-vector multiplications. Circulant preconditioners are still the choice for preconditioners, whose construction unfortunately takes $O(n^2)$ time.

4. The covariance kernel. The covariance kernel $\phi(\boldsymbol{r}; \boldsymbol{\theta})$ defines the covariance matrix K with entries $K_{ij}(\boldsymbol{\theta}) = \phi(\boldsymbol{x}_i - \boldsymbol{x}_j; \boldsymbol{\theta})$. Among the many choices of covariance kernel ϕ, one with a finite support often results in a sparse covariance matrix K where nonzeros are located only on a few diagonals. A benefit of such a kernel is that the cost of matrix-vector multiplications is only linear to the number of sampling sites. However, a compact kernel sometimes results in bumpy surfaces of (the SAA) of the gradient of the log-likelihood (see (2.17)):

$$\Psi_j^N(\boldsymbol{\theta}) = \sum_{i=1}^{N} \boldsymbol{u}_i^T \left(\left((K(\boldsymbol{\theta})^{-1}\boldsymbol{y})(K(\boldsymbol{\theta})^{-1}\boldsymbol{y})^T - K(\boldsymbol{\theta})^{-1} \right) \frac{\partial K(\boldsymbol{\theta})}{\partial \theta_j} \right) \boldsymbol{u}_i,$$

which pose severe numerical difficulty in locating the root. An illustration is shown in Figure 4.1(a), where the kernel

$$\phi(\boldsymbol{r}; \boldsymbol{\theta}) = \varphi\left(\sqrt{\frac{r_1^2}{\theta_1^2} + \frac{r_2^2}{\theta_2^2}} \right)$$

is a spline with compact support:

$$\varphi(r) = \begin{cases} 1 - 6r^2 + 6r^3 & r \in [0, 1/2), \\ 2(1-r)^3 & r \in [1/2, 1), \\ 0 & r \in [1, \infty). \end{cases}$$

Therefore, we do not consider compact kernels.

The choice of kernel in this work is the Matern function

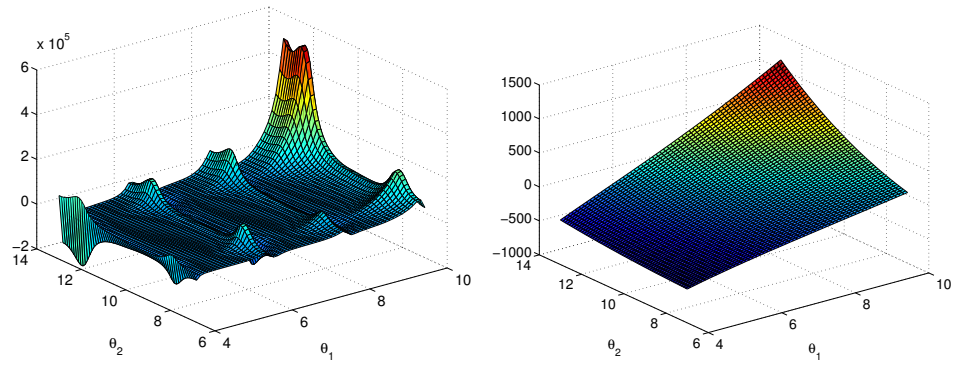
$$\varphi(r) = \frac{1}{2^{\nu-1}\Gamma(\nu)} (\sqrt{2\nu} r)^{\nu} \mathsf{K}_{\nu}(\sqrt{2\nu} r),$$

chosen for its superior theoretical statistical properties [30], where Γ is the Gamma function and K_{ν} is the modified Bessel function of the second kind of order ν. When ν is some integer plus $1/2$, the Matern function can be computed without evaluations of Bessel functions. We choose $\nu = 3/2$, which results in

$$\varphi(r) = (1 + \sqrt{3} r) \exp(-\sqrt{3} r). \tag{4.1}$$

We will demonstrate two uses of the Matern function in later numerical experiments. The first use is in the tensor product form,

$$\phi(\boldsymbol{r}; \boldsymbol{\theta}) = \sigma^2 \varphi\left(\left| \frac{r_1}{\theta_1} \right| \right) \varphi\left(\left| \frac{r_2}{\theta_2} \right| \right), \qquad \boldsymbol{\theta} = [\theta_1, \theta_2, \sigma]^T, \tag{4.2}$$



(a) Compact kernel. (b) Matern kernel.

