Fast Direct Methods for Gaussian Processes

Sivaram Ambikasaran, Daniel Foreman-Mackey, Leslie Greengard, *Member, IEEE*, David W. Hogg, and Michael O'Neil, *Member, IEEE*

Abstract—A number of problems in probability and statistics can be addressed using the multivariate normal (Gaussian) distribution. In the one-dimensional case, computing the probability for a given mean and variance simply requires the evaluation of the corresponding Gaussian density. In the n-dimensional setting, however, it requires the inversion of an $n \times n$ covariance matrix, C, as well as the evaluation of its determinant, $\det(C)$. In many cases, such as regression using Gaussian processes, the covariance matrix is of the form $C = \sigma^2 I + K$, where K is computed using a specified covariance kernel which depends on the data and additional parameters (hyperparameters). The matrix C is typically dense, causing standard direct methods for inversion and determinant evaluation to require $\mathcal{O}(n^3)$ work. This cost is prohibitive for large-scale modeling. Here, we show that for the most commonly used covariance functions, the matrix C can be hierarchically factored into a product of block low-rank updates of the identity matrix, yielding an $\mathcal{O}(n \log^2 n)$ algorithm for inversion. More importantly, we show that this factorization enables the evaluation of the determinant $\det(C)$, permitting the direct calculation of probabilities in high dimensions under fairly broad assumptions on the kernel defining K. Our fast algorithm brings many problems in marginalization and the adaptation of hyperparameters within practical reach using a single CPU core. The combination of nearly optimal scaling in terms of problem size with high-performance computing resources will permit the modeling of previously intractable problems. We illustrate the performance of the scheme on standard covariance kernels.

Index Terms—Gaussian process, covariance function, covariance matrix, determinant, hierarchical off-diagonal low-rank, direct solver, fast multipole method, Bayesian analysis, likelihood, evidence

1 Introduction

A common task in probability and statistics is the computation of the numerical value of the posterior probability of some parameters $\boldsymbol{\theta}$ conditional on some data $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ using a multivariate Gaussian distribution. This requires the evaluation of

$$p(\boldsymbol{\theta}|\boldsymbol{x}, \boldsymbol{y}) \propto \frac{1}{|\det(C(\boldsymbol{x};\boldsymbol{\theta}))|^{1/2}} e^{-\frac{1}{2}\boldsymbol{y}^t C^{-1}(\boldsymbol{x};\boldsymbol{\theta})\boldsymbol{y}} p(\boldsymbol{\theta}), \qquad (1)$$

where $C(\boldsymbol{x};\boldsymbol{\theta})$ is an $n \times n$ symmetric, positive-definite *covariance* matrix. The explicit dependence of C on particular parameters $\boldsymbol{\theta}$ is shown here, and may be dropped in the proceeding discussion. In the one-dimensional case, C is simply the scalar variance. Thus, computing the probability requires only the evaluation of the corresponding Gaussian. In the n-dimensional setting, however, C is typically dense, so that its inversion requires $O(n^3)$ work as does the evaluation of its determinant $\det(C)$. This cost is prohibitive for large n.

- *S. Ambikasaran is with the Courant Institute, New York University, New York, NY 10025. E-mail: sivaram@cims.nyu.edu.*
- *D. Foreman-Mackey and D. W. Hogg are with the Department of Physics, New York University, New York, NY 10025. E-mail: {danfm, david.hogg}@nyu.edu.*
- *L. Greengard is with the Courant Institute, New York University, New York, NY 10025, and the Simons Center for Data Analysis, Simons Foundation, New York, NY. E-mail: greengard@cims.nyu.edu.*
- *M. O'Neil is with the Courant Institute and the Polytechnic School of Engineering, New York University, New York, NY 10025. E-mail: oneil@cims.nyu.edu.*

Manuscript received 24 Mar. 2014; revised 4 Apr. 2015; accepted 10 June 2015. Date of publication 21 June 2015; date of current version 13 Jan. 2016. Recommended for acceptance by J. Zhu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPAMI.2015.2448083

In many cases, the covariance matrix C is assumed to be of the form $C(\boldsymbol{x}) = \sigma^2 I + K(\boldsymbol{x})$, where $K_{ij}(\boldsymbol{x}) = k(x_i, x_j)$. This happens when the model for the data assumes some sort of uncorrelated additive measurement noise having variance σ^2 in addition to some structured covariance described by the kernel k. The function $k(x_i, x_j)$ is called the covariance function or covariance kernel, which, in turn, can depend on additional parameters, $\boldsymbol{\theta}$. Covariance matrices of this form universally appear in regression and classification problems when using Gaussian process priors [49]. Because many covariance kernels are similar to those that arise in computational physics, a substantial body of work over the past decades has produced a host of relevant fast algorithms, first for the rapid application of matrices such as K [21], [24], [29], [31], [64], and more recently on their inversion [4], [7], [10], [15], [21], [30], [39], [43]. We do not seek to further review the literature here, except to note that it is still a very active area of research.

Using the approach outlined in [4], we will show that under suitable conditions, the matrix C can be hierarchically factored into a product of block low-rank updates of the identity matrix, yielding an $\mathcal{O}(n \log^2 n)$ algorithm for inversion. More importantly (and perhaps somewhat surprising), we show that our factorization enables the evaluation of the determinant, $\det(C)$, in $\mathcal{O}(n \log n)$ operations. Together, these permit the efficient direct calculation of probabilities in high dimensions. Previously existing methods for inversion and determinant evaluation were based on either rough approximation methods or iterative methods [8], [13], [17], [54], [55]. These schemes are particularly ill-suited for computing determinants. Although bounds exist for sufficiently random and diagonally dominant matrices, they are often inadequate in the general case [11]. We briefly review existing accelerated methods for Gaussian processes

in Section 2.4 and present a cursory heuristic comparison with our covariance matrix factorization.

Gaussian processes are the tool of choice for many statistical inference or decision theory problems in machine learning and the physical sciences. They are ideal when requirements include flexibility for the modeling of continuous functions. However, applications are limited by the computational cost of matrix inversion and determinant calculation. Furthermore, the determinant of the covariance matrix is required for Gaussian process likelihood evaluations (i.e., computation of any actual value of the probability of the data under the covariance hyperparameters, or evidence). Existing linear algebraic schemes for direct matrix inversion and determinant calculation are prohibitively expensive when the likelihood evaluation is placed *inside* an outer optimization or Markov chain Monte Carlo (MCMC) sampling loop.

In this paper we will focus on describing and applying our new methods for handling large-scale covariance matrices (dense and full- or high-rank) to avoid the computational bottlenecks encounters in regression, classification, and other problems when using Gaussian process models. We motivate the algorithms by explaining where their need arises only in Gaussian process regression, but similar calculations are frequently encountered in other regimes under Gaussian process priors. Other applications, such as marginalization and adaptation of hyperparameters are relatively straightforward, and the computational bottlenecks of each are highly related.

The paper is organized as follows. Section 2 reviews some basic facts about Gaussian processes and the resulting formulas encountered in the case of a one-dimensional regression problem. Prediction, marginalization, adaptation of hyperparameters, and existing approximate accelerated methods are also discussed. Section 3 discusses the newly developed matrix factorization for hierarchical off-diagonal low-rank (HODLR) matrices, for which factorization requires only $\mathcal{O}(n \log^2 n)$ work. Subsequent applications of the operator and its inverse scale as $\mathcal{O}(n \log n)$. Many popular covariance functions used for Gaussian processes yield covariance matrices satisfying the HODLR requirements. While other hierarchical methods could be used for this step, we focus on the HODLR decomposition because of its simplicity and applicability to a wide range of covariance functions. We would like to emphasize that the algorithm will work for any covariance kernel, but the scaling of the algorithm might not be optimal; for instance, if the covariance kernel has a singularity or is highly oscillatory without damping. Further, in Section 4, we show that the determinant of an HODLR decomposition can be computed in $\mathcal{O}(n \log n)$ operations. Section 5 contains numerical results for our method applied to some standard covariance functions for data embedded in varying dimensions. Finally, in Section 6, we summarize our results and discuss any shortcomings and other applications of the method, as well as future avenues of research.

The conclusion contains a cursory description of the corresponding software packages in C++ and Python which implement the numerical schemes of this work. These open-source software packages have been made available since the time of submission.

2 Gaussian Processes and Regression

In the past two decades, Gaussian processes have gained popularity in the fields of machine learning and data analysis for their flexibility and robustness. Often cited as a competitive alternative to neural networks because of their rich mathematical and statistical underpinnings, practical use in large-scale problems remains out of reach due to computational complexity. Existing direct computational methods for manipulations involving large-scale covariance matrices require $\mathcal{O}(n^3)$ calculations. This causes regression/prediction, parameter marginalization, and optimization of hyperparameters to be intractable problems. This scaling can be reduced in special cases via several approximation methods, discussed in Section 2.4, however for dense, highly coupled covariance matrices no suitable direct methods have been proposed. Here, by *direct method* we mean one that constructs the inverse and determinant of a covariance matrix to within some pre-specified numerical tolerance *directly* instead of iteratively. The matrix inverse can then be stored for use later, much as standard LU or QR factorizations. The numerical tolerance can be measured in the spectral or Frobenius norms, and our algorithm is able to easily achieve approximations on the order of 10^{-12}. Often, near machine precision ($\sim 10^{-15}$) is attainable.

The following sections contain an overview of regression via Gaussian processes and the large computational tasks that are required at each step. The one-dimensional regression case is discussed for simplicity, but similar formulae for higher dimensions and classification problems are straightforward to derive. In higher dimensions, the corresponding computational methods scale with the same asymptotic complexity, albeit with larger constants. For a thorough treatment of regression using Gaussian processes, see [42], [49].

The canonical linear regression problem we will analyze assumes a model of the form

$$y = f(x) + \epsilon, \tag{2}$$

where $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$ is some form of uncorrelated measurement noise. Given a dataset $\{x_i, y_i\}$, the goal is to infer f, or equivalently some set of parameters that f depends on. We will enforce the prior distribution of the unknown function f to be a Gaussian process,

$$f \sim \mathcal{GP}(m, k), \tag{3}$$

where $k = k(x, x')$ is some admissable covariance function (k corresponds to positive definite covariance matrices), possibly depending on some unknown hyperparameters, and $m = m(x)$ is the expected mean of f. The task of fitting hyperparameters is discussed in Sections 2.2 and 2.3 of the paper. Table 1 lists some of the frequently used covariance functions for Gaussian processes.

