Non-smooth Bayesian Optimization in Tuning Scientific Applications

The International Journal of High Performance Computing Applications
XX(X):1–32

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Hengrui Luo[1,3], Younghyun Cho[2] and James W. Demmel[2], Igor Kozachenko[2], Xiaoye S. Li[1] and Yang Liu[1]

Abstract

Tuning algorithmic parameters to optimize the performance of large, complicated computational codes is an important problem involving finding the optima and identifying regimes defined by non-smooth boundaries in black-box functions. Within the Bayesian optimization framework, the Gaussian process surrogate model produces smooth mean functions, but functions in the tuning problem are often non-smooth, which is exacerbated by the fact that we usually have limited sequential samples from the black-box function. Motivated by these issues encountered in tuning, we propose a novel Gaussian process model called a clustered Gaussian process (cGP), where the components are dynamically updated by clustering. In our studies, the performance of cGP can be better than stationary GPs in nearly 90% of the experiments and better than non-stationary GPs in nearly 70% of the repeated experiments while requiring less computational cost. cGP provides a novel approach for dynamic GP, computes more efficiently than recursive partitioning, and discovers non-smoothness regimes. We provide extensive experiments including high-performance computing (HPC) and industrial simulation functions to show the effectiveness of our methods.

Keywords

Bayesian optimization, auto-tuning, non-smooth, surrogate modeling, Gaussian regression models

1 Introduction

In the realm of high-performance computing (HPC), performance model tuning is critical to leverage the full potential of computing resources. Bayesian optimization (BO) is an effective method for parameter tuning in HPC. BO employs probabilistic models, typically Gaussian Processes, to predict performance and iteratively samples parameters where the expected improvement is maximum. This intelligent sampling strategy aids in efficiently exploring the parameter space and reducing the number of expensive evaluations of "black-box" performance models.

Bayesian optimization operates within a "surrogate modeling optimization framework." Here, the term "surrogate modeling" refers to the practice of approximating expensive or complex black-box functions f with simpler, computationally cheap models. These surrogate models (Gramacy 2020) are designed to mimic the behavior of the actual function and provide insights into its properties without having to evaluate it frequently. By iteratively refining the surrogate model using data pairs $(\boldsymbol{x}, y) = (\boldsymbol{x}, f(\boldsymbol{x}))$ from the true function f, Bayesian optimization can identify optimal or near-optimal parameters with fewer function evaluations. This approach is particularly useful in scenarios where function evaluations are costly, such as in computer simulations or HPC tasks. The cited literature provides in-depth explorations of these techniques and their applications (Booker et al. 1999; Snoek et al. 2012; Shahriari et al. 2016; Gramacy 2020). The sequential sampling scheme is guided by maximizing the acquisition function (Gramacy and Polson 2011; Gramacy and Ludkovski 2015). *Gaussian process* (GP) modeling is a predominantly popular choice for building surrogate models g in Bayesian optimization (Liu et al. 2021; Cho et al. 2023a,b). We sequentially draw expensive samples from the black-box function f and update the surrogate model g with the samples drawn. If the surrogate model g approximates the black-box function f sufficiently well, we can find the true optimum $f_{\max} = \max_{\boldsymbol{x}} f(\boldsymbol{x})$ of the black-box function in the sense that we can find a configuration $\hat{\boldsymbol{x}}$ whose function value $f(\hat{\boldsymbol{x}})$ is close to the actual $f_{\max}$.

The main application that motivates us is the tuning problem in computational science. In the tuning problem, an expensive black-box performance function f (e.g., the running time of a complicated simulation) has a moderate dimensional (e.g., 4- and 7-dimensional spaces in the tuning context as shown by later examples in Section 4) variable $\boldsymbol{x}$ (e.g., a set of algorithmic parameters) that needs to be selected to maximize the performance. In addition, another important challenge is the potential non-smoothness of the black-box performance function. Although our method offers a distinct focus in the realm of BO with GP for HPC tasks, the non-smoothness in this framework has not been addressed before. Unlike traditional approaches,

[1]Lawrence Berkeley National Laboratory, Berkeley, CA
[2]University of California, Berkeley, Berkeley, CA
[3]Rice University, Houston, TX

Corresponding author:
Hengrui Luo, Rice University, Houston, TX, USA,
Email: hl180@rice.edu

our technique harnesses advanced probabilistic modeling and optimization strategies, ensuring enhanced performance and accuracy. Notably, our method stands out in Bayesian optimizations with a focus on potentially non-smooth functions arising from HPC applications while the state-of-the-art BO methods predominantly handle smooth black-box functions (Klus et al. 2021; Maddox et al. 2021; Lamparth et al. 2022; Raponi et al. 2020; Schweidtmann et al. 2021). While traditional roofline and other performance models are constructed using static datasets and often maintain costly, comprehensive approximations, our dynamic GP model is tailored for less understood mechanisms with noisy data and focuses on near-optimal performance approximation, sacrificing precision in less relevant parameter regions. This is distinct from other works which address performance modeling in dense linear algebra (Benner et al. 2016; Peise et al. 2015; Alonso et al. 2015), or tuning scientific codes for scalability and performance (Calotoiu et al. 2013; Fabregat-Traver et al. 2016), where both lines of research will require a much bigger budget for training. Traditional HPC tuning often relies on heuristics or exhaustive search, lacking inherent handling of uncertainties. In contrast, Bayesian Optimization (BO) for HPC employs probabilistic models, efficiently balancing exploration and exploitation, scaling well with high-dimensional spaces, and offering flexibility across diverse HPC challenges. The GP model usually "over-smooths" the data (Cramér and Leadbetter 2013), and has difficulty in capturing potential non-smoothness in the black-box performance function. This feature of GP surrogates makes it difficult to model performance functions in tuning problems, where non-smoothness determines the change of regimes (i.e., different partitions) in performance. The tuning problem brings us the following challenges and limitations that demand new ideas in building surrogates. Our contributions in each area are summarized below.

- *Limited samples.* For tuning problems, we need to fit a GP surrogate model with limited sequential samples due to the costly evaluation of the function associated with the application. Limited and sequential samples make fitting GP surrogate models more difficult and dependent on the choice of samples in each sampling step. With limited samples, it is difficult to estimate all the parameters introduced by over-parameterized (non-stationary) kernels.
  Our first contribution is that our model allows dynamic updates of the partitions and can learn from limited samples, outperforming some simple stationary and non-stationary GPs.

- *High-dimensional generalization.* Bayesian optimization for the tuning problem usually has a natural moderately high-dimensional domain (e.g., $\mathbb{R}^d$ with $d \geq 3$), but it is not immediately clear how to generalize these existing partition-based GP methods (Gramacy and Apley 2015; Herlands et al. 2016) into a high-dimensional domain. Besides, the number of parameters in surrogate models also grows quickly in high dimensions.
  Our second contribution is that our model utilizes the input-response pairs and models with independent GP components, and hence scales up naturally to high-dimensional large datasets.

- *Restrictive partition shape.* Most existing partition-based GP methods have quite restrictive partition shapes (i.e., determined by a system of inequalities $t_i^- \leq x_i \leq t_i^+, i = 1, \cdots, d$. (Gramacy and Lee 2008; Chipman et al. 1998)). When the true non-smooth partition boundaries are not aligned with the coordinate axes, rectangular partitions would not be appropriate for non-smoothness modeling since we cannot expect non-smoothness along rectangular boundaries.
  Our third contribution is that our model can discover the regimes defined by non-smooth boundaries with flexible boundaries based on the novel cluster-classification scheme. The model is particularly suitable for the online context, as an alternative to recursive partitioning (Gramacy and Apley 2015; Chipman et al. 2010).

Our approach to this problem is an innovative local GP (Gramacy and Apley 2015) surrogate model, where the partition shape is flexible and dynamically updated along with the sampling. The cGP model offers a structured representation of the function's non-smooth regimes, making this approach distinct. In practice, we find this dynamic partition approach more efficient compared to non-stationary kernels. This novel surrogate not only improves performance but also identifies non-smooth regimes well.

As motivating applications, we consider both univariate and multivariate performance functions that arise in specific tuning problems (e.g., matmul and SuperLU). For instance, in a matrix multiplication (matmul) problem $AB = C$, the practice of blocking large matrices A, B, C and performing block-wise multiplication is done to improve the computational performance (Hong and Kung 1981; Blackford et al. 2002; Bilmes et al. 1997; Whaley et al. 2001; Zee et al. 2016). The black-box function f is the measured computational speed (`Mflops/s`) of this matmul operation, which is determined by the efficiency of the block matrix multiplication algorithm along with the potential measurement noise. The tuning parameter in this problem is the block size for these blocked matrices. The optimal block size is machine-dependent since processor cache sizes can differ, and depends on the structure of the innermost loops. We want to "tune" the optimal block size to attain the fastest matmul for large matrices. However, for large matrices, we cannot exhaustively try every possible block size with a limited budget but tune with a limited budget.

