Scaling Multidimensional Inference for Structured Gaussian Processes

Elad Gilboa, *Student Member, IEEE* , Yunus Saatçi, and John P. Cunningham

Abstract—Exact Gaussian process (GP) regression has $\mathcal{O}(N^3)$ runtime for data size N, making it intractable for large N. Many algorithms for improving GP scaling approximate the covariance with lower rank matrices. Other work has exploited structure inherent in particular covariance functions, including GPs with implied Markov structure, and inputs on a lattice (both enable $\mathcal{O}(N)$ or $\mathcal{O}(N \log N)$ runtime). However, these GP advances have not been well extended to the multidimensional input setting, despite the preponderance of multidimensional applications. This paper introduces and tests three novel extensions of structured GPs to multidimensional inputs, for models with additive and multiplicative kernels. First we present a new method for inference in additive GPs, showing a novel connection between the classic backfitting method and the Bayesian framework. We extend this model using two advances: a variant of projection pursuit regression, and a Laplace approximation for non-Gaussian observations. Lastly, for multiplicative kernel structure, we present a novel method for GPs with inputs on a multidimensional grid. We illustrate the power of these three advances on several data sets, achieving performance equal to or very close to the naive GP at orders of magnitude less cost.

Index Terms—Gaussian processes, backfitting, projection-pursuit regression, Kronecker matrices

1 Introduction

Gaussian processes (GP) have become a popular tool for nonparametric Bayesian regression. Naive GP regression has $\mathcal{O}(N^3)$ runtime and $\mathcal{O}(N^2)$ memory complexity, where N is the number of observations. At ten thousand or more observations, this problem is for all practical purposes intractable, given current hardware.

A variety of approaches are suggested in the literature for improving the computational complexity of GP for large data sets. Some approximate the GP using simpler models on a lower dimensional subspace, e.g., kernel convolution [1], [2], moving averages [3], or fixed number of basis functions [4]. Other approaches enable fast computation by working in the spectral domain or using algorithms based on the fast Fourier transform (FFT) [5], [6], [7]. Though these methods confer great advantage in the univariate case, extensions to the multivariate case often require restrictive assumptions [7]. A significant amount of research has also gone into sparse approximations, including covariance tapering [8], [9], [10], conditional independence to inducing inputs [11], [12], or a Gaussian Markov random field approximation [13]. However, the results of these algorithms depend strongly on the properties of the data [11], [13]. Since different assumptions fit different data sets, it is imperative to explore alternative avenues for attaining scalability.

The central aim of this paper is to introduce methods for *structured* GPs on multidimensional inputs. While efficient methods for structured GPs are known in the case of scalar inputs, many regression applications involve multivariate inputs. We will extend two common types of GP structures, namely: models with additive kernels and models with multiplicative kernels. These structures have a long history and are well used in GP literature [13], [14]. We present three novel advances which allow efficient and sometimes exact inference, or at least a superior runtime-accuracy tradeoff than existing methods. Our first advance extends the additive model using a variant of projection pursuit regression (PPGPR-Greedy), which significantly improves the expressivity of the model. Our second advance is an efficient method for multidimensional GPs with non-Gaussian likelihood (Additive-LA). Our third advance is an algorithm for GPs with a multiplicative kernel structure, which arises naturally, for most common kernels, when multidimensional inputs are on a lattice (GP-grid). We also extend this algorithm to cases of incomplete grids and non-i.i.d. noise.

- *E. Gilboa is with the Preston M. Green Department of Electrical and System Engineering, Washington University in St. Louis, 14049 Agusta Dr., Chesterfield, MO 63017. E-mail: gilboae@ese.wustl.edu.*
- *Y. Saatçi is with the Department of Engineering, University of Cambridge, 47 Consort Avenue, CB2 9AE, Cambridge CB2 9AE, Cambridgeshire, United Kingdom. E-mail: saatchi@cantab.net.*
- *J.P. Cunningham is with the Department of Statistics, Columbia University, Room 1026 SSW, MC 4690, 1255 Amsterdam Ave, New York, NY 10027. E-mail: jpc2181@columbia.edu.*

Manuscript received 15 Sept. 2012; revised 15 Mar. 2013; accepted 10 Sept. 2013. Date of publication 30 Sept. 2013; date of current version 14 Jan. 2015. Recommended for acceptance by R.P. Adams, E. Fox, E. Sudderth, and Y.W. Teh. 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.2013.192

1.1 Gaussian Process Regression

In brief, GP regression is a Bayesian method for nonparametric regression, where a prior distribution over continuous functions is specified via a Gaussian process (the use of GP in machine learning is well described in [12]). A GP is a distribution over a function $\mathbf{f}$ indexed by an input space X such that any finite selection of input locations $\mathbf{x}_1, \ldots, x_N \in X$ gives rise to a multivariate Gaussian density over the associated targets $\mathcal{N}(\mathbf{m}_N, \mathbf{K}_N)$, where $\mathbf{m}_N = m(\mathbf{x}_1, \ldots, \mathbf{x}_N)$ is the mean vector and $\mathbf{K}_N = \{k(\mathbf{x}_i, \mathbf{x}_j)\}_{i,j}$ is the covariance matrix, for mean function $m(\cdot)$ and covariance function $k(\cdot, \cdot)$. Here we are

specifically interested in the basic equations for GP regression, which involve two steps. First, for given data $\mathbf{y}$ (zero-mean data), we calculate the predictive mean and covariance at M unseen inputs as:

$$\boldsymbol{\mu}_* = \mathbf{K}_{MN}\big(\mathbf{K}_N + \sigma_n^2\mathbf{I}_N\big)^{-1}\mathbf{y}, \tag{1}$$

$$\boldsymbol{\Sigma}_* = \mathbf{K}_M - \mathbf{K}_{MN}\big(\mathbf{K}_N + \sigma_n^2\mathbf{I}_N\big)^{-1}\mathbf{K}_{NM}. \tag{2}$$

For model selection, since the function $k(\cdot,\cdot;\theta)$ is parameterized by hyperparameters such as amplitude and length-scale (which we group into θ), we must consider the marginal likelihood $Z(\theta)$:

$$\log Z(\theta) = -\frac{1}{2}\big[\mathbf{y}^\top\big(\mathbf{K}_N + \sigma_n^2\mathbf{I}_N\big)^{-1}\mathbf{y} + N\log(2\pi) + \log\big|\mathbf{K}_N + \sigma_n^2\mathbf{I}_N\big|\big]. \tag{3}$$

Here we use this log marginal likelihood to optimize over the hyperparameters in the usual way [12]. The runtime of GP regression and hyperparameter learning is $\mathcal{O}(N^3)$ due to $(\mathbf{K}_N + \sigma_n^2\mathbf{I}_N)^{-1}$, which is present in all equations.

2 Extending Structured GP on Multiple Input Dimensions

We will extend two common types of GP structures, namely models with additive kernel structure: $\mathbf{K} = \mathbf{K}_1 + \mathbf{K}_2 + \cdots + \mathbf{K}_D$; and models with multiplicative kernel structure: $\mathbf{K} = \mathbf{K}_1 \otimes \mathbf{K}_2 \otimes \cdots \otimes \mathbf{K}_D$. We first deal with the additive case (Section 2.1), extending it to an efficient and more expressive projected additive GP regression (Section 2.1.1), and secondly innovate to the case of non-Gaussian likelihoods (Section 2.1.2). Thirdly, we address the case of multiplicative kernel structure (Section 2.2).

2.1 GP with Additive Kernels

We begin by considering the simplifying (and overly restrictive) assumption of creating a multidimensional GP from additive single-dimensional GPs. Additive GP regression can be described using the following generative model:

$$\begin{aligned} y_i &= \sum_{d=1}^{D}\mathbf{f}_d(\mathbf{X}_{i,d}) + \epsilon \qquad i = 1,\ldots,N, \\ \mathbf{f}_d(\cdot) &\sim \mathcal{GP}\big(\mathbf{0}, k_d(\mathbf{x}_d,\mathbf{x}_d';\theta_d)\big) \qquad d = 1,\ldots,D, \\ \epsilon &\sim \mathcal{N}(0,\sigma_n^2), \end{aligned}$$

where $\mathbf{X}_{i,d}$ is the dth component of input i, $k_d(\cdot,\cdot)$ is the kernel of the scalar GP along dimension d, θ_d represents the dimension-specific hyperparameters, $D \geq 1$ is the dimensionality of the input space, and σ_n^2 is the (global) noise hyperparameter [14].

Given an additive GP model, it is possible to efficiently solve it using a variation of the classical backfitting algorithm (Algorithm 1). In brief, the backfitting algorithm fits an additive model over a D-dimensional space with the same overall asymptotic complexity of a scalar input space [15], [16]. The backfitting technique is particularly useful for GPs because convergence to the exact posterior mean is guaranteed [17], and can be viewed as a GP extension of the original Bayesian backfitting method in [18]. For full details see our supporting work [19].

Algorithm 1: Efficient Computation of Additive GP Posterior Mean via Backfitting

inputs : Training data $\{\mathbf{X},\mathbf{y}\}$. Suitable covariance function. Hypers $\theta = \{\theta_1,\ldots,\theta_D,\sigma_n^2\}$.
outputs: Posterior training means: $\sum_{d=1}^{D}\boldsymbol{\mu}_d$, where $\boldsymbol{\mu}_d \equiv \mathbb{E}(\mathbf{f}_d|\mathbf{y},\mathbf{X},\theta_d,\sigma_n^2)$.

```
Zero-mean the targets y;
Initialize the μ_d (e.g., to 0);
while The change in μ_d is above a threshold do
    for d = 1,...,D do
        μ_d ← E(f_d | y − Σ_{j≠d} μ_j, x_d, θ_d, σ_n^2);   ▷ Use Gauss-Markov processes.
    end
end
```

While this approach to solve additive GPs is encouraging, we are still left with a series of unidimensional GP problems, each of size N. Fortunately, if we further assume some special structure (e.g., kernel corresponding to a state-space model, equispaced inputs, etc.), we can use efficient methods to reduce complexity to $\mathcal{O}(N)$. Here we use the well-studied structure of Gauss-Markov Processes. As a starting point, we briefly review the use of Gauss-Markov processes for efficient GP regression over scalar inputs.