2.1 Prediction

One of the main uses for the previous model (especially in machine learning) is to predict, with some estimated confidence, $f(\tilde{x})$ for some new input data point $\tilde{x}$. This is equivalent to calculating the conditional distribution $\tilde{y}|\boldsymbol{x}, \boldsymbol{y}, \tilde{x}$. We will not assume any parametric form of f, and enforce

TABLE 1
Common Covariance Kernels Used in Gaussian Processes

Name	Covariance function
Ornstein-Uhlenbeck	$\exp\left(-\lvert x-y\rvert\right)$
Gaussian	$\exp\left(-\lvert x-y\rvert^2\right)$
Matérn family	$\sigma^2\frac{\left(\sqrt{2\nu}\lvert x-y\rvert\right)^{\nu}}{\Gamma(\nu)2^{\nu-1}}K_{\nu}\left(\sqrt{2\nu}\lvert x-y\rvert\right)$
Rational Quadratic	$\frac{1}{(1+\lvert x-y\rvert^2)^{\alpha}}$

structure only through the observed data and the choice of the mean function m and covariance function k. Additionally, for the time being, assume that k is fixed (i.e., hyperparameters are either fixed or absent). Given the data $\boldsymbol{x} = (x_1\ x_2\ \ldots\ x_n)^t$ and $\boldsymbol{y} = (y_1\ y_2\ \ldots\ y_n)^t$, it is easy to show that the conditional distribution (likelihood) of $\boldsymbol{y}$ is given by

$$\boldsymbol{y}|\boldsymbol{x} \sim \mathcal{N}\big(\boldsymbol{m}(\boldsymbol{x}),\, \sigma_\epsilon^2 I + K(\boldsymbol{x})\big), \tag{4}$$

where the mean vector and covariance matrix are:

$$\begin{aligned} \boldsymbol{m}(\boldsymbol{x}) &= (m(x_1)\ m(x_2)\ \ldots\ m(x_n))^t, \\ K_{ij}(\boldsymbol{x}) &= k(x_i, x_j). \end{aligned} \tag{5}$$

The conditional distribution of a predicted function value, $\tilde{y} = f(\tilde{x})$, can then be calculated as

$$\tilde{y}|\boldsymbol{x}, \boldsymbol{y}, \tilde{x} \sim \mathcal{N}(\tilde{f}, \tilde{\sigma}^2), \tag{6}$$

with

$$\begin{aligned} \tilde{f} &= \boldsymbol{k}(\tilde{x}, \boldsymbol{x})\big(\sigma_\epsilon^2 I + K(\boldsymbol{x})\big)^{-1}\boldsymbol{y}, \\ \tilde{\sigma}^2 &= k(\tilde{x}, \tilde{x}) - \boldsymbol{k}(\tilde{x}, \boldsymbol{x})\big(\sigma_\epsilon^2 I + K(\boldsymbol{x})\big)^{-1}\boldsymbol{k}(\boldsymbol{x}, \tilde{x}). \end{aligned} \tag{7}$$

In the previous formulas, for the sake of simplicity, we have assumed that the mean function $\boldsymbol{m} = 0$. The vector $\boldsymbol{k}(\tilde{x}, \boldsymbol{x})$ is the column vector of covariances between $\tilde{x}$ and all the known data points $\boldsymbol{x}$, and $\boldsymbol{k}(\boldsymbol{x}, \tilde{x}) = \boldsymbol{k}(\tilde{x}, \boldsymbol{x})^t$ [49]. We have therefore reduced the problem of prediction and confidence estimation (in the expected value sense) down to matrix-vector multiplications. For large n, the cost of inverting the matrix $\sigma^2 I + K$ is expensive, with direct methods for dense systems scaling as $\mathcal{O}(n^3)$. A direct algorithm for the rapid inversion of this matrix is one of the main contributions of this paper.

2.2 Hyperparameters and Marginalization

As mentioned earlier, often the covariance function k used to model the data $\boldsymbol{x}$, $\boldsymbol{y}$ depends on some set of parameters, $\boldsymbol{\theta}$. For example, in the case of a Gaussian covariance function

$$k(x, x'; \boldsymbol{\theta}) = \beta + \frac{1}{\sqrt{2\pi}\sigma}e^{\frac{-(x-x')^2}{2\sigma^2}}, \tag{8}$$

the column vector of hyperparameters is given by $\boldsymbol{\theta} = (\beta, \sigma)^t$.

Often hyperparameters correspond to some physically meaningful quantity of the data, for example, a decay rate or some spatial scale. In this case, these parameters are fixed once and for all according to the specific physics or dynamics of the model. On the other hand, hyperparameters may be included for robustness or uncertainty quantification and must be marginalized (integrated) away before the final posterior distribution is calculated. In this case, for *relevant* hyperparameters $\boldsymbol{\theta}$ and *nuisance* parameters $\boldsymbol{\eta}$, in order to compute the evidence one must compute marginalization integrals of the form

$$p(\boldsymbol{y}, \boldsymbol{x}|\boldsymbol{\theta}) \propto \int \frac{1}{|\det(C)|^{1/2}}e^{-\frac{1}{2}(\boldsymbol{y}^t C^{-1}\boldsymbol{y})}\, p(\boldsymbol{\eta})\, d\boldsymbol{\eta}, \tag{9}$$

where $C = C(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\eta})$ is a covariance matrix corresponding to the Gaussian process prior and $p(\boldsymbol{\eta})$ is some prior on $\boldsymbol{\eta}$. If there are r nuisance parameters, this is an r-dimensional integral whose numerical integration requires $\mathcal{O}(q^r)$ quadrature nodes, where q is roughly the number of quadrature nodes needed for one-dimensional marginalization. Unless C^{-1} and $\det(C)$ can be calculated rapidly for varying samples of the nuisance parameters, the direct calculation of this integral is not possible. Rapid algorithms for constructing C^{-1} and $\det(C)$ would allow for the *direct* marginalization of nuisance parameters, thereby *directly* constructing the probability of the data, or the marginal evidence. This is in contrast with several existing approximate Monte Carlo methods for computing the above integral (e.g., importance sampling, MCMC, etc.), which are not direct, and which converge with only half-order accuracy (i.e., the numerical accuracy of the integral only decreases as $m^{-1/2}$, where m is the number of Monte Carlo samples). These sampling methods may decrease the number of inversions of C for varying parameters, but do not completely avoid this cost.

2.3 Adaptation of Hyperparameters

Alternatively, there exist situations in which the hyperparameters $\boldsymbol{\theta}$ do not arise out of physical considerations, but rather one would like to infer them as *best fit parameters*. This entails minimizing some regression norm with respect to the parameters,

$$\min_{\boldsymbol{\theta}} |\boldsymbol{y} - f(\boldsymbol{x})|$$

or rather maximizing a parameter likelihood function (point estimation using a Bayesian framework):

$$\max_{\boldsymbol{\theta}} p(\theta|\boldsymbol{x}, \boldsymbol{y}, f).$$

In either case, some manner of non-linear optimization must be performed because of the non-linear dependence of *every entry* of the covariance matrix C on the hyperparameters $\boldsymbol{\theta}$.

Regardless of the type of optimization scheme selected, several evaluations of the evidence, likelihood, and/or Gaussian regression must be performed—each of which requires evaluation of the inverse of the covariance matrix, C^{-1}. In order to achieve the maximum rate of convergence of these opimization algorithms, the full likelihood (or evidence) is required, i.e., the numerical value of the determinant of C is need. Unless the determinant and inverse can be re-calculated and applied to the data $\boldsymbol{x}$ rapidly, optimizing over all possible $\boldsymbol{\theta}$'s is not a computationally tractable problem. We skip the discussion of various optimization procedures relevant to the adaptation of hyperparameters in Gaussian processes [49], but only point out that virtually all of them require the re-computation of the inverse covariance matrix C^{-1}.

2.4 Accelerated Methods

A variety of linear-algebraic methods have been proposed to accelerate either the inversion of $C = I + K$, the computation of its determinant, or both. If K is of low-rank, say p, then it is straightforward to compute C^{-1} and $\det(C)$ using the the Sherman-Morrison-Woodbury formula [36], [53], [61] and the Sylvester determinant theorem [1] in $O(p^2n)$ operations. However, unless an analytical form of the low-rank property of the covariance kernel is known, some type of dense numerical linear algebra must be performed. Usually, constructing general low-rank approximations to $n \times n$ matrices requires at least $\mathcal{O}(pn^2)$ operations, where p is the *numerical rank* of the matrix. By numerical rank we loosely mean that there are p (normalized) singular values larger than some specified precision, ϵ. This definition of numerical rank is consistent with spectral norm, and closely related to the Frobenius norm.

Almost all of the dense matrix low-rank approximations construct some suitable factorization (approximation) of K:

$$K \approx Q_{n\times p}\, K^s_{p\times p}\, Q^T_{p\times n}\,, \tag{10}$$

where one can think of Q^T as *compressing* the action of K onto a subset of points $\{x_{i1}, \ldots, x_{ip}\}$, K^s as the covariance kernel acting on that subset, and Q as *interpolating* the result to the full set of n points $\{x_1, x_2, \ldots, x_n\}$ [45], [54], [55], [56].

Iterative methods can also be applied. These are particularly effective when there is a fast method to compute the necessary matrix-vector products. For Gaussian covariance matrices, this can be accomplished using the fast Gauss transform [29] and its higher-dimensional variants using kd-trees (see, for example, [52], [63]). Alternatively, when $k(x_i, x_j)$ is a convolution kernel and the data are equispaced, the Fast Fourier Transform (FFT) can be used to accelerate the matrix vector product [18]. For non-equispaced data, non-uniform fast Fourier transforms (NUFFTs) are applicable [19], [20], [28]. As mentioned in the introduction, analysis-based fast algorithms can also be used for specific kernels [17] or treated using the more general "black-box" or "kernel-independent" fast multipole methods [21], [24], [64].

In some instances, the previously described linear algebraic or iterative methods can be avoided all-together if an analytical decomposition of the kernel is known. For example, much of the mathematical machinery needed to develop the fast Gauss transform [29] relies on careful analysis of generating functions (or related expansions) of the Gaussian kernel. For example,

$$e^{-(t-s)^2/\delta} = \sum_{j=0}^{\infty} \frac{1}{j!} \left(\frac{s - s_0}{\sqrt{\delta}}\right) h_j\left(\frac{t - s_0}{\sqrt{\delta}}\right), \tag{11}$$

expresses the Gaussian as a sum of separated functions in s, t, centered about s_0, scaled by δ, and where h_j is the jth degree Hermite function. Similar formulas, often referred to as addition formulas or multipole expansions in the physics literature, can be derived for other covariance kernels. As another example, one could build a low-rank representation of covariance matrices generated by the Matérn kernel using formulas of the form:

$$K_\nu(w - z) = \sum_{j=-\infty}^{\infty} (-1)^j K_{\nu+j}(w) I_j(z), \tag{12}$$

where w,z are complex variables for which $|w| < |z|$ and I_j is the modified Bessel function of the second kind or order j. See [46], [60] for a full treatment of formulas of this type. Relationships such as the previous ones lead (almost) directly to fast algorithms for the forward application of the associated covariance matrices. Directly building the inverse matrix (and evaluating the determinant) is more complicated.