1.1 Non-smoothness in the black-box function

Although different choices of covariance kernels would adjust the smoothness of the surrogate model g, a GP surrogate would not accurately model non-smoothness. In Figure 1, we show the fitted mean function of GP and GP surrogate models (with a deterministic partition) in the unrealistic best-case situation where we know the exact location of the discontinuity. All models are based on the same 10 equally spaced pilot samples drawn from equally

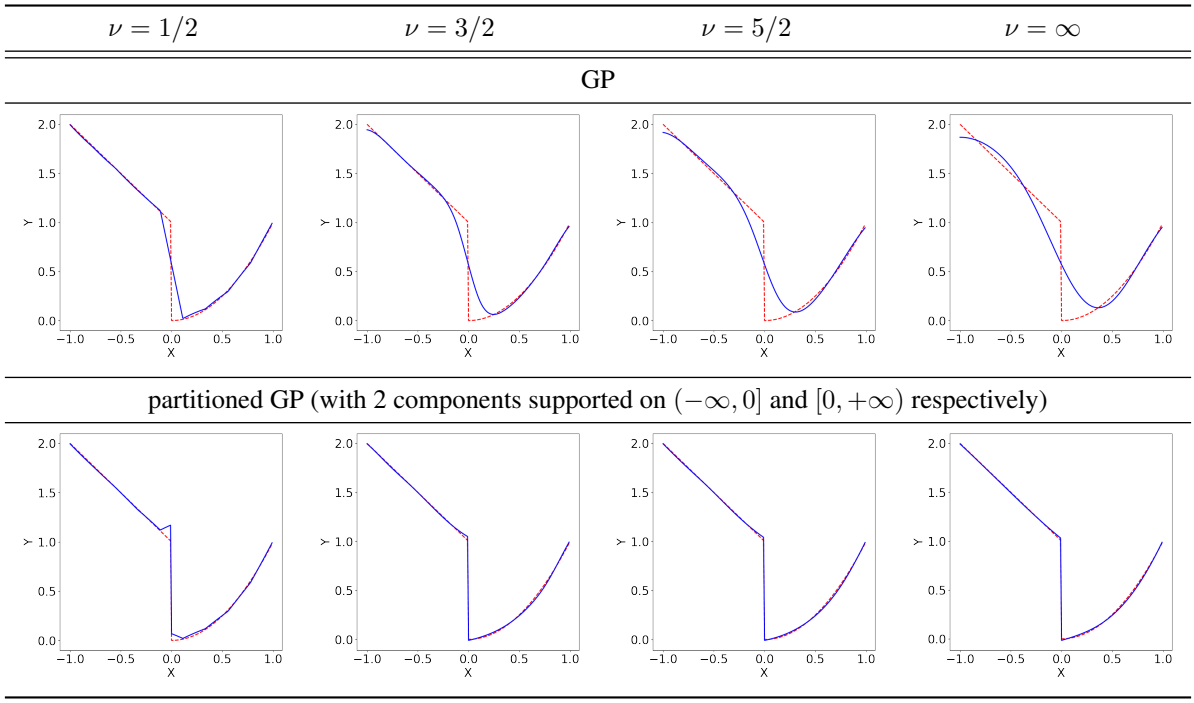


Figure 1. Mean function extracted from the fitted GP with Matérn kernels of different degrees of freedom (first row) and partitioned GP (second row, with components supported at $(-\infty, 0]$ and $[0, +\infty)$ respectively) surrogate model with 10 equally spaced samples on $[-1, 1]$ from f_1. The blue solid line is the fitted surrogate mean function and the red dashed line is the true function f_1.

spaced locations on $[-1, 1]$ from f_1 shown in Figure 1, with Gaussian noise with variance 0.01. We provide surrogate fits with Matérn covariance kernels with parameters $\nu = 1/2, 3/2, 5/2, \infty$, which adjusts the smoothness of the GP surrogate.

For the simple GP surrogate models displayed in the first row, near the non-smooth point $x = 0$, we see a lack-of-fit between the black-box function and the fitted mean function of the surrogate model. We also fit an GP surrogate model $g = g_1 + g_2$ where g_1 and g_2 are supported on $(-\infty, 0]$ and $[0, +\infty)$ respectively. From the second row of the figure, we can see that this approach captures the non-smoothness that occurs at $x = 0$, and the minimum obtained from g is closer to the true minimum at 0 than for a single GP with any kernel. An important observation is that an appropriate partition of the input domain helps the surrogate model capture the non-smoothness, compared to stationary and non-stationary (Noack et al. 2023) GP surrogate models and tree-process models (Forest (random decision tree forest, (Geurts et al. 2006)), GBRT (Ke et al. 2017)) that are specifically designed to handle non-smoothness in fixed datasets as defined next (Head et al. 2018). The existence of non-smoothness in f presents challenges to the optimization since the optima may no longer be close to the true optimum when the black-box functions are non-smooth as shown in Figure 1 and Table 1.

Definition 1. *(Non-smoothness) Letting f be a function of variable $\boldsymbol{x} \in \mathbb{R}^d$, we define a* non-smooth point $\boldsymbol{x}_0$ *of f to be a point whose first-order gradient $\nabla_{\boldsymbol{x}} f$ is either unbounded or does not exist at some point $\boldsymbol{x}$ in any open neighborhood of $\boldsymbol{x}_0$.*

As a proxy for this formal mathematical concept, we consider those points x_2 near a (fixed) point x_1 at which the finite difference $|f(x_1) - f(x_2)|/\|x_1 - x_2\|$ exceeds a certain threshold to be "non-smooth" but those points where the finite difference ratio falls below that threshold to be "smooth". Such a consideration motivates our later construction where we use $(x, \xi y)$ as the clustering criterion in our algorithm, with a factor ξ for threshold adjustment. Detailed discussions are delayed to Section 2.2.

In the surrogate modeling context, we only take finitely many sample points. The non-smoothness must be reflected by the surrogate model, not the discrete sample points. This conflict between mathematical and modeled non-smoothness cannot be resolved without high-order derivative modeling (e.g., Solak et al. (2003)).One solution is to have yet another GP surrogate to model first-order derivatives and then use that as a criterion for clustering. This scheme can be generalized to higher order derivatives.

The algorithm we introduce later does not use any formal definition directly, but we are motivating the choice of the clustering criterion $(\boldsymbol{x}, \xi y)$ (for the moment $\xi = 1$, and its role will be clear later) using a loose definition of non-smoothness, which will enable the surrogate fitting to take non-smoothness into account. Note that, although the multiplier ξ allows us to treat the input x and output y asymmetrically, the actual value of ξ is not the main factor that makes our method outperform others. The main contributing factors are $\xi \neq 0$ and the introduction of the cluster-classification scheme using *both* x and y. This practice helps us in this task of non-smoothness detection. Our work is different from the past work using data-based cluster GP like Nguyen-Tuong et al. (2009), Liu et al. (2015) and Sung et al. (2019). All three of these papers can be considered as special cases of our metric with $\xi = 0$, where only x is used in the clustering. But then a new location

cannot leverage the joint information provided by the pairs $(x, \xi y)$.

In addition, our work adds an additional step of classification and focuses on the online context, which is suggested as a future work in Sung et al. (2019). Another major difference is that, we separate the classification from the cluster operation, allowing a natural adoption of various classification algorithms instead of expensive cross-validation as used by Sung et al. (2019) or a predetermined number of clusters in Liu et al. (2015) (i.e., their number of local models Q). As far as we know, all the aforementioned works with localization focus on using a weighted average from different GPs instead of choosing one GP for prediction, due to the difficulty in cluster assignment at new locations. This problem is non-existent since we separate the expectation-maximization (EM) fitting step (Sung et al. 2019) into a cluster-classification regime, since we can either provide prediction classification probabilities (if using DGM as explained below) to each cluster as weights; or simply take the prediction classification labels (if using K-NN as explained below) to each cluster as assignments.

Our main contribution is the novel clustered GP (cGP) model for the new problem of non-smoothness in the tuning context. The cGP model provides a clear and interpretable approach for constructing a partitioned GP surrogate dynamically, as an alternative to the existing recursive partitioning method. The cGP model generalizes previous work by incorporating both x and y and a clear separation of cluster-classification steps that works in both offline and online contexts. In addition, our framework works well for limited samples and scales up with parallelism.

In Section 2, we describe our methodology for building the cGP surrogate model. The model leverages the partitioning scheme in Section 2.1 induced by the cluster-classification step to accommodate the potential non-smoothness as explained in Section 2.2. We review related work in Section 3. Section 4 provides simulated experiments with benchmark functions (Section 4.1) and an analysis for real-world tuning problems, including matmul in Section 1 and SuperLU (Section 4.2). We conclude with a discussion of the cGP surrogate model and future work in Section 5.

2 Methodology

2.1 Partitioning the input domain

The observation we made in Figure 1 is that by fitting partitioned Gaussian components to datasets on appropriate partitions, the non-smoothness can be captured in the fitted model. As a generalization of the partitioned GP shown in Figure 1, we believe a partition-based GP surrogate model is a good candidate for handling non-smoothness in the tuning context. With normalization of the inputs, we make the following

Assumption: Our input domain is a d-dimensional unit hypercube $H^d = [0,1]^d \subset \mathbb{R}^d$ $(d \geq 1)$ for the simplicity of discussion. Our methodology can be modified to work for $\mathbb{R}^d$ and $\mathbb{Z}^d$.

It is natural to think that different components in such a surrogate model should be fitted over different partition regions determined by the non-smooth points. For instance, in Figure 1, the only non-smooth point for f_1 is $\boldsymbol{x}_* = 0$ and its complement divides the domain $\mathbb{R}$ into $(-\infty, 0)$ and $(0, +\infty)$.

Partition methods are popular in modeling irregular patterns. For example, a tree-like partition is utilized in order to handle non-stationarity in GP modeling (Chipman et al. 1998, 2010; Luo et al. 2022). Furthermore, overly parameterized change-surface models (Martinez-Cantin 2015; Herlands et al. 2016) can be perceived as partitioning by weight functions or change-surfaces. The effect of partitioning is restricting each surrogate model component to the corresponding piece of the domain. Existing partition-based GP models are mostly for offline situations, but in our context below, the partition is dynamically updated along with the sequential sampling.

We want to point out that it is not straightforward to put every partition-based GP into the online context.

One way is to use the partition-based GP as the simple GP in the acquisition function. From our experiments, this makes the sequential samples concentrate near local optima (within one partition) and ignore the non-smoothness and heterogeneity in variances (across different partition components).

Another way is to extract the partition and use our Algorithm 1 to re-weight the acquisition function to force sequential samples to explore all available partitions. This is more sophisticated in implementation, and for both tree-based multi-resolution GP (Gramacy and Apley 2015) and local (non-compactly supported) GP (Broderick and Gramacy 2010) this partition extraction cannot be implemented in an online fashion trivially.

Therefore, we only compare to simple GP, with different choices of kernels, which is the well accepted baseline for surrogate-based optimizations.