2.1.1 Gauss-Markov Processes

Although Gauss-Markov processes are well studied, their use for exact and efficient GP regression is under-appreciated. A GP with a kernel corresponding to a state-space model can be viewed as a Gauss-Markov process, enabling linear runtime. Gauss-Markov processes can be viewed as the solution of an order-m linear, stationary stochastic differential equation (SDE), given by:

$$\frac{d^m f(x)}{dx^m} + a_{m-1}\frac{d^{m-1}f(x)}{dx^{m-1}} + \cdots + a_1\frac{df(x)}{dx} + a_0 f(x) = w(x), \tag{4}$$

where $w(x)$ is a zero-mean white noise Gaussian process over any scalar input (often time), and f is also a GP (see [20] for an introduction to SDEs). We can rewrite Eq. (4) as a vector Markov process:

$$\frac{d\mathbf{z}(x)}{dx} = \mathbf{A}\mathbf{z}(x) + \mathbf{L}w(x), \tag{5}$$

where

$$\mathbf{z}(x) = \left[f(x), \frac{df(x)}{dx}, \ldots, \frac{d^{m-1}f(x)}{dx^{m-1}}\right]^\top, \tag{6}$$

and where $\mathbf{L} = [0,0,\ldots,1]$, and $\mathbf{A}$ is the coefficient matrix. The Markov structure of Eq. (5) is the key enabler of efficiency gains.

Earlier work [19], [21], derived the SDEs corresponding to several commonly used covariance functions including the Matérn family, spline kernels, and a good approximation to

the exponentiated-quadratic kernel. Once the SDE is known, the Kalman filter and Rauch-Tung-Striebel smoothing (belief propagation) can be used to perform GP regression in $\mathcal{O}(N)$ time and memory, a noteworthy leap in efficiency.[1]

Although Algorithm 1 allows for an efficient calculation of the posterior mean, to calculate the posterior variances and learn hyperparameters we must investigate further. Under the additivity assumption, and using the Gauss-Markov properties, the latent variables $\mathbf{Z}$ consist of D Markov chains. The resulting model regresses a sum of D Gauss-Markov processes $\mathbf{Z}_i$ (which are independent a priori). However, the true posterior $p(\mathbf{Z}_1, \ldots, \mathbf{Z}_D | \mathbf{y}, \mathbf{X}, \theta)$ is hard to handle computationally because all variables $\mathbf{Z}_i$ are coupled in the posterior. Although everything is still Gaussian, we are no longer able to use the efficient state-space methods of the Gauss-Markov process, returning us to the original computational intractability at large N. Thus, we require an approximate inference technique such as variational Bayesian expectation maximization (VBEM) or Markov Chain Monte Carlo (MCMC) (e.g., [22]). Full details of both approximation methods are deferred to our supporting work [19]. We will use the results of these additive GP methods (Additive-VB and Additive-MCMC) for comparison to the more advanced projected additive GP regression, presented in the next section.

2.1.2 Efficient Projected Additive GP Regression

So far, we have shown how the assumption of additivity can be exploited to derive non-sparse GP regression algorithms which scale as $\mathcal{O}(N)$. These considerable efficiency gains can however decrease accuracy and predictive power versus a full unstructured GP, due to the limited expressivity of the simple additive model. To address this, we now demonstrate a relaxation of the additivity assumption *without* sacrificing the $\mathcal{O}(N)$ scaling, by considering *projection pursuit* GP regression (PPGPR), a novel fusion of the classical projection pursuit regression algorithm with GP regression. The graphical model illustrating this idea is given in Fig. 1. We refer to the following projected additive GP prior:

$$y_i = \sum_{p=1}^{P} \mathbf{f}_p(\boldsymbol{\phi}_{p,i})) + \epsilon, \tag{7}$$

$$\boldsymbol{\phi}_p = \mathbf{X}\mathbf{w}_p, \tag{8}$$

$$\mathbf{f}_p(\cdot) \sim \mathcal{GP}(\mathbf{0}, k_p(\boldsymbol{\phi}_p, \boldsymbol{\phi}_p'; \theta_p)), \tag{9}$$

$$\epsilon \sim \mathcal{N}\left(0, \sigma_n^2\right),$$

for $i = 1, \ldots, N$. Each of the linear projections $\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_P\} \in \mathbb{R}^D$ projects the D dimensional input space to a different scalar input space. Forming linear combinations of the inputs before feeding them into an additive GP model significantly enriches the flexibility of the functions supported by the prior above, including many terms which are formed by taking products of covariates, and thus can capture relationships where the covariates jointly affect the target variable [23], [24]. In fact, Eqs. (7-9) are identical to the standard neural network model where the nonlinear activation functions are modeled using GPs.

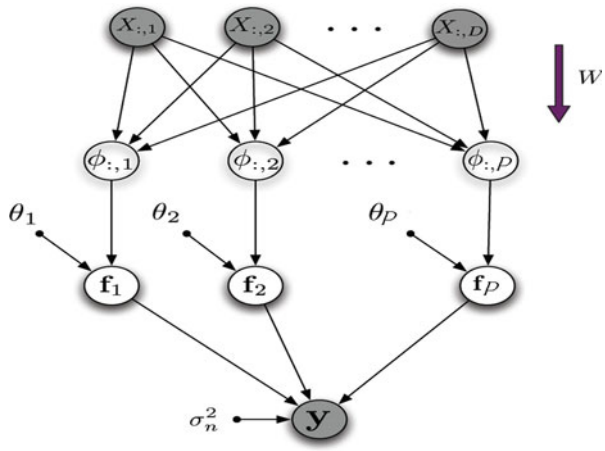

Fig. 1. Graphical model for Projected Additive GP Regression. In general, $P \neq D$. We present a greedy algorithm to select P, and jointly optimize $\mathbf{W}$ and $\{\theta_p\}_{p=1}^{P}$.

The one-dimensional projection ($P = 1$) is known in the literature as the Gaussian process single-index model (GP-SIM), and was recently extended to the case of multiple output regression [25]. For kernels that can be represented as state-space models, we use an EM algorithm, where the M-step involves optimizing the marginal likelihood with respect to $\mathbf{w}_1$ and θ. Notice that every step of parameter learning scales as $\mathcal{O}(N)$, since at every step we need to compute the marginal likelihood of a scalar GP (and its derivatives). These quantities are computed using the Kalman filter by differentiating the Kalman filtering process with respect to $\mathbf{w}_1$ and θ, as described in [17] (supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPAMI.2013.192).

The GP-SIM model is often used on the merits of simple inference and interpretable results [26], [27]; however better flexibility is gained by allowing for larger P. Extending P to larger values is unwieldy, as noted in [28], unless used with low complexity models (e.g., neural networks [28], or simple tree based approaches [29]). To deal with this, we use a simple greedy approach with a low number of hyperparameters (e.g., Matérn, squared exponential kernels), and limit the size of P.

The greedy algorithm is similar to classical projection pursuit regression [30]. In this case, at each iteration we find the optimal projection weights $\mathbf{w}_p$. The greedy nature of the algorithm allows the learning of the dimensionality of the feature space, P, rather naturally—we add feature dimensions until there is no significant change in performance over a validation set. In our implementation, we chose to initialize the weights $\mathbf{w}_p$ via linear regression of $\mathbf{X}$ onto the residual vector. Other initialization schemes, such as random initialization, will also work; however, these schemes typically require multiple runs to avoid convergence to poor local optima.

1. Note that the Gauss-Markov process framework requires sorted input points. Else, a preprocessing step of $\mathcal{O}(N \log N)$ is needed.

PPGPR is used for learning $\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_P\}$ and $\{\theta_1, \theta_2, \ldots, \theta_P\}$; for inference, the backfitting approach of Section 2.1 is used. The PPGPR algorithm offers a bridge between the flexibility of the naive Full-GP, and the efficiency of its approximate additive counterpart. For completeness, we also note that we implemented an MCMC alternative to this greedy approach for learning PPGPR, but we found it underperformed in all cases, and thus we do not report on it.

2.1.3 Generalized Additive GP Regression

Here we extend these efficient methods to non-Gaussian likelihood functions, as used in classification. Much work has been done to address these likelihoods via MCMC sampling or analytic approximations [31], but these works have only considered important scalability questions via sparsification approaches [32], [33], [34]. A natural and important question is how to extend our structured GP methods to this non-Gaussian case.

Here we derive an accurate and efficient $\mathcal{O}(N)$ algorithm using Laplace's approximation,[2] and we show that this method is a Bayesian analog of the classical local scoring technique [36].

Given the likelihood $p(\mathbf{y}|\mathbf{f})$ is non-Gaussian, we use the standard Laplace approximation:

$$p(\mathbf{f}|\mathbf{y}, X, \theta) \approxeq \mathcal{N}(\hat{\mathbf{f}}, \Lambda^{-1}), \tag{10}$$

where $\hat{\mathbf{f}} \equiv \arg\max_{\mathbf{f}} p(\mathbf{f}|\mathbf{y}, X, \theta)$ and the approximated covariance matrix $\Lambda \equiv -\nabla\nabla \log p(\mathbf{f}|\mathbf{y}, X, \theta)|_{\mathbf{f}=\hat{\mathbf{f}}}$. We define the following objective:

$$\Omega(\mathbf{f}) \equiv \log p(\mathbf{y}|\mathbf{f}) + \log p(\mathbf{f}|X, \theta). \tag{11}$$

$\hat{\mathbf{f}}$ is found by applying Newton's method (the likelihood is typically log-concave in $\mathbf{f}$, resulting in a convex program). If we assume that $\mathbf{f}$ is drawn from an additive GP, then it follows that the required gradient and Hessian (for Newton iterations) are:

$$\nabla\Omega(\mathbf{f}) = \nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f}) - \mathbf{K}_{\text{add}}^{-1}\mathbf{f}, \tag{12}$$

$$\nabla\nabla\Omega(\mathbf{f}) = \underbrace{\nabla\nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f})}_{\equiv -\mathbf{W}} - \mathbf{K}_{\text{add}}^{-1}. \tag{13}$$

This makes the Newton iteration:

$$\begin{aligned}\mathbf{f}^{(k+1)} &\leftarrow \mathbf{f}^{(k)} \\ &\quad + \left(\mathbf{K}_{\text{add}}^{-1} + \mathbf{W}\right)^{-1} \cdot \left(\nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f})|_{\mathbf{f}^{(k)}} - \mathbf{K}_{\text{add}}^{-1}\mathbf{f}^{(k)}\right) \\ &= \mathbf{K}_{\text{add}}(\mathbf{K}_{\text{add}} + \mathbf{W}^{-1})^{-1} \cdot \left[\mathbf{f}^{(k)} + \mathbf{W}^{-1}\nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f})|_{\mathbf{f}^{(k)}}\right],\end{aligned} \tag{14}$$

where the Woodbury matrix identity is used to get the final result. Note that Eq. (14) is precisely the form of a posterior inference (Eq. (1)) for target vector $[\mathbf{f}^{(k)} + \mathbf{W}^{-1}\nabla_{\mathbf{f}} \log p(\mathbf{y}|\mathbf{f})|_{\mathbf{f}^{(k)}}]$ and a diagonal noise term $\mathbf{W}^{-1}$, implying (i) that Algorithm 1 will solve each Newton iteration, and (ii) $\hat{\mathbf{f}}$ can be calculated in $\mathcal{O}(N)$ time. Wrapping backfitting iterations inside a global Newton iteration is precisely how the local-scoring algorithm is run to fit a generalized additive model [36].