Before we move on, there are several other accelerated methods which are popular in the Gaussian process community (namely greedy approximations, sub-sampling, and the Nyström method) [49]. We would like to briefly describe the accelerations that can be obtained by interpreting Gaussian processes via a state-space model [37], [51], [57].

Stochastic linear differential equations (causal, and driven by Gaussian noise) and state-space models are intimately connected with Gaussian processes and (stationary) covariance functions via the Wiener-Khintchine theorem [16]. In particular, this observation allows one to construct spectral density approximations to stationary covariance kernels which in turn give rise to a corresponding state-space process. This process can then be analyzed using Kalman filters and other smoothers, which often have linear computational complexity time for single point inference [37], [40]. The accuracy of this inference lies in the quality of the spectral density approximation, which is usually expressed as a rational function. This finite-rank spectral density approximation via rational functions can be interpreted much in the same way as approximating the Gaussian process covariance matrix as in equation (10)—once this finite-rank approximation is constructed, the resulting matrix inversion scales as $\mathcal{O}(p^2n)$ by the Sherman-Morrison-Woodbury formula (see Section 3). Applying state-space models to *parameter* inference problems, instead of smoothing or functional inference problems, is more subtle but several methods from signal processing are useful. For a clear exposition on this topic, see [12], [51]. Often the asymptotic computational cost of the state-space model analysis will be similar to the algorithm of this paper because both methods are using rank considerations to approximate the covariance structure—the algorithm of this paper uses a spatial-hierarchical method, whereas fast state-space methods use a spectral approximation of temporal data.

One last theme for increasing the scalability of Gaussian processes to big data sets is to introduce some notion of sparsity [38], [45]. Many of the previous accelerated methods can be interpreted as introducing sparsity at the covariance kernel level—i.e., by approximating the matrix K as a finite rank operator. The resulting approximation is dense, but *data-sparse*. Alternatively, one may introduce sparsity at the level of the actual matrix K by thresholding small elements away from the diagonal. The resulting K may retain high (or full) numerical rank, but the actual matrix is sparse, thereby enabling sparse matrix algebra to be performed which has reached a high level of acceleration in modern computing environments. Sparsity may also be introduced by the inclusion of data-generating latent variables (related to the state-space interpretation of Gaussian processes), similar to hidden Markov models [12], [38], [51].

It should be noted that all of the previous methods for accelerating Gaussian process calculations involve some sort of approximation. Depending on the method, either the resulting covariance matrix is approximated (using a low-rank factorization) or the actual covariance kernel is approximated (using a low-rank representation, or by approximating the actual Gaussian process by a finite-rank chain, as in the case of the state-space models). In each case, the analysis of the approximating Gaussian process is different because the approximation take place at different levels in the mathematics. Our accelerated direct method, which is described in the next section, makes an approximation at the level of the covariance matrix. This is akin to viewing the covariance matrix as a continuous linear operator, and not an arbitrary data matrix. Often this approximation is negligible as it is near to machine precision in finite digit arithmetic.

Lastly, the evaluation of determinants is a somewhat different matter. Most of the previously described accelerated approximations in this section are unable to evaluate the determinant in less than $\mathcal{O}(n^3)$ time since this is equivalent to constructing some matrix factorization or all of the eigenvalues. Taylor series approximations [47] and Monte Carlo methods have been suggested [9], as well as conjugate gradient-type methods combined with trace estimators [17]. For additional approximation methods, see the text [49]. In general, however, it is difficult to obtain accurate values for the determinant in a robust and reliable manner. Thus, the development of a fast, accurate, and direct method is critical in making large-scale Gaussian process modeling useful for for *exact inference* problems.

3 Hierarchical Matrices

A large class of dense matrices, for example, matrices arising out of boundary integral equations [65], radial basis function interpolation [4], kernel density estimation in machine learning, and covariance matrices in statistics and Bayesian inversion [5], [6], can be efficiently represented as data-sparse hierarchical matrices. After a suitable ordering of columns and rows, these matrices can be recursively subdivided and certain sub-matrices at each level can be well-represented by low-rank matrices.

We refer the readers to [2], [10], [14], [15], [27], [33], [34], [35] for more details on this approach. Depending on the subdivision structure and low-rank approximation technique, different hierarchical decompositions exist. For instance, the fast multipole method [31] accelerates the calculation of long-range gravitational forces for n-body problems by hierarchically compressing the associated matrix operator using low-rank considerations. The algorithm of this paper makes use of sorting data points according to a kd-tree, which has the same formalism in arbitrary dimension. The data is sorted recursively, one dimension at a time, yielding a data structure which can be searched in at most $\mathcal{O}(n)$ time, and often much faster. Once the sorting is completed, the data points can be globally re-ordered according to, for example, a Z-order or Z-curve. It is this ordering which generates a correspondence between individual data points and matrix columns and rows. Based on the particular covariance kernel and the data structure used (an adaptive versus a uniform sorting), the resulting algorithm will perform slightly differently, but with the same asymptotic scaling.

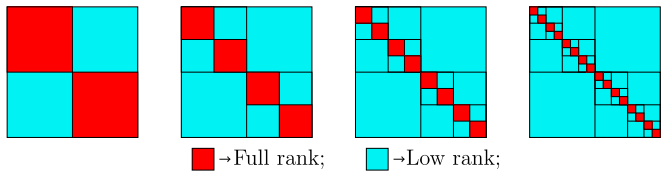

Fig. 1. The same HODLR matrix at different levels.

In this article, we will be working with the class of hierarchical matrices known as HODLR matrices [4], though the ideas extend for other classes of hierarchical matrices as well. As the name suggests, this class of matrices has off-diagonal blocks that are efficiently represented in a recursive fashion. A graphical representation of this class of matrices is shown in Fig. 1. Each block represents the *same* matrix, but viewed on different hierarchical scales to show the particular rank structure.

We first give an example of a simple two-level decomposition for real symmetric matrices, and then describe the arbitrary-level case in more detail. In a slight abuse of notation, in order to be consistent with previous sources describing HODLR matrices, we will refer to the decomposition of a matrix K, which is *not necessarily* the same K as previously mentioned in the covariance matrix case, namely in $C = I + K$.

Algebraically, a real symmetric matrix $K \in \mathbb{R}^{n\times n}$ is termed a two-level HODLR matrix, if it can be written as:

$$K = \begin{bmatrix} K_1^{(1)} & U_1^{(1)} V_1^{(1)^T} \\ V_1^{(1)} U_1^{(1)^T} & K_2^{(1)} \end{bmatrix}, \tag{13}$$

with the diagonal blocks given as

$$\begin{aligned} K_1^{(1)} &= \begin{bmatrix} K_1^{(2)} & U_1^{(2)} V_1^{(2)^T} \\ V_1^{(2)} U_1^{(2)^T} & K_2^{(2)} \end{bmatrix}, \\ K_2^{(1)} &= \begin{bmatrix} K_3^{(2)} & U_2^{(2)} V_2^{(2)^T} \\ V_2^{(2)} U_2^{(2)^T} & K_4^{(2)} \end{bmatrix}, \end{aligned} \tag{14}$$

where the $U_i^{(j)}$, $V_i^{(j)}$ matrices are $n/2^j \times r$ matrices and $r \ll n$. In practice, the rank of the U, V matrices will fluctuate slightly based on the desired accuracy of the approximation. In general, all off diagonal blocks of all factors on all levels can be well-represented by a low-rank matrix, i.e., on each level, $U_i^{(j)}, V_i^{(j)}$ are tall and thin matrices. It is easy to show that the matrix structure given in equations (13) and (14) can be manipulated to provide a factorization of the original matrix as a product of matrices, one of which is block-diagonally dense, and the rest of which are block-diagonal low-rank updates to the identity matrix. This is shown in Fig. 2.

The above is merely a description of the structure of matrices which meet the HODLR requirements, but not a description of how to actually construct the factorization. There are two aspects which need to be discussed: (i) constructing the low-rank approximations of all the off-diagonal blocks, and (ii) using these low-rank approximations to recursively build a factorization of the form shown in Fig. 2.

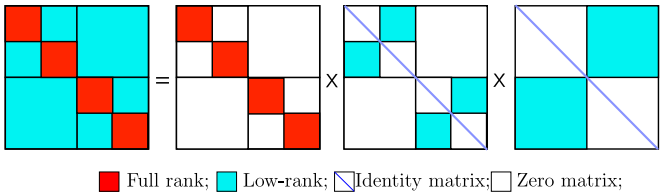

Fig. 2. A two-level factorization of an HODLR matrix.

We now describe several methods for constructing the low-rank approximations in the next section.

3.1 Fast Low-Rank Approximation of Off-Diagonal Blocks

The first key step is to have a computationally efficient way of obtaining the low-rank factorization of the off-diagonal blocks. Given any matrix $A \in \mathbb{R}^{m\times n}$, the *optimal* low-rank approximation (in the least-squares sense) is obtained using the singular value decomposition (SVD) [25]. The downside of using the SVD is that the computational cost of direct factorizations scales as $\mathcal{O}(mnr)$, where r is the numerical rank of the matrix. In practice, r is obtained on-the-fly such that the factorization is accurate to some specified precision ϵ. For our algorithm to be computationally tractable, we need a fast low-rank factorization. More precisely, we need algorithms that scales at most as $\mathcal{O}(r^2 n)$ to obtain a rank r factorization of a $n \times n$ matrix. Thankfully, there has recently been tremendous progress in obtaining fast low-rank factorizations of matrices. These techniques can be broadly classified as either *analytic* or *linear-algebraic* techniques.

If the matrix entries are obtained as evaluations from a smooth function, as is the case for most of the covariance matrices in Gaussian processes, we can rely on approximation theory based analytic techniques like interpolation, multipole expansion, eigenfunction expansion, Taylor series expansions, etc. to obtain a low-rank decomposition. In particular, if the matrix elements are given in terms of a smooth function f, as in the Gaussian process case,

$$A_{ij} = f(x_i, x_j), \tag{15}$$

then polynomial interpolation methods can be used to efficiently approximate the matrix A with near spectral accuracy. Barycentric interpolation formulae such as those recently discussed by Townsend and Trefethen and others [58], [59] serve to effectively factorize A into

$$A \approx E\tilde{A}P, \tag{16}$$

where $\tilde{A}$ is a matrix obtained by sampling the *function* f at suitable chosen nodes, e.g., Chebyshev interpolation nodes. The matrices E, P are then obtained via straightforward interpolation formulas. The accuracy of the approximation can be estimated from spectral analysis of the interpolating Chebyshev polynomial, and the approximation can be computed in $\mathcal{O}(r \max(m, n))$ time.

