Speeding Up Neighborhood Search in Local Gaussian Process Prediction

Robert B. Gramacy
Booth School of Business
The University of Chicago
Chicago, IL 60637
(rbgramacy@chicagobooth.edu)

Benjamin Haaland
Duke-NUS Graduate Medical School
National University of Singapore
Singapore 119077
(ben.haaland@isye.gatech.edu)

Recent implementations of local approximate Gaussian process models have pushed computational boundaries for nonlinear, nonparametric prediction problems, particularly when deployed as emulators for computer experiments. Their flavor of spatially independent computation accommodates massive parallelization, meaning that they can handle designs two or more orders of magnitude larger than previously. However, accomplishing that feat can still require massive computational horsepower. Here we aim to ease that burden. We study how predictive variance is reduced as local designs are built up for prediction. We then observe how the exhaustive and discrete nature of an important search subroutine involved in building such local designs may be overly conservative. Rather, we suggest that searching the space radially, that is, continuously along rays emanating from the predictive location of interest, is a far thriftier alternative. Our empirical work demonstrates that ray-based search yields predictors with accuracy comparable to exhaustive search, but in a fraction of the time—for many problems bringing a supercomputer implementation back onto the desktop. Supplementary materials for this article are available online.

KEY WORDS: Active learning; Approximate kriging; Big data; Nearest neighbor; Nonparametric regression; Sequential design of experiments.

1. INTRODUCTION

Gaussian process (GP) regression is popular for response surface modeling wherever surfaces are reasonably smooth, but where otherwise little is known about the input–output relationship. GP regression models are particularly popular as emulators for computer experiments (Sacks et al. 1989; Santner, Williams, and Notz 2003), whose outputs tend to exhibit both qualities. Moreover, computer experiments are often deterministic, and it turns out that GPs are one of a few flexible regression approaches, which can interpolate while also predicting accurately with appropriate coverage out-of-sample. Unfortunately, GP regression requires \( O(N^3) \) dense matrix decompositions, for \( N \) input–output pairs, so implementations struggle to keep up with today’s growing pace of data collection. Modern supercomputers make submitting thousands of jobs as easy as submitting one, and therefore \( N = 27 \) runs are no longer a prototypically sized computer experiment (Chen et al. 2015).

Most modern desktops cannot decompose dense matrices as large as \( 10^3 \times 10^4 \), due primarily to memory constraints. They struggle to perform tens to hundreds of much smaller, \( 10^3 \times 10^3 \) decompositions required for numerical inference of unknown parameters, say by maximum likelihood, in reasonable time. Although humbly small by comparison to other literatures, like genetics or marketing, those limits define “big data” for computer experiments: the canonical models cannot cope with the modern scale of computer simulation.

A scramble is on for fast, approximate, alternatives (e.g., Kaufman et al. 2012; Sang and Huang 2012; Gramacy and Apley 2015), and a common theme is sparsity, leading to fast matrix decompositions. Some approaches increase data size capabilities by one-to-two orders of magnitude. Yet even those inroads are at capacity. Practitioners increasingly prefer much cruder alternatives, for example, trees (Chipman, Ranjan, and Wang 2012; Gramacy, Taddy, and Wild 2012; Pratola et al. 2014), which struggle to capture the smoothness of most simulators.

Another approach is to match supercomputer simulation with supercomputer emulation, for example, using graphical processing units (GPUs, Franey, Ranjan, and Chipman 2012), cluster, and symmetric-multiprocessor computation, and even all three together (Paciorek et al. 2015), for (dense matrix) GP regression. This too has led to orders of magnitude expansion in capability, but may miss the point: emulation is meant to avoid further computation. Hybrid approximate GPs and big-computer resources have been combined to push the boundary even farther (Eidsvik et al. 2014). Gramacy, Niemi, and Weiss (2015) later showed how GPUs and/or thousands of clustered CPUs could be combined to handle designs as large as \( N = 10^6 \) in about an hour.

This article makes three contributions to this literature, focusing on a particular sparsity-inducing local GP approximation developed by Gramacy and Apley (2015). First, we study a greedy search subroutine, applied locally and independently for each element of a potentially vast predictive grid. Each search identifies small local subdesigns by a variance reduction heuristic—the main sparsity-inducing mechanism—and we show how it organically facilitates a desirable trade-off between local and (more)
global site selection. We then observe highly regular patterns in the variance reduction surface searched in each iteration of the greedy scheme, motivating our second contribution. We propose swapping out an exhaustive discrete search for a continuous one having far narrower scope: along rays emanating from each predictive site. Our empirical work demonstrates that the new scheme yields accurate predictions in time comparable to a GPU-cluster computing implementation, yet only requires a modern desktop. Third, to acknowledge that rays can be inefficient in highly anisotropic contexts, we illustrate in our supplementary material how a thrifty prescaling step from a crude global analysis leads to improved out-of-sample performance. Finally, an implementation is provided in an updated 1aGP package for R (Gramacy 2013, 2015).

The remainder of the article is outlined as follows. In Section 2, we survey the modern literature for fast approximate GP regression with an emphasis on sparsity and local search. Section 3 explores the structure of local designs, recommending the simple heuristic presented in Section 4. In Section 5, we provide implementation details and illustrations on real and simulated data from the recent literature. Section 6 concludes with a brief discussion. Accompanying supplementary material extends the empirical studies in Sections 3 and 5.

2. FAST APPROXIMATE GAUSSIAN PROCESS REGRESSION

Here, we review the basics of GP regression, emphasizing computational limitations and remedies separately leveraging sparsity and distributed computation. That sets the stage for a modern recasting of a localization technique from the spatial statistics literature that is able to leverage both sparsity and big computing paradigms. Yet even that method has inefficiencies, which motivates our contribution.