FIG. 4.1. *Plots of* $\Psi_1^N(\boldsymbol{\theta})$ *for different covariance matrices* $K(\boldsymbol{\theta})$ *generated from different kernels.*

whereas the second use is the native form,

$$\phi(\boldsymbol{r};\boldsymbol{\theta}) = \sigma^2\varphi\left(\sqrt{\frac{r_1^2}{\theta_1^2}+\frac{r_2^2}{\theta_2^2}}\right), \qquad \boldsymbol{\theta} = [\theta_1,\theta_2,\sigma]^T. \tag{4.3}$$

The surface $\Psi_1^N(\boldsymbol{\theta})$ resulting from (4.3) is shown in Figure 4.1(b).

4.1. Taylor coefficients in FMM. With the choice of kernel (4.3), the Taylor coefficients a in (3.6) can be evaluated with the help of those of the other three kernels $\psi_1 = \exp(-\sqrt{3}r)/r$, $\psi_2 = \exp(-\sqrt{3}r)$, $\psi_3 = \sqrt{3}r\exp(-\sqrt{3}r)$. Let $b_{\boldsymbol{k}}$, $c_{\boldsymbol{k}}$, $d_{\boldsymbol{k}}$ be their Taylor coefficients, respectively:

$$b_{\boldsymbol{k}} = \frac{1}{\boldsymbol{k}!}D_{\boldsymbol{y}}^k\psi_1(\boldsymbol{x}_i,\boldsymbol{y}_c), \quad c_{\boldsymbol{k}} = \frac{1}{\boldsymbol{k}!}D_{\boldsymbol{y}}^k\psi_2(\boldsymbol{x}_i,\boldsymbol{y}_c), \quad d_{\boldsymbol{k}} = \frac{1}{\boldsymbol{k}!}D_{\boldsymbol{y}}^k\psi_3(\boldsymbol{x}_i,\boldsymbol{y}_c).$$

We have

$$a_{\boldsymbol{k}} = c_{\boldsymbol{k}} + d_{\boldsymbol{k}}, \tag{4.4}$$

and the following recurrence formulas:

$$b_{\boldsymbol{k}} = \frac{1}{L_1L_2||\boldsymbol{k}||r^2}\left((2||\boldsymbol{k}||-1)\sum_{i=1}^{2}L_{j(i)}(x_i-y_i)b_{\boldsymbol{k}-\boldsymbol{e}_i} - (||\boldsymbol{k}||-1)\sum_{i=1}^{2}L_{j(i)}b_{\boldsymbol{k}-2\boldsymbol{e}_i}\right)$$
$$+\frac{\sqrt{3}}{L_1L_2||\boldsymbol{k}||r^2}\left(\sum_{i=1}^{2}L_{j(i)}((x_i-y_i)c_{\boldsymbol{k}-\boldsymbol{e}_i}-c_{\boldsymbol{k}-2\boldsymbol{e}_i})\right), \tag{4.5}$$

$$c_{\boldsymbol{k}} = \frac{\sqrt{3}}{||\boldsymbol{k}||}\left(\sum_{i=1}^{2}\frac{1}{L_i}((x_i-y_i)b_{\boldsymbol{k}-\boldsymbol{e}_i}-b_{\boldsymbol{k}-2\boldsymbol{e}_i})\right), \tag{4.6}$$

$$d_{\boldsymbol{k}} = \frac{\sqrt{3}}{||\boldsymbol{k}||}\left(\sum_{i=1}^{2}\frac{x_i-y_i}{L_i}(\sqrt{3}c_{\boldsymbol{k}-\boldsymbol{e}_i}-b_{\boldsymbol{k}-\boldsymbol{e}_i})-\sum_{i=1}^{2}\frac{1}{L_i}(\sqrt{3}c_{\boldsymbol{k}-2\boldsymbol{e}_i}-b_{\boldsymbol{k}-2\boldsymbol{e}_i})\right), \tag{4.7}$$

for $||\boldsymbol{k}|| \geq 1$, where $L_1 = \theta_1^2$, $L_2 = \theta_2^2$, $j(1) = 2$, $j(2) = 1$; $b_0 = \psi_1(r)$, $c_0 = \psi_2(r)$, $d_0 = \psi_3(r)$, and $b_{\boldsymbol{k}}, c_{\boldsymbol{k}}, d_{\boldsymbol{k}} = 0$ if any $k_i < 0$. Recurrence relations for $\partial\phi/\partial\theta_\ell$ are similar but tedious in notation; because of the page limit they are not shown here.