The above calculates the posterior Laplace approximation. To efficiently approximate the marginal likelihood, we use the Taylor expansion of the objective function $\Omega(\mathbf{F})$ for $\mathbf{F} \equiv [\mathbf{f}_1; \ldots; \mathbf{f}_D]$ to obtain:

$$\log p(\mathbf{y}|\mathbf{X}) \approx \Omega(\hat{\mathbf{F}}) - \frac{1}{2}\log\det\left(\tilde{\mathbf{W}} + \tilde{\mathbf{K}}^{-1}\right) + \frac{ND}{2}\log(2\pi) \tag{15}$$

$$\begin{aligned}&= \log p(\mathbf{y}|\hat{\mathbf{F}}) - \frac{1}{2}\hat{\mathbf{F}}^{\top}\tilde{\mathbf{K}}^{-1}\hat{\mathbf{F}} \\ &\quad - \frac{1}{2}\log\det\left(\tilde{\mathbf{K}} + \tilde{\mathbf{W}}^{-1}\right) - \frac{1}{2}\log\det(\tilde{\mathbf{W}}),\end{aligned} \tag{16}$$

where $\tilde{\mathbf{K}}$ is a block diagonal tiling of $\mathbf{K}_1, \mathbf{K}_2, \ldots, \mathbf{K}_D$, and $\tilde{\mathbf{W}}$ is a block diagonal tiling of the single (diagonal) $\mathbf{W}$ matrix. This problem separates blockwise into matrix inverse and log determinant problems over unidimensional GPs. Since each GP has Gauss-Markov process structure, the Kalman filter can be used to calculate these terms efficiently [19].

2.1.4 Parallelization of State-Space Models

The above algorithms can be parallelized to achieve even greater speed up. For a fixed set of hyperparameters, the values of evolution and emission matrices of the SDE for all locations are independent, and hence can be precalculated in parallel. This fact enables a very simple parallelization scheme across multiple threads. We will further discuss these gains in Section 3.1.1.

2.2 GP with Multiplicative Kernels

Another natural and common structure to consider is multiplicative kernels. A tensor product kernel has the form $\mathbf{K} = \mathbf{K}_1 \otimes \mathbf{K}_2 \otimes \cdots \otimes \mathbf{K}_D$. Unlike in the additive section, the one-dimensional kernels $\mathbf{K}_d$ will require no additional assumptions on the kernel structure itself; however, the multiplicative structure requires inputs to be on a multidimensional grid. This is commonly seen for regression problems, e.g., regular measurements at evenly spaced weather stations, or video captured by a CCD camera. Much work has gone to applications of scalar lattice data, such as Toeplitz and spectral approaches [5], [6], [37]. However, these methods are very restrictive when extended to multidimensional grid input [7]. Another approach that is used extensively in the fields of computer vision and image analysis is the Gaussian-Markov random field (GMRF) model; however, this model often proves too simplistic for real data, and is known to be inconsistent over different subsets of lattice data [38]. Other work exploits separability with the Kronecker product [39], but this approach is rarely used in practice because it requires the restrictive assumptions of noiseless, full (regular) grid measurements. Here we present a novel extension of the Kronecker method to perform exact inference in $\mathcal{O}(DN^{\frac{D+1}{D}})$ time for cases of incomplete grids, missing observations, and variable noise.

A covariance function is a tensor product kernel if it computes covariances which can be written as a separable

2. Though literature often prefers the EP approximation [35], the computational properties of the Laplace are far more attractive and, we have found, do not involve a large loss in performance.

product over dimensions $d = 1, \ldots, D$. The assumption of tensor product kernel is quite general as most commonly-used kernels are of this form; for example, the squared exponential kernel factorizes as

$$k(\mathbf{x_i}, \mathbf{x_j}) = \sigma_f^2 \exp\left\{ -\sum_d \frac{1}{2\ell_d^2} \left(x_i^{(d)} - x_j^{(d)}\right)^2 \right\} = \sigma_f^2 \prod_{d=1}^{D} \exp\left\{ -\frac{1}{2\ell_d^2} (x_{i,d} - x_{j,d})^2 \right\}. \tag{17}$$

When the inputs lie on a grid, the kernel decomposes to a Kronecker product over the kernels of the individual dimensions [19], [39], [40]. To exploit this structure, we first describe matrix-vector multiplication across a Kronecker product, which is an $\mathcal{O}(N^2)$ operation using standard matrix-vector multiplication. However, we will show that with problems of this form, it is possible to attain close to linear runtime using tensor algebra, which will later be the key component in our GP-grid method. Using the Kronecker property $(\mathbf{B} \otimes \mathbf{C})\text{vec}(\mathbf{X}) = \text{vec}(\mathbf{C}\mathbf{X}\mathbf{B}^\top)$, we have

$$\left(\bigotimes_{d=1}^{D} \mathbf{A}_d \right) \mathbf{b} = \text{vec}\left(\mathbf{A}_D \mathbf{B} \left(\bigotimes_{d=1}^{D-1} \mathbf{A}_d \right)^\top \right), \tag{18}$$

where $\mathbf{B}$ is a matrix with dimensions $N^{1/D} \times N^{\frac{D-1}{D}}$,[3] and $\mathbf{b} = \text{vec}(\mathbf{B})$. We can write Eq. (18) as

$$\text{vec}\left(\left(\text{vec}^{-1}\left(\text{vec}\left(\left(\bigotimes_{d=1}^{D-1} \mathbf{A}_d \right) (\mathbf{A}_D \mathbf{B})^\top \right) \right) \right)^\top \right), \tag{19}$$

where we define the operator $\text{vec}^{-1}(\cdot)$ as $\text{vec}^{-1}(\text{vec}(\mathbf{A})) = \mathbf{A}$. The inner component of Eq. (19) can be written as

$$\text{vec}\left(\left(\bigotimes_{d=1}^{D-1} \mathbf{A}_d \right) (\mathbf{A}_D \mathbf{B})^\top \mathbf{I}_{N^{\frac{1}{D}}} \right) = \mathbf{I}_{N^{\frac{1}{D}}} \otimes \left(\bigotimes_{d=1}^{D-1} \mathbf{A}_d \right) \text{vec}((\mathbf{A}_D \mathbf{B})^\top). \tag{20}$$

Notice that Eq. (20) is in the same form as Eq. (18). By repeating Eqs. (19)-(20) over all D dimensions, and noting that $(\bigotimes_{d=1}^{D} \mathbf{I}_{N^{\frac{1}{D}}})\mathbf{b} = \mathbf{b}$, we see that the original matrix-vector product can be written as

$$\left(\bigotimes_{d=1}^{D} \mathbf{A}_d \right) \mathbf{b} = \text{vec}\big([\mathbf{A}_1 \cdots [\mathbf{A}_{D-1}[\mathbf{A}_D \mathbf{B}]^\top]^\top]^\top\big), \tag{21}$$

where the bracket notation denotes transpose and reshape, i.e., $[\mathbf{AB}]^\top = \text{reshape}((\mathbf{AB})^\top)$. Iteratively solving Eq. (21) requires $\mathcal{O}(DN^{\frac{D+1}{D}})$, because each of D ($D \ll N$) matrix-matrix multiplications requires $\mathcal{O}(N^{\frac{D+1}{D}})$ operations, which is much smaller than the original $\mathcal{O}(N^2)$. As an interesting side note, the product in Eq. (21) can be viewed as a contraction operation in tensor algebra [19]. Per standard practice in large scale optimization, we never actually write down (or store in memory) the large kernel matrix (which will cost $\mathcal{O}(N^2)$ in time and memory complexity); instead we treat it as linear operator acting on its vector arguments. Algorithm 2 gives pseudo-code illustrating Eq. (21).

Algorithm 2: kron_mvprod

▷ Efficient matrix-vector multiply for Kronecker matrices
inputs : D matrices $[\mathbf{A}_1 \ldots \mathbf{A}_D]$, length-$N$ vector $\mathbf{b}$
outputs: $\boldsymbol{\alpha}$, where $\boldsymbol{\alpha} = \left(\bigotimes_{d=1}^{D} \mathbf{A}_d\right)\mathbf{b}$

```
x ← b;
for d ← D to 1 do
    G_d ← size(A_d);
    X ← reshape(x, G_d, N/G_d);
    Z ← A_d X      ▷ Matrix-tensor product
    Z ← Z^⊤        ▷ Tensor rotation
    x ← vec(Z);
end
α ← x;
```

2.2.1 GP-Grid with Spherical Noise

The critical second step is to note that our inversion of interest $(\mathbf{K} + \sigma_n^2 \mathbf{I}_N)^{-1}$ is not a Kronecker product, due to the perturbation on the main diagonal. Nevertheless, it is possible to sidestep this problem using the eigendecomposition:

$$\left(\mathbf{K} + \sigma_n^2 I_N\right)^{-1} \mathbf{y} = \mathbf{Q}\left(\Lambda + \sigma_n^2 I_N\right)^{-1} \mathbf{Q}^\top \mathbf{y}. \tag{22}$$

Importantly, the eigenvector matrix $\mathbf{Q}$ will also be a Kronecker product made up of the eigenbases of each Kronecker block $\mathbf{K}_d$ such that $\mathbf{K} = \bigotimes_{d=1}^{D} \mathbf{K}_d$. To efficiently solve Eq. (22), we first perform eigendecompositions of covariances along the individual dimensions to get $[\mathbf{Q}_d, \Lambda_d]$. Next, we calculate Eq. (22) in three steps:

$$\boldsymbol{\alpha} \leftarrow \texttt{kron\_mvprod}\left(\left[\mathbf{Q}_1^\top, \ldots, \mathbf{Q}_D^\top\right], \mathbf{y}\right), \tag{23}$$

$$\boldsymbol{\alpha} \leftarrow \left(\Lambda + \sigma_n^2 I_N\right)^{-1} \alpha, \tag{24}$$

$$\boldsymbol{\alpha} \leftarrow \texttt{kron\_mvprod}([\mathbf{Q}_1, \ldots, \mathbf{Q}_D], \alpha), \tag{25}$$

where we efficiently used `kron_mvprod` twice and note that the matrix $\Lambda + \sigma_n^2 I_N$ is easy to invert as it is diagonal.

In summary, we exploited two important realizations: both the eigendecomposition and matrix-vector product can be done efficiently using properties of Kronecker product. We call this new algorithm "GP-grid", which allows for exact GP inference at orders of magnitude lower cost than the naive GP technique.