2.1 Kriging and Sparsity

A Gaussian process (GP) is a prior over functions (see, e.g., Stein 1999), with finite-dimensional distributions defined parsimoniously by a mean and covariance, often paired with an error model (also Gaussian) for noisy data. However for regression applications, a likelihood perspective provides a more expedient view of the salient quantities. In a typical GP regression, N data pairs \( D_N = (X_N, Y_N) \), comprised of an \( N \times p \)-dimensional design \( X_N \) and an \( N \)-vector of scalar responses \( Y_N \), is modeled as \( Y_N \sim \mathcal{N}(f_0(X_N), \Sigma_0(X_N)) \), where \( \theta \) are a small number of parameters that relate the mean \( f \) and covariance \( \Sigma \) to covariates \( X_N \). Linear regression is a special case where \( f_0(X_N) = X_N \beta \) and \( \Sigma_0(X_N) = \tau^2 I_N \).

In the nonlinear case, it is typical, especially for computer experiments (e.g., Santner, Williams, and Notz 2003), to have a zero mean and therefore move all of the “modeling” into a correlation function \( K_\theta(x, x') \) so that \( Y_N \sim \mathcal{N}(0, \tau^2 K_N) \), where \( K_N \) is an \( N \times N \) positive definite matrix comprised of pairwise evaluations \( K_\theta(x_i, x_j) \) on the rows of \( X_N \). Choice of \( K_\theta(\cdot, \cdot) \) determines the decay of spatial correlation throughout the input space, and thereby stationarity and smoothness. A common first choice is the so-called isotropic Gaussian: \( K_\theta(x, x') = \exp[-\sum_{k=1}^p (x_k - x'_k)^2 / \theta] \), where correlation decays very rapidly with lengthscale determined through \( \theta \). Since \( K_\theta(x, x) = 1 \), the resulting regression function is an interpolator, which is appropriate for many deterministic computer experiments. For noisy data, or for more robust modeling (protecting against numerical issues as well as inappropriate choice of covariance, for example, assuming stationarity) of deterministic computer experiments (Gramacy and Lee 2012), a nugget can be added to \( K_\theta \). Here, we primarily use the isotropic Gaussian formulation, and fix a small nugget for numerical stability. Some variations in empirical work are provided in the supplementary material. All new methodology described herein can be generalized to any correlation family that is differentiable in \( \theta \).

GP regression is popular because inference (for \( \theta \)) is easy, and (out-of-sample) prediction is highly accurate and conditionally on \( \theta \) analytic. It is common to deploy a reference \( \pi(\tau^2) \propto 1/\tau^2 \) prior (Berger, De Oliveira, and Sanso 2001) and obtain a marginal likelihood for \( \theta \)

\[
p(Y_N|K_\theta(\cdot, \cdot)) = \frac{\Gamma[N/2]}{(2\pi)^{N/2}|K_N|^{1/2}} \cdot \frac{\psi_N}{2}^{N/2},
\]

where \( \psi_N = Y_N^T K_N^{-1} Y_N \),

which has analytic derivatives, leading to fast Newton-like schemes for maximizing.

The predictive distribution \( p(y(x)|D_N, K_\theta(\cdot, \cdot)) \) is Student-\( t \) with degrees of freedom \( N \),

\[
\mu(x|D_N, K_\theta(\cdot, \cdot)) = K_N(x) K_N^{-1} Y_N,
\]

and scale \( \sigma^2(x|D_N, K(\cdot, \cdot)) = \psi_N[ K_\theta(x, x) - K_N(x) K_N^{-1} K_N(x) ] / N \),

where \( k_N(x) \) is the \( N \)-vector whose \( i \)th component is \( K_\theta(x_i, x_i) \). Using properties of the Student-\( t \), the variance of \( Y(x) \) is \( V_N(x) = \text{var}[Y(x)|D_N, K_\theta(\cdot, \cdot)] = \sigma^2(x|D_N, K_\theta(\cdot, \cdot)) \times N/(N - 2) \). The form of \( V_N(x) \), expanding out in a “football shape” away from the elements of \( X_N \), has attractive uses in sequential design applications (Gramacy and Lee 2009).

The trouble with all this is \( K_N^{-1} \) and \( |K_N| \), appearing in several instances in Equations (1)–(3), and requiring \( O(N^3) \) computation for decomposing dense matrices. That limits data size to \( N \approx 1000 \) in reasonable time—less than 1 hr for inference and prediction on a commensurately sized \( N \)-predictive-grid, say—without specialized hardware. Franey, Ranjan, and Chipman (2012) showed how graphical processing unit (GPU) matrix decompositions can accommodate \( N \approx 5000 \) in similar time. Paciorek et al. (2015) added an order of magnitude to \( N \approx 60,000 \) with a combination of cluster computing (with nearly 100 nodes, 16 cores each) and GPUs.

An alternative is to induce sparsity on \( K_N \) and leverage fast sparse-matrix libraries, or to avoid large matrices altogether. Kaufman et al. (2012) used a compactly supported correlation (CSC) function, \( K_\theta(\cdot, \cdot) \), that controls the proportion of \( K_N \) entries that are nonzero. Snelson and Ghahramani (2006) worked with a reduced global design of pseudo-inputs, while Cressie and Johannesson (2008) exploited a truncated basis. Such sparsity-inducing approaches extend the dense-matrix alternatives by an order of magnitude. For example, Kaufman et al. (2012) illustrated with an \( N = 20,000 \) cosmology dataset.
However, there is a need for bigger capability. For example, Pratola et al. (2014) chose a thrifty sum of trees model that allowed for cluster-style (message passing interface, MPI) implementation to perform inference for an \( N = 7 \) million sized design distributed over hundreds of computing cores. Until recently, such a large dataset was well out of reach of GP-based methods, whether by sparse approximation or distributed computation.