5. Numerical results. In this section, we show comprehensive numerical results to demonstrate the practicality of the combination of various techniques considered in this paper for solving the maximum likelihood problem involving a Gaussian process. Emphasis is placed on the scalability of the techniques. For this purpose, the experiments were conducted based on the following cases:

1. Regular grid: The kernel ϕ is defined in (4.2). A BCCB preconditioner is used.
2. Regular grid: The kernel ϕ is defined in (4.3). A BCCB preconditioner is used.
3. Deformed grid: The kernel ϕ is defined in (4.3). A circulant preconditioner is used.

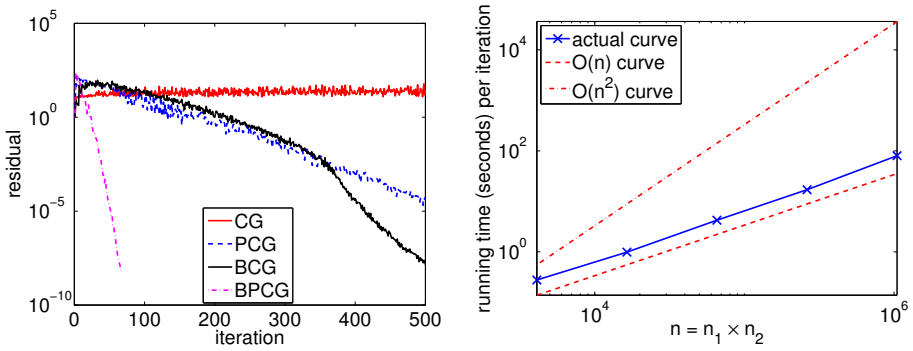
These three cases will be frequently referenced throughout this section. The deformed grid is generated by scaling the y-coordinates of the sites of a regular grid by a quadratic function, which is 1 in the middle of the range of x and 0.5 at the extremes. The deformed grid was illustrated in [5]; because of limited space, we do not replicate the figure here.

Each experiment was conducted on a Linux desktop with multiple (4 to 16) cores and 4 to 32 GB of memory. (Not all the experiments were done on the same machine, but the machine architecture was the same in each experiment and each comparison, to be fair.) The code was implemented by using the Matlab language, except that occasionally some loop- or recursion-intensive functions were written in C/C++ and called by Matlab via the mex function scheme. Note that in principle the implementation was serial but not parallel, even though the Matlab scheduler would make full or partial use of all the cores for some operations. The purpose of the experiments is to demonstrate that we can handle a Gaussian process with as many sites as $2^{20} \approx 10^6$ on a single desktop machine and that the proposed techniques are scalable. In a future parallel implementation on supercomputers, we expect that the size of the data that can be processed can grow by several orders of magnitude.

5.1. Performance of BPCG. The first task is to verify that BPCG is a suitable solver for solving the linear systems. Figure 5.1(a) shows a typical convergence history for the four CG solvers: CG (the standard CG), PCG (preconditioned CG), BCG (block CG), and BPCG. The linear system (covariance matrix $K(\boldsymbol{\theta})$) was defined based on Case 1, where the size of the grid is 64×64 and $\boldsymbol{\theta} = [4, 14, 3]^T$. The right-hand sides were randomly generated. In all experiments throughout this section, by "residual" we mean the norm of the residual vector divided by the norm of the right-hand side. In the case of block solvers, 100 right-hand sides were used for testing, and the residual for each iteration was the maximum of the residuals across all linear systems.

From Figure 5.1(a), we can see that the standard CG solver shows no indication of convergence in 500 iterations and that PCG and BCG converge very slowly. On the other hand, BPCG converges to tolerance 10^{-8} in around 70 iterations. Thus, BPCG is favorable for the type of linear systems at hand. The convergence histories for Case 2 and Case 3 are similar and are not shown here.

5.2. Scalability of BPCG. Encouraged by the rapid convergence, we further tested the scalability of the solver by varying the size of K (or equivalently, the size of the grid). We used the same parameter $\boldsymbol{\theta}$ as in the above experiment, generated 100 random right-hand sides, and set the tolerance of the BPCG solver to be 10^{-8}. The number of iterations are shown in Table 5.1, and the actual running times per



(a) Convergence history of BPCG and others. (b) Scalability of BPCG.