2.2 Clustered GP (cGP) surrogate model

2.2.1 Dynamic partitioning via input-response pairs.

Partitioning is central to our solution to the problem of non-smoothness of the black-box function, where we use the input-response pair $(\boldsymbol{x}, y)$ to dynamically update the partition at each iteration.

The proposed surrogate model uses the *decision boundaries* (of classifiers trained by cluster results on observations) to model partition boundaries directly, with a modified acquisition function weighted by the cluster sizes used in sequential sampling. We can introduce clustering based on the joint input-response pair $(\boldsymbol{x}, \xi y)$ in accordance with our sample-based definition of non-smoothness.

When $\boldsymbol{x}_0$ is not a non-smooth point, by our Definition 1, we have a small ϵ-neighborhood of $\boldsymbol{x}_0$ and in this neighborhood the gradient of f exists and is bounded. Given our assumption that our domain is H^d for continuous functions, we can find an $\boldsymbol{x}^*$ in this neighborhood such that

$$|f(\boldsymbol{x}_1) - f(\boldsymbol{x}_2)| \leq \|\boldsymbol{x}_1 - \boldsymbol{x}_2\| \cdot \|\nabla_{\boldsymbol{x}^*} f(\boldsymbol{x})\|, \qquad (1)$$

The rationale for using the joint input-response pair $(\boldsymbol{x}, y), y = f(\boldsymbol{x})$ to determine whether $\boldsymbol{x}_0$ is a smooth point is that, if we can find such a pair of $(\boldsymbol{x}_1, y_1)$ and $(\boldsymbol{x}_2, y_2)$ that $\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ is small but $|y_1 - y_2|$ is large then the necessary condition for some point $\boldsymbol{x}_0$ being a smooth point is violated. Therefore, $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2)$ should belong to different clusters.

To summarize, desirable properties are satisfied if we set up the clustering method in our proposed cluster-classification approach as follows.

1. When $\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ is small, but $|y_1 - y_2|$ large, we put $(\boldsymbol{x}_1, y_1)$ and $(\boldsymbol{x}_2, y_2)$ into two distinct clusters.

2. When $\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ is small, and $|y_1 - y_2|$ is small, we expect $(\boldsymbol{x}_1, y_1)$ and $(\boldsymbol{x}_2, y_2)$ to belong to the same cluster.

3. When $\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ is large, the value $|y_1 - y_2|$ can be small or large, regardless of smoothness of f. So we do not make a decision about whether $(\boldsymbol{x}_1, y_1)$ and $(\boldsymbol{x}_2, y_2)$ are in the same cluster.

One of the main changes of perspective we propose is to use not only the input $\boldsymbol{x}$ but also the corresponding response y as clustering criteria for the clustering step. Based on these considerations, we propose clustering the $(d+1)$-dimensional pairs $(\boldsymbol{x}, y), \boldsymbol{x} \in H^d, y \in \mathbb{R}^1$ using one of the methods discussed below, and then use these cluster labels to train a classifier. This classifier would label points and induce a partition scheme on the input domain, and hence on the new locations. One advantage of such clustering is that we do not have to model the support of each partitioned component nor change-surfaces explicitly. The procedure of sequential sampling with this cluster-classification surrogate model is summarized below and in Figure 10.

Main Procedure.

1. **(Clustering step)** Cluster pairs $(\boldsymbol{x}, y)$, with the clustering method you choose (e.g., if you do not know the number of clusters, Bayesian Dirichlet GP mixture (DGM) (Teh et al. 2010) works; if you know the number of clusters, k-means (Devroye et al. 2013) works). Attach cluster labels to samples $\boldsymbol{x}$ and form classes among samples. This step labels the observed locations and takes cluster labels as class labels for each location $\boldsymbol{x}$.

2. **(Classifying step)** Classify $\boldsymbol{x}_{new}$ by a classification method trained by the labeled samples $\boldsymbol{x}$ (e.g., K-nearest neighbors (K-NN, Devroye et al. (2013)) provides a local classification; a support vector machine provides a global classification.) The new predictive locations $\boldsymbol{x}_{new}$ (without knowledge of corresponding responses) can be assigned to one and only one cluster class. In each sampling step, the clustering step (re-)labels the observations; and the classifying step is (re-)trained using the labels. The trained classifier based on the clustered samples can classify arbitrary input points.

3. **(Surrogate fitting step)** Construct GP models over the partition and fit with either a frequentist or Bayesian approach (Snoek et al. 2012; Shahriari et al. 2016).

4. **(Acquisition maximization step)** Maximize the modified acquisition functions(See details in Algorithm 1) for each component (e.g., expected improvement, which uses maxima of the black-box function) using a selected (gradient-based or global) optimization algorithm to determine the next sequential sample.

Here we want to revisit the cluster criterion $(\boldsymbol{x}, \xi y)$ proposed in Section 1.1. For bounded data, where we know the upper and lower bounds of $\boldsymbol{x}$ in certain applications, we can normalize the parameter domain and then $\xi = 1$ will serve as a reasonable choice if both $\boldsymbol{x}$ and y are in the range of $[0, 1]$. However, in a black-box optimization application, a normalization is usually impossible due to the fact that new sequential samples may increase the upper bound or decrease the lower bound (for both $\boldsymbol{x}$ and y). Instead of normalizing both $\boldsymbol{x}$ and y, we introduce the multiplier ξ in order to express the "relative importance" of input and response.

For example, if the input $\boldsymbol{x}$ is at a magnitude of at most 1024 (e.g., data block size); yet the response is at a magnitude of at most 10,000 (e.g., megaflops), then we may adjust $\xi = 0.1$ for better performance in the cluster-classification step. The advantage is that this builds a more realistic cluster metric and the introduction of ξ proves to work better compared to normalizing at each sequential step.

The clustered GP method could be considered as a special partition-based local GP, where the partition is induced by dynamically updating clusters of the observed locations and their responses (i.e., the pairs $(\boldsymbol{x}, y)$) in each iteration. Alternatively, the clustered GP could also be thought of as using a mixture of compactly supported kernels, where the support is updated in each iteration. The idea of our method is to partition the domain using the decision boundaries given by a clustering method, which generalizes naturally to a domain of any dimension.

We use k-means and DGM in the cluster-classification steps for all subsequent examples. Determining the smoothness of a black-box function is not straightforward. Practically, we initiate with a reasonably large k, observing if the model with DGM converges to $k = 1$ (smooth) or $k > 1$ (non-smooth). We do not require smoothness knowledge, as cGP adaptively estimates potential non-smoothness with DGM. If the user is aware of non-smooth boundaries, employing k-means with a known k enhances convergence. However, our implementation supports a wide variety of clustering and classification algorithms used in machine learning. In the cluster-classification scheme, k-means uses the proximity of pairs $(x, \xi y)$ to cluster sampled observations; and the K-NN ($K = 1$ means that we simply allocate the new location to the nearest cluster) using only the proximity of x to classify *any* location x without the knowledge of y. Again, we want to emphasize that the choice of $\xi > 0$ and the generic cluster-classification scheme distinguish our cGP from other partition-based GPs, including Nguyen-Tuong et al. (2009), Liu et al. (2015) and Gramacy and Apley (2015).

Revisiting Table 1, we can see that cGP behaves quite differently from using a non-stationary kernel along with the usual GP surrogate model. Although the cGP indeed creates a sequence of non-stationary kernels along with the sampling, the additive form and the kernel parameter estimates of each kernel can be different from one sampling step to the next. In contrast, using a single non-stationary kernel would assume the same functional form but change kernel parameter estimates as sampling proceeds. This latter approach would not allow us to adjust the partitioning of the input domain and hence cannot learn the geometry of the underlying curve.

Table 1. Average statistics $\Delta \arg\max$ and $\Delta \max$ for the repeated 50 experiments (each experiment uses different 10 random pilot samples with 100 total evaluations) for synthetic functions f_2 and f_3. The smallest statistics (indicating the best model in terms of these metrics) in each column are shown in bold fonts. Contour plots of these functions are in Figure 8. in Appendix A. Note that for non-smooth function f_3, even with the same number of kernel parameters, the performance of non-stationary (NS) kernel (with additional additive dot product term $k_{\sigma_0}(x_1, x_2) = \sigma_0^2 + x_1 \cdot x_2$ on the original kernel) is not competitive with cGP($k = 2$, DGM) or cGP($k = 2$, k-means), both with 4 kernel parameters as shown in the $\#\kappa$ column indicating the number of kernel parameters. We also provide k-means with 2,3,4,5 clusters and DGM with 2,3,4 clusters.

Surrogate	$\#\boldsymbol{\kappa}$	f_2 (smooth)		f_3 (non-smooth)	
		$\Delta \arg\max$	$\Delta \max$	$\Delta \arg\max$	$\Delta \max$
GP ($k = 1$)	2	0.018718	0.000349	0.082762	0.006721
cGP ($k = 2$, k-means)	4	0.034503	0.001182	0.067821	0.004524
cGP ($k = 3$, k-means)	6	0.041824	0.001730	0.085938	0.007180
cGP ($k = 4$, k-means)	8	0.028472	0.000808	0.078971	0.006084
cGP ($k = 5$, k-means)	10	0.026969	0.000724	0.090119	0.007937
cGP ($k = 2$, DGM)	4	/	/	0.082752	0.006623
cGP ($k = 3$, DGM)	6	/	/	0.078508	0.006031
cGP ($k = 4$, DGM)	8	/	/	**0.067031**	**0.004412**
cGP (partitioned)	4	/	/	0.160008	0.022961
GP (NS-Matern1/2)	4	0.037845	0.001430	0.084346	0.007064
GP (NS-Matern3/2)	4	**0.011321**	**0.000128**	0.119220	0.014014
GP (NS-Matern5/2)	4	0.048933	0.002389	0.120635	0.014344
Forest	>10	0.714731	0.005801	0.661403	0.037539
GBRT	>10	0.726816	0.004925	0.602178	0.020917