2.2 Local Search

In computer experiments, where emulation or surrogate modeling emphasizes accurate prediction (2)–(3), Gramacy and Apley (2014) showed how orders of magnitude faster inference and prediction can be obtained by modernizing local kriging from the spatial statistics literature (Cressie 1991, pp. 131–134). Focusing on quickly obtaining accurate predictive equations at a particular location, \( x_j \), local kriging involves choosing data subsets \( D_j(x) \subseteq D_N \), based on \( n < N \) whose \( X_n(x) \) values are close to \( x \). This recognizes that data far from \( x \) have vanishingly small influence on prediction given the typical choices of rapidly decaying correlation functions \( K_\theta(\cdot, \cdot) \). The simplest choice is to fill \( D_j(x) \) with \( n \) nearest neighbors (NNs) \( X_n(x) \) to \( x \) in \( X_N \), along with responses \( Y_n(x) \).

This is a sensible idea. As long as \( D_n \) is determined purely by computational considerations, that is, not by looking \( Y_N \) values, the result is a valid probability model for \( Y(x) \mid D_N \) (Datta et al. 2015). For modest \( n \), prediction and inference is fast and accurate, and as \( n \) gets large predictors increasingly resemble their full-data analogs with a variance \( V_n(x) \) that is organically inflated (higher variance for smaller \( n \)) relative to \( V_N(x) \). Recently, Emory (2009) showed that the NN version works well for a wide swath of common choices of \( K_\theta(\cdot, \cdot) \). However, there is documented scope for improvement. Vecchia (1988) and Stein, Chi, and Welty (2004) argued that the NN designs are suboptimal—i.e., pays to have a local design with more spread in the input space. However, finding an optimal local design \( D_n(x) \), under almost any criteria, would involve a combinatorially huge search even for modest \( n \) and \( N \).

Gramacy and Apley (2015) showed how a greedy iterative search for local designs, starting with a small NN set \( D_{n_0}(x) \) and building up to \( D_n(x) \) by augmenting \( D_j(x) \), for \( j = n_0, \ldots, n - 1 \) through a simple objective criteria leads to better predictions than NN. Importantly, the greedy and NN schemes can be shown to have the same computational order, \( O(n^3) \), when an efficient updating scheme is deployed for each \( j \rightarrow j + 1 \). The idea of building up designs iteratively for faster calculations is not new (Gramacy and Polson 2011; Haaland and Qian 2011), however the focus has previously been global. Gramacy and Apley’s local search chooses an \( x_{j+1} \) from the remaining set \( X_N \setminus X_j(x) \) to maximally reduce predictive variance at \( x \). The local designs contain a mixture of NNs to \( x \) and somewhat farther out “satellite” points, which we explore further in Section 3.

The resulting local predictors are at least as accurate as other sparse methods, like CSC, but incur an order of magnitude lower computational cost. Since calculations are independent for each predictive location, \( x \), prediction over a dense grid can be trivially parallelized to leverage multiple cores on a single (e.g., desktop) machine. However somewhat disappointingly, Gramacy and Apley also observed that the resulting greedy local predictors are not more accurate per-unit-computational cost than local NN predictors derived from an order of magnitude larger \( n \). In other words, the search for \( x_{j+1} \) over a potentially huge number of candidates \( |X_N \setminus X_j(x)| = N - j \), for \( j = n_0, \ldots, n \) is expensive relative to decomposing a larger (but still small compared to \( N \)) matrix for GP inference and prediction.

Gramacy, Niemi, and Weiss (2015) later recognized that that search, structured to independently entertain thousands of \( x_{j+1} \) under identical criteria, is ideal for exporting to a GPU. Depending on the size of the search, improvements were 20–100-fold, leading to global improvements (over all \( j = n_0, \ldots, n \)) of 5–20x, substantially out-pacing the accuracy-per-flop rate of a big-NN alternative. Going further, they showed how a GPU/multi-CPU/cluster scheme could combine to accurately emulate at one-million-sized designs in about an hour—a feat unmatched by other GP-based methodology.

Further savings are obtained by recognizing that searching over all \( N - j \) candidates is overkill, whether via GPU or otherwise, since many are very far from \( x \) and thus have almost no influence on prediction. Searching over \( N' \ll N - j \) NNs, say \( N' = 1000 \) when \( N = 1 \) million yields substantial speedups, but multi-node/multi-core resources would still be required in large data contexts. Importantly, one must take care not to choose \( N' \) so small as to preclude entertaining the very points, well outside the NN set, which lead to improvements over NN. Considering the observed regularity of the greedy local designs, illustrated below, there does nevertheless seem to be potential for drastically shortcutting an exhaustive search without limiting scope, thereby avoiding the need for specialized computing architecture.

3. EXPLORING LOCAL INFLUENCE ON PREDICTION

The numbers in Figure 1 indicate local designs \( X_{n=50}(x) \) built up in a greedy fashion, successively choosing the next location by a reduction in variance criteria, discussed in detail shortly. Focus first on the black numbers, based on an exhaustive search. The reference predictive location, \( x = (−1.725, 1.725)^\top \), is situated off of the input design \( X_{N=40401} \), which is a regular \( 201 \times 201 \) grid in \([-2, 2]^2\). Observe that the local sub-design contains a roughly \( \frac{2}{3} : \frac{1}{3} \) split comprising of the nearest elements in \( X_N \) to \( x \) and ones farther out. However, even the farther-out elements are closely relative to the full scope of the grid in \([-2, 2]^2\). Observe that although many NNs are chosen in early iterations, the farthest-out locations are not exclusive to the final iterations of the search. As early as \( j = 10 \), a far-out excursion was made, and as late as \( j = 50 \) an NN was chosen.

On first encounter, it may be surprising that having farther-out design elements helps reduce local predictive variance differentially more than points much closer in. One naturally wonders what trade-offs are being made in the objective criteria for design, and to what extent they can be attributed to a particular parameterization of \( K_\theta(\cdot, \cdot) \). These two aspects are explored below, leading to novel technical and empirical observations.
with an eye toward exploiting that trade-off for fast search in Section 4.