On the other hand, if there is no *a-priori* information of the matrix, then linear-algebraic methods provide an attractive way of computing fast low-rank decompositions. These include techniques like pseudo-skeletal approximations [26], interpolatory decomposition [21], randomized algorithms [23], [41], [62], rank-revealing LU [44], [48], adaptive cross approximation [50], [66] (which is a minor variant of partial-pivoted LU), and rank-revealing QR [32]. Though purely analytic techniques can be faster since many operations can be pre-computed, algebraic techniques are attractive for constructing black-box low-rank factorizations. The algorithm of this paper relies on an implementation of approximate partial-pivoted LU, which we will now discuss.

Briefly, we construct factorizations of off-diagonal blocks via a partial-pivoted LU decomposition which executes in $\mathcal{O}(rn)$ time. Heuristically, this factorization constructs a series of rank-one matrices whose sum approximates the original matrix, i.e., we wish to write

$$A \approx \sum_{k=1}^{r} \boldsymbol{\alpha}_k \boldsymbol{u}_k \boldsymbol{v}_k^T. \tag{17}$$

The vectors $\boldsymbol{u}_k$, $\boldsymbol{v}_k$ are computed from the columns and rows of A.

The linear complexity is achieved by checking the resulting approximation against only a sub-sampling of the original matrix. If the underlying matrix (covariance kernel) is sufficiently smooth, then this sub-sampling error estimation will result in an approximation which is accurate to near machine precision. For other matrices or covariance kernels which are highly oscillatory or contain small-scale structure, this method will not scale and will likely yield a less-accurate approximation. In this case, analytic methods are preferable as they will be more efficient and provide suitable high-accuracy approximations. We omit a pseudo-code description of this algorithm, as it is a well-know linear algebra procedure, and instead refer to Section 2.2, Algorithm 6 of [50].

The next section presents the fast matrix factorization of the entire covariance matrix once the low-rank decomposition of the off-diagonal blocks has been obtained using one of the above mentioned techniques. We offer a concise, but complete description of the factorization in order to make the exposition self-contained. For a longer and more detailed discussion of the material, see [4].

3.2 HODLR Matrix Factorization

The overall idea behind the $\mathcal{O}(n \log^2 n)$ factorization of an $n \times n$, κ-level (where $\kappa \sim \log n$) HODLR matrix as described in [4] is to factor it as a product of $\kappa + 1$ block diagonal matrices,

$$K = K_\kappa K_{\kappa-1} K_{\kappa-2} \ldots K_1 K_0, \tag{18}$$

where, except for K_κ, $K_k \in \mathbb{R}^{n\times n}$ is a block diagonal matrix with 2^k diagonal blocks, each of size $n/2^k \times n/2^k$. More importantly, each of these diagonal blocks is a low-rank update to the identity matrix. The first factor K_κ is formed from dense block diagonal sub-matrices of the original matrix, K. Aside from straightforward block-matrix algebra, the main tool used in constructing this factorization is the Sherman-Morrison-Woodbury formula [36], [53], [61]. To simplify the notation assume for a moment that K is an $n \times n$ matrix, where $n = 2^m$ for some integer m. For example, a two-level HODLR matrix described in equations (13) and (14) can be factorized as:

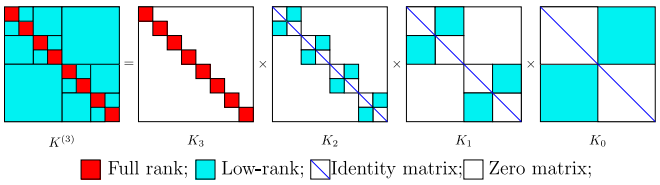

Fig. 3. Factorization of a three level HODLR matrix.

$$\begin{bmatrix} K_1^{(2)} & 0 & 0 & 0 \\ 0 & K_2^{(2)} & 0 & 0 \\ 0 & 0 & K_3^{(2)} & 0 \\ 0 & 0 & 0 & K_4^{(2)} \end{bmatrix} \begin{bmatrix} I_{n/4} & \tilde{K}_{12}^{(2)} & 0 & 0 \\ \tilde{K}_{21}^{(2)} & I_{n/4} & 0 & 0 \\ 0 & 0 & I_{n/4} & \tilde{K}_{34}^{(2)} \\ 0 & 0 & \tilde{K}_{43}^{(2)} & I_{n/4} \end{bmatrix} \begin{bmatrix} I_{n/2} & \tilde{K}_{12}^{(1)} \\ \tilde{K}_{21}^{(1)} & I_{n/2} \end{bmatrix}, \tag{19}$$

where I_m is the $m \times m$ identity matrix, and the matrices $\tilde{K}_{ij}^{(k)}$ are low-rank. Similarly, Fig. 3 graphically depicts the factorization of a level 3 HODLR matrix.

In the case of a one-level factorization, we can easily write down the computation. Let the matrix K be:

$$K = \begin{bmatrix} A_{11} & UV^T \\ VU^T & A_{22} \end{bmatrix}, \tag{20}$$

where we assume that U, V have been computed using one of the algorithms of the previous section. Then the *only* step in the decomposition is to factor out the terms A_{11}, A_{22}, giving:

$$K = \begin{bmatrix} A_{11} & 0 \\ 0 & A_{22} \end{bmatrix} \begin{bmatrix} I_{n/2} & A_{11}^{-1}UV^T \\ A_{22}^{-1}VU^T & I_{n/2}. \end{bmatrix}. \tag{21}$$

We see that the computation involved was to merely apply the inverse of the dense block diagonal factor to the corresponding rows in the remaining factor. Furthermore, since the matrix UV^T was low-rank, so is $A_{11}^{-1}UV^T$. Unfortunately, a one-level factorization such as this is still quite expensive: it required the direct inversion of A_{11}, A_{22}, each of which are $n/2 \times n/2$ matrices. The procedure must be done recursively across $\log n$ levels in order to achieve a nearly optimal algorithm.

$$K = \begin{bmatrix} A_{11} & 0 & 0 & 0 \\ 0 & A_{22} & 0 & 0 \\ 0 & 0 & A_{33} & 0 \\ 0 & 0 & 0 & A_{44} \end{bmatrix} \begin{bmatrix} I_{n/4} & A_{11}^{-1}U_1^{(2)}V_1^{(2)^T} & 0 & 0 \\ A_{22}^{-1}V_1^{(2)}U_1^{(2)^T} & I_{n/4} & 0 & 0 \\ 0 & 0 & I_{n/4} & A_{33}^{-1}U_2^{(2)}V_2^{(2)^T} \\ 0 & 0 & A_{44}^{-1}V_2^{(2)}U_2^{(2)^T} & I_{n/4} \end{bmatrix} \begin{bmatrix} I_{n/2} & A_1^{-1}U_1^{(1)}V_1^{(1)^T} \\ A_2^{-1}V_1^{(1)}U_1^{(1)^T} & I_{n/2} \end{bmatrix} \tag{22}$$

Before describing the general scheme, we give the full two-level factorization using the notation of equations (13) and (14). The full factorization in this two-level scheme is given in equation (22) (spanning two columns). The matrices A_1 and A_2 appearing in the off-diagonal expressions are given by:

$$A_1 = \begin{bmatrix} A_{11} & U_1^{(2)}V_1^{(2)^T} \\ V_1^{(2)}U_1^{(2)^T} & A_{22} \end{bmatrix}$$
$$A_2 = \begin{bmatrix} A_{33} & U_2^{(2)}V_2^{(2)^T} \\ V_2^{(2)}U_2^{(2)^T} & A_{44} \end{bmatrix}. \tag{23}$$

This factorization is an indication of how to construct the ultimate κ-level factorization as it only required the direct construction of the inverse of dense matrices of size $n/4 \times n/4$. If this procedure is repeated recursively, the only dense inversions required are of $n/2^\kappa \times n/2^\kappa$ matrices.

At first glance, it may look as though the computation of A_1^{-1}, A_2^{-1} is expensive, and will scale as $\mathcal{O}(n^3/8)$. However, these matrices are of the form:

$$\begin{bmatrix} A & UV^T \\ VU^T & B \end{bmatrix} = \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} + \begin{bmatrix} U & 0 \\ 0 & V \end{bmatrix} \begin{bmatrix} 0 & V^T \\ U^T & 0 \end{bmatrix}. \tag{24}$$

If the inverses of A, B are known (and they are in this case, they were computed on a finer level), and U, V are low-rank matrices, then the inverse of the full matrix can be computed rapidly using the Sherman-Morrison-Woodbury formula:

$$(A + LSR)^{-1} = A^{-1} - A^{-1}L\big(S^{-1} + RA^{-1}L\big)^{-1}RA^{-1}.$$

If S is of small rank, then the inner inverse can be computed very rapidly.

To summarize, see Fig. 4 for rough pseudo-code describing how to construct a general κ-level HODLR factorization. We avoid too much index notation, please see [4] for a full detailed algorithm.

This pseudo-code computes a factorization of the original matrix K. We have not yet computed the inverse K^{-1}. The inverse can be computed by directly applying the Sherman-Morrison-Woodbury formula to each term in the factorization

$$K = K_\kappa K_{\kappa-1} \ldots K_1 K_0. \tag{25}$$

Since each term is block diagonal or a block diagonal low-rank update to the identity matrix, the inverse factorization can be computed in $\mathcal{O}(n \log n)$ time.

Before moving on we would like to point out that in the case where the data points at which the kernel is to be evaluated at are not approximately uniformly distributed, the performance of the factorization may suffer, but only slightly. A higher level of compression could be obtained in the off-diagonal blocks if the hierarchical tree structure is constructed based on spatial considerations instead of *point count*, as is the case with some *kd*-tree implementations.

The next section gives a brief estimate of the computational complexity of constructing a HODLR-type factorization.

3.3 Computational Complexity

Constructing a HODLR-type factorization can be split into two main steps: (i) computing the low-rank factorization of all off-diagonal blocks, and (ii) using these low-rank approximations to recursively factor the matrix into roughly $\mathcal{O}(\log n)$ pieces.

Require: Factorization precision $\epsilon > 0$
Require: Size of smallest sub-matrix on finest level, $p_{\max}$ {the size of the smallest diagonal block}
Require: Matrix entry evaluation routine, $f(i,j)$
$\kappa \leftarrow \lfloor \log_2 n/p_{max} \rfloor$
for $j = 1$ to κ **do**
  for all i **do**
    Compute the low-rank factorization $U_i^{(j)} V_i^{(j)^T}$ of all off-diagonal blocks in the κ-level hierarchy to precision ϵ
  end for
end for
Form the block diagonal matrix K_κ using the block diagonals of the original matrix K
for $j = \kappa - 1$ down to 1 **do**
  for $\ell = j - 1$ down to 1 **do**
    Apply the inverse of the block diagonals of K_j to each left low-rank factor, $U_i^{(\ell)}$, $V_i^{(\ell)}$, in the remaining off-diagonal blocks {Factor K_j out of the remaining matrix}
  end for
end for
K_0 is the last factor, off the form $I +$ low-rank, obtained by having applied all earlier factor inverses to off-diagonal blocks on finer levels

Fig. 4. Pseudo-code for constructing an HODLR factorization.