Table 2. This table shows how frequently cGP performs better than the baseline GP surrogate (both with stationary Matérn 3/2 kernel) on Bukin N.6, with different exploration rates, in 100 tests with random seeds.We record the fraction of tests where cGP provides equal or better results (outside parentheses) and the fraction where cGP provides strictly better results (inside parentheses). In the last column, we compare cGP (with non-stationary Matérn 3/2 kernel) and the simple GP (with non-stationary Matérn 3/2 kernel).

k	seqSize	expRate=0.5	expRate=0.8	expRate=1.0	Non-stationary
2	10	0.96 (0.34)	0.95 (0.28)	0.98 (0.30)	0.84 (0.33)
	90	0.95 (0.80)	0.95 (0.79)	0.97 (0.78)	0.64 (0.49)
	190	0.95 (0.87)	0.94 (0.87)	0.97 (0.89)	0.64 (0.54)
3	10	0.96 (0.33)	0.96 (0.26)	0.98 (0.22)	0.91 (0.24)
	90	0.95 (0.83)	0.98 (0.85)	0.96 (0.87)	0.63 (0.49)
	190	0.98 (0.91)	0.97 (0.88)	0.95 (0.90)	0.69 (0.60)
4	10	0.94 (0.30)	0.97 (0.18)	0.99 (0.14)	0.91 (0.25)
	90	0.97 (0.87)	0.95 (0.83)	0.98 (0.89)	0.50 (0.39)
	190	0.97 (0.92)	0.98 (0.92)	0.98 (0.93)	0.71 (0.64)

2.2.2 Model specification. Following the Gaussian process modeling notation (Rasmussen and Williams 2006), the overall clustered GP surrogate model (with k clusters) for a sample $\{y_1(\boldsymbol{x}_1), \cdots, y_n(\boldsymbol{x}_n)\}$ at n different locations $\boldsymbol{X} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n\}$ takes the following vector form:

$$\boldsymbol{y}(\boldsymbol{X}) = (y_1(\boldsymbol{x}_1), \cdots, y_n(\boldsymbol{x}_n))^T = \sum_{j=1}^{k} \boldsymbol{f}_j(\boldsymbol{X}) \cdot \mathbf{1}_j(\boldsymbol{X}) + \boldsymbol{\epsilon}, \tag{2}$$

where the $\boldsymbol{f}_j(\boldsymbol{x}) = (f_j(\boldsymbol{x}_1), \cdots, f_j(\boldsymbol{x}_n))^T$ denotes the vector of the j-th scalar function evaluated at n different locations $\boldsymbol{X} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n\}$ of all k Gaussian mean components; the overall Gaussian noise $\boldsymbol{\epsilon}$ has mean 0 and variance σ_ϵ^2. The notation $\mathbf{1}_j(\boldsymbol{x})$ denotes an indicator function returning 1 if $\boldsymbol{x}$ is classified as belonging to the j-th class and 0 otherwise. The partition is determined by the support of the indicator functions and can also be updated per iteration; this dynamic update of partitioning is different from the existing GP surrogate models.

In contrast to a hierarchical partitioning model (Luo et al. 2022; Park and Choi 2010), the partitioned components of this model are mutually independent due to the disjoint partition scheme induced by our cluster-classification procedure. In the cGP model (2), each $\boldsymbol{f}_j$ is modeled by a zero-mean Gaussian process prior, i.e., $\boldsymbol{f}_j(\boldsymbol{x}) \sim N_n(\mathbf{0}, \boldsymbol{K}_j)$, where the notation $N_d(\boldsymbol{m}, \Sigma)$ denotes the d-dimensional Gaussian distribution with mean vector $\boldsymbol{m}$ and covariance matrix Σ, with covariance matrix $\boldsymbol{K}_j = [K_{\boldsymbol{\kappa}}(\boldsymbol{x}_r, \boldsymbol{x}_s) \cdot \mathbf{1}_j(\boldsymbol{x}_r) \cdot \mathbf{1}_j(\boldsymbol{x}_s)]_{r,s=1}^n$ and the covariance kernel $K_{\boldsymbol{\kappa}}$ modeling the dependence between $\boldsymbol{x}$ within the same j-th class. We use $\boldsymbol{\kappa}$ to denote all kernel parameters for kernel $K = K_{\boldsymbol{\kappa}}$ below for brevity.

The model parameters are the kernel parameters $\boldsymbol{\kappa}$ that determines K and the error variance σ_ϵ^2. One approach to fit the model is to estimate the kernel parameters by maximizing the model likelihood function defined by (2), which is usually known as the "frequentist approach" (Snoek et al. 2012, 2014; Gramacy 2020).

Our idea for the main procedure above follows the discussion on input-response pairs in Section 2.2.1, and Figure 10. We create labels for locations in the input domain based on $(d+1)$-dimensional pairs $(\boldsymbol{x}_{obs}, y_{obs})$ by (unsupervised) clustering. Then we compute the decision boundary of a classifier trained by $\boldsymbol{x}_{obs}$ and its cluster labels, to induce a partitioning of the input domain in each iteration step. We expect that sample points that belong to different regimes determined by non-smoothness would be assigned to different partition components. With this trained classifier (supervised), we can classify any point $\boldsymbol{x}_{new}$ in the H^d and form a partition over the domain.

We compute individual acquisition functions based on each partitioned component, and divide by the number of sequential samples in this component. After optimizing each acquisition individually, we pick from the component with the largest acquisition function (details are in Algorithm 1). Then, the (weighted) acquisition function (by default we use the expected improvement (EI) as the acquisition function) based on the surrogate model is maximized over each component. This weighted acquisition function can be applied to different kinds of acquisition functions like UCB (Upper Confidence Bound) or PI (Probability of Improvement) (Hernández-Lobato et al. 2014); we provide a theoretic justification of this weighting scheme in the partitioned surrogate model in Appendix B. In short, such a weighting scheme and the exploration rate below would ensure convergence of the Bayesian optimization procedure based on cGP.

We call this model the **clustered Gaussian process** (cGP) model. This model not only dynamically updates the partition according to input-response pairs but also independently fits the model components for sequential sampling.

The clustering labels define classes and can be used for training classifiers. The classifier partitions the domain H^d. In addition, we can maintain an interpretable model which indicates where the discontinuities appear. Compared to other partitioning-based methods (Gramacy and Apley 2015; Luo et al. 2022), our cluster-classification algorithm creates a novel partition scheme for the partition-based surrogate GP model and generalizes to high-dimensional domains well while fitting in parallel.

To the best of our knowledge, the dynamic partition scheme has not been used to construct a GP surrogate in an online context before (i.e., where the sampling happens sequentially). Our cGP model brings a brand new alternative to recursive local GP (Luo et al. 2022) or structured GP (Gramacy and Apley 2015). Our cGP also proposes a novel method with computational efficiency and parallelism over a typical GP with non-stationary kernels. We also want to emphasize that the purpose of cGP is not only to provide a way of dealing with non-smoothness, but also to identify where the non-smoothness occurs in the domain.

In the following Algorithm 1, we use the following notations: $\texttt{GP}(X_{n,d}, Y_n)$ is a Gaussian process surrogate model fitted with the data matrix $X_{n,d}$ and response Y_n. $\texttt{EI}(x, g)$ is an acquisition function based on the surrogate model g, evaluated at $x \in H^d$. $\texttt{CLUSTER}(X_{n,d}, Y_n, k)$ is a clustering method (e.g., k-means) performed on $X_{n,d}, Y_n$ that returns the labels of a data matrix $X_{n,d}$ as an $n \times 1$ vector. $\texttt{CLASSIFY}(X_{n,d}, (X'_{n',d}, L'_{n',1}))$ is a classification method (e.g., K-NN) that is trained by a set of labeled data $(X'_{n',d}, L'_{n',1})$; and returns the labels of a data matrix $X_{n,d}$ as an $n \times 1$ vector. `unique` is a function that returns unique values in the vector v. `CLUSTER` runs the clustering operation that reclusters and relabels all samples in X, on return the labels for these samples are computed and stored in $dataLabel$. On the entry of `CLASSIFY`, $dataLabel$ is read-only and on return j_x is the cluster that x belongs to. That means the points can be moved to a different cluster when sequential samples are appended and the partition regime can be learned dynamically, which is an important piece of information when we only have limited sampling budget in expensive scientific applications.

2.2.3 Exploration rate and penalty. In the implementation of the cGP model, we introduce the notion of exploration rate, following the practice described by Bull (2011) (in Definition 4). The *exploration rate* is the probability that the next sequential example would be sampled according to the acquisition maximization. When the exploration rate is exactly 0, the sequential samples are all randomly chosen

Algorithm 1: Clustered Gaussian process (cGP) surrogate model algorithm.

Input: $X_{n_0,d}$ (data matrix of pilot samples in H^d),
X_{new} (data matrix of predictive locations in H^d),
k (optional, the number of clusters),
τ, (the exploration rate),
$maxSampleSize$ (the maximal number of samples that can be drawn from f)
Output: list g of GP surrogate model components corresponding to clusters
$X = X_{n_0,d}$
while $i < maxSampleSize$ **do**
  Generate a random number $u \in [0, 1]$
  if $u \leq \tau$ **then**
    $dataLabel$ = $\texttt{CLUSTER}(X, Y, k)$
    $clusLabel$ = $\texttt{unique}(dataLabel)$
    (In practice, we also eliminate clusters with too small sizes to avoid mis-fits.)
    for j **in** $clusLabel$ **do**
      X_j = $X[dataLabel == j, :]$
      Y_j = $Y[dataLabel == j]$
      $g[j]$ = $\texttt{GP}(X_j, Y_j)$
      $x_0[j]$ = $\arg\max_{x \in H^d} \texttt{EI}(x, g[j])$
      $ei_0[j]$ = $\texttt{EI}(x_0[j], g[j])$
      (The $\texttt{CLASSIFY}(x, (X, dataLabel))$ is used to predict the point partition label when we evaluate the acquisition at a certain point.)
      j_0 = $\arg\max_{j \in clusLabel} ei_0[j]/|dataLabel == j|$
      Append $x_0[j_0]$ and function values $f(x_0[j_0])$ to X, Y
    end for
  else
    Append random location $X_i = x1$ and its function values $Y_i = f(x1)$ to X and Y
  end if
end while
$dataLabel = \texttt{CLUSTER}(X, Y, k)$
for x **in** X_{new} **do**
  $j_x = \texttt{CLASSIFY}(x, (X, dataLabel))$
  Predict the value of surrogate function at x using GP model $g[j_x]$.
end for

without referring to the acquisition function at all. When the exploration rate is exactly 1, the sequential samples are all sampled according to the maximizer of the acquisition function. We introduce this element to allow different sequential sampling schemes and investigate its effect in Section 4.1. The exploration rate is set to be 1.0 by default.