2.2.2 Generalizing GP-Grid for an Incomplete Grid and Variable Noise

The above assumptions of (i) a full grid and (ii) spherical noise are too simplistic for many applications, and can cause GP-grid to substantially underfit (this claim will be substantiated later in the results). First, examples of incomplete grids can often occur due to either missing values (e.g., malfunctioning sensors), or when a region of interest has an irregular shape (e.g., a segment of an image). Second, the constant noise assumption is also not valid in many real

3. For clarity of the derivation we assume the dimensions are all equal; however, more generally the dimensionality is $G_D \times \prod_{d=1}^{D-1} G_d$, where G_d is the number of elements in dimension d, and which presents no change to the algorithmic complexity.

systems as sensor noise can vary between sensors or can be signal dependent. Other work considers an additional GP to infer the noise model [25], [41] or uses gradient based maximum likelihood [24]. These approaches are general but computationally burdensome and have the potential for overfitting. In [40] the authors deal with missing observations by removing their locations from the covariance matrix and sampling to calculate the posterior. On the other hand, here we extend GP-grid to exactly and efficiently handle both incomplete grids and variable noise.

We handle both of these enhancements by considering non-spherical noise, first completing the grid with the introduction of dummy observations in the missing locations. The GP formulation allows us to introduce these variables without corrupting the GP inference. We view these incomplete data as measured data with high noise $\mathbf{y}_{\text{dummy}} \sim \mathcal{N}(\mathbf{f}, \epsilon^{-1}\mathbf{I}_w)$, where $\epsilon \to 0$, and w is the number of dummy variables. Using Eqs. (1) and (2), and reordering the observation vector so that the w dummy variables are at the end of the observation vector $\mathbf{y} = [\mathbf{y}_n, \mathbf{y}_w]^T$, $(N = n + w)$, we get

$$\boldsymbol{\mu}_* = \mathbf{K}_{MN}(\mathbf{K}_N + \mathbf{D})^{-1}\mathbf{y}, \tag{26}$$

$$\Sigma_* = \mathbf{K}_M - \mathbf{K}_{MN}(\mathbf{K}_N + \mathbf{D})^{-1}\mathbf{K}_{NM}. \tag{27}$$

Notice that the $\mathbf{D}$ matrix is no longer $\sigma_n^2\mathbf{I}$, but

$$\mathbf{D} = \begin{bmatrix} \mathbf{V} & 0 \\ 0 & \epsilon^{-1}\mathbf{I}_w \end{bmatrix}, \tag{28}$$

where $\mathbf{V}$ is a diagonal matrix corresponding to the noise of the true observations, $[\mathbf{V}]_{ii} = \sigma_i^2$. The introduction here of $[\mathbf{V}]_{ii} = \sigma_i^2$ (as opposed to a multiple of the identity) allows us to address the desire for variable noise with this same framework.

First we show that the introduction of the dummy variables does not affect the results of the GP analysis. The analysis will pertain to Eq. (26); however, the same approach can be used for Eq. (27). Rewriting Eq. (26), where we reordered the dummy observation to be at the end, we get

$$\boldsymbol{\mu}_* = \begin{bmatrix} \mathbf{K}_{Mn} \\ \mathbf{K}_{Mw} \end{bmatrix} \begin{bmatrix} \mathbf{K}_n + \mathbf{V} & \mathbf{K}_{nw} \\ \mathbf{K}_{nw}^\top & \mathbf{K}_w + \epsilon^{-1}\mathbf{I}_w \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{y}_n \\ \mathbf{y}_w \end{bmatrix}. \tag{29}$$

Using the block matrix inversion lemma for inverting a matrix $[AB; CE]$, we get

$$\begin{bmatrix} (A - BE^{-1}C)^{-1} & -A^{-1}B(E - CA^{-1}B)^{-1} \\ -E^{-1}C(A - BE^{-1}C)^{-1} & (E - CA^{-1}B)^{-1} \end{bmatrix}, \tag{30}$$

where $A = \mathbf{K}_n + \mathbf{V}$, $B = \mathbf{K}_{nw}$, $C = \mathbf{K}_{nw}^\top$, and $E = \mathbf{K}_w + \epsilon^{-1}\mathbf{I}_w$. We can then take the limit by writing the terms in Eq. (30) as:

$$E^{-1} = \epsilon(\epsilon\mathbf{K}_w + \mathbf{I}_w)^{-1} \xrightarrow{\epsilon \to 0} \mathbf{0}, \tag{31}$$

$$(A - BE^{-1}C)^{-1} \xrightarrow{\epsilon \to 0} A^{-1}, \tag{32}$$

$$-E^{-1}C(A - BE^{-1}C)^{-1} \xrightarrow{\epsilon \to 0} \mathbf{0}, \tag{33}$$

$$(E - CA^{-1}B)^{-1} \xrightarrow{\epsilon \to 0} \mathbf{0}, \tag{34}$$

$$-A^{-1}B(E - CA^{-1}B)^{-1} \xrightarrow{\epsilon \to 0} \mathbf{0}. \tag{35}$$

Putting the terms together, we rewrite Eq. (29) as

$$\boldsymbol{\mu}_* = \mathbf{K}_{Mn}(\mathbf{K}_n + \mathbf{V})^{-1}\mathbf{y}_n, \tag{36}$$

which is the exact GP over the non-dummy variables. Thus we can introduce these dummy variables to enable the Kronecker method without corrupting the result. To efficiently solve Eq. (26), we will use the preconditioned conjugate gradient (PCG) method. We iteratively solve

$$\mathbf{C}^\top(\mathbf{K}_N + \mathbf{D})\mathbf{C}\mathbf{x} = \mathbf{C}^\top\mathbf{y}, \tag{37}$$

where the preconditioner matrix $\mathbf{C} = \mathbf{D}^{-\frac{1}{2}}$. Note that the preconditioner masks out the dummy locations, in addition to the usual standardization of CG.

The PCG algorithm can be efficiently calculated using the `kron_mvprod` algorithm (Algorithm 2). Using PCG, Eq. (37) can be computed in $\mathcal{O}(QN^{\frac{D+1}{D}})$, where Q is the number of PCG iterations, and in practice is usually trivially small compared to N. PCG also allows elimination of the memory burden.

We have thus dealt with the posterior, but in order to learn the hyperparameters we also need to efficiently calculate the $\log(\det(\mathbf{K}_N + \mathbf{D}))$ term in the marginal likelihood (Eq. (3)). Since the complexity of solving the logdet term is $\mathcal{O}(N^3)$, we instead approximate it as

$$\log(\det(\mathbf{K}_N + \mathbf{D})) \approx \log(\det(\mathbf{K}_N + \gamma(\mathbf{D})\mathbf{I}_N)), \tag{38}$$

where $\gamma(\mathbf{D})$ is a scalar function of the variable noise matrix $\mathbf{D}$. With this approximation, we can easily do the logdet and derivative calculations via the Kronecker eigendecomposition (as in Eq. (22)), reducing the complexity again to $\mathcal{O}(N^{\frac{D+1}{D}})$. For the approximation we chose to use the geometric mean of the elements in $\mathbf{D}$, formally $\gamma(\mathbf{D}) = (\prod_{i=1}^N \mathbf{D}_{ii})^{\frac{1}{N}}$. Approximating this variable noise with spherical noise represents destroying the eccentricity of that ellipsoid while preserving its volume. Intuitively, this approximation will be reasonable as long as the eccentricity of $\mathbf{D}$ is not aligned with $\mathbf{K}_N$ (since the addition represents a convolution of these two ellipsoids). Because the diagonal structure of $\mathbf{D}$ has axis aligned eccentricity, and because the eccentricity of $\mathbf{K}_N$ is always quite off-axis (kernel stationarity implies that the marginal in each axis is identical, namely the kernel variance σ_f^2), we should suffer a small approximation penalty for this logdet calculation when $\mathbf{D}$ has a reasonable range of values, as indeed we observe empirically. This empirical investigation also showed that this approximation preserved the robustness of the algorithm to initial conditions.

3 Results

Here we will compare the three methods for scalable GP inference, namely: PPGPR (Section 2.1.1), Additive-LA (Section 2.1.2), and GP-grid (Section 2.2.2) (all shown in blue), to a naive Full-GP (black), other GP approaches (reds), and other relevant machine learning methods (green). We will examine both the runtimes and accuracies of the methods, and perform an efficiency analysis.

3.1 Runtime Complexity

We begin by comparing runtime performance (in seconds), taking into account both learning and prediction.

3.1.1 PPGPR-Greedy

Here we will compare the runtime performance of PPGPR to other GP based methods for multidimensional regression. In each experiment, we used $M = 1{,}000$ points. If a particular algorithm has a stochastic component to it (e.g., if it involves MCMC) its performance will be averaged over 10 runs. Every experiment was composed of training (i.e., smoothing and hyperparameter learning given $\{\mathbf{X}, \mathbf{y}\}$) and testing phases.

We test the following algorithms (with the following names): the full naive GP regression implementation (Full-GP), additive models using VBEM inference (Additive-VB) and MCMC inference (Additive-MCMC), projected additive models using greedy projection pursuit of Section 2.1.1 (PPGPR-Greedy) and a variation using MCMC (PPGPR-MCMC). Finally, for the sparse GP regression method we used the sparse pseudo-input Gaussian process (SPGP) [42]. For SPGP, to be conservative, we did not learn the pseudo inputs (which can potentially greatly increase the algorithm complexity and runtime) but rather used a random subset of the inputs as the active set. For both the SPGP and the Full GP, we used the GPML Matlab Code version 3.1 [43]. Also note that, for Additive-VB and PPGPR-Greedy we have set the number of outer loop iterations (the number of VBEM iterations for the former, and the number of projections for the latter) to be at maximum 10 for all N. Increasing this number increased the cost with no meaningful change to accuracy, so this is a reasonable choice. All algorithms were run both as a single thread and using a parallel multicore, but since SPGP and Full GP do not offer efficient implementation of the parallel schemes, their results were the same for both cases.[4]

We used synthetic data generated by:

$$y_i = \sum_{d=1}^{D} \mathbf{f}_d(\mathbf{x}_{i,d}) + \epsilon \qquad i = 1, \ldots, N, \tag{39}$$

$$\begin{aligned} \mathbf{f}_d(\cdot) &\sim \mathcal{GP}(\mathbf{0}; k_d(\mathbf{x}_d, \mathbf{x}'_d; [1, 1])) \quad d = 1, \ldots, D, \\ \epsilon &\sim \mathcal{N}(0, 0.01), \end{aligned} \tag{40}$$

where $k_d(\mathbf{x}_D, \mathbf{x}'_d; [1, 1])$ is given by the Matérn(7/2) kernel with unit lengthscale and amplitude [12]. We used $D = 8$ dimensions, and collected runtimes for a set of values for N ranging from 1,000 to 50,000.

Fig. 2 illustrates the significant computational savings attained by exploiting the structure of the additive kernel, with the PPGPR method of Section 2.1.1 (shown in blue) having minimal runtime compared to other GP methods. As expected, the log-log slope of the Full-GP is close to three (2.52) due to its cubic complexity, and all the approximation algorithms have runtimes that scale linearly (0.97, 0.62, 1.01, 0.98, 0.97) with the input size. We can also see that parallel processing of the state-space model matrices offers further improvement in scaling. These results serve only as a rough estimate, because the performance can depend on the chosen algorithm parameters, such as: number of outer loop iterations in the Additive-VB, number of projections in PPGPR-Greedy, or number of samples in the MCMC methods. This runtime/accuracy consideration should be used when comparing the efficiency of the algorithms.