For an $n \times n$ matrix which admits the HODLR structure, as shown in, Fig. 1, there are approximately $\kappa \approx \log_2 n/p$, where p is the size of the diagonal block on the finest level (this is a user-defined parameter). Ignoring the diagonal blocks, this means there are two blocks of size $n/2 \times n/2$, four blocks of size $n/4 \times n/4$, etc. Finding the low-rank approximation of an $n/2^j \times n/2^j$ off-diagonal block using cross approximation requires $\mathcal{O}(rn/2^j)$ flops, where r is the ϵ-rank of the sub-matrix. Constructing all such factorizations requires $\mathcal{O}(rn \log n)$.

Once the approximations are obtained, the matrix must be pulled apart into its HODLR factorization. Let us remember that there are $\kappa \approx \log_2 n$ levels in the HODLR structure. In order to factor K_κ (the matrix of dense block diagonals), as in equation (18), out of the original matrix K, we must apply the inverse of the corresponding block diagonal to all the left low-rank factors, $U_i^{(\kappa)}$, $V_i^{(\kappa)}$ as in equations (21) - (23). In general, a $p \times p$ inverse must computed and applied to all left low-rank factors, of which there are $\mathcal{O}(\kappa)$. The inverse calculation is $\mathcal{O}(p^3)$, and the subsequent application is $\mathcal{O}(prn/2^j)$, $j = 1, \ldots, \kappa$, which dominates the inverse calculation. There are 2^κ such applications, yielding the cost *for only the first factorization level* to be $\mathcal{O}(prn \log n)$. Applying the same reasoning as each subsequent factor K_j, $j = \kappa - 1, \ldots, 0$, along with the complexity result,

$$\sum_{j=1}^{\log n} j = \mathcal{O}(\log^2 n), \tag{26}$$

yields a total complexity for the factorization stage of $\mathcal{O}(n \log^2 n)$. The factors of p, r have been dropped as it is assumed that $p, r \ll n$.

Given a HODLR-type factorization, it is straightforward to show that the computational complexity of determining the inverse scales as $\mathcal{O}(n \log n)$. There are $\mathcal{O}(\log n)$ factors, and since each level is constructed as low-rank updates to the identity, invoking the Sherman-Morrison-Woodbury formula yields the inverse in $\mathcal{O}(n)$ time. This gives a total runtime of $\mathcal{O}(n \log n)$.

4 Determinant Computation

As discussed earlier, once the HODLR factorization has been obtained, Sylvester's determinant theorem [1] enables the computation of the determinant at a cost of $\mathcal{O}(n \log n)$ operations. This computationally inexpensive method for direct determinant evaluation enables the efficient *direct* evaluation of probabilities. We now briefly review the algorithm used for determinant evaluation.

Theorem 4.1 (Sylvester's Determinant Theorem). *If $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times m}$, then*

$$\det(I_m + AB) = \det(I_n + BA),$$

where $I_k \in \mathbb{R}^{k \times k}$ is the identity matrix. In particular, for a rank p update to the identity matrix,

$$\det(I_n + U_{n \times p} V_{p \times n}) = \det(I_p + V_{p \times n} U_{n \times p}).$$

Remark 4.2. The computational cost associated with computing the determinant of a rank p update to the identity is $\mathcal{O}(p^2 n)$. The dominant cost is computing the matrix-matrix product $V_{p \times n} U_{n \times p}$.

Furthermore, we recall two basic facts regarding the determinant. First, the determinant of a block diagonal matrix is the product of the determinants of the individual blocks of the matrix. Second, the determinant of a square matrix is completely multiplicative over the set of square matrices, that is to say,

$$\det(A_1 A_2 \cdots A_n) = \det(A_1) \det(A_2) \cdots \det(A_n). \tag{27}$$

Using the HODLR factorization in equation (18) and these two facts, we have:

$$\begin{aligned} \det(K) = \det(K_\kappa) \det(K_{\kappa-1}) \det(K_{\kappa-2}) \cdots \\ \det(K_2) \det(K_1) \det(K_0). \end{aligned} \tag{28}$$

Each of the determinants on the right hand side of equation (28) can be computed as a product of the determinants

TABLE 2
Timings for Gaussian Covariance Functions in One, Two, and Three Dimensions

	One-dimensional data					Two-dimensional data					Three-dimensional data				
n	Assembly	Factor	Solve	Det.	Error	Assembly	Factor	Solve	Det.	Error	Assembly	Factor	Solve	Det.	Error
$10,000$	0.12	0.11	0.008	0.01	10^{-13}	0.56	0.50	0.018	0.03	10^{-13}	15.4	17.3	0.113	0.91	10^{-12}
$20,000$	0.15	0.23	0.016	0.03	10^{-13}	1.16	0.99	0.028	0.05	10^{-13}	30.9	33.1	0.224	1.06	10^{-12}
$50,000$	0.47	0.71	0.036	0.12	10^{-12}	2.74	2.44	0.067	0.12	10^{-13}	75.5	76.3	0.434	1.68	10^{-11}
$100,000$	1.24	1.46	0.052	0.24	10^{-12}	5.43	5.08	0.165	0.23	10^{-12}	149	166	0.923	3.11	10^{-11}
$200,000$	2.14	3.12	0.121	0.39	10^{-13}	12.4	14.4	0.485	0.44	10^{-12}					
$500,000$	6.13	10.2	0.388	0.56	10^{-12}	31.7	37.3	1.33	1.17	10^{-12}					
$1,000,000$	14.1	23.2	0.834	1.52	10^{-12}	70.8	79.2	3.15	2.24	10^{-12}					

The matrix entries are given as $C_{ij} = 2\delta_{ij} + \exp(-||r_i - r_j||^2)$, *where* r_i *are random uniformly distributed points in the interval* $[-3, 3]^d$ $(d = 1, 2, 3)$.

of diagonal blocks. Each of these diagonal blocks is a low-rank perturbation to the identity, and hence, the determinant can be computed using Sylvester's Determinant Theorem 4.1. It is easy to check that the computational cost for each of the determinants $\det(K_i)$ is $\mathcal{O}(n)$, therefore the total computational cost for obtaining $\det(K)$ is $\mathcal{O}(\kappa n)$, and we recall that $\kappa \sim \log n$.

5 Numerical Results

In this section, we discuss the performance of the previously described algorithm for the inversion and application of covariance matrices $C = I + K$, as well as the calculation of the normalization factor, i.e., the determinant $\det(C)$. Detailed results for n-dimensional datasets $\boldsymbol{x}$, where each x_i is a point in one, two, and three dimensions, are provided for the Gaussian and multiquadric covariance kernels. We also provide benchmarks in one dimension for covariance matrices constructed from exponential, inverse multiquadric, and biharmonic covariance functions. Unless otherwise, stated, all the proceeding numerical experiments have been run on a MacBook Air with a 1.3 GHz Intel Core i5 processor and 4 GB 1600 MHz DDR3 RAM. In all these cases, the matrix entry C_{ij} is given as

$$C_{ij} = \sigma_i^2 \delta_{ij} + k(r_i, r_j),$$

where $k(r_i, r_j)$ is a particular covariance function evaluated at two points, r_i and r_j. It should be noted that in certain cases, the matrix K with entries $K_{ij}(\boldsymbol{r}) = k(r_i, r_k)$ might itself be a rank deficient matrix and that this has been exploited in the past to construct fast schemes. However, this is not always the case, for instance, if the covariance function is an exponential. In this situation, the rank of K is in fact full-rank. Even if the matrix K were to be formally rank deficient, in practice, the rank might be very large whereas the ranks of the off-diagonal blocks in the hierarchical structure are very small. Another major advantage of this hierarchical approach is that it is applicable to a wide range of covariance functions and can be used in a black-box, plug-and-play, fashion.

In each of the following tables, timings are provided for the assembly, factorization, and inversion of the covariance matrix (denoted by the columns *Assembly*, *Factor*, and *Solve*). Assembly of the matrix refers to computing all of the low-rank factorizations of the off-diagonal blocks (including the time required to construct a kd-tree on the data), and Factor refers to computing the $\log n$ level factorization described earlier. The time for computing the determinant of the $n \times n$ matrix is given in the *Det.* column, and the error provided is approximately the relative l_2 precision in the solution to a test problem $C\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{b}$ was generated *a priori* from the known vector $\boldsymbol{x}$.

5.1 Gaussian Covariance

Here the covariance function, and the corresponding entry of K, is given as

$$k(r_i, r_j) = \exp\left(-|r_i - r_j|^2\right), \tag{29}$$

where r_i, r_j are points in one, two, or three dimensions. The results for the Gaussian covariance kernel have been aggregated in Table 2. Scaling of the algorithm for data embedded in one, two, and three dimensions is compared with the direct calculation in Fig. 5.

5.2 Multiquadric Covariance Matrices

Covariance functions of the form

$$k(r_i, r_j) = \sqrt{\left(1 + |r_i - r_j|^2\right)} \tag{30}$$

are known as multiquadric covariance functions, one class of frequently used radial basis functions. Analogous numerical results are presented below in one, two, and three dimensions as were in the previous section for the Gaussian

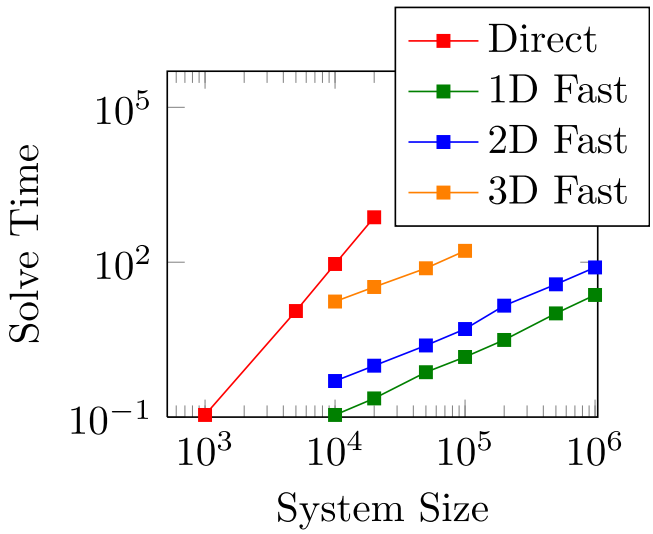

Fig. 5. Comparison of time required to factorize the covariance matrix in the case of a Gaussian covariance kernel in one, two, and three dimensions. The conventional direct calculation is independent of dimension but scales as $\mathcal{O}(n^3)$.