Another adjustment we introduce in our implementation is to add a boundary penalty function to the acquisition function (Park et al. 2011). In the problem of handling non-smoothness, it seems that it is of benefit to sample more near the boundary of the partition. This penalty function can also be updated sequentially to account for the updated partitions as well. We did not use any penalty in the current paper, but recommend a careful penalty adjusted to the design and constraints of the domain or no penalty by default.

We also point out that cGP is a GP within a single component by its construction. In an extreme situation, we pick only two points, $x_1 = -0.5$ and $x_2 = 0.5$ for the f_1 function with $f_1(x_1) = 1.5, f_1(x_2) = 0.25$ as shown in Figure 1. If we treat x_1, x_2 as one cluster, then we fail to capture the non-smooth point $x_* = 0$ and will have a constant mean function at 0.875. If we treat x_1, x_2 as two clusters, then we will have piecewise constant functions of values 1.5 and 0.25 respectively with a non-smooth change-point at 0. This problem is intrinsic to any partition-based method with a mis-specified partition scheme.

3 Related work for non-smooth modeling

Derivative-free optimization methods (Rios and Sahinidis 2013) are tailored for real-world non-smooth problems but may not adaptively capture the black-box function's structure. Random search with evolutionary algorithms (Bergstra and Bengio 2012), while robust, might not be adept at capturing specific non-smooth characteristics. Traditional methods like CMA-ES (Igel et al. 2007) are resilient against minor non-smoothness but might falter with abrupt discontinuities (as opposed to our adjustable ξ parameters). Adaptive sampling techniques, though efficient, might not be equipped for the intricacies of non-smooth black-box functions Zilberstein et al. (1999). From the large literature on black-box optimization, we focus on GP surrogate-based methods where the samples can be appended sequentially. Our cGP model can be considered as an innovation to GP surrogate from related work in the following three aspects.

The first line of related work is the online change-point detection problem with GP models (Adams and MacKay 2007; Saatçi et al. 2010). In a one-dimensional domain (e.g., a time-series), a *change-point* means that the function behavior is different before and after this point. The problem involving change-point detection investigates whether and when such a change of behavior occurs. The setup of sequential sampling and inference is known as *online* detection, in contrast to *offline* detection where the samples are fixed and the inference is retrospective (Basseville and Nikiforov 1993; Saatçi et al. 2010). (Smith 1975) studied the problem with a Bayesian approach, and then (Barry and Hartigan 1993) put the offline inference into a fully Bayesian framework, and an alternative Bayesian method is developed by (Adams and MacKay 2007). This kind of online Bayesian change-point model relies on the one-dimensional time domain, making it hard to generalize to higher dimensions. Our cGP model takes a different approach to clustering, which is general enough for higher dimensions.

The second line of related work are the non-stationary GP models, especially in the spatial statistics (2-dimensional domain). GP models with a covariance kernel that not only depends on pairwise distances could be referred as *non-stationary* (Paciorek and Schervish 2004). In the context of offline surrogate models, (Krause and Guestrin 2007) propose a partition mixture model to address the non-stationarity. Furthermore, (Martinez-Cantin 2015) suggest a global-local approach for potential non-smoothness. In a different direction, (Herlands et al. 2016) model general change-surfaces. Our cGP model works as well as or better for optima search in the limited sample situation, with better computational speed. Our method does not introduce extra

parameters for non-stationarity but use a more natural cluster approach, which generalizes at a lower computational cost.

The third line of related work is in the context of online surrogate models. When the derivative information is not available, (Stoyanov et al. 2017) models piecewise constant functions for cracking patterns using hierarchical grid-based methods, and generalizes to piecewise polynomials in (Stoyanov 2018). We do not see an immediate adaptation of the grid-based method into Bayesian optimization framework, which gives up the sequential sampling procedure. When the derivative information is available, (Solak et al. 2003) suggested to model observation derivatives directly with the known generative equations. However, this is not the usual situation in the tuning or data analysis context we focus on, since we usually only observe the sample pairs without higher order information.

4 Experiments

4.1 Synthetic studies.

4.1.1 Smooth function. First, we want to consider a smooth black-box function and see if cGP is competitive against (stationary and non-stationary) GP surrogates when there is no non-smoothness. In this situation, we expect a simple GP surrogate to perform the best, but expect that cGP models are also competitive. Since when we choose the number of clusters $k = 1$, the cGP model reduces to the usual GP model, we also expect that GP and cGP would have similar fits when k is small. We take the smooth function $f_2(\boldsymbol{x}) = f_2(x_1, x_2) = 1/(1 + (x_1 - 0.25)^2 + (x_2 - 0.25)^2)$.

We examine one sample of cGP fits (with Matérn $3/2$ kernels) for $k = 1, 2, \cdots, 5$ and average performance in Table 1. For Forest and GBRT models, we use the state-of-the-art implementations in scikit-optimize==0.9.0 maintained as sequel to Head et al. (2018), and the default hyper-parameters for each model. When we try to maximize the function, we observe two key metrics

$$\Delta \arg\max := \| \arg\max_{\boldsymbol{x}}(f) - \text{sample maximum point } \boldsymbol{x}\|_2$$

the ℓ_2 distance between sample maximum point $\boldsymbol{x}$ and the exact maximum point, and

$$\Delta \max := |f_{\max} - \text{sample maximal observed } y|$$

which is the difference between sample maximum function value y and the exact maximum value $f_{\max} = f(\arg\max_{\boldsymbol{x}} f)$. Smaller metrics $\Delta \arg\max$ indicate better performance of the surrogate model, which leads to a sample maximum closer to the truth. Therefore, smaller metrics $\Delta \max$ indicates the surrogate model finds a (possibly noisy) observed value that is closer to the true function value.

We can see that for this f_2 with the unique maximum 1 attained at $(1/4, 1/4)$, cGP gives worse performance than the simple GP models for smooth underlying function f_2, but they are reasonably close to simple GP on average. From this, we can see that cGP performs very closely to the simple GP surrogate as shown in Table 1.

This set of synthetic experimental results supports that the cGP method performs reasonably well when the black-box function is smooth and would not create a lot of non-smoothness in the fitted surrogate model. Averaging over multiple runs with different pilot samples leads to similar results, therefore, we conclude that adopting the cGP model would create comparable tuning results on average compared to GP surrogates even if there is no non-smoothness at all.

4.1.2 Non-smooth function. Second, we want to see how cGP performs when non-smoothness exists in the black-box function, compared to GP. We create some non-smoothness in the function f_2 along the line $x_2 = 0$. We use the indicator function $\mathbf{1}(\cdot)$ to complete the construction. $f_3(\boldsymbol{x}) = f_3(x_1, x_2) = \mathbf{1}(x_2 > 0) \cdot (1/(1 + (x_1 - 0.25)^2 + (x_2 - 0.25)^2)) + \mathbf{1}(x_2 \leq 0) \cdot (0.25/(1 + (x_1^2 + x_2^2)))$ with the same unique exact maximum point (1/4,1/4) as f_2. Therefore, if the non-smoothness introduced by the indicator functions does not affect the surrogate-based optimization at all, then we would be able to locate the maximum value of f_3 as easily as f_2. We examine the mean of cGP fits (with Matérn $3/2$ kernels) for $k = 1, 2, \cdots, 4$ using both k-means and DGM.

Since the cGP model consists of two main parts, the dynamic cluster-classification step to get the partitions and modeling on the partitions, we want to explore how the cGP model works when the exact partitioning is known, and compare to the results when the partitions must be computed. Following the practice in Figure 1, we get a partitioned cGP assuming that we know the partition is given by $x_2 = 0$.

There are several interesting observations in Table 1. On one hand, our Algorithm 1 can be shown to converge when the exact partition is fixed, see Proposition 1 in Appendix B. On the other hand, this should not be confused with the fact that Table 1 indicates the fixed partition (i.e., cGP(partitioned)) leads to the worst result.

We do not guarantee any optimal convergence rate, but point out that the convergence exists, at a certain rate. In fact, in our cGP algorithm, the partition regime changes dynamically at every sampling iteration, forcing a more radical exploration and possibly encouraging faster convergence near the boundary. This observation for online data with sequential sampling is not contradictory to the image we saw in Figure 1, where the dataset is offline and fixed. When $k = 3$, the non-smoothness at $x_2 = 0$ cannot be modeled very appropriately with either k-means or DGM (except for k=2 DGM) as we observe. The cGP with DGMs all have smaller $\Delta \arg\max$ compared to GP and reasonable improvement, while cGP models with k-means ($k = 2, 4$) and DGM ($k = 2, 3, 4$) have better $\Delta \max$ and $\Delta \arg\max$ compared to a simple GP on average as observed from Table 1. This means that with the same limited sample budget, cGP models (with appropriate clustering) get closer to the true maximum value. We observe that a cGP model with DGM performs better in terms of identifying clusters (except for k=2 DGM).

One more observation we made based on the average statistics shown in Table 1 is that we can see that compared to f_2, f_3 tends to have larger statistics, no matter what surrogate model we use. This strengthens and is consistent with our intuition established in Figure 1 that when there exists non-smoothness, the Bayesian optimization based on surrogate models would encounter difficulties due to the lack of fit near the non-smooth points. In this non-smooth example, we only change the maximal number of clusters allowed in DGM.

Furthermore, the choice of clustering algorithm in the cluster-classification step matters. When $k = 2$ the fit of cGP with k-means clustering seems to capture the non-smoothness well while retaining the behavior that is similar to the underlying function. However, in practice, we do not know how many clusters are there. One approach is to treat k as a hyper-parameter that is subject to further tuning with model selection criteria or cross-validation on real data.