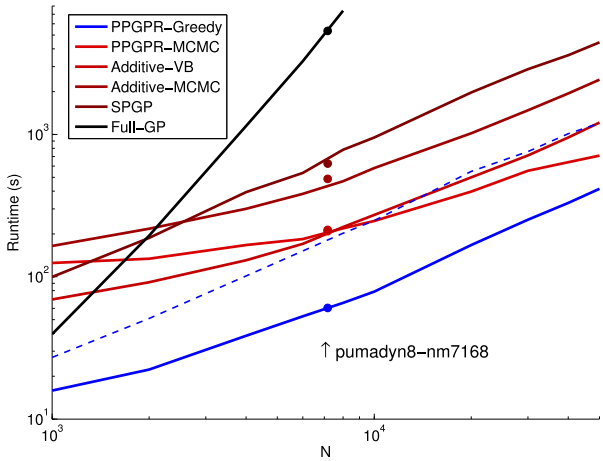

Fig. 2. A comparison of runtimes for efficient Bayesian additive GP regression, with $D = 8$. The algorithms ran on a Linux server, in a multicore parallel scheme using eight processors (solid lines). For comparison, we also added the runtime of PPGPR-Greedy using a single thread (dash lines). At $\mathrm{N} = 7{,}168$, we added an overlay of the runtime results for the pumadyn8-nm data set (described in Section 3.2.1), showing that this figure is representative of runtime in real data sets.

Additionally, runtime on a modern computer is by no means a perfect measure of algorithmic complexity. Nonetheless, we will see that the results of Fig. 2 agree with all the results from the real data sets. For example, in Fig. 2 we overlay the results of one of the real data sets, and one sees a close correspondence between synthetic and real data. Thus, these and subsequent results are highly representative and assert the primary point of this section: our main contribution—PPGPR (the greedy scheme in blue)—has roughly linear runtime, versus the cubic scaling of the naive Full-GP.

3.1.2 Additive-LA

Here we will compare the runtime performance of our Additive-Laplace method (Section 2.1.2) to other common methods for multidimensional binary classification. As in Section 3.1.1, we used $M = 1{,}000$ points and the Matérn(7/2) covariance function.

We used synthetic data generated by:

$$\begin{aligned} y_i &\sim \mathrm{Bernoulli}(p_i) \qquad i = 1, \ldots, N, \\ p(\cdot) &= g\Big(\sum_d \mathbf{f}_d(\cdot)\Big), \\ \mathbf{f}_d(\cdot) &\sim \mathcal{GP}(\mathbf{0}; k_d(\mathbf{x}_d, \mathbf{x}'_d; \theta_d)) \qquad d = 1, \ldots, D, \end{aligned} \tag{41}$$

where $g(\cdot)$ is the logistic link function.

Within the GP framework, we compared generalized additive GP Regression from Section 2.1.2 (Additive-LA), standard GP classification with Laplace's approximation (Full-GP) [12], the sparse GP methods of informative vector machine (IVM) [34], and fully independent conditional (FIC) [11]. For completeness, we also include support vector

4. When discussing parallel schemes we refer to only the learning stage. As in all GP frameworks, parallelism can always be used for prediction, since we are only interested in the predictive marginals per test point. However, this does not have any noticeable effect on the runtime and is thus unimportant to the comparison.

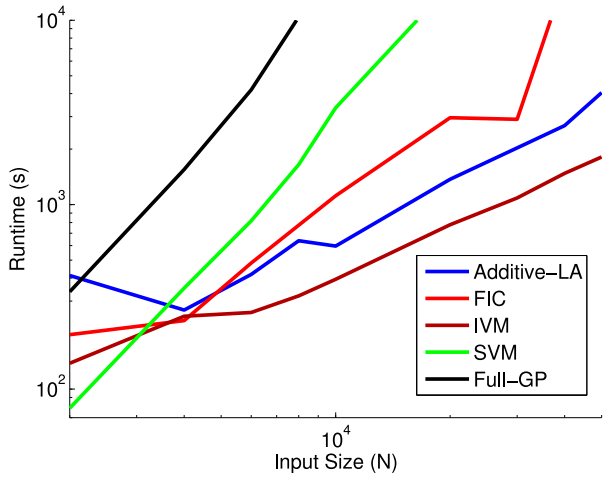

Fig. 3. This figure shows the runtime of the classification algorithms for the synthetic data set with $D = 8$. For the learning stage we used 50 iterations, and we did prediction on 1,000 points. The log-log slopes of the algorithms are: Full-GP $= 2.75$, Additive-LA $= 1.07$, FIC $= 1.53$, IVM $= 0.80$, SVM $= 2.16$.

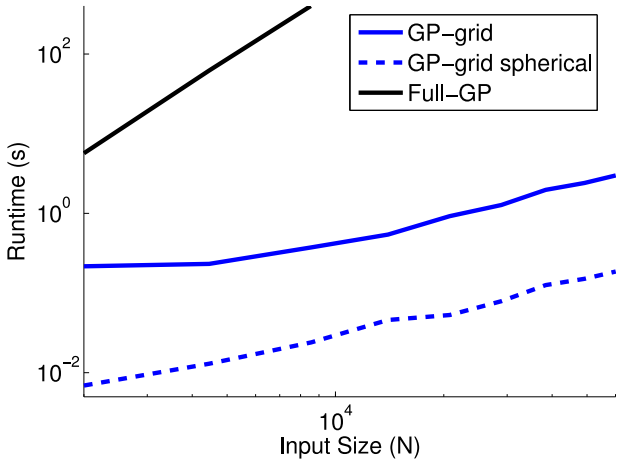

Fig. 4. Runtime complexity of naive Full-GP, GP-grid, and GP-grid spherical. The runtime illustrated is for a single calculation of the negative log marginal likelihood and its derivatives (dNLML). The ratio of input size to the complete grid size (n/N) is 0.7. The slope for the naive Full-GP is 2.9, for GP-grid is 1.1, and for GP-grid spherical is 1.0 (based on the last eight points). This empirically verifies the improvement in scaling.

machines (SVM) [44].[5] For the Full-GP we used GPML Matlab Code version 3.1 [43]; for FIC we used the GPstuff Matlab package [46]; for SVM we used LIBSVM [47]; and for IVM we used the implementation given in [34]. We tested the algorithms on the synthetic data from the model above using eight dimensions while varying the number of inputs $N = [2; 4; 6; 8; 10; 20; 30; 40; 50] \times 10^3$. We stopped running the Full-GP at $N = 10,000$ as it took too long to finish. A comparison of the runtime results is shown in Fig. 3. To be consistent, we used exactly 25 iterations for all algorithms during the learning stage. As can be seen from the figure, Additive-LA offers excellent scaling for large input sizes. The only algorithm that offers faster runtime than the Additive-LA is IVM. This can be expected as the IVM only uses the information in the active set. Our algorithm, on the other hand, makes use of all the data, and is thus able to achieve a more accurate estimation, as the results in Section 3.2.2 demonstrate.

3.1.3 GP-Grid

Here we will compare the runtime performance of the GP-grid method from Section 2.2.2 to both the naive Full-GP regression method, and GP-grid with spherical noise (GP-grid spherical) from Section 2.2.1. We conduct the comparison using a segment of real image data, where we mask out part of the picture. At each iteration the size of the window is increased, thereby increasing both the number of input locations n (pixels we did not mask out) and dummy locations w (masked pixels). For the naive Full-GP we consider only the input locations (without the dummy locations), but for the GP-grid algorithms we used the dummy locations to complete the grid, $N = n + w$. Fig. 4 illustrates the time complexity of the three algorithms as a function of input size n (pixels). The time complexity presented for all algorithms is for a single calculation of the negative log marginal likelihood (NLML) and its derivatives (dNLML), which are the needed calculations in GP learning (and which carry the complexity of the entire GP regression algorithm). Both the naive Full-GP and GP-grid spherical methods calculate the mentioned values for four parameters including a learned global noise parameter, the two lengthscales, and the signal variance. GP-grid learns just three parameters (but accounts for variable noise). In GP-grid, the noise model is:

$$\sigma_i^2 = 0.2495 I_i + 15.9858, \tag{42}$$

where at location i, σ_i^2 is the noise variance and I_i is the measured image intensity. This model was chosen as a sensible model of known camera properties [48]. In all cases we used the Matèrn(5/2) covariance function. As can be seen in Fig. 4, GP-grid scales just superlinearly with the input size, while Full-GP is cubic. Furthermore, the introduction of the variable noise and dummy variables does increase the computational time of the GP-grid compared to GP-grid spherical; however, it does not hurt the scalability of the algorithm (and we will show that it has significant performance implications that easily warrant this increase).

3.2 Performance

Next, we extend the comparison to real data sets, which will allow thorough accuracy comparisons.

3.2.1 PPGPR-Greedy

We use two standard performance measures on the test set: normalized mean square error (NMSE) and mean negative log probability (MNLP):

$$\text{NMSE} = \frac{\sum_{i=1}^{N_*} (\mathbf{y}_*(i) - \boldsymbol{\mu}_*(i))^2}{\sum_{i=1}^{N_*} (\mathbf{y}_*(i) - \overline{y})^2},$$

$$\text{MNLP} = \frac{1}{2N_*} \sum_{i=1}^{N_*} \left[\frac{(\mathbf{y}_*(i) - \boldsymbol{\mu}_*(i))^2}{\mathbf{v}_*(i)} + \log \mathbf{v}_*(i) + \log 2\pi \right],$$

where $\boldsymbol{\mu}_* \equiv E(\mathbf{f}_*|X, \mathbf{y}, X_*, \theta)$, $\mathbf{v}_* \equiv \mathbb{V}(\mathbf{f}_*|X, \mathbf{y}, X_*, \theta)$, and $\overline{y}$ is the training-set average target value. These measures have

5. To calculate the MNLL, we used the probabilistic predictions from the SVM using cross-validation and the cross-entropy metric [45].