TABLE 3
Timings for Multiquadric Covariance Functions in One, Two, and Three Dimensions

	One-dimensional data					Two-dimensional data					Three-dimensional data				
n	Assembly	Factor	Solve	Det.	Error	Assembly	Factor	Solve	Det.	Error	Assembly	Factor	Solve	Det.	Error
$10,000$	0.08	0.13	0.006	0.02	10^{-13}	0.77	0.86	0.022	0.04	10^{-13}	19.0	23.2	0.135	1.32	10^{-12}
$20,000$	0.11	0.22	0.011	0.03	10^{-13}	1.41	1.42	0.042	0.06	10^{-13}	38.7	45.1	0.276	1.65	10^{-11}
$50,000$	0.34	0.65	0.030	0.13	10^{-13}	3.31	3.43	0.082	0.15	10^{-12}	87.8	97.8	0.578	2.28	10^{-11}
$100,000$	0.85	1.44	0.059	0.22	10^{-12}	6.54	6.95	0.177	0.31	10^{-12}	164	195	1.24	3.84	10^{-10}
$200,000$	1.56	3.12	0.147	0.44	10^{-13}	14.1	15.9	0.395	0.59	10^{-11}					
$500,000$	4.72	8.33	0.363	0.94	10^{-12}	38.2	42.1	1.12	1.69	10^{-11}					
$1,000,000$	10.9	17.1	0.814	1.94	10^{-12}	79.9	90.3	2.38	3.39	10^{-11}					

The matrix entries are given as $C_{ij} = \delta_{ij} + \sqrt{1 + ||r_i - r_j||}$, where r_i are random uniformly distributed points in the interval $[-3, 3]^d$ $(d = 1, 2, 3)$.

covariance function. Table 3 contains the results. Scaling is virtually identical to the Gaussian case, and we omit the corresponding plot.

5.3 Exponential Covariance

Covariances functions of the form

$$k(r_i, r_j) = \exp(-|r_i - r_j|) \tag{31}$$

are known as exponential covariance functions. One-dimensional numerical results are presented in Table 4. Fig. 6 compares scaling for various kernels.

5.4 Inverse Multiquadric and Biharmonic

The inverse multiquadric and biharmonic kernel (also known as the thin plane spline) are frequently used in radial basis function interpolation and *kriging* in geostatistics. These kernels are given by the formulae

$$k_1(r_i, r_j) = \frac{1}{\sqrt{1 + |r_i - r_j|^2}}, \qquad k_2(r_i, r_j) = |r_i - r_j|^2 \log |r_i - r_j|, \tag{32}$$

respectively. Timing results are presented in Tables 5 and 6, and comparison with the exponential kernel is shown in Fig. 6.

TABLE 4
Timings for One-Dimensional Exponential Covariance Functions

	Time taken in seconds				
n	Assembly	Factor	Solve	Det.	Error
10^4	0.13	0.06	0.003	0.02	10^{-13}
2×10^4	0.23	0.11	0.008	0.03	10^{-13}
5×10^4	0.64	0.32	0.020	0.10	10^{-12}
10^5	1.41	0.70	0.039	0.23	10^{-13}
2×10^5	2.86	1.42	0.076	0.42	10^{-12}
5×10^5	8.63	3.47	0.258	0.67	10^{-12}
10^6	18.8	8.05	0.636	1.35	10^{-12}

The matrix entry is given as $C_{ij} = \delta_{ij} + \exp(-|r_i - r_j|)$, where r_i are random uniformly distributed points in the interval $[-3, 3]$.

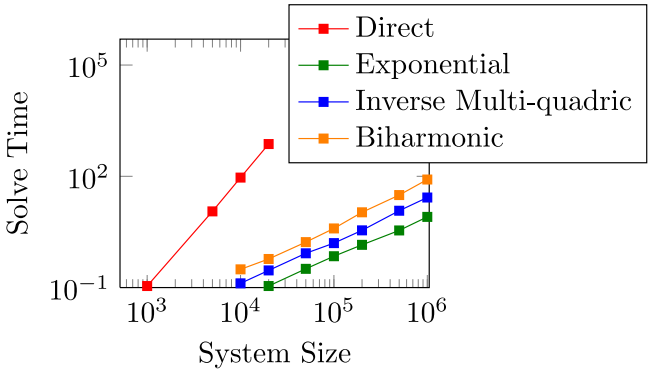

Fig. 6. Comparison of time required to factorize a variety of covariance matrices arising from exponential, inverse multiquadric, and biharmonic covariance kernels for one-dimensional data.

5.5 Scaling in High Dimensions

In this section we report results on the scaling of the algorithm described in this paper when the data (independent variables, $\boldsymbol{x}$) lie in high euclidean dimensions. We perform two experiments. First, we run our algorithm on data lying in the hypercube $[-3, 3]^d$ for various values of d. In this scenario, we actually see an *increase* in computational speed

TABLE 5
Timings for One-Dimensional Inverse Multiquadric Covariance Functions

	Time taken in seconds				
n	Assembly	Factor	Solve	Det.	Error
10^4	0.11	0.13	0.006	0.02	10^{-13}
2×10^4	0.17	0.29	0.017	0.04	10^{-13}
5×10^4	0.47	0.84	0.037	0.12	10^{-12}
10^5	1.07	1.58	0.072	0.21	10^{-12}
2×10^5	2.18	3.49	0.158	0.44	10^{-12}
5×10^5	6.43	11.8	0.496	0.71	10^{-11}
10^6	14.2	26.8	1.02	1.49	10^{-11}

The matrix entry is given as $C_{ij} = \delta_{ij} + 1/\sqrt{1 + |r_i - r_j|^2}$, where r_i are random uniformly distributed points in the interval $[-3, 3]$.

TABLE 6
Timings for One-Dimensional Biharmonic Covariance Function

	Time taken in seconds				
n	Assembly	Factor	Solve	Det.	Error
10^4	0.28	0.31	0.015	0.03	10^{-12}
2×10^4	0.61	0.59	0.028	0.06	10^{-12}
5×10^4	1.62	1.68	0.067	0.15	10^{-12}
10^5	3.61	3.93	0.123	0.34	10^{-12}
2×10^5	8.03	10.7	0.236	0.65	10^{-12}
5×10^5	26.7	31.2	0.632	1.41	10^{-11}
10^6	51.3	81.9	1.28	3.40	10^{-11}

The matrix entry is given as $C_{ij} = 2\delta_{ij} + |r_i - r_j|^2 \log(|r_i - r_j|)$, where r_i are random uniformly distributed points in the interval $[-3, 3]$.

TABLE 7
Timings for Gaussian Covariance Functions in d-Dimensions

		Time taken in seconds				
n	Dim d	Assembly	Factor	Solve	Det.	Error
5000	1	0.02	0.087	0.002	0.001	10^{-12}
5000	2	1.17	2.043	0.027	0.014	10^{-11}
5000	4	9.67	21.25	0.073	0.088	10^{-13}
5000	8	10.4	7.773	0.041	0.039	10^{-08}
5000	16	0.09	0.168	0.002	0.002	10^{-17}
5000	32	0.15	0.061	0.004	0.001	10^{-16}
5000	64	0.21	0.059	0.001	0.001	10^{-16}

The matrix entries are given as $C_{ij} = 2\delta_{ij} + \exp(-||r_i - r_j||^2)$, where r_i are random uniformly distributed points in the unscaled interval $[-3, 3]^d$ (where $d = 1, \ldots, 64$).

and accuracy as d is increased after some point. These results are reported in Table 7. However, this is not a fair result. As d increases, the expected value of $r = ||r_i - r_j||$ increases, causing, at least in the case of an unscaled Gaussian covariance kernel, for many matrix entries to be very close to zero.

The second experiment was with the same set of parameters, except the data are located in the *scaled* hypercube, $[-3/\sqrt{d}, 3/\sqrt{d}]^d$. These results are reported in Table 8. This is equivalent to rescaling euclidean distance, or rescaling the covariance kernel. We see that the scalings for this experiment saturate once $d \approx 10$ due to the fact that we are not increasing the number of data points n along with d. For fixed n, the data become very sparse as d increases and the ranks of the off-diagonal blocks in the associated covariance matrix remain the same. What we mean to say by this is that $5,000$ points in a ten-dimensional space is massively under-sampling any sort of spatial structure, this is equivalent to grid of about two or three points per dimension ($2.34^{10} \approx 5000$). Running the algorithm with this type of data is equivalent to doing dense linear algebra since the ranks of all off-diagonal blocks are close to full. This is a manifestation of the curse of dimensionality. However, it's possible that one may encounter both types of data (scaled versus unscaled) in real-world situations.

These experiments were run on a faster laptop, namely a MacBook Pro with a 3.0 GHz Intel Core i7 processor and 16 GB 1600 MHz DDR3 RAM. No sophisticated software optimizations were made.

TABLE 8
Timings for Gaussian Covariance Functions in d-Dimensions

		Time taken in seconds				
n	Dim d	Assembly	Factor	Solve	Det.	Error
5000	1	0.02	0.087	0.002	0.001	10^{-12}
5000	2	0.70	1.157	0.017	0.005	10^{-11}
5000	4	33.6	108.2	0.165	0.360	10^{-12}
5000	8	40.8	138.4	0.186	0.359	10^{-15}
5000	16	41.9	138.7	0.182	0.358	10^{-15}
5000	32	43.4	174.1	0.174	0.354	10^{-15}
5000	64	42.2	142.2	0.181	0.346	10^{-15}

The matrix entries are given as $C_{ij} = 2\delta_{ij} + \exp(-||r_i - r_j||^2)$, where r_i are random uniformly distributed points in the scaled interval $[-3/\sqrt{d}, 3/\sqrt{d}]^d$ (where $d = 1, \ldots, 64$).

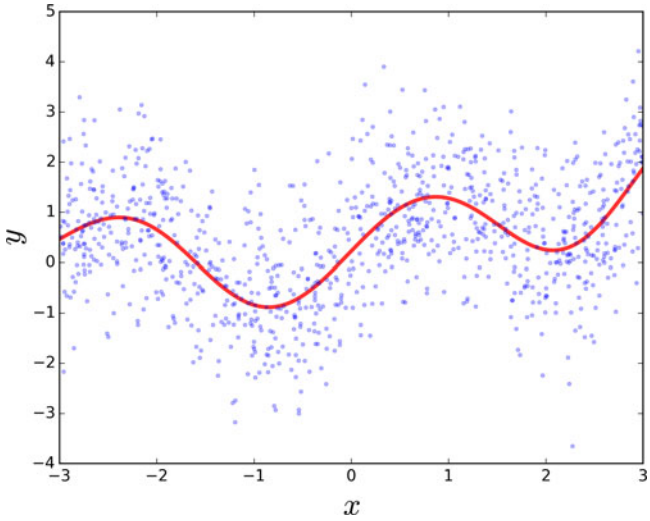

Fig. 7. Regression obtained from Gaussian process prior for synthetic data drawn from the model in equation (33).