FIG. 5.1. *Performance of BPCG.*

iteration are plotted in Figure 5.1(b) (the curve for Case 1 almost overlaps with that for Case 2; thus only the former is shown).

TABLE 5.1
Number of iterations for different sizes of linear systems.

Grid size	64×64	128×128	256×256	512×512	1024×1024
Case 1, # iter	72	102	110	128	149
Case 2, # iter	87	153	191	214	263

As expected, the running times for solving the linear systems conform to the theoretical cost $O(n \log n)$. (In a log-log plot, an $n \log n$ curve looks indistinguishable from a straight line of slope 1.) This running time is achieved by ensuring that both the matrix-vector multiplication and the multiplication with the preconditioner have a cost of this order. One can see from the table that as the size of the matrix increases, the needed number of iterations increases moderately. According to this trend, for larger systems the number of iterations will be relatively small compared with n.

5.3. Scalability of FMM. The above experiment empirically confirms what is known theoretically: the cost of one BPCG iteration is dominated by the matrix-vector multiplication and the multiplication of the preconditioner. To our knowledge, however, when performing matrix-vector multiplications, FMM runs much slower than FFT, even though the two have the same asymptotic cost. Figure 5.2 shows the running time for performing one matrix-vector multiplication as the matrix size varies. Nevertheless, the plot shows better scaling of the running time than $O(n^2)$. The parameter for generating the covariance matrix was $\boldsymbol{\theta} = [7, 10, 3]^T$, and the parameters for the treecode implementation were $p = 14$, $N_0 = 500$, and $c = 0.4$.

5.4. Overall parameter estimation process. In this section, we present the results of estimating the parameter $\boldsymbol{\theta} = [\theta_1, \theta_2, \sigma]^T$ of the Gaussian process given a sample $\boldsymbol{y}$ generated from $\mathcal{N}(\mathbf{0}, K(\boldsymbol{\theta}))$ with convergence guarantee (2.9)–(2.10) provided by the SAA interpretation of the approach. The covariance matrix K was defined based on the parameter $\boldsymbol{\theta} = [7, 10, 3]^T$. The solver for solving the stochastic nonlinear equation $\boldsymbol{\Psi}^N(\boldsymbol{\theta}) = \mathbf{0}$, where $N = 100$, was the Matlab function `fsolve`, which by default used the trust-region dogleg algorithm. The Jacobian was estimated

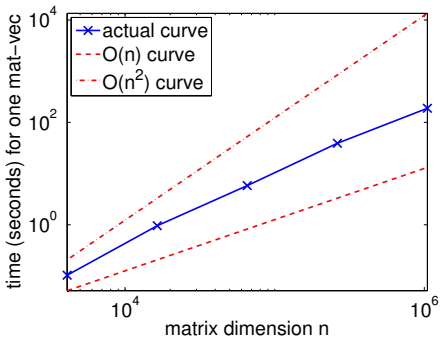
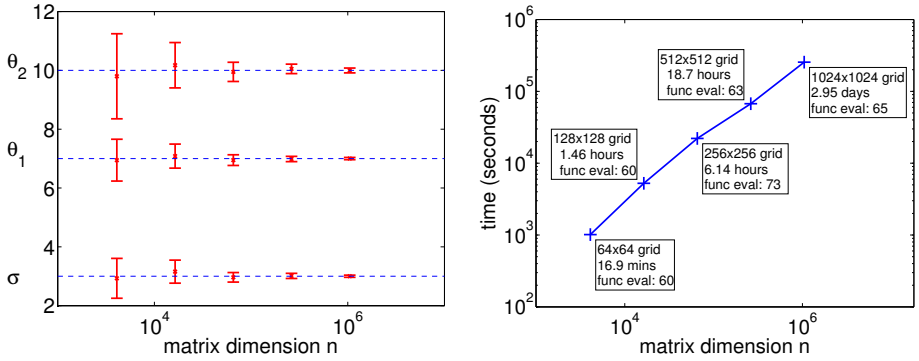


FIG. 5.2. *Scalability of FMM.*

by finite difference. The tolerance of the linear system solver BPCG was set to 10^{-8} as before.