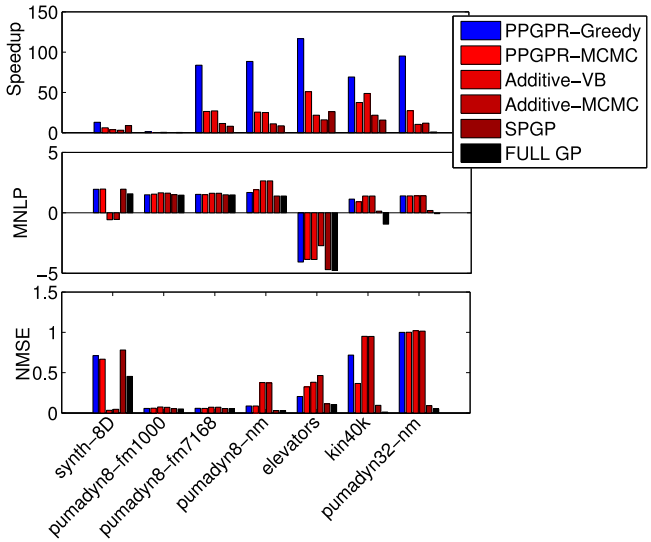

Fig. 5. These figures offer a comparison between the different GP methods discussed in the text, taking into account both speedup and accuracy. For comparison we used several known data sets from the literature and ran the algorithms on a multicore (eight-core) computer. The top figure illustrates the speedup of the approximation algorithms runtimes with respect to the Full-GP (exact inference) runtime. The bottom two figures show two metrics for calculating regression accuracy.

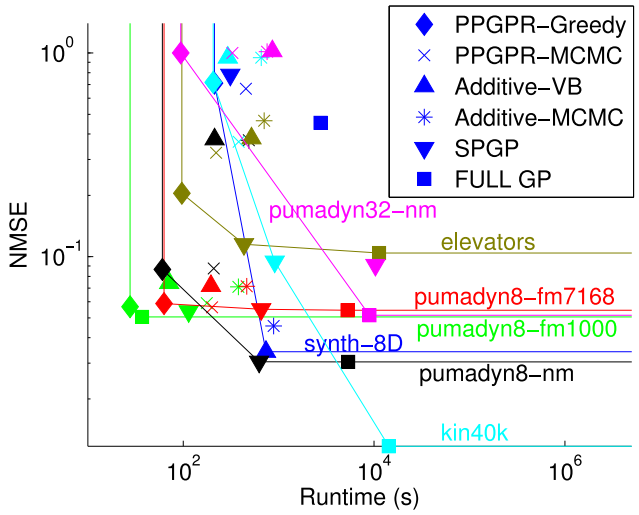

Fig. 6. The two fundamental desiderata of our algorithms are accuracy and speed. Here we plot error versus runtime to quantify the tradeoff between these two objectives using the notion of Pareto efficiency. Every algorithm is represented using a unique marker and with a color scheme chosen according to the data sets. For each data set, the Pareto efficient frontier is shown as a color line passing through the efficient algorithms for that data set.

been chosen to be consistent with those commonly used in the sparse GP regression literature.

We test seven well-known data sets: `synth-8D` ($N = 8{,}000$ synthetic data from Section 3.1.1). The `pumadyn` family is a robotic arm data set, and consists of three data sets: `pumadyn8-fm1000` ($N = 1{,}000$, fairly linear data with $D = 8$ dimensions), `pumadyn8-fm7168` ($N = 7{,}168$, fairly linear data with $D = 8$ dimensions), `pumadyn32-nm` ($N = 7{,}168$, highly nonlinear data with $D = 32$). `Elevators` data set consists of the current state of the f16 aircraft ($N = 8{,}752$, 17-dimensional) [49], and `kin40k` is a highly nonlinear data set ($N = 10{,}000$, eight-dimensional).[6] Fig. 5 demonstrates the central analysis of this section. In each subplot, we calculate speedup (as an inverse multiple of the Full-GP runtime), MNLP, and NMSE across all seven data sets and six algorithmic options. We compared the same methods as in Section 3.1.1. The top subplot in Fig. 5 indicates the substantial speedups offered by all algorithms over the full GP, with the exception only of the $N = 1{,}000$ data set (`pumadyn8-fm1000`; this is not surprising given small N). Further, our PPGPR-Greedy achieves the largest speedup across all data sets, and in most cases the error (MNLP and NMSE in the second and third subplots) is the same as competing methods. We also see that the simple additive models almost always underperform in accuracy, which is as expected given their limited expressivity; thus our PPGPR innovation of Section 2.1.2 is well warranted. The one exception where Additive-VB outperforms PPGPR-Greedy is the synthetic data set. However, this is expected as we used an additive model to generate data and the greedy nature of PPGPR-Greedy causes it to underperform. In the final two data sets, we see that SPGP and the full GP have better accuracy. This may be explained as both these data sets are highly nonlinear, making the additive assumption inaccurate.

PPGPR-Greedy achieves the best runtimes but at times with an accuracy cost. Thus we want to quantify the notion of a runtime-accuracy tradeoff. We plot all data sets and algorithms in a runtime versus error plot (Fig. 6), and we use the economics concept of Pareto efficiency: *efficient* points in the runtime versus error plot represent algorithms with minimum runtime for a given error rate. Pareto inefficient algorithms are then those points that are unambiguously inferior. The efficient frontier is the convex hull of all {runtime, error} points (algorithms) for a given data set. This will give us a clear picture of which algorithms are optimal choices across a range of data sets. Fig. 6 details this, with one efficient frontier for each data set (a given color). Each algorithm has a given marker type. This immediately shows what one would expect: the full GP implementation is typically most accurate, but only if one is willing to invest substantial runtime. Secondly, most often the PPGPR-Greedy is the other efficient choice for a substantially reduced runtime, albeit higher error.

Three algorithms stand out in their overall efficiency: PPGPR-Greedy (efficient in all seven data sets), SPGP (efficient in 4), and full GP (efficient in six data sets). Unsurprisingly, the additive model is typically inferior to the more expressive PPGPR model. The PPGPR-Greedy is the *only* efficient algorithm for *all* data sets as it achieves the fastest runtime. However, more interestingly, it also achieves very good accuracy results making most other algorithms inefficient. Of course, any trivial algorithm could achieve efficiency by having minimal runtime and arbitrary error, but the data demonstrates that this is not the case with our algorithms: the PPGPR-Greedy error in almost all data sets is competitive or better than all alternatives. As can be seen by the results of this section, PPGPR-Greedy is a bridge between the fast and overly simplified additive model and the slower more flexible Full-GP. Its efficiency to a wide

6. `Pumadyn` and the synthetically generated `Kin40k` data sets are from the DELVE archive. `Elevators` from KEEL archive.

TABLE 1
Performance Comparison of Efficient Additive GP Classification Algorithms with Commonly-Used Classification Techniques on Larger Data Sets

Algorithm	Error Rate	MNLL	Runtime (s)
Synthetic Additive Data ($N = 4000$, $M = 1000$, $D = 8$)			
Additive-LA	0.28	0.59	161
Full-GP	0.60	0.74	2244
FIC - 50	0.28	1.06	525
FIC - 400	0.45	1.36	850
IVM - 50	0.28	0.69	65.1
SVM	0.29	0.58	345
Magic Gamma Telescope ($N = 15216$, $M = 3804$, $D = 10$)			
Additive-LA	0.14	0.34	2345
Full-GP	NA	NA	NA
FIC - 50	0.14	0.36	3340
FIC - 1522	0.14	0.36	7331
IVM - 50	0.66	0.69	118
SVM	0.12	0.30	8070
IJCNN ($N = 49990$, $M = 91701$, $D = 13$)			
Additive-LA	0.05	0.16	14505
Full-GP	NA	NA	NA
FIC - 50	0.05	1.18	4390
FIC - 4999	0.08	0.82	16728
IVM - 50	0.09	0.69	369
SVM	0.02	0.05	22170

range of data sets stems mainly from its ability to capture more degrees of freedom and is considerably faster to train than SPGP. However, its performance will depend on the structure of the data set, and its accuracy will degrade for highly nonlinear data sets with high dimensionality (e.g., `kin40k` and `pumadyn32-nm`). Thus, we believe PPGPR-greedy to be most appropriate in cases of moderate nonlinearity and large data size.

3.2.2 Additive-LA

In this section we will compare the generalized additive-GP from Section 2.1.2 to other kernel classifiers (both Bayesian and non-Bayesian). We use common performance metrics from the sparse GP classification literature, enabling straightforward comparison with other experimental results. In this paper we will focus on the task of binary classification; however in principle, extensions to tasks such as multi-class classification and Poisson regression can be performed without affecting asymptotic complexity. For performance measures we use algorithm runtime (in seconds), test error rate, and mean negative log-likelihood (MNLL):

$$\text{Error Rate} = \frac{\#(\text{incorrect classifications})}{\#(\text{test cases})}, \tag{43}$$

$$\text{MNLL} = \frac{1}{N_*}\sum_{i=1}^{N_*}[y_i \log \hat{p}_i + (1 - y_i)\log(1 - \hat{p}_i)]. \tag{44}$$

For both the test error rate and MNLL measures lower values indicate better performance.

We tested the classification algorithms from the previous section on the synthetic data and on two additional popular data sets: `Magic Gamma Telescope` [49], and `IJCNN` [50].

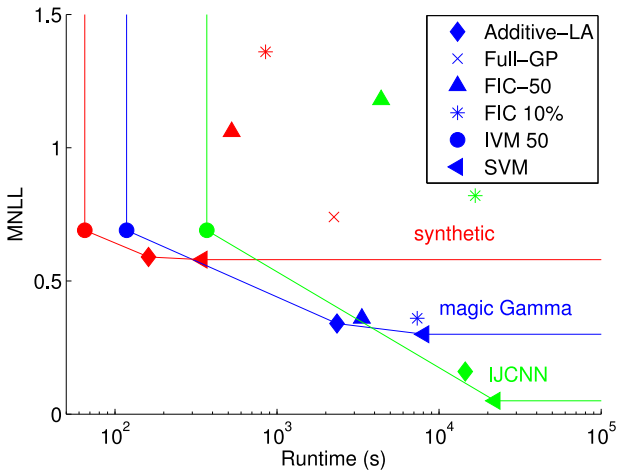

Fig. 7. As in Fig. 6, here we plot error (MNLL) versus runtime to quantify the tradeoff between these two objectives using Pareto efficiency.

We again only allowed 25 iterations in the learning stage. For sparse methods we tested two activeset sizes: 50, and $0.1N$. Table 1 summarizes the classification results across all algorithms and data sets. Each column gives the classification error rate, MNLL, and runtime.

Similar to Section 3.2.1, we quantify the notion of a runtime-accuracy tradeoff using the Pareto efficiency in Fig. 7. As opposed to the regression case, here the Full-GP is never efficient, while the Additive-LA is highly efficient. The three algorithms that stand out in their overall efficiency are: IVM-50 (efficient in all three data sets), Additive-LA (efficient in 2), and SVM (efficient in 3). The IVM has the best runtime performance across all data sets; however, it underperforms substantially compared to our Additive-LA method. Furthermore, while the SVM is often efficient, it carries significant runtime burden which may not justify the minor improvement in accuracy. These results show that the Additive method should be considered as a competitive balance of speed and accuracy. Though the results are not as compelling as in the regression case, the Additive-LA is perhaps the most viable choice when a Bayesian solution is needed.