5.6 Regression Performance

In this section we demonstrate the relationship between various parameters in our algorithm and regression performance. The two main parameters that need to be set in our algorithm are the factorization precision ϵ (see Fig. 4) and the maximum size of the smallest sub-matrix on the finest level, p_{max}. For a fixed ϵ, changing p_{max} does *not* affect the $RMSE$ (root-mean-square error) of the regression (up to machine precision errors), it merely affects the overall runtime. For sufficiently large p_{max}, the scheme ceases to be a multi-level algorithm. We merely state this as a fact, and do not report the data. We set $p_{max} = 20$ in the following numerical experiment.

However, for varying values of ϵ, we present the difference between the $RMSE$ for the exact (dense linear algebra) regression and the regression obtained using the matrix factorization algorithm of this paper. We generate $1,024$ datapoints $\{x_j, y_j\}$ from the model:

$$y = \sin(2x) + \frac{1}{8}e^x + \epsilon, \tag{33}$$

where $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$ and x_j is chosen randomly in the interval $(-3, 3)$. The non-parametric regression curve (or estimate) under a Gaussian process prior (with zero mean) is then calculated at the same points x_j as in (6) and (7):

$$\hat{\boldsymbol{y}} = K(\boldsymbol{x})\big(\sigma_\epsilon^2 I + K(\boldsymbol{x})\big)^{-1}\boldsymbol{y}, \tag{34}$$

where we have chosen our covariance kernel to be consistent with the previous numerical experiments:

$$k(x, x') = e^{-(x-x')^2}. \tag{35}$$

No effort was made to adapt the covariance kernel to the synthetic data. For $\sigma_\epsilon = 1.0$, the inferred curve through the data is shown in Fig. 7. If dense linear algebra is used to invert $I + K$ to obtain the *exact* estimate $\boldsymbol{y}_{exact}$, the $RMSE$ for this data is:

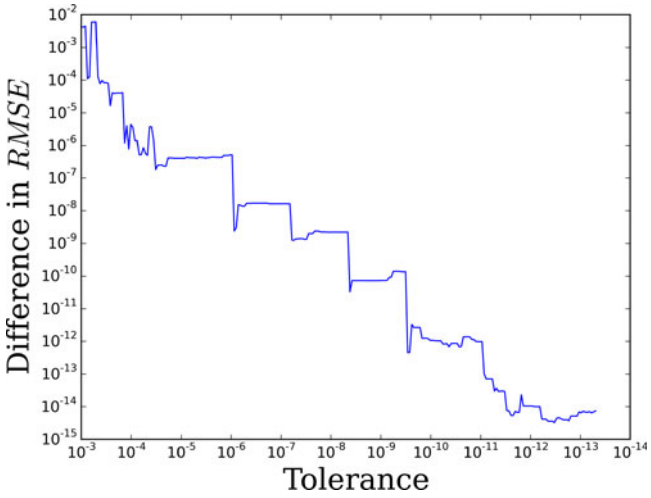

Fig. 8. Difference in the root-mean-square error of the Gaussian process regression using the algorithm of this paper and that of the direct calculation as a function of the factorization precision ϵ.

$$RMSE = \frac{\|\boldsymbol{y} - \hat{\boldsymbol{y}}_{exact}\|}{\sqrt{1024}} = 0.9969571605921435. \tag{36}$$

Fig. 8 shows the *absolute difference* between the $RMSE$ obtained from dense linear algebra and that obtained from our accelerated scheme as a function of factorization precision ϵ. Unsurprisingly, from this plot we determine that, indeed, the difference in regression performance (at least as measured by $RMSE$) is proportional to ϵ.

6 Conclusions

In this paper, we have presented a fast, accurate, and nearly optimal hierarchical direct linear algebraic algorithms for computing determinants, inverses, and matrix-vector products involving covariances matrices encountered when using Gaussian processes. Similar matrices appear in problems of classification and prediction; our method carries over and applies equally well to these problems. Previous attempts at accelerating these calculations (inversion and determinant calculation) relied on either sacrificing fidelity in the covariance kernel (e.g., thresholding), constructing a global low-rank approximation to the covariance kernel, or paying the computational penalty of dealing with dense, full-rank covariance matrices. Our HODLR-based algorithm obviates the need for this compromise. Our *observation* that many covariance matrices of mathematical statistics have fine-grained, compressible hierarchical structure that provides access to the inverse may find use in many applications in the future.

The source code for the algorithm has been made available on GitHub. The HODLR package for solving linear systems and computing determinants is available at https://github.com/sivaramambikasaran/HODLR [3] and the Python Gaussian process package [22], *george*, has been made available at https://github.com/dfm/george. Both packages are open source, the HODLR package is released under the MPL2.0 license and *george* is released under the MIT license. Details on using these packages are available at their respective online repositories.

In its present form, our method degrades in performance when the n-dimensional data has a covariance function based on points in $\mathbb{R}^d$ with $d > 3$, as well as when the covariance function is oscillatory. Part of the performance loss cannot be avoided due to the curse of dimensionality. High-dimensional data is simply more complicated than low-dimensional data causing the off-diagonal blocks to have larger ranks (at least in the scenario of more and more data samples). The other part of the performance loss is in the compression. For high-dimensional data, analytic interpolatory low-rank approximations will provide faster and more robust approximations. Extensions of our approach to these cases is a subject of current research. We are also investigating high-dimensional anisotropic quadratures for marginalization and moment computation.

Acknowledgments

The authors would like to thank Iain Murray for several useful and detailed discussions. Research by S. Ambikasaran and M. O'Neil was supported in part by the Air Force Office of Scientific Research under NSSEFF Program Award FA9550-10-1-0180 and AIG-NYU Award #A15-0098-001. Research by L. Greengard was supported in part by the Simons Foundation and the Air Force Office of Scientific Research under NSSEFF Program Award FA9550-10-1-0180. Research by D. Foreman-Mackey and D. W. Hogg was supported in part by NASA grant NNX12AI50G, US National Science Foundation (NSF) grant IIS-1124794, the Gordon and Betty Moore Foundation, and the Alfred P. Sloan Foundation.

References

[1] A. G. Akritas, E. K. Akritas, and G. I. Malaschonok, "Various proofs of Sylvester's (determinant) identity," *Math. Comput. Simul.*, vol. 42, no. 4, pp. 585–593, 1996.

[2] S. Ambikasaran, "Fast algorithms for dense numerical linear algebra and applications," Ph.D. dissertation, Stanford Univ., Stanford, CA, USA, 2013.

[3] S. Ambikasaran. (2013). A fast direct solver for dense linear systems [Online]. Available: https://github.com/sivaramambikasaran/HODLR

[4] S. Ambikasaran and E. Darve, "An $\mathcal{O}(n \log n)$ fast direct solver for partial hierarchically semi-separable matrices," *J. Sci. Comput.*, vol. 57, no. 3, pp. 477–501, 2013.

[5] S. Ambikasaran, J. Y. Li, P. K. Kitanidis, and E. Darve, "Large-scale stochastic linear inversion using hierarchical matrices," *Comput. Geosci.*, vol. 17, no. 6, pp. 913–927, 2013.

[6] S. Ambikasaran, A. K. Saibaba, E. F. Darve, and P. K. Kitanidis, "Fast algorithms for Bayesian inversion," in *Computational Challenges in the Geosciences*. New York, NY, USA: Springer, 2013, pp. 101–142.

[7] A. Aminfar, S. Ambikasaran, and E. Darve, "A fast block low-rank dense solver with applications to finite-element matrices," *arxiv.org/abs/1403.5337*, 2014.

[8] M. Anitescu, J. Chen, and L. Wang, "A matrix-free approach for solving the parametric Gaussian process maximum likelihood problem," *SIAM J. Sci. Comput.*, vol. 34, pp. A240–A262, 2012.

[9] R. P. Barry and R. K. Pace, "Monte carlo estimates of the log determinant of large sparse matrices," *Linear Algebra Its Appl.*, vol. 289, pp. 41–54, 1999.

[10] S. Börm, L. Grasedyck, and W. Hackbusch, "Hierarchical matrices," *Lecture Notes*, vol. 21, 2003.

[11] R. P. Brent, J.-A. H. Osborn, and W. D. Smith, "Bounds on determinants of perturbed diagonal matrices," *arxiv.org/abs/1401.7084*, 2013.

[12] O. Cappé, E. Moulines, and T. Rydén, *Inference in Hidden Markov Models*. New York, NY, USA: Springer-Verlag, 2005.