3.1 From Global to Local Design

Designing \( X_N \) to minimize predictive variance, averaged globally over the input space \( \mathcal{X} \), is a sensible objective leading to so-called A-optimal designs (see, e.g., Santner, Williams, and Notz 2003). However, minimizing over \( N \) free coordinates in \( p \)-dimensional space is combinatorially complex. Seo et al. (2000) showed that approximately A-optimal designs can be obtained by proceeding sequentially, that is, greedily: building up \( X_N \) through choices of \( x_{j+1} \) to augment \( X_j \) to minimize variance globally. In particular, they considered choosing \( x_{j+1} \) to minimize \( \Delta V_j(x_{j+1}) = \int_X V_j(x) - V_j(x|x_{j+1}) \, dx \), where \( V_j(x|x_{j+1}) \) is the new variance after \( x_{j+1} \) is added into \( X_j \), obtained by applying Equations (2) and (3) with a \( (j+1) \)-sized design \( X_j; X_{j+1} \). All quantities above, and below, depend implicitly on \( \theta \). Now, minimizing future variance is equivalent to maximizing a quantity proportional to a \textit{reduction} in variance:

\[
\int_X k_j^2(x)G_j(x_{j+1})v_j(x_{j+1})k_j(x) + 2k_j^2(x)g_j(x_{j+1})K(x_{j+1}, x) + K(x_{j+1}, x)^2/v_j(x_{j+1}) \, dx
\]

(4)

where \( G_j(x') = g_j(x')g_j^T(x') \), \( g_j(x') = K_j^{-1}k_j(x')/v_j(x') \), and \( v_j(x_{j+1}) = K_0(x_{j+1}, x_{j+1}) - k_j^2(x_{j+1})K_j^{-1}k_j(x_{j+1}) \).

The designs that result are difficult to distinguish from ones derived from other space-filling criteria. The advantage is that greedy selection avoids a combinatorially huge search.

Figure 1. Example local designs constructed by a greedy application of a reduction in variance criterion (4). The numbers indicate the order in which each location is chosen. The candidates form a regular 201 \( \times \) 201 grid in \([-2, 2]^2 \); black numbers are from an exhaustive search (Section 3), whereas the green ones are from the search method proposed in Section 4.

Gramacy and Apley (2015) argued that the \textit{integrand} in (4) is a sensible heuristic for local design. It tabulates a quantity proportional to reduction in variance at \( x \), obtained by choosing to add \( x_{j+1} \) into the design, which is the dominating component in an estimate of the future mean squared prediction error (MSPE) at that location. When applied sequentially to build up \( X_n(x) \) via \( X_j(x) \) and \( x_{j+1} \) for \( j = n_0, \ldots, n - 1 \), the result is again an approximate solution to a combinatorially huge search. However, the structure of the resulting local designs \( X_n(x) \), with near as well as far points, is counterintuitive (see Figure 1). The typical rapidly decaying \( K_\theta(\cdot, \cdot) \) should substantially devalue locations far from \( x \). Therefore, considering two potential locations, an \( x_{j+1} \) close to \( x \) and \( x_{j+1}' \) farther away, one wonders: how could the latter choice, \( x_{j+1}' \) with \( y_{j+1}' \)-value modeled as vastly less correlated with \( y(x) \) than \( y_{j+1} \) via \( x_{j+1} \), be preferred over the closer \( x_{j+1} \) choice?

The answer is remarkably simple, and has little to do with the form of \( K_\theta(\cdot, \cdot) \). The integrand in Equation (4) looks quadratic in \( K_\theta(x_{j+1}, x) \), which measures a “distance,” in terms of correlation \( K_\theta \), between the reference predictive location \( x \) and the potential new local design location \( x_{j+1} \). That would seem to suggest maximizing the criteria involves maximizing \( K_\theta(x_{j+1}, x) \), that is, choosing \( x_{j+1} \) as close as possible to \( x \). But that is not the only part of the expression that involves \( x_{j+1} \). Observe that the integrand is also quadratic in \( g_j(x_{j+1}) \), a vector measuring “inverse distance,” via \( K_j^{-1} \), between \( x_{j+1} \) and the current local design \( X_j(x) \). So there are two competing aims in the criteria: minimize “distance” to \( x \) while maximizing “distance” (or minimizing “inverse distance”) from the existing design. Upon further reflection, that tradeoff makes sense. The value of a potential new data element \( (x_{j+1}, y_{j+1}) \) depends not just on its proximity to \( x \), but also on how potentially different that information is from where we already have (lots of) it, at \( X_j(x) \).

This observation also provides insight into the nature of global A-optimal designs. We now recognize that the integral in Equation (4), often thought to be essential to obtain space-filling behavior in the resulting design, is but one contributing aspect. Assuming that potential design sites for \( X_N \) or \( X_n(x) \subset \mathcal{X}_n \) are limited, say to a predefined mesh of values (note that this is always the case when working with a finite precision computer implementation) and that it is not possible to observe the true output \( y(x) \) at locations \( x \) where you are trying to predict (that would cause the integrand in (4) to be minimized trivially, and obliterate the need for emulation), the GP predictor prefers design sites that are somewhat spread out relative to where it will be used to predict, globally or locally, regardless of the choice of \( K_\theta(\cdot, \cdot) \).

3.2 Ribbons and Rings

Having established it is possible for the criteria to prefer new \( x_{j+1} \) farther from \( x \), being repelled by \( X_j(x) \) already nearby, it is natural to wonder about the extent of that trade-off in particular examples: of iteration \( j \), choice of parameters \( \theta \), and operational considerations such as how searches are initialized (a number \( n_0 \) of NNSs) and any numerical considerations (i.e., nugget \( \eta \)). Here we restrict \( X_N \) to the 201 \( \times \) 201 grid coinciding with the example in the previous subsection. Figure 2 shows snapshots along the trajectory of four separate greedy
Figure 2. Progress and reduction in variance surfaces at various stages of the search under different initial conditions and GP parameters. Whites are higher values; reds are lower values. The four snapshots in each column were taken at $j = \{9, 10, 20, 50\}$; $j = \{9, 25, 26, 50\}$; $j = \{24, 25, 38, 49\}$; and $j = \{1, 4, 30, 50\}$, respectively. The full sequence for each case is provided as supplementary material.

searches (one in each column) for prediction at the same $x$ location, shown as a solid dot. The image plots show the reduction in variance criteria (integrand of (4)), with lighter shades being higher values, and open circles representing the current local design $X_j(x)$. The full set of plots for each of $n = 50$ greedy selections are provided as supplementary material.