3.2.3 GP-Grid

Application to image data. In this section we present the performance of GP-grid for real image data interpolation, and the improvement compared to commonly used image interpolation methods. As a reminder, GP-grid is an exact GP algorithm so the previous runtime-accuracy tradeoff comparison is not needed (since our method is always superior to Full-GP). Hence here we briefly present the application to images to show that it is a competitive method against other image processing methods. We use three novelties from Section 2.2.2 to test this method: the use of GP itself (enabled by GP-grid), the ability of GP to accept segmented data (see Eqs. (26–27), and the ability of GP to accept a known noise model (Eq. (42)).

For comparison, we used real images acquired by a CCD imaging array,[7] where over a thousand pictures of the same four scenes were taken. We manually segmented the images

7. Kodak KAI-4022 4-Mega pixel.

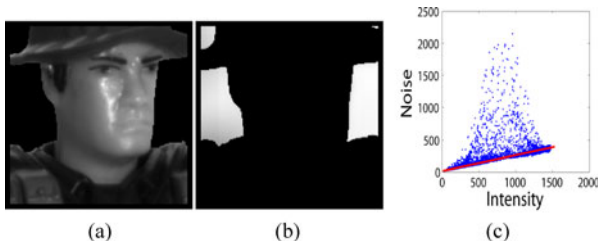

Fig. 8. An example of the face image separated to an object segment (Fig. 8a) and a background segment (Fig. 8b), which are used for interpolation comparison along with their empirical noise versus intensity model (Fig. 8c). The red line corresponds to the camera specific linear noise model in Eq. (42).

into two exclusive segments, one of the object and one of the background, an example of which is shown in Fig. 8. In all pictures the empirical noise model fit reasonably well to the camera-specific noise model used in Eq. (42) (line shown in red). To estimate the true image, we averaged over the majority of the pictures, leaving a small subset for testing. In order to test interpolation performance, we interpolated the entire image using only a subset of the image (down-sampled by 1/4, a factor of two in both the vertical and horizontal directions). All images are 200×200 pixels, hence, even their down-sampled version will be impractical for Full-GP. The interpolated images were then compared to their corresponding averaged images for accuracy analysis. For an accuracy criterion, we compared the standardized mean square error (SMSE) between the interpolated images and the average images, as defined in [12]. Note that in all the comparisons we intentionally changed the test conditions so they would be most favorable to non-GP methods: we discarded the pixels by the border pixels (five pixels width), and allowed non-GP methods access to the entire image. These choices are conservative as the non-GP methods fail particularly badly at the edges, and so we discarded those results to clarify that these improvements have nothing to do with the failure modes of other methods. We chose to compare GP-grid with the common interpolation algorithms: bilinear, bicubic, bicubic-spline (Bic-sp) and NEDI [51]. Although this is by no means an exhaustive comparison, it allows for a benchmark for comparison with GP performance.

We ran GP-grid using both the Matérn(1/2) and Matérn (5/2) covariance functions, and learned the hyperparameters: lengthscales (l_1, l_2), signal variance (σ_f^2), and noise variance (σ_n^2) [12]. For brevity, we will use GP-grid$(\cdot)$ for GP-grid Matérn$(\cdot)$, and we will add "sph" when we used GP-grid to learn the spherical noise variance hyperparameter σ_n^2 (Section 2.2.1, as opposed to Section 2.2.2). We tested the algorithms on images such as the one from Fig. 8 when taken as a whole (W), object segment (O), and background segment (B). As a reference, we also added GP-grid (1/2) spherical and GP-grid(5/2) spherical. As Table 2 shows, the GP-grid algorithm with the camera specific noise model improved performance in all images compared with GP-grid spherical and best overall interpolation results of all the algorithms tested. The improvement over GP-grid spherical is perhaps most evident in the moose object image, where both the GP-grid spherical algorithms severely underfit the results. These improvements may be in part due to the fact that the GP-grid is the only algorithm with knowledge of a noise model (Eq. (42)). The GP framework is a natural choice for enabling this noise model, and its use is critically enabled by our GP-grid method (which is the point of this section).

Application to temperature data. Finally, to show the extension of our GP-grid algorithm to higher dimensional data, we show a spatio-temporal example $(D = 3)$ of monthly land surface temperatures in North America.[8] Fig. 9 (left column) shows monthly temperature readings from 1950. We used temperature readings from nine months (excluding April, August, and December) as our training set, corresponding to $n = 24{,}939$ data points on an irregular grid. This data comprises 44 percent of the full $66 \times 71 \times 12$ grid. and is irregular both in time (held-out test data) and space (incomplete land coverage). Note that this data size is already well beyond the range of a Full-GP (c.f., Fig. 4), and the incomplete grid structure precludes the use of GP-grid spherical (Section 2.2.1). Hence, our GP-grid method (Section 2.2.2) is a critical enabler of this application. We chose the Matérn(5/2) kernel and learned the hyperparameters (including global noise σ_n). The GP-grid inference results are shown the second column for the held-out test set of April, August and December, with their corresponding 95 percent confidence intervals (two standard deviations of the posterior are plotted) in the third column. Note the higher posterior variance in December (which had only past data, not past and future as in April and August), indicating that this data has significant temporal structure in addition to its spatial structure. Our GP-grid required only 4.6 seconds for inference and 5 minutes for learning, in a data set of roughly 25 thousand points where other GP methods are intractable. The critical point of these results is that our computational advances enable GP to be applied in a new application domain where large data sets are the norm.

4 Discussion and Conclusion

Gaussian processes are perhaps the most popular nonparametric Bayesian method in machine learning, but their adoption across other fields—and notably in application domains—has been limited by their burdensome scaling properties. While important sparsification work has somewhat addressed this scalability issue, the problem is by no means closed. Here we focused on structured GP models, making nontrivial advances to existing state-space and lattice-input GP methods in order to extend structured GP techniques into the multidimensional input domain. Our results (Section 3) illustrate across a range of data and different algorithms that structured models are most often superior to the state of the art sparse methods (SPGP).

Our PPGPR-greedy algorithm (Section 2.1.1) combines the computational efficiency of additive GP models (Section 2.1) with the expressivity of a multidimensional coupled model. The result (Section 3.2.1) is an algorithm that has a superior runtime-accuracy tradeoff than competing algorithms. While its accuracy was often slightly lower than

8. Data from the Joint Institute for the study of Atmosphere and Ocean. (http://jisao.washington.edu/data/satmergedarctic).

TABLE 2
Comparison of Standardized MSE Interpolation Results for Images with Additive Variable Gaussian Noise

Alg	Sphere			Moose			Cone			Face		
	O	B	W	O	B	W	O	B	W	O	B	W
GP-grid(1/2)	0.39	0.40	0.69	1.08	0.35	**0.86**	0.56	0.39	1.74	1.28	0.88	2.67
GP-grid(5/2)	**0.33**	**0.15**	0.73	1.10	**0.08**	0.91	**0.52**	**0.17**	**1.24**	**1.10**	0.70	**1.98**
GP-grid(1/2) sph	0.40	0.30	0.71	2.98	0.11	1.54	0.84	0.19	3.81	1.30	0.69	3.02
GP-grid(5/2) sph	0.38	0.23	0.92	4.37	0.08	3.05	3.33	0.19	4.99	1.73	**0.64**	2.97
Blinear	0.56	0.56	**0.68**	1.12	0.56	0.97	0.64	0.61	1.63	1.27	1.06	2.60
Bicubic	0.59	0.61	0.83	**1.01**	0.57	0.91	0.69	0.67	2.05	1.32	1.03	2.69
Bic-sp	0.74	0.76	0.80	1.16	0.76	1.06	0.80	0.82	1.24	1.23	1.12	2.04
NEDI	0.56	0.53	0.74	1.11	0.52	0.97	0.72	0.59	1.97	1.61	1.21	3.86

We tested each image when taken as a whole (W), object segment (O), and background segment (B).

a naive Full-GP, the linear scaling properties of PPGPR mean that it can be efficiently used across a much broader range of data sizes and applications.

Of course, in some cases the researcher will prefer the SPGP method over PPGPR-Greedy. We see this as an inherent fact in approximation techniques: various methods will be more appropriate in different settings. Our results (Section 3) investigated this runtime-accuracy tradeoff, using both metrics on real data sets and meta-analyses of Pareto efficiency. These results thus enable the researcher to make an informed choice about a GP method for a given data size, data complexity, and available computational resource.

To the point of runtime-accuracy tradeoff, there are sometimes opportunities for great scaling advantages with no accuracy tradeoff whatsoever, as we demonstrated with the case of multiplicative kernel structure (Section 2.2.2). Notably, this GP-grid method opens up an entirely new set of applications for GP, such as image and video processing, or financial engineering applications such as implied volatility surfaces. Our future work is pursuing these application domains.

As a last computational point, as growth in computational speed is increasingly driven by parallelism, it will become increasingly important to use GP schemes that naturally incorporate parallel processing, to efficiently deal with the rapid growth of future data sets. Our PPGPR-Greedy method stands out in this regard versus both the naive full GP and SPGP, and the results of Section 3 reiterate this fact.

Understanding how our existing nonparametric models can scale and be used in real data, and how these models connect to other areas of statistics, will increase the utility of machine learning algorithms in general. This is perhaps most important with Gaussian processes, which promise a wide range of useful applications. The code is available at https://mloss.org/software/view/501/ and https://mloss.org/software/view/503/.

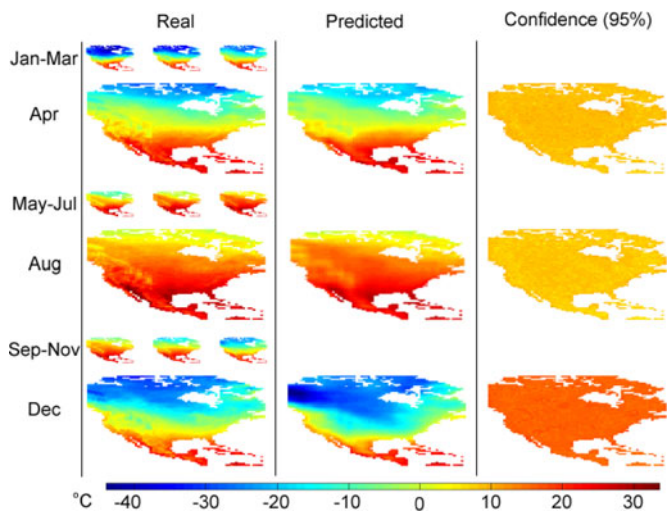

Fig. 9. Average monthly land surface temperatures in North America in 1950. The left column presents the real measurements. Small images (nine months) were used as a training set, and April, August, and December were used as a held-out test set. The middle and right columns show the corresponding GP-grid posterior mean and 95 percent confidence intervals for April, August, and December.

ACKNOWLEDGMENTS

Image data were from Shengkui Gao and Dr. Viktor Gruev at the advanced sensors research laboratory in Washington University in St. Louis. The authors thank Dr. Arye Nehorai for helpful conversations and support.