For Case 1, an initial guess $\boldsymbol{\theta}_0 = [4, 14, 1]^T$ was used. For various sizes of grid (64×64, 128×128, 256×256, 512×512, and 1024×1024), the estimated $\boldsymbol{\theta}$ (together with 95% confidence intervals) are shown in Figure 5.3(a). One can see that the estimated parameters are close to those that define the covariance matrix K particularly for large grid sizes. The confidence intervals all cover the true $\boldsymbol{\theta}$; and as the grid size increases, the intervals become narrower and narrower. This result implies that the proposed SAA approach is particularly favorable for large-scale problems, which is precisely the regime we are aiming for.

More detailed information on solving the nonlinear equations is provided in Figure 5.3(b). The scaling of the total running times is roughly to the order $O(n)$ or $O(n \log n)$. This is achieved by the $O(n \log n)$ cost of solving linear systems $K(\boldsymbol{\theta})$, and roughly a constant number of function evaluations of $\boldsymbol{\Psi}^N(\boldsymbol{\theta})$. The figure shows that the number of function evaluations is roughly 70. The trend shown in the figure can be used to estimate the running time for larger grids.

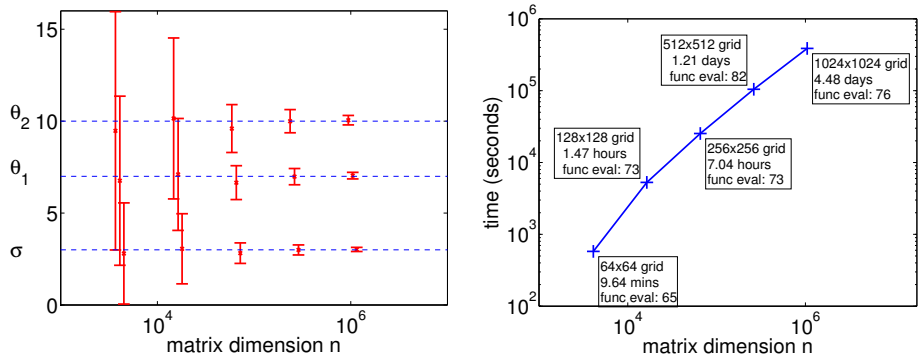


(a) Est. parameters with confidence interval. (b) Running time versus matrix dimension n.

FIG. 5.3. *Results for Case 1.*

For Case 2, an initial guess $\boldsymbol{\theta}_0 = [5, 14, 1]^T$ was used, and the corresponding results are shown in Figures 5.4(a) and 5.4(b). (In Figure 5.4(a) the positions of the intervals are slightly shifted horizontally so that they can be clearly seen; the actual

grid sizes were the same as those in Case 1.) As with Case 1, the results here are similarly expected: sufficiently accurate estimates, narrower and narrower confidence intervals, $O(n \log n)$ scaling of running times, and around 70 function evaluations.



(a) Est. parameters with confidence interval. (b) Running time versus matrix dimension n.

FIG. 5.4. *Results for Case 2.*

For Case 3, the same initial guess $\boldsymbol{\theta}_0 = [5, 14, 1]^T$ was used, but only the 64×64 grid was tested. Despite the encouraging results of the previous two cases, the extremely long running time of matrix-vector multiplications via FMM prevented us from reporting results for larger grids. For the 64×64 grid, parameters of FMM were set the same as those in §5.3, to ensure that the error of matrix-vector products fell around 10^{-10}. Results are in the following:

```
Estimated parameters:
  theta_1: 6.6453,   95% confidence interval: [3.0171, 10.2735]
  theta_2: 9.3134,   95% confidence interval: [4.2572, 14.3696]
  sigma:   2.7271,   95% confidence interval: [0.5839,  4.8704]
Function evaluations: 65
Running time: 331306 seconds (3.83 days)
```

Except for the running time, the results are similar to those in the previous two cases. Extrapolating from these results, we expect that the estimation of the parameters will be more and more accurate as the size of the grid increases. We also expect that the $O(n \log n)$ running time holds for Case 3, since the number of function evaluations do not vary much. We note again that it is the large hidden constant, rather than the asymptotic cost, that prevents us from running larger-scale tests. In future work involving a parallel implementation on supercomputers, experiments with much larger n can be conducted and results be reported.