First some high level observations. The red ribbons and rings, carving out troughs of undesirable locations for future sampling, are fascinating. They persist, with some refinement, as greedy selections progress (down the rows), choosing elements of $X_j(x)$, the open circles. Moreover, these selections, whether near to $x$ or far, impact future choices spanning great distances in both space and time, that is, impacting $X_{j+k}$ many iterations $k$ into the future.

Now some lower level observations along the columns. Focusing on the first one, notice how the local design initially spans out in four rays emanating from $x$. But eventually points off those rays are chosen, possibly forming new rays. The third column shows a similar progression, although the larger $\eta$ combined with smaller $\theta$ leads to a bigger proportion of NNs. The first and second rows in those columns show successive designs for $j$ and $j + 1$ (however with different $j$), and it is interesting to see how those choices impact the resulting reduction in variance surface. In the first column, the choices “tie” a ribbon of red closer to $x$ in one corner of the design space. In the third, where the same behavior might be expected, instead the surface flattens out and a red ring is created around the new point.

The second and fourth column show more extreme behavior. In the second, the large $\theta$ and $\eta$ lead to many NNs being chosen.
However, eventually the design does fan out along rays, mimicking the behavior of the first and third columns. The fourth column stands out as exceptional. That search is initialized at $n_0 = 1$ and uses a nearly zero nugget $\eta$. The first dozen or so locations follow an arc emanating from $x$, shown in the first two rows. Eventually, in the third row, the arc circles around on itself, creating a ring of points. However, now some points off of the ring have also been selected, both near and far. The final row shows the ring converting into a spiral. More importantly, it shows a local design starting to resemble those from the other columns: NNs with satellite points positioned (loosely) along rays.

Our takeaway from this empirical analysis is that the pattern of local designs is robust to the choice of correlation parameters, although the proportion of NNs to satellite points, and the number of rays and their orientation is sensitive to those choices. We worked with a gridded design, which lent a degree of regularity to the illustration. In other experiments (not shown), we saw similar behavior as long as $X_N$ was space-filling. However, obviously if $X_N$ does not accommodate selecting local designs along rays, then an $X_n(x)$ could not exhibit them. Below we suggest a searching strategy leveraging the patterns observed here, but which does not preclude discovering different local structure in a less idealized setting.

4. EXPLOITING LOCAL INFLUENCE FOR EFFICIENT SEARCH

Here, we exploit the partition between NNs and satellite points placed along rays observed in Section 3.2. We replace an exhaustive search among candidates $X_N \setminus X_j(x)$ with a continuous line search that can be solved quickly via a standard one-dimensional (1D) numerical optimizer, and then “snap” the solution back to the nearest element of the candidate set. Recognizing that the greedy local design pattern is sensitive to initialization and parameterization, as exemplified in Figure 2, our scheme is conservative about choosing search directions: looking along rays but not emphasizing the extension of existing ones.

Consider an existing local design $X_j(x)$ and a search for $X_{j+1}$ along rays emanating from $x$. For example, the left panel of Figure 3 shows rays at iteration $j = 20$. Determining the start and end of the rays—that is, the actual line segments—is discussed shortly. For each ray, the corresponding 1D graph of reduction in variance along the segment is shown in the right panel. Of the 35 rays, 5 have a reduction in variance, which is maximized away from its origin, at $x$, and therefore away from the NN set. Those are shown in black, with blue triangles denoting the optima, and green diamonds indicating the closest element of the candidate set, $X_N \setminus X_{20}(x)$, to that point. The right panel shows that if a random ray were chosen, the chances would be 5/35 that a nonNN point would be selected. For reference, we note that the next satellite point is chosen by exhaustive search at $j = 27$.

We select the nearest candidates in $X_N \setminus X_j(x)$ to $x$ as potential starting points for a ray shooting in the opposite direction. There is no need to search between those nearest candidates and $x$ since there are, by definition, no remaining candidates there. We allow the segment to stretch to 10 times the distance between the starting point and $x$, or to the edge of the bounding box of the full candidate set $X_N$. The rays in Figure 3 follow this rule. This choice allows the segments to grow in length as NNs accumulate in $X_j(x)$; both the starts and ends of rays will be spaced farther from each other and from $x$ as $j$ increases. Although we show many rays in the figure, only a select few are considered in each iteration $j$. These are based on a pseudo-random sequence on the NN candidates, following

$$k = (j - n_0 + 1) \mod \lfloor \sqrt{j - n_0 + 1} \rfloor.$$

Then, if the dimension of the input space is $p$, we take the $k$th, $k + 1$st, . . . , $k + p$th closest elements to $x$ in $X_N \setminus X_n(x)$ as starting points for ray searches. The start of the sequence, $k$, is designed to be a round robin over the nearest elements, where the scope of the “round” increases as $j$ increases, discouraging successive searches from identical starting points. Creating $p$ rays acknowledges that as the input dimension grows so too does the complexity of search, however number of rays could...
be a user-defined choice adjusting the speed (fewer rays is faster) and fidelity (but also cruder) as the situation recommends. We note that ray starting points could also be chosen randomly, however that would result in a stochastic local design \( X_s(x) \), and therefore stochastic predictions, which would be undesirable in many contexts.