[13] K. Chalupka, C. K. I. Williams, and I. Murray, "A framework for evaluating approximation methods for Gaussian process regression," *J. Mach. Learn. Res.*, vol. 14, pp. 333–350, 2013.
[14] S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, and T. Pals, "A fast solver for HSS representations via sparse matrices," *SIAM J. Matrix Anal. Appl.*, vol. 29, no. 1, pp. 67–81, 2006.
[15] S. Chandrasekaran, M. Gu, and T. Pals, "A fast ULV decomposition solver for hierarchically semiseparable representations," *SIAM J. Matrix Anal. Appl.*, vol. 28, no. 3, pp. 603–622, 2006.
[16] C. Chatfield, *The Analysis of Time Series: An Introduction*, 6th ed. Boca Raton, FL, USA: Chapman and Hall, 2003.
[17] J. Chen, L. Wang, and M. Anitescu, "A fast summation tree code for Matérn kernel," *SIAM J. Sci. Comput.*, vol. 36, pp. A289–A309, 2013.
[18] C. Dietrich and G. Newsam, "Fast and exact simulation of stationary Gaussian processes through circulant embedding of the covariance matrix," *SIAM J. Sci. Comput.*, vol. 18, p. 1088, 1997.
[19] A. Dutt and V. Rokhlin, "Fast Fourier transforms for nonequispaced data," *SIAM J. Sci. Comput.*, vol. 14, no. 6, pp. 1368–1393, 1993.
[20] A. Dutt and V. Rokhlin, "Fast Fourier transforms for nonequispaced data, II," *Appl. Comput. Harmonic Anal.*, vol. 2, no. 1, pp. 85–100, 1995.
[21] W. Fong and E. Darve, "The black-box fast multipole method," *J. Comput. Phys.*, vol. 228, no. 23, pp. 8712–8725, 2009.
[22] D. Foreman-Mackey. (2014). Fast Gaussian processes for regression [Online]. Available: https://github.com/dfm/george, 2014.
[23] A. Frieze, R. Kannan, and S. Vempala, "Fast Monte-Carlo algorithms for finding low-rank approximations," *J. ACM*, vol. 51, no. 6, pp. 1025–1041, 2004.
[24] Z. Gimbutas and V. Rokhlin, "A generalized fast multipole method for nonoscillatory kernels," *SIAM J. Sci. Comput.*, vol. 24, no. 3, pp. 796–817, 2003.
[25] G. Golub and C. Van Loan, *Matrix Computations*, vol. 3. Baltimore, MD, USA: Johns Hopkins Univ. Press, 1996.
[26] S. Goreinov, E. Tyrtyshnikov, and N. Zamarashkin, "A theory of pseudoskeleton approximations," *Linear Algebra Its Appl.*, vol. 261, nos. 1–3, pp. 1–21, 1997.
[27] L. Grasedyck and W. Hackbusch, "Construction and arithmetics of $\mathcal{H}$-matrices," *Computing*, vol. 70, no. 4, pp. 295–334, 2003.
[28] L. Greengard and J.-Y. Lee, "Accelerating the nonuniform fast fourier transform," *SIAM Rev.*, vol. 46, pp. 443–454, 2004.
[29] L. Greengard and J. Strain, "The fast Gauss transform," *SIAM J. Sci. Stat. Comput.*, vol. 12, pp. 79–94, 1991.
[30] L. Greengard, D. Gueyffier, P.-G. Martinsson, and V. Rokhlin, "Fast direct solvers for integral equations in complex three-dimensional domains," *Acta Numerica*, vol. 18, no. 1, pp. 243–275, 2009.
[31] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *J. Comput. Phys.*, vol. 73, no. 2, pp. 325–348, 1987.
[32] M. Gu and S. Eisenstat, "Efficient algorithms for computing a strong rank-revealing QR factorization," *SIAM J. Sci. Comput.*, vol. 17, no. 4, pp. 848–869, 1996.
[33] W. Hackbusch and S. Börm, "Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices," *Computing*, vol. 69, no. 1, pp. 1–35, 2002.
[34] W. Hackbusch, "A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices," *Computing*, vol. 62, no. 2, pp. 89–108, 1999.
[35] W. Hackbusch and B. N. Khoromskij, "A sparse $\mathcal{H}$-matrix arithmetic," *Computing*, vol. 64, no. 1, pp. 21–47, 2000.
[36] W. Hager, "Updating the inverse of a matrix," *SIAM Rev.*, vol. 31, pp. 221–239, 1989.
[37] J. Hartikainen and S. Särkkä, "Kalman filtering and smoothing solutions to temporal Gaussian process regression models," in *Proc. IEEE Int. Workshop Mach. Learn. Signal Process.*, 2010, pp. 379–384.
[38] J. Hensman, N. Fusi, and N. D. Lawrence, "Gaussian processes for big data," in *Proc. 29th Conf. Uncertainty Artif. Intell.*, 2013, pp. 282–290.
[39] K. L. Ho and L. Greengard, "A fast direct solver for structured linear systems by recursive skeletonization," *SIAM J. Sci. Comput.*, vol. 34, no. 5, pp. 2507–2532, 2012.
[40] J. Y. Li, S. Ambikasaran, E. Darve, and P. K. Kitandis, "A Kalman filter powered by $\mathcal{H}^2$-matrices for quasi-continuous data assimilation problems," *Water Resour. Res.*, vol. 50, pp. 3734–3749, 2014.
[41] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert, "Randomized algorithms for the low-rank approximation of matrices," *Proc. Nat. Acad. Sci. USA*, vol. 104, no. 51, p. 20167, 2007.
[42] D. J. MacKay, "Introduction to Gaussian processes," *NATO ASI Ser. F Comput. Syst. Sci.*, vol. 168, pp. 133–166, 1998.
[43] P.-G. Martinsson and V. Rokhlin, "A fast direct solver for boundary integral equations in two dimensions," *J. Comput. Phys.*, vol. 205, no. 1, pp. 1–23, 2005.
[44] L. Miranian and M. Gu, "Strong rank revealing LU factorizations," *Linear Algebra Its Appl.*, vol. 367, pp. 1–16, 2003.
[45] J. Q. Candela and C. E. Rasmussen, "A unifying view of sparse approximate Gaussian process regression," *J. Mach. Learn. Res.*, vol. 6, pp. 1939–1959, 2005.
[46] F. W. J. Olver, D. W. Lozier, R. F. Boisvert, and C. W. Clark, Eds., *NIST Handbook of Mathematical Functions*. New York, NY, USA: Cambridge Univ. Press, 2010.
[47] R. K. Pace and J. P. LeSage, "Chebyshev approximation of log-determinants of spatial weight matrices," *Comput. Stat. Data Anal.*, vol. 45, pp. 179–196, 2004.
[48] C.-T. Pan, "On the existence and computation of rank-revealing LU factorizations," *Linear. Algebra Its Appl.*, vol. 316, no. 1, pp. 199–222, 2000.
[49] C. E. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2006.
[50] S. Rjasanow, "Adaptive cross approximation of dense matrices," presented at the Int. Assoc. Boundary Element Methods, Austin, TX, USA, 2002.
[51] S. Särkkä, A. Solin, and J. Hartikainen, "Spatio-temporal learning via infinite-dimensional Bayesian filtering and smoothing," *IEEE Signal Process. Mag.*, vol. 30, no. 4, pp. 51–61, Jul. 2013.
[52] Y. Shen, A. Ng, and M. Seeger, "Fast Gaussian process regression using KD-trees," in *Proc. Adv. Neural Inf. Process. Syst.*, 2006, pp. 1225–1232.
[53] J. Sherman and W. J. Morrison, "Adjustment of an inverse matrix corresponding to a change in one element of a given matrix," *Ann. Math. Stat.*, vol. 21, pp. 124–127, 1950.
[54] A. J. Smola and P. L. Bartlett, "Sparse greedy Gaussian process regression," in *Proc. Adv. Neural Inf. Process. Syst.*, 2001, pp. 619–625.
[55] E. Snelson and Z. Ghahramani, "Local and global sparse Gaussian process approximations," in *Proc. Artif. Intell. Statist.*, 2007, vol. 11, pp. 524–531.
[56] E. L. Snelson, "Flexible and efficient Gaussian process models for machine learning," Ph.D. dissertation, Gatsby Comput. Neurosci. Unit, Univ. College London, London, U.K., 2007.
[57] A. Solin and S. Särkkä, "Infinite-dimensional Bayesian filtering for detection of quasi-periodic phenomena in spatio-temporal data," *Phys. Rev. E*, vol. 88, no. 5, p. 052909, 2013.
[58] A. Townsend and L. N. Trefethen, "Continuous analogues of matrix factorizations," *Proc. Roy. Soc. A*, vol. 471, p. 20140585, 2015.
[59] L. N. Trefethen, *Approximation Theory and Approximation Practice*. Philadelphia, PA, USA: SIAM, 2013.
[60] G. N. Watson, *A Treatise on the Theory of Bessel Functions*. New York, NY, USA: Cambridge Univ. Press, 1995.
[61] M. A. Woodbury, "Inverting modified matrices," Statist. Res. Group, Princeton Univ., Princeton, NJ, USA, Tech. Rep. 42, 1950.
[62] F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert, "A fast randomized algorithm for the approximation of matrices," *Appl. Comput. Harmonic Anal.*, vol. 25, no. 3, pp. 335–366, 2008.
[63] C. Yang, R. Duraiswami, and L. Davis, "Efficient kernel machines using the improved fast Gauss transform," in *Proc. Adv. Neural Inf. Process. Syst.*, 2005, pp. 1561–1568.
[64] L. Ying, G. Biros, and D. Zorin, "A kernel-independent adaptive fast multipole algorithm in two and three dimensions," *J. Comput. Phys.*, vol. 196, no. 2, pp. 591–626, 2004.
[65] L. Ying, "Fast algorithms for boundary integral equations," in *Multiscale Modeling and Simulation in Science*. New York, NY, USA: Springer, 2009, pp. 139–193.
[66] K. Zhao, M. N. Vouvakis, and J.-F. Lee, "The adaptive cross approximation algorithm for accelerated method of moments computations of EMC problems," *IEEE Trans. Electromagn. Compat.*, vol. 47, no. 4, pp. 763–773, Nov. 2005.

Sivaram Ambikasaran received the bachelor's and master's degree in aerospace engineering from the Indian Institute of Technology Madras in 2007. Thereafter, he received the master's degree in statistics, the master's & PhD degrees in computational mathematics from Stanford University in 2013, where he worked on fast direct solvers for large dense matrices. He is currently a Courant instructor at New York University. His research focuses on designing efficient, fast, scalable algorithms for mathematical problems arising out of physical applications.

Daniel Foreman-Mackey received the BSc degree in physics from McGill University in 2008, the MSc degree in physics from Queen's University, Canada, in 2010, and the PhD degree in physics from New York University in 2015. He will start as a Carl Sagan postdoctoral fellow at the University of Washington in the summer of 2015. His research is focused on comprehensive probabilistic data analysis projects in astrophysics, and specifically the field of exoplanets.

Leslie Greengard received the BA degree in mathematics from Wesleyan University in 1979, the PhD degree in computer science from Yale University in 1987, and the MD degree from Yale University in 1987. From 1987 to 1989, he was a US National Science Foundation (NSF) Postdoctoral fellow at Yale University and at the Courant Institute of Mathematical Sciences, NYU, where he has been a faculty member since 1989. He was the director of the Courant Institute from 2006 to 2011 and is presently a director of the Simons Center for Data Analysis at the Simons Foundation. His research interests include fast algorithms, acoustics, electromagnetics, elasticity, heat transfer, fluid dynamics, computational biology, and medical imaging. He is a member of the National Academy of Sciences and the National Academy of Engineering. He is a member of the IEEE.

David W. Hogg received the SB degree from the Massachusetts Institute of Technology and the PhD degree in physics from the California Institute of Technology. His research has touched on large-scale structure in the Universe, the structure and formation of the Milky Way and other galaxies, and extra-solar planets. He currently works on engineering and data analysis aspects of large astronomical projects.

Michael O'Neil received the AB degree in mathematics from Cornell University in 2003, and the PhD degree in applied mathematics from Yale University in 2007. He was a postdoctoral fellow at the Courant Institute of Mathematical Sciences, NYU from 2010-2014. Presently, he is an assistant professor of mathematics at the Courant Institute, NYU and the Polytechnic School of Engineering, NYU. His research focuses on problems in computational electromagnetics, acoustics, and magnetohydrodynamics, integral equations, fast analysis-based algorithms, and computational statistics. He is a member of the IEEE.