REFERENCES

[1] D. Higdon, "Space and Space–Time Modeling Using Process Convolutions," *Quantitative Methods for Current Environmental Issues*, p. 37, Springer-Verlag, 2002.
[2] G. Xia and A.E. Gelfand, "Stationary Process Approximation for the Analysis of Large Spatial Data Sets," technical report, Duke Univ., 2006.
[3] C.K. Winkle and N. Cressie, "A Dimension-Reduced Approach to Space-Time Kalman Filtering," *Biometrika*, vol. 86, pp. 815-829, 1999.
[4] N. Cressie and G. Johannesson, "Fixed Rank Kriging for Very Large Spatial Data Sets," *J. Royal Statistical Society: Series B (Statistical Methodology)*, vol. 70, no. 1, pp. 209-226, 2008.
[5] M. Fuentes, "Approximate Likelihood for Large Irregularly Spaced Spatial Data," *J. Am. Statistical Assoc.*, vol. 102, pp. 321-331, 2007.
[6] C. Paciorek, "Bayesian Smoothing with Gaussian Processes Using Fourier Basis Functions in the SpectralGP Package," *J. Statistical Software*, vol. 19, no. 2, 2007.
[7] J. Fritz, I. Neuweiler, and W. Nowak, "Application of FFT-Based Algorithms for Large-Scale Universal Kriging Problems," *Math. Geosciences*, vol. 41, pp. 509-533, 2009.
[8] A.J. Storkey, "Truncated Covariance Matrices and Toeplitz Methods in Gaussian Processes," *Proc. Ninth Int'l Conf. Artificial Neural Networks*, vol. 1, pp. 55-60, Sept. 1999.
[9] R. Furrer, M.G. Genton, and D. Nychka, "Covariance Tapering for Interpolation of Large Spatial Data Sets," *J. Computational and Graphical Statistics*, vol. 15, pp. 502-523, 2006.
[10] C. Kaufman, M. Schervish, and D. Nychka, "Covariance Tapering for Likelihood-Based Estimation in Large Spatial Data Sets," *J. Am. Statistical Assoc.*, vol. 103, no. 484, pp. 1545-1555, 2008.
[11] J. Quiñonero-Candela and C. Rasmussen, "A Unifying View of Sparse Approximate Gaussian Process Regression," *J. Machine Learning Research*, vol. 6, pp. 1939-1959, Dec. 2005.
[12] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[13] H. Rue and H. Tjelmeland, "Fitting Gaussian Markov Random Fields to Gaussian Fields," *Scandinavian J. Statistics*, vol. 29, no. 1, pp. 31-49, 2002.
[14] D. Duvenaud, H. Nickisch, and C.E. Rasmussen, "Additive Gaussian Processes," *Proc. Advances in Neural Information Processing Systems (NIPS)*, pp. 226-234, 2011.
[15] L. Breiman and J. Friedman, "Estimating Optimal Transformations for Multiple Regression," *Computer Science and Statistics: Proc. 16th Symp. the Interface*, pp. 121-134, 1985.
[16] T. Hastie and R. Tibshirani, *Generalized Additive Models*. Chapman & Hall, 1990.
[17] E. Gilboa, Y. Saatçi, and J.P. Cunningham, "Scaling Multidimensional Gaussian Processes Using Projected Additive Approximations," *JMLR: W&CP*, vol. 28, 2013.
[18] T. Hasite and R. Tibshirani, "Bayesian Backfitting," *Statistical Science*, vol. 15, pp. 193-223, 2000.
[19] Y. Saatçi, "Scalable Inference for Structured Gaussian Process Models," PhD dissertation, Univ. of Cambridge, 2011.
[20] L. Arnold, *Stochastic Differential Equations: Theory and Practice*. Krieger Publishing Corporation, 1992.
[21] J. Hartikainen and S. Särkkä, "Kalman Filtering and Smoothing Solutions to Temporal Gaussian Process Regression Models," *Proc. IEEE Int'l Workshop Machine Learning for Signal Processing (MLSP)*, pp. 379-384, Aug. 2010.
[22] C.M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2007.
[23] F. Vivarelli and C.K.I. Williams, "Discovering Hidden Features with Gaussian Processes Regression," *Proc. Advances in Neural Information Processing Systems*, vol. 11, 1998.
[24] E. Snelson and Z. Ghahramani, "Variable Noise and Dimensionality Reduction for Sparse Gaussian Processes," *Proc. Uncertainty in Artifical Intelligence (UAI 22)*, 2006.
[25] A.G. Wilson, D.A. Knowles, and Z. Ghahramani, "Gaussian Process Regression Networks," *Proc. Int'l Conf. Machine Learning (ICML)*, 2012.
[26] R. Gramacy and H. Lian, "Gaussian Process Single-Index Models as Emulators for Computer Experiments," *Technometrics*, vol. 54, pp. 30-41, 2012.
[27] P.R. Hahn and L.F. Burgette, "The Mesa Distribution: An Approximation Likelihood for Simultaneous Nonlinear Quantile Regression," technical report, Univ. of Chicago, 2012.
[28] T. Hastie, R. Tibshirani, and J.H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. second ed., Springer-Verlag, 2009.
[29] H.A. Chipman, E.I. George, and R.E. McCulloch, "BART: Bayesian Additive Regression Trees," *The Annals of Applied Statistics*, vol. 4, no. 1, pp. 266-298, 2010.
[30] J. Friedman and W. Stuetzle, "Projection Pursuit Regression," *J. Am. Statistical Assoc.*, vol. 76, pp. 817-823, 1981.
[31] T. Minka, "A Family of Algorithms for Approximate Bayesian Inference," PhD thesis, MIT, 2001.
[32] A. Naish-Guzman and S. Holden, "The Generalized FITC Approximation," *Proc. Advances in Neural Information Processing Systems 21*, pp. 534-542, 2007.
[33] J. Vanhatalo and A. Vehtari, "Sparse Log Gaussian Processes via MCMC for Spatial Epidemiology," *J. Machine Learning Research*, vol. 1, pp. 73-89, 2007.
[34] N.D. Lawrence, M. Seeger, and R. Herbrich, "Fast Sparse Gaussian Process Methods: The Informative Vector Machine," *Proc. Advances in Neural Information Processing Systems 17*, pp. 625-632, Dec. 2003.
[35] H. Nickisch and C. Rasmussen, "Approximations for Binary Gaussian Process Classification," *J. Machine Learning Research*, vol. 9, pp. 2035-2078, Dec. 2008.
[36] T. Hastie and R. Tibshirani, "Generalized Additive Models (with Discussion)," *Statistical Science*, vol. 1, pp. 297-318, 1986.
[37] J.P. Cunningham, K.V. Shenoy, and M. Sahani, "Fast Gaussian Process Methods for Point Process Intensity Estimation," *Proc. 25th Int'l Conf. Machine Learning (ICML)*, pp. 192-199, 2008.
[38] P. McCullagh, "What Is a Statistical Model?" *The Annals of Statistics*, vol. 30, no. 5, pp. 1225-1534, 2002.
[39] C. Williams, M. Baran, and E. Bonilla, "A Note on Noise-Free Gaussian Process Prediction with Separable Covariance Functions and Grid Designs," , Dec. 2007.
[40] J. Luttinen and A. Ilin, "Efficient Gaussian Process Inference for Short-Scale Spatio-Temporal Modeling," *Journal of Machine Learning Research (JMLR): W&CP*, vol. 22, pp. 741-750, 2012.
[41] P.W. Goldberg, C.K. Williams, and C.M. Bishop, "Regression with Input-Dependent Noise: A Gaussian Process Treatment," *Proc. Advances in Neural Information Processing Systems*, vol. 10, 1998.
[42] E. Snelson and Z. Ghahramani, "Sparse Gaussian Processes Using Pseudo-Inputs," *Proc. Advances in Neural Information Processing Systems 18*, pp. 1257-1264, Dec. 2006.
[43] C. Rasmussen and H. Nickisch, "Gaussian Processes for Machine Learning (GPML) Toolbox," *J. Machine Learning Research*, vol. 11, pp. 3011-3015, Dec. 2010.
[44] C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine Learning*, vol. 20, no. 3, pp. 273-297, 1995.
[45] J.C. Platt, "Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods," *Advances in Large Margin Classifiers*, pp. 61-74, MIT Press, 1999.
[46] J. Vanhatalo, J. Riihimäki, J. Hartikainen, P. Jylänki, V. Tolvanen, and A. Vehtari, "Bayesian Modeling with Gaussian Processes Using the MATLAB Toolbox GPstuff (v3.3)," *arXiv*, 2012.
[47] C. Chang and C. Lin, "LIBSVM: A Library for Support Vector Machines," *ACM Trans. Intelligent Systems and Technology*, vol. 2, article 27, 2011.
[48] R. Perkins and V. Gruev, "Signal-to-Noise Analysis of Stokes Parameters in Division of Focal Plane Polarimeters," *Optics Express*, vol. 18, pp. 25815-25824, 2010.
[49] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, "KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework," *J. Multiple-Valued Logic and Soft Computing*, vol. 17, no. 2-3, pp. 255-287, 2011.
[50] C. Chang and C.J. Lin, "IJCNN 2001 Challenge: Generalization Ability and Text Decoding," *Proc. IEEE Int'l Joint Conf. Neural Networks (IJCNN)*, pp. 1031-1036, 2001.
[51] X. Li and M.T. Orchard, "New Edge-Directed Interpolation," *IEEE Trans. Image Processing*, vol. 10, no. 10, pp. 1521-1527, Oct. 2001.

Elad Gilboa (S'12) received the BSc degree in electrical and computer engineering and the ME degree in biomedical engineering from Technion University, Israel, in 2004 and 2009. He is currently working toward the PhD degree in electrical and system engineering at Washington University in St. Louis. His research interests are in statistical signal processing, machine learning, and parallel nonlinear optimization. He is a student member of the IEEE.

Yunus Saatçi received the MSc degree in Informatics from the University of Edinburgh, United Kingdom, in 2007, and the PhD degree from Cambridge University, United Kingdom, under the supervision of Dr. Carl E. Rasmussen and Professor Zoubin Ghahramani in 2011. During the PhD degree he launched a Machine Learning consultancy (QuantMachines Ltd.) with Dr. Rasmussen. His research interests include Bayesian methods for supervised, unsupervised, and reinforcement learning, with focus on making these methods scalable.

John P. Cunningham received the BA degree in computer science from Dartmouth University in 2002, the MS and PhD degrees in electrical engineering from Stanford University in 2006 and 2009, after which he did postdoctoral work in the Machine Learning Group at Cambridge University, United Kingdom, where he was the Sackler Engineering fellow of Christ's College. He is an assistant professor in statistics at Columbia University. His research includes information engineering and machine learning, particularly their application to real world data.