Any 1D optimization scheme can be used to search over the convex combination parameter \( s \in [0, 1] \), spanning the endpoints of the line segment. We use the derivative-free \( \text{Brent}_\text{fmin} \) method (Brent 1973), which is the workhorse behind R’s \texttt{optimize} function, leveraging a C implementation of Fortran library routines (http://www.netlib.org/fmm/fmin.f). A bespoke derivative-based method could speed convergence, however we prefer the well-engineered library routine in our setting of repeated application over \( j = n_0, \ldots, n \) for each of a potentially huge set of \( x \)’s. We recognize that \( \text{Brent}_\text{fmin} \)’s initialization biases the solver toward solutions in the interior of the search space, while we observe (see, e.g., Figures 2 and 3) that there is nearly always a mode at the NN (\( s = 0 \)), in addition to possibly multiple others. Therefore, we check against the \( s = 0 \) mode as a post-processing step.

Optimizing over rays replaces an exhaustive discrete search with a continuous one, providing a solution \( X_{p+1}^* \) off the candidate set. So the final step involves “snapping” \( X_{p+1}^* \) onto \( x_{j+1}^* \in X_N \setminus X_N(x) \), by minimizing Euclidean distance. When performing that search, we explicitly forbid the ray starting location(s) from being chosen, unless its location agrees precisely with \( x_{j+1}^* \), that is, unless \( \text{Brent}_\text{fmin} \) returns \( s = 0 \). This prevents choosing an NN location known \textit{not} to maximize the search objective criteria.

Besides these modifications—pseudo random round-robin ray choice, numerical optimization over a convex combination, and snapping back to the candidate set—the proposed scheme is identical to the one described in Section 2.2. The green numbers in Figure 1 show that the local designs based on ray search are qualitatively similar to the exhaustive ones. What remains is to demonstrate comparable out-of-sample predictive performance at substantially reduced computational cost.

5. IMPLEMENTATION AND EMPIRICAL COMPARISON

The methodological developments described above, and all local GP comparators, are implemented in the \texttt{laGP} package on CRAN. The code is primarily in C and uses \texttt{OpenMP} for multicore parallelization, as described by Gramacy and Apley (2015). Our experiments distribute the local design over eight threads on a four-core hyperthreaded 2010-model iMac. GPU subroutines for parallelized exhaustive search of the reduction in variance criteria are also provided in the package, however we do not include any new runs leveraging that feature in the empirical work reported here. We will, however, provide a brief to the timings on GPUs and multi-node implementations described by Gramacy, Niemi, and Weiss (2015). Our supplementary material provides further comparison to GPU timings (SM, sec. 2), a variation involving modeling with separable covariance structures (locally and globally) (SM, sec. 3), an expansion of benchmark datasets (SM, sec. 4), and a study of how performance varies depending on input dimension and number of rays (SM, sec. 4).

All local designs are coupled with local inference for the lengthscale, \( \theta_{1}(x) \) \( |X_s(x) \), via Newton-like maximization of the local posterior probabilities. We also consider second-stage re-designs where subdesigns were recalculated based on the local parameter estimates obtained from the first sub-designs, that is, computing \( X_s(x)\theta_{a}(x) \) after \( \theta_{a}(x)X_s(x) \). NN comparators do not benefit from a second-stage redesign, since the NN set is independent of \( \theta \). All exhaustive search variations use a limited set of NN candidates for each \( x \) of size \( N' \ll N \) to keep computational costs manageable for those comparators. We primarily use \( N' = 1000 \), however there are some exceptions to match experiments reported in earlier work. No such limits are placed on ray-based searches, however snapping to the nearest element of the candidate set could be sped up slightly with similarly narrowed search scope. Unless an exception is noted, each ray-based search in our experiments uses \( p \) rays for \( p \)-dimensional \( X_N \), however in the software this is a knob that can be adjusted by the user.

5.1 A Synthetic 2D Dataset

Consider a synthetic two-dimensional (2D) dataset, whose \( 201 \times 201 \) gridded input design \( X_N = \{ 0, 0 \}^{40,401} \) was used for illustrations earlier in the article. The response follows \( f(x_1, x_2) = -w(x_1)w(x_2) \), where \( w(x) = \exp(-(x - 1)^2) + \exp(-0.8(x + 1)^2) - 0.05 \sin(8(x + 0.1)) \).

Table 1, top, summarizes a predictive experiment involving a \( 99 \times 99 \) grid of 9801 locations spaced to avoid \( X_N \). Since \( X_N \) is extremely dense in the 2D space, we opted for just one ray (rather than \( p = 2 \)) at each local design search. Using two rays gives slight improvements on root mean square error (RMSE), but requires nearly twice the computation time for the larger \( n = 200 \) experiments. A more in-depth study of how the number of rays is related to accuracy is deferred to SM Section 4. The most important result from the table is that the ray-based method requires about 10%–20% of the time compared to the exhaustive ("Ext") search, and performs at least as well by RMSE. Design by rays is also more accurate, and 10\( \times \) faster than, a large NN comparator. The simple option of randomly sub sampling a design of size 1000 ("sub"), and then performing inference and prediction under an isotropic model, requires commensurate computing resources but leads to poor accuracy results. (Estimating a separable global model on a subset of the data leads to similar accuracy/coverage results, since the data-generating mechanism has a high degree of radial symmetry, at greater computational expense required to numerically optimize over a vectorized \( \theta \) parameter.) Choosing the subset randomly also adds variability: a 95% interval for RMSE over 30 repeats is (0.0007, 0.0416). Space-filling sub designs can substantially reduce that variability but at potentially great computational cost. For example, calculating a 1000-sized maximum entropy sub design would cause compute times to go up by at least two orders of magnitude (with no change in mean performance) due to the multitude of \( K_{1000} \) calculations that would require.