4.1.3 Nearly non-smooth function. As a third example, we study a well-known benchmark function called Bukin N.6 function (displayed in Figure 9) defined on a 2-dimensional domain. For now, let us see how different exploration rates would affect the performance of cGP surrogate models in terms of $\Delta \max$. In this set of experiments, we first investigate the distribution of optima found by different surrogate models. In the first row of Figure 2, we repeat the optimization for 100 different random seeds and we can see that when there are moderate or sufficient sequential samples ($n = 90, 190$), cGP surrogates produce $f_{\min}$'s that are consistently much lower than the optima given by the GP surrogate with the same sample size. When there are only limited samples ($n = 10$), cGP seems to have small improvements.

It is not hard to see in the second row of Figure 2 that, partition schemes are actually capturing the level sets of Bukin N.6 function rather accurately and sample more near the optima. Quantitatively, in Figure 2 we provide the percentages that cGP surrogates (under different combination of exploration rates) outperform their GP counterparts among 100 runs with distinct random seeds. We not only provide the percentages that cGP gives equal or strictly smaller $f_{\min}$, but also the percentage that cGP gives strictly smaller $f_{\min}$. The percentage that cGP gives equal or strictly smaller $f_{\min}$ should be over 50% to justify the use of cGP, and a larger percentage of strictly better optima indicates the superiority of cGP. A nearly 70% supports that cGP is strictly better than GP with non-stationary kernel (Matérn 3/2 with an additive term), showing qualitative difference between GP and cGP. However, we also noted that when there are only $n = 10$ sample budgets, the non-stationary kernel is generally preferred since we do not have to identify different numbers of clusters. However, when we compare cGP with stationary kernels against GP with stationary kernel (Matérn 3/2), in almost $\geq 94\%$ of cases, cGP outperforms GP (Figure 2). The dynamic partitioning serves as a "boost" to both stationary and non-stationary kernels.

This experiment shows a consistent and qualitative advantage of cGP over GP surrogates. More importantly, it also shows that different configurations of hyper-parameters in the cGP model can be adjusted to yield better performance over the same function. The partition scheme accompanying the cGP surrogate fit also reveals the nontrivial geometric structure of the black-box function. In general, if the black-box function has optima in a relatively small region compared to the whole domain or the function has non-smoothness (also sharp changes) near optima, then the cGP surrogate may reach better optima compared to the GP surrogate, and run faster.

4.1.4 Complexity Analysis. A complexity analysis shows that constructing a simple GP has complexity $\mathcal{O}(n^3)$ while cGP with k clusters only has complexity $\mathcal{O}\left(\left(\frac{n}{k}\right)^3 \cdot k\right) = \mathcal{O}\left(\frac{n^3}{k^2}\right)$ on average, assuming that the clusters are of equal sizes on average (although this assumption may not be true in some extreme cases). Another source of complexity in cGP comes from the cluster-classification step. This step is composed of two operations. For d-dimensional dataset, the (t-step iterative) k-means clustering takes $\mathcal{O}(k \cdot n \cdot d \cdot t)$ operations (Slonim et al. 2013), while a brute-force K-NN classifier takes $\mathcal{O}(K \cdot n \cdot d)$ operations (Bhatia et al. 2010). Both algorithms are linear in the sample size n and d when we can fix or bound the number of clusters k and the number of neighbors K. The overall complexity is $\mathcal{O}(n^3/k^2 + kndt + Knd)$. When the clustering algorithm is DGM, the complexity depends on the implementation (e.g., variational inference or MCMC) but is usually comparable to k-means in practice (See Figure 11).

Therefore, although cGP is designed for a limited sample budget, it is much more scalable and parallelizable than GP with non-stationary kernels. Furthermore, cGP can identify different kinds of behavior as shown in Figure 2 rather faithfully through the cluster-classification procedure (on page 5) while the non-stationary cannot.

Furthermore, we can observe that if we allow cGP to take the same non-stationary kernels for each component, then the cGP still predominately outperforms the GP with non-stationary kernels. Our dynamic partitioning can be interpreted as one way to introduce non-stationarity into the surrogate model, yet we firmly pointed out that it is not the same as using non-stationary kernels since it changes partition each iteration. This is reflected by the fact that in roughly 70% of cases it is better than GP with non-stationary kernels and it is usually computationally faster. We will only consider the baseline comparison using stationary kernels hereafter.

4.1.5 Summary. In the smooth function example, we observe that cGP with small k performs similarly to GP. When there is no non-smoothness in the black-box function, we can still use cGP instead of GP without much loss of performance. In fact, GP is a special case of cGP with $k = 1$.

In the non-smooth function example, we observe that cGP with different k's can outperform GP when k is appropriately chosen. When there is non-smoothness in the black-box function, we can use cGP to improve optimization results. cGP would produce a partition scheme that illustrates the non-smooth behavior in the black-box function with appropriate hyper-parameters. For parameter ξ, we treat it as a data-driven normalizing constant to balance the scales of input x and response y, aligning with joint clustering. Parameter k denotes the maximum number of clusters in DGM, but adjusts dynamically during modeling. With prior knowledge, a fixed k can be chosen using k-means. The exploration rate, typically $[0.5, 1]$, helps avoid local minima. A default of $[0.95, 1]$ is recommended for optimal tuning. Furthermore, cGP shows small sensitivity to the combination of ξ, k, and exploration rates, when using DGM or k-means clustering in our algorithm, which makes it convenient to use even with default hyperparameters.

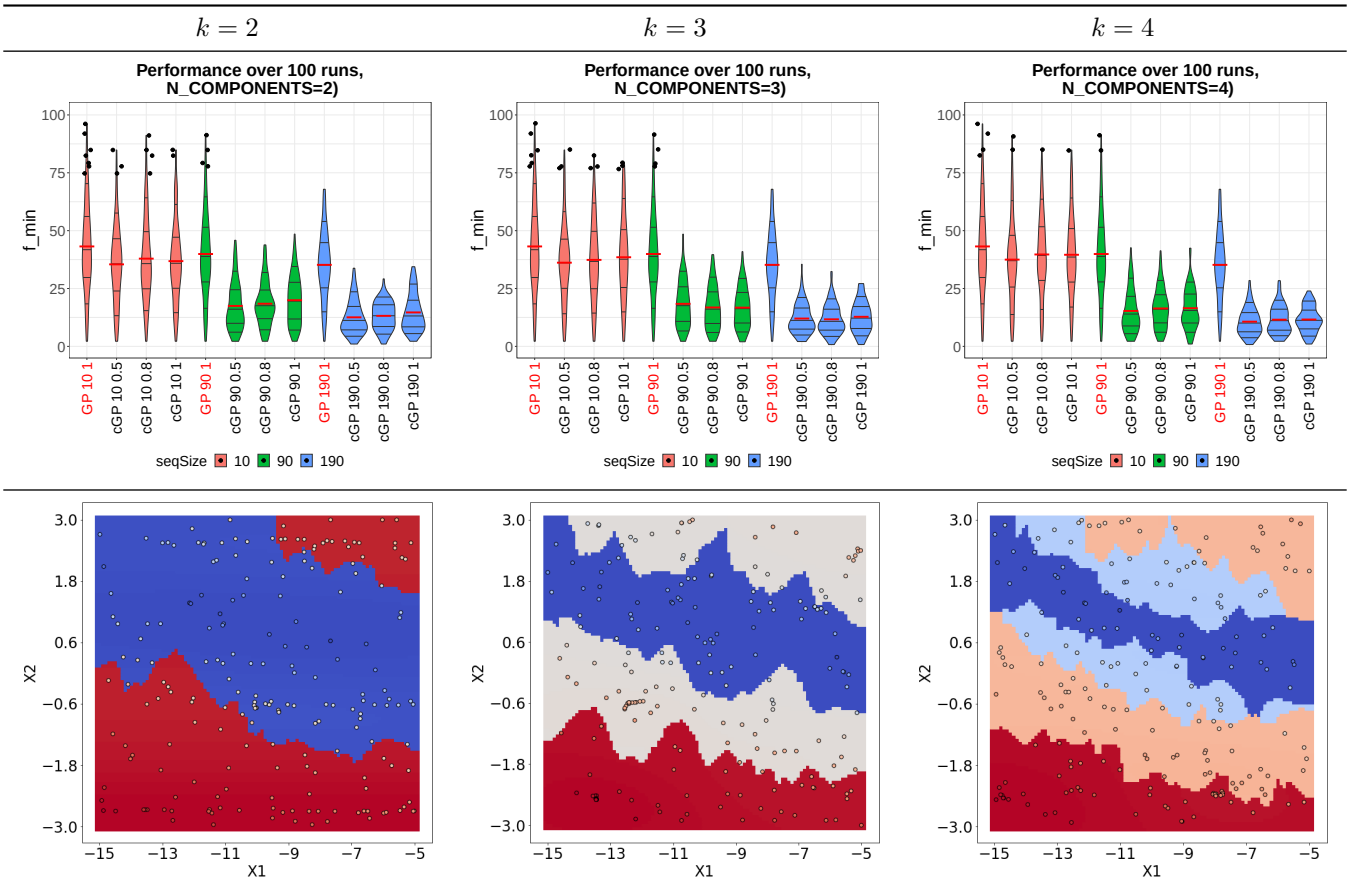


Figure 2. The violin plot of optima (minimum) obtained from the cGP surrogate model with exploration rates (τ) of 0.5, 0.8 and 1.0, 10 pilot samples and a sequential sample of size (seqSize) $n = 10, 90, 190$ from Bukin N.6 function. We display the mean using a red segment, and the performance in terms of distribution of optima from sequential samples from GP and cGP surrogate models, with different numbers of components, exploration rates and sequential sample sizes.
We also provide corresponding partition schemes (different clustering components are indicated by different colors) for one fit of cGP surrogate model with 190 sequential samples in the second row of the figure.