6. Concluding remarks. We have introduced a matrix-free approach for computing the solution of the maximum likelihood problem for Gaussian processes. The approach is based on a sample average approximation, which also provides the means for generating a confidence interval based on convergence in distribution results for sample average approximation approaches. In turn, this requires only the action of the inverse of the covariance matrix on a vector, which can be carried by preconditioned Krylov methods using only matrix-vector multiplications. For data on a regular grid, the matrix-vector multplications can be carried out by circulant em-

bedding in $O(n \log n)$ time; for data on an arbitrary grid, the latter is accomplished by a fast multipole method, even for covariance kernels that do not have compact support. This removes the reliance of the analysis step of Gaussian processes on the use of a Cholesky factorization, which is the prevailing current practice [26]. If the Krylov methods are preconditioned successfully, the approach can scale well, as we have demonstrated on three cases involving the Matern kernel. Future work will involve extensive testing of the approach on increasingly large problems—in particular for the fast multipole approach that, as we have demonstrated, scales well but is still expensive for medium-scale tests because of high initial cost. Our future work will address the rapid increase in data set size in the applied sciences.

Acknowledgments. We are grateful to Michael Stein for comments and suggestions. This work was supported by the U.S. Department of Energy under contract DE-AC02-06CH11357.

REFERENCES

[1] J. Barnes and P. Hut, *A hierarchical $O(N \log N)$ force-calculation algorighm*, Nature, 324 (1986), pp. 446–449.
[2] R. H. Chan, X.-Q. Jin, and M.-C. Yeung, *The circulant operator in the Banach algebra of matrices*, Linear Algebra Appl., 149 (1991), pp. 41–53.
[3] R. H.-F. Chan and X.-Q. Jin, *An Introduction to Iterative Toeplitz Solvers*, SIAM, 2007.
[4] T. F. Chan, *An optimal circulant preconditioner for Toeplitz systems*, SIAM J. Sci. and Stat. Comput., 9 (1988), pp. 766–771.
[5] J. Chen, M. Anitescu, and Y. Saad, *Computing $f(A)b$ via least squares polynomial approximations*, SIAM Journal on Scientific Computing, 33 (2011), pp. 195–222.
[6] J. Chiles and P. Delfiner, *Geostatistics: modeling spatial uncertainty*, Wiley, New York, 1999.
[7] N. Cressie, *Statistics for spatial data*, Wiley-Interscience, New York, 1993.
[8] P. J. Diggle, J. A. Tawn, and R. A. Moyeed, *Model-based geostatistics*, Applied Statistics, 47 (1998), pp. 299–350.
[9] Z. Duan and R. Krasny, *An adaptive treecode for coputing nonbounded potential energy in classical molecular systems*, J. Comput. Chem., 23 (2001), pp. 1549–1571.
[10] A. C. Faul, G. Goodsell, and M. J. D. Powell, *A Krylov subspace algorithm for multiquadric interpolation in many dimensions*, IMA Journal of Numerical Analysis, 25 (2005), pp. 1–24.
[11] R. Furrer, M. Genton, and D. Nychka, *Covariance tapering for interpolation of large spatial datasets*, Journal of Computational and Graphical Statistics, 15 (2006), pp. 502–523.
[12] M. Gibbs and D. MacKay, *Efficient implementation of Gaussian processes*, Cavendish Lab., Cambridge, UK, Tech. Rep, (1997).
[13] A. Gray and A. Moore, *'N-Body'problems in statistical learning*, Advances in Neural Information Processing Systems, (2001), pp. 521–527.
[14] N. A. Gumerov and R. Duraiswami, *Fast radial basis function interpolation via preconditioned Krylov iteration*, SIAM J. Sci. Comput., 29 (2007), pp. 1876–1899.
[15] K. Heitmann, D. Higdon, C. Nakhleh, and S. Habib, *Cosmic calibration*, The Astrophysical Journal, 646 (2006), pp. L1–L4.
[16] D. Higdon, M. Kennedy, J. Cavendish, J. Cafeo, and R. Ryne, *Combining field data and computer simulations for calibration and prediction*, SIAM Journal on Scientific Computing, 26 (2004), pp. 448–466.
[17] M. Hutchinson, *A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines.*, COMMUN. STAT. SIMUL. COMPUT., 19 (1990), pp. 433–450.
[18] M. Kennedy and A. O'Hagan, *Bayesian calibration of complex computer models*, Journal of the Royal Statistical Society Series B, 63 (2001), pp. 425–464.
[19] R. Krasny and L. Wang, *Fast evaluation of multiquatric RBF sums by a Cartesian treecode*, SIAM Sci. Comp., (2011), p. accepted. to appear.
[20] P. Li, H. Johnston, and R. Krasny, *A Cartesian treecode for screened Coulomb interactions*, J. Comp. Phys., 228 (2009), pp. 3858–3868.
[21] K. Lindsay and R. Krasny, *A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow*, J. Comput. Phys., 172 (2001), pp. 879–907.