Some other observations from the top of the table include that a second-stage design consistently improves accuracy, and that a large (\( n = 200 \)) search, which would require unreasonable computation (without a GPU) if exhaustive, is feasible via rays and leads to the best overall predictors. Al-
Table 1. Performance of local GP variations on simple 2D experiment. The rows of the top of the table are sorted by the (out-of-sample) RMSE column. As a point of reference, a mean-only model has an in-sample RMSE of about 0.2085. The times reported for two-stage methods include those from the first stage. The rows on the bottom of the table show the time required for variations on local design size \( n \), and NN candidate set \( N' \).

| Meth | \( n \) | Stage | Secs | RMSE | SD | 95%c |
|------|-------|-------|------|------|----|------|
| Ray  | 200   | 2     | 2948.9 | 0.0002 | 0.0048 | 1.00 |
| Ray  | 200   | 1     | 1695.4 | 0.0004 | 0.0047 | 1.00 |
| Ray  | 50    | 2     | 87.8  | 0.0008 | 0.0061 | 1.00 |
| Ext  | 50    | 2     | 436.8 | 0.0008 | 0.0058 | 1.00 |
| Ray  | 50    | 1     | 45.6  | 0.0009 | 0.0061 | 1.00 |
| NN   | 200   | 1     | 710.3 | 0.0010 | 0.0027 | 1.00 |
| Ext  | 50    | 1     | 390.4 | 0.0010 | 0.0060 | 1.00 |
| NN   | 50    |       | 11.5  | 0.0023 | 0.0045 | 1.00 |
| Sub  | 1000  |       | 50.7  | 0.0369 | 0.0398 | 0.94 |

| n  | \( N' \) | Ext | C/GPU | Rays | iMac |
|----|---------|-----|------|------|------|
| 50 | 1000    | 91  | 46   |      |      |
| 50 | 2000    | 120 | 46   |      |      |
| 128| 2000    | 590 | 377  |      |      |

though some methods report lower predictive SD than others, note that local approximations yield perfect coverage, which is typical with deterministic computer experiment data. The “sub” comparator achieves coverage close to the nominal 95% rate, but that ought not impress considering its poor RMSE results. Achieving nominal coverage is easy at the expense of accuracy, for example, via constant \( \mu \) and \( \sigma^2 \) under a global Gaussian model. Achieving a high degree of accuracy at the expense of over-covering, and hence obtaining a conservative estimate of uncertainty, is a sensible tradeoff in our context.

Gramacy, Niemi, and Weiss (2015) reported relative speedup times for a variant on the above experiment using dual-socket 8-core 2.6 GHz 2013-model Intel Sandy Bridge compute cores, and up to two connected Nvidia Tesla M2090 GPU devices, which massively parallelized the exhaustive search subroutine. The best times from those experiments, involving 16 threads and both GPs and one-stage design, are quoted on the bottom of Table 1. The accuracy of the predictions follows trends from the top of the table, however the timing information reveals that searching via rays is competitive with the exhaustive search even when that search subroutine is offloaded to a GPU. As the problem gets bigger (\( n = 128, N' = 2000 \)), the GPU exhaustive search and CPU ray-based search are competitive, but the ray method does not require limiting search to the \( N' \) nearby locations.

5.2 The Borehole Data

The borehole experiment (Worley 1987; Morris, Mitchell, and Yi Visiaker 1993) involves an eight-dimensional input space, and our use of it here follows the setup of Kaufman et al. (2012); more details can be found therein. Table 2 shows the results of an experiment mimicking one from Gramacy, Niemi, and Weiss (2015): out-of-sample prediction with problems of increasing size \( N \). The designs and predictive sets (also of size \( N' \)) are from a joint random Latin hypercube sample. As \( N \) is increased so is the local design size \( n \), and that there is steady reduction of out-of-sample MSE down the rows of the table. Gramacy, Niemi, and Weiss also increased \( N' \), the size of the local candidate set, with \( N \), however the ray searches are not limited in this way.

First focus on the timing and accuracy (MSE) results in the “iMac via rays” columns. The four columns to the left summarize the earlier experiment(s). By comparing time columns, we see that the 4-core hyperthreaded iMac/rays implementation is about 2.5× times slower than the combined effort of 80 cores and 2 GPs, or about 11.5× times slower than using more than 1500 cores. The amount of computing time for the largest run, at 9 hr for more than one-million predictions on a one-million-sized design, is quite impressive for a nearly 5 year old desktop, compared to modern supercomputers (in earlier columns). However, looking at the accuracy column(s), we see that while the ray-based search is giving nearly identical results for the largest emulations, it is less accurate for the smaller ones. We attribute this to the size of the input space relative to the density of the smaller design(s), with two implications: (1) when designs are small, searching exhaustively is cheap, so a larger candidate set (\( N' \)) relative to design size (\( N \)) can be entertained; (2) at the same time, since the design is sparse in high dimension when \( N \) is small, the rays intersect fewer candidates, reducing the chances that the \( x_{j+1} \) is close to the solution \( x_{{j+1}} \) found along ray(s). In a large-\( p \) small-\( N \) setup, it may be best to search exhaustively.

The final pair of columns in Table 2 show timings and accuracies for runs with rays distributed over 96 Intel/Sandy Bridge 16-core machines. Observe how the running times are comparable to the exhaustive version, shown in fourth column, until the pair (\( N = 32,000, n = 50 \)). Neither method is fully utilizing the massively parallel potential of this supercomputer on “smaller” data. The exhaustive method is more accurate, if just slightly slower up until this point, so perhaps that method may be preferred in the “smaller” (but still huge for GPs) setting. But then the timings diverge, with rays showing big efficiency gains and nearly identical accuracy. By (\( N = 1,024,000, n = 60 \)) rays emulate more than 20 times faster. We then allowed rays to explore larger problems, to see what size data could be emulated in about an hour, ultimately finding that we can accommodate an 8× larger experiment while allowing the approximation fidelity to rise commensurately up to \( n = 66 \). This computational feat is unmatched in the computer modeling literature, whether via GPs or otherwise.

6. DISCUSSION

Approximate Gaussian process (GP) prediction is becoming essential for emulating simulation experiment output of the size generated by modern supercomputers. In this article, we focused on a local approach to emulation, building up local subdesigns containing input locations nearby to where predictions are required. By empirically studying the topology of surfaces depicting potential for reduction in predictive variance, the main
objective behind the local design scheme, we discovered a regularity that was not being exploited by an exhaustive search. Instead, we suggested a continuous 1D line search along rays emanating from the predictive location of interest, mimicking patterns of local designs observed in practice. We then showed, empirically, how the resulting ray-based designs yielded comparable out-of-sample predictive performance in a fraction of the time.