In the nearly non-smooth Bukin N.6 function example (Figure 9), we observe that cGP with different exploration rates and sample sizes outperforms GP under the same conditions, but different configurations of these hyper-parameters can affect the optimization performance of cGP. In this example, cGP produces a partition scheme illustrating a non-trivial structure of the black-box function and yields better optima. In terms of complexity, cGP is more efficient when using the same type of (stationary or non-stationary) kernels compared to simple GP.

cGP is computationally more efficient and has the potential of detecting the non-smoothness regime in the black-box function, and cGP usually performs at least as well as GP (with non-stationary or stationary kernels).

4.2 HPC Applications.

4.2.1 One-dimensional dense matmul tuning. Let us revisit the matmul tuning problem we mentioned in Section 1. The black-box function is the computational speed function of the operation $AB = C$ using blocked (tiled) loops, where the $n \times n$ matrices A, B, C have double precision entries, and consist of $N \times N$ blocks each of size $b \times b$ such that $n = Nb$. See the pseudo code implementation in Appendix D.1. The parameter we attempt to tune is the shared block size b to maximize the computational speed function $f(b)$.

The exact optimal value of b depends on the details of how the loops are organized, compiler optimizations, and hardware architecture. More details of optimization for matrix multiplication are explained in (Bilmes et al. 1997), (Whaley et al. 2001) and (Zee et al. 2016).

In what follows, we use matmul as a motivating example, where clear non-smooth and different regimes can be observed, for the clustered GP surrogate. cGP helps us identify the non-smooth regime, which is important in this tuning problem.

The machine on which we conduct this experiment is a Cori Haswell computing node, with an L1 cache size of 32KB, L2 cache size of 256KB and L3 cache size of 40960KB. All cache-line sizes are 64 bytes, i.e., 8 double precision words. The node has two Intel Xeon Processor E5-2698 v3 2.3 GHz processor (with a theoretical peak of 1.2T Flops) with 128GB memory. If we evaluate the computational speed of each block size b from 1 to 1000 ($n = 1000$) (when the block size is 1 or 1000, there is no blocking), we observe obvious non-smooth behaviors as the block size increases, due to the overflow of L1 and L2 cache. The L3 cache is large enough to fit all 3 1000×1000 matrices A, B and C.

A value of b that results in less than one matrix fitting leads to poor cache performance, while fitting three matrices ensures effective utilization of spatial locality. The transition between these regimes of block size b, depending on algorithmic details, reflects the intricate interplay between computational requirements and hardware constraints, leading to discontinuities in the peak flops.

The choice of b that minimizes memory traffic also (potentially) maximizes speed. Based on multiple runs, we observe that there is little noise in the data and so we do not consider averaging it further here. In this dataset, we can make preliminary observations (see Figures 3 and 12 in Appendix D) that there are several different kinds of behavior, apart from the noise:

When $b < 160$, the speed increases rapidly with some oscillations and then drops to a plateau level. The oscillations, with local peaks at b equal to multiples of 8, are to be expected since the cache-line size is 8 words. The optimal configuration is attained at $(\boldsymbol{x}_{\max}, f_{\max}) = (112, 2010.702)$, with the 2 next best configurations at $(120, 2001.35)$ and $(128, 2000.758)$. It is no surprise that the performance falls off rapidly past $b = 128$, reaching a plateau around $b = 160$. When $160 \leq b \leq 500$, the computational speed has a stable trend and rough behavior, with some fluctuations caused by memory alignment and code implementation. Again, most of the small peaks occur at multiples of 8. When $b \geq 500$, there is another significant non-smoothness (the trend can also be identified starting from $b = 400$) until a roughly constant computational speed is attained. To reduce randomness, we use the recorded dataset which is collected from historical runs instead of obtained in real-time as the black-box function f defined on $[1, 1000] \cap \mathbb{Z}$.

We can get a first impression of the difference between GP and cGP surrogates by looking at Figure 3. When the total sample size is 20 (i.e., with 10 pilot and 10 sequential samples), we can see that there is a big difference between the surrogate fits of GP and cGP, although their sample maxima are close. This difference is due to the fact that the clustering algorithm we choose gives us a correct cluster-based partition when there are as few samples as 20. When the sample size is 100, we observe that the cGP (with $\xi = 0.1$) identifies the drops starting from $b = 160$ and $b = 500$ and attains maxima as good as $b = 128$ (with $f(128) = 2000.758$, close to the optimal $f(112) = 2010.702$), and the same as GP. While its performance is similar to GP with larger sample sizes, we can see that cGP correctly identifies the change of function behavior caused by the drop of computational speed near $b = 160$ and another behavior that changes rapidly when $b \geq 500$ even with very limited samples. Within each partition regime, the cGP provides a better fit in the sense that its mean function captures the steep slopes caused by cache size. The main point we try to make here is that cGP can provide additional partition information while still maintaining a similar performance as GP in searching for maximum (see Figure 15). So far, we consider all sequential samples selected by acquisition maximization (exploration rate 1).

Table 3 showcases results for varying sample sizes and exploration rates across columns (a), (b), and (c). In (a), cGP consistently surpasses the GP baseline, notably at $n = 90$ with a 1.0 exploration rate value of 0.54. Column (b) highlights cGP's efficiency, often needing fewer iterations for optimal block size. Meanwhile, (c) reveals a stable estimated cluster count around 2.8 to 2.9 for different sample sizes, underscoring cGP's consistent performance. As sample size grows, both surrogate models exhibit similar optima-reaching behaviors, but cGP more frequently attains the true optimum $b = 112$. A 0.8 exploration rate in cGP enhances performance in this data application. The average component count stabilizes near 2.9, aligning with the observed three partition regimes (see Figure 12) for the complete dataset, with smaller sample sizes occasionally under-sampling the $[1, 80] \cap \mathbb{Z}$ domain partition.

In this matmul tuning example, the cause of non-smooth points (i.e., overflow of cache) and different kinds of behavior is clear and observed in the recorded dataset as shown in Figure 12. cGP surrogates produce better fits than the GP model when the number of samples is small. When there are enough samples, the cGP model clearly identifies different partition regimes much better with a reasonable performance in optimization (See Table 3). For this realistic example, we also use the cutting-edge SMAC3(=1.2.0) (Lindauer et al. 2022) with the same number of evaluations of different It is also clear that each partitioned component has a different normalization for the data within this label, hence different degrees of smoothness are better elicited in the surrogate fit. The non-smooth change-points between components are also captured in the cGP model. In short, cGP provides non-smooth information about the black-box function without loss of optimization performance compared to GP in this application.

We focus on the power of the cGP model in identifying non-smoothness induced partition in the current section. For a related artificial tuning problem, varying the matrix size while keeping the block size fixed (see Appendix D).

4.2.2 Two-dimensional dense matmul tuning. We also investigate cGP effectiveness for dense matrix-matrix multiplication runtime tuning using two cache block sizes as tuning parameters. We use the double-precision matmul DGEMM routine from the open source BLIS (BLAS-like Library Instantiation Software) framework (Zee et al. 2016), a state-of-the-art BLAS (Basic Linear Algebra Subprograms) implementation which can be instantiated for various CPU microarchitectures. The experiments were performed on a single core, single hardware thread of a single CPU on one of the Perlmutter supercomputer nodes at NERSC (NERSC 2024). All other cores of that CPU were idle to exclude interference with our single-core single-hardware-thread run time measurements. The employed Perlmutter supercomputer node has two AMD EPYC 7763 @Max 3.5 GHz CPUs (codename Milan with Zen 3 microarchitecture) and 512 GB of DDR4 total memory. Each CPU consists of 8 dies with 8 cores and 16 hardware threads. The cache hierarchy for the CPU core is as follows: 32 KB L1 Data, 32 KB L1 Instruction, 512 KB L2, 32 MB L3. Each core provides a theoretical peak performance of 56 GFLOPS for a single hardware thread. The BLIS framework was instantiated for the Zen 3 microarchitecture configuration. Runtime experiments were gathered under the Linux 5.14.21 based operating system. Source code was compiled by the

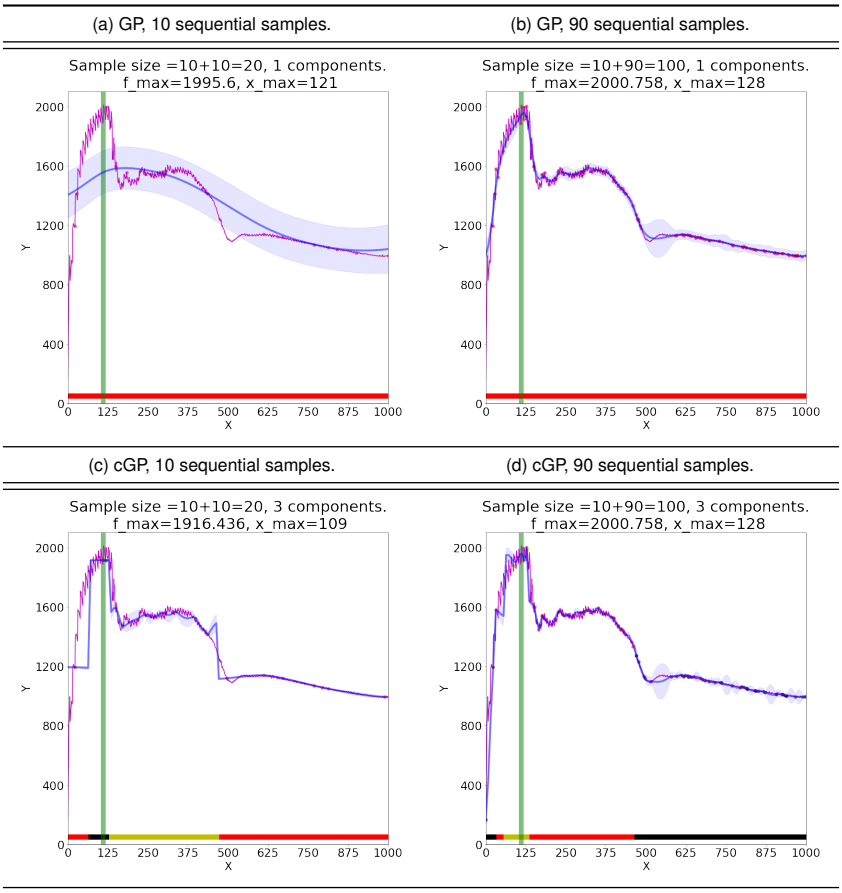


Figure 3. The computational speed function and fitted surrogate models for one-dimensional tuning of matmul. The dark blue solid line and light blue shaded area are the mean and variance of fitted prediction models. $(\boldsymbol{x}_{\max}, f_{\max})$ records the optimal block size and the actual optimal speed is illustrated by the green vertical line on the figure. Bottom colored bars with different colors indicate to which cluster certain portions of the domain belong (but color has no semantic meaning). We also use magenta solid lines to overlay the truth from Figure 12 for comparison purposes. (The exploration rates of these experiments are all 1.)

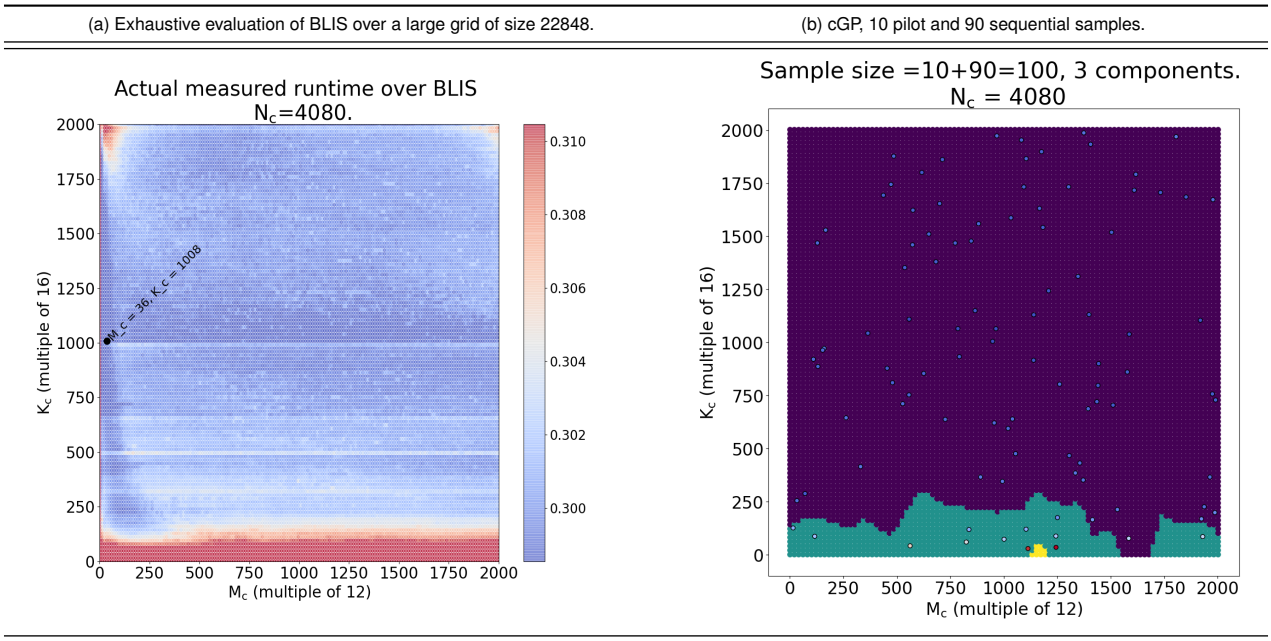


Figure 4. The computational time (s) function and fitted cGP mean function for two-dimensional matmul tuning. (The exploration rates of these experiments are all 1.) Panel (a) exhibits the actual recorded matmul performance on the Perlmutter as detailed in Section 4.2.2, with the best runtime of 0.295779 s for the parameter values $M_c = 36$ and $K_c = 1008$. Panel (b) illustrates the fitted cGP surrogate model (with $k = 3$ and 2-neighbors in k-means classifier) when $\tau = 1.0$.

Table 3. The tuning performance of 1-dimensional matmul problem. (a) The percentage of cGP fitted surrogates with optima equal or better than the baseline GP surrogate under different exploration rates. (b) The number of cGP (and GP) fitted surrogates that attain the actual optimal matrix size $\boldsymbol{x}_{\max}$ =112 under different exploration rates in 100 different random seeds. (c) The average number of clustering components in the fitted cGP surrogate models among 100 different random seeds.

		(a)			(b)				(c)		
Exp. rate		1.0	0.8	0.5	1.0	0.8	0.5	GP	1.0	0.8	0.5
Sample size	$n = 10$	0.72	0.66	0.62	2	2	1	2	2.71	2.64	2.63
	$n = 30$	0.57	0.54	0.48	2	5	3	4	2.89	2.83	2.85
	$n = 50$	0.55	0.51	0.45	4	8	6	9	2.84	2.91	2.86
	$n = 70$	0.55	0.57	0.49	14	14	11	11	2.88	2.90	2.91
	$n = 90$	0.54	0.63	0.54	15	19	16	13	2.94	2.94	2.97

GNU C compiler, version 12.3.0. All experiments were performed in double-precision floating-point arithmetic on random, column-major indexed real matrices that were stored in memory aligned to 4 KB boundary.

The matmul multiplies an $M \times K$ matrix by a $K \times N$ matrix. The parameters that we tuned were M_c and K_c, the two cache block sizes used to partition the matrix-matrix multiplication problem down to so-called "block-panel" subproblem implemented as a micro kernel. The third cache block size N_c was set to the default value of 4080. Note that M_c and N_c have a constraint that they have to be a multiple of register block sizes for the micro-kernel in M and N dimensions $M_r = 6$ and $N_r = 8$ respectively (Zee et al. 2016). The BLIS framework uses $M_c = 72$, $K_c = 256$, $N_c = 4080$ as the default optimal values for Zen 3 microarchitecture.

To understand the true performance landscape, we first performed an exhaustive search by collecting the run time data points for the multiplication of two 2000×2000 matrices by varying M_c from 1 to 1992 in steps of size 12, and by varying K_c from 1 to 1984 in steps of size 16, and for fixed $N_c = 4080$. We measured the run time for each data point 8 times and took the minimum value in order to reduce fluctuations. The collected 20,875 run time data points form a 167×125 2D parameter grid which is presented in Figure 4 (a). We measured the shortest runtime of 0.295779s for the parameter values $M_c = 36$ and $K_c = 1008$ (compared to 0.299294s for the default BLIS parameter values $M_c = 72$ and $K_c = 256$).

Then we ran the cGP, GP, Forest, GBRT and SMAC surrogate models on the obtained true run time data. From Table 4, the cGP models with different exploration parameters (τ=0.5, τ=0.8, τ=1.0) consistently found nearly identical optimal matmul run times 0.297 s after 90 sequential samples compared to >0.298 s found by other models. And we can also see that even with the same number of sequential samples, cGP usually outperforms the other models (better than BLIS default but worse than the exhaustive search), suggesting that the exploration rate may not have had a significant impact on the optimal run time within this set of benchmarks. In Table 4, on the other hand, the Forest, GBRT, and SMAC models found higher (worse) run times than the cGP models for all the cases. Uniquely, according to Figure 4 (a), GP also recovered the non-smoothness boundary near $K_c \approx 100$ and force the sequential sampling near these boundaries as shown in Figure 4 (b). Such a boundary cannot be detected by the other models (e.g., Forest, GBRT, SMAC, or standard GP) without performing an expensive exhaustive search.

The close competition among cGP models and the GP model suggest that Gaussian Process based methods are robust in finding the optimal execution time for the BLIS benchmark and can be used for identifying the non-smoothness in this HPC application.

4.2.3 Merged-dimensional Tuning of hypre. In this section, we present another tuning example using the hypre application Falgout et al. (2005) for tuning the runtime (in seconds) for solving the convection-diffusion equation on a 20^3 Cartesian grid with the algebraic multigrid (AMG) algorithms. In this example, we choose two tuning parameters, i.e., truncated factor for AMG interpolation (continuous variable ranging from 0 to 1) and type of coarsening algorithms (categorical variable with 4 categories). We call this a merged-dimensional tuning by merging the two tuning parameters into one continuous parameter x ranging from 0 to 4. More specifically, truncation factor value ϵ and coarsen type $k \in \{0, 1, 2, 3\}$ are mapped to a single parameter value $x = k + \epsilon$. We expect this treatment to create at most 3 discontinuities at $x = 1, 2, 3$.

In Figure 5 we executed our experiment on one Intel i9-12900T node and the objective function (in magenta solid lines) shows that this scientific application has much higher noise but we can still observe a drop near the parameter value 1. This means that we only need to use 2 components in cGP. From Table 5, we can observe that in this noisy black-box function, the Forest and GBRT optimizers cannot perform as well as shown in the synthetic functions Table 1 due to the easily over-fitting nature of the forest and gradient-boosting ensemble. In the same experiment (see Figure 5), we examine the partitioning discovered by our unique cGP with 100 samples. For exploration rate 1.0 and 0.8, the cGP partitioning only discovered the high execution time and low execution time regions, but already witness improvement over the rest competing models. We believe this shall be credited to the denoising nature of the (additive) GP model under this high signal-to-noise ratio (i.e., the magnitude of noise is comparable to the magnitude of the trend/shape exhibited by the black-box function) scenario caused by hypre. The cGP with exploration rate 0.5 also discovered the

jump of the execution time black-box function near tuning parameter value 1. And from Table 5 the exploration rate 0.5 also gets bit more improvement when the sequential sample size grows to 90.

From this scientific application, we learn that learning the geometry that causes the non-smoothness in the black-box function can actually help the surrogate-based optimization. Comparing the matmul and hypre applications, we can see that cGP has been relatively robust regardless of the (supposedly unknown) signal-to-noise ratio in the black-box function. We have incorporated the large-scale hypre application along