[22] C. A. Micchelli, *Interpolation of scattered data: Distance matrices and conditionally positive definite functions*, Constr. Approx., 2 (1986), pp. 11–22.
[23] A. A. Nikishin and A. Y. Yeremin, *Variable block CG algorithms for solving large sparse symmetric positive definite linear systems on parallel computers, I: General iterative scheme*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 1135–1153.
[24] J. Nocedal and S. Wright, *Numerical optimization*, Springer Verlag, 2006.
[25] D. P. O'Leary, *The block conjugate gradient algorithm and related methods*, Linear Algebra Appl., 29 (1980), pp. 293–322.
[26] C. Rasmussen and C. Williams, *Gaussian processes for machine learning*, MIT Press, Cambridge, Massachusets., 2006.
[27] K. P. Schmitt, M. Anitescu, and D. Negrut, *Efficient sampling for spatial uncertainty quantification in multibody system dynamics applications*, International Journal for Numerical Methods in Engineering, 80 (2009), pp. 537–564.
[28] A. Shapiro, D. Dentcheva, and A. Ruszczyński, *Lectures on Stochastic Programming: Modeling and Theory*, MPS/SIAM Series on Optimization 9, Philadelphia, PA, 2009.
[29] J. Skilling, *Bayesian numerical analysis*, Physics & Probability: Essays in honor of Edwin T. Jaynes, (1993), pp. 207–221.
[30] M. Stein, *Interpolation of spatial data: Some theory for Kriging*, Springer-Verlag, Berlin, 1999.
[31] J. Tsitsiklis, D. Bertsekas, and M. Athans, *Distributed asynchronous deterministic and stochastic gradient optimization algorithms*, Automatic Control, IEEE Transactions on, 31 (1986), pp. 803–812.
[32] E. E. Tyrtyshnikov, *Optimal and superoptimal circulant preconditioners*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 459–173.

Appendix. Example of difference between SAA nonlinear equations and optimization approaches. We illustrate that the SAA terms do not have the same relationship as the exact means, in that one is not the gradient of the other. This phenomenon is brought about by the fact that

$$\frac{d}{d\theta}\operatorname{tr}(\log(K)) = \operatorname{tr}\left(K^{-1}\frac{dK}{d\theta}\right)$$

but

$$\frac{d}{d\theta}\log(K) \neq K^{-1}\frac{dK}{d\theta} \text{ or } \frac{dK}{d\theta}K^{-1}.$$

We now illustrate this situation with an example. Let

$$K = \exp\left(\begin{bmatrix} a & b \\ b & a \end{bmatrix}\right) = e^a \begin{bmatrix} \cosh(b) & \sinh(b) \\ \sinh(b) & \cosh(b) \end{bmatrix},$$

where a and b are functions of a single variable θ. To simplify notations, we let $c = \cosh(b)$ and $s = \sinh(b)$. Then

$$\frac{d}{d\theta}\log(K) = \begin{bmatrix} a' & b' \\ b' & a' \end{bmatrix}, \qquad K^{-1} = e^{-a}\begin{bmatrix} c & -s \\ -s & c \end{bmatrix}, \qquad \frac{dK}{d\theta} = a'e^a\begin{bmatrix} c & s \\ s & c \end{bmatrix} + e^a\begin{bmatrix} s & c \\ c & s \end{bmatrix}.$$

Clearly,

$$K^{-1}\frac{dK}{d\theta} = \frac{dK}{d\theta}K^{-1} = \begin{bmatrix} a' & 1 \\ 1 & a' \end{bmatrix}$$

It then immediatly follows that, in general, for a vector $\boldsymbol{u}$ in $\mathbb{R}^2$ we have that

$$\boldsymbol{u}^T K^{-1}\frac{dK}{d\theta}\boldsymbol{u} \neq \frac{d}{d\theta}\boldsymbol{u}^T\log(K)\boldsymbol{u} = \boldsymbol{u}^T\begin{bmatrix} a' & b' \\ b' & a' \end{bmatrix}\boldsymbol{u},$$

Therefore we cannot use the left-hand side to compute the derivative of the right-hand side.