We discussed how conclusions from the exploratory analysis in Section 3 were independent of the choice of correlation function $K_\theta(\cdot, \cdot)$, in part justifying our use of the simple isotropic Gaussian family for our empirical work. We remarked that this choice only constrains the process locally. Global predictive properties may still be nonisotropic, and even nonstationary if parameterization is allowed to vary over the input space; see Gramacy and Apley (2015) for further discussion. When the response surfaces are highly anisotropic on a global scale, like with the borehole data, prescaling the inputs based on a crude global analysis can yield substantial improvements (see SM, sec. 3); further discussion is offered by Gramacy et al. (2015) and in the laGP software vignette (Gramacy 2015). There are many other situations in which the data might be better modeled by richer families of $K_\theta(\cdot, \cdot)$, and these certainly would impact local designs. Our supplementary material provides trajectories of local designs, like the ones in Figure 2, for several relevant choices. Qualitatively however, similarities outnumber differences. Ribbons and rings are still present in the reduction in variance surface, and the designs that result are still a mix of NNs and satellite points.

Finally, while searching with rays led to big speedups and accurate approximate emulation for large designs, such schemes are not always appropriate. We saw that when designs are small and the dimension is large—meaning that the density of points in the space is low—searching exhaustively is likely to lead to better out-of-sample prediction and, in the supercomputing setup, better value for a computational budget. Increasing the number of rays may help improve on accuracy, although the extra computation may not be warranted in small-to-moderately sized emulation problems. We also remark here that another disadvantage, especially in the distributed supercomputing context, is that ray-based searches can have uncertain times to convergence. We have observed that the slowest searches can require twice the number of objective (reduction in variance) evaluations than the average over a large set of problems ranging over both $j$ and $\mathbf{x}$. The exhaustive search, by comparison, has a deterministic runtime assuming operating system “noise” is low. Random convergence times can present a load balancing challenge when trying to make the most out of a supercomputing resource. However, as the timings in Table 2 show, the ray-based searches are so fast for the largest problems that perhaps such a small inefficiency might go unnoticed.

**SUPPLEMENTARY MATERIALS**

Our online supplement is separated into four sections. Section 1 shows detailed local sequential design trajectories and reduction in variance surfaces. These include the complete trajectories corresponding to the snapshots shown in Figure 2, as well new sets under an alternative (less smooth) covariance specification. SM Section 2 provides a comparison, primarily pitting ray search on a desktop to exhaustive search on a GPU-enhanced cluster, on a real benchmark dataset from previous studies—the Langley Glide-Back Booster. SM Section 3 augments the comparison from Section 5.2 on the borehole data to include
full-data GP comparators on random data subsets, and separable covariance variations of ray-based local methods and random subset predictors. Finally, SM Section 4 provides enhanced comparison on new benchmark datasets emphasizing predictive accuracy versus computation time in several variations including isotropic/separable modeling, and number of rays versus input dimension.

ACKNOWLEDGMENTS

This work was completed in part with resources provided by the University of Chicago Research Computing Center. The authors are grateful for the valuable comments of an anonymous referee, which led to substantial improvements in the revised draft.

[Received August 2014. Revised February 2015.]

REFERENCES

Berger, J., De Oliveira, V., and Sanoso, B. (2001), “Objective Bayesian Analysis of Spatially Correlated Data,” Journal of the American Statistical Association, 96, 1361–1374. [295]

Brent, R. (1973), Algorithm for Minimization Without Derivatives, Englewood Cliffs, NJ: Prentice-Hall. [300]

Chen, H., Loeppky, J., Sacks, J., and Welch, W. (2015), “Analysis Methods for Computer Experiments: How to Assess and What Counts?” Statistical Science. [294]

Chipman, H., Ranjan, P., and Wang, W. (2012), “Sequential Design for Computer Experiments With a Flexible Bayesian Additive Model,” Canadian Journal of Statistics, 40, 663–678. [294]

Cressie, N. (1991), Statistics for Spatial Data (revised ed.), New York: Wiley. [296]

Cressie, N., and Johannesson, G. (2008), “Fixed Rank Kriging for Very Large Data Sets,” Journal of the Royal Statistical Society, Series B, 70, 209–226. [295]

Datta, A., Banerjee, S., Finley, A. O., and Gelfand, A. E. (2015), “Hierarchical Nearest-Neighbor Gaussian Process Models for Large Geostatistical Datasets,” Journal of the American Statistical Association, to appear, DOI: 10.1080/01621459.2015.1044091. [296]

Eidsvik, J., Shaby, B. A., Reich, B. J., Wheeler, M., and Niemi, J. (2014), “Estimation and Prediction in Spatial Models With Block Composite Likelihoods,” Journal of Computational and Graphical Statistics, 23, 295–315. [294]

Emory, X. (2009), “The Kriging Update Equations and Their Application to the Selection of Neighboring Data,” Computational Geosciences, 13, 269–280. [296]

Franey, M., Ranjan, P., and Chipman, H. (2012), “A Short Note on Gaussian Process Modeling for Large Datasets Using Graphics Processing Units,” Technical Report, Acadia University. arXiv: 1203.1269 [294,295]

Gramacy, R. (2015), “laGP: Large-Scale Spatial Modeling via Local Approximate Gaussian Processes in R,” Journal of Statistical Software; available as a vignette in the laGP package. [295,302]

Gramacy, R., Bingham, D., Holloway, J. P., Grosskopf, M. J., Kuranz, C. C., Rutter, E., Tramham, M., and Drake, P. R. (2015), “Calibrating a Large Computer Experiment Simulating Radiative Shock Hydrodynamics,” Annals of Applied Statistics. [302]

Gramacy, R., and Lee, H. (2012), “Cases for the Nugget in Modeling Computer Experiments,” Statistics and Computing, 22, 713–722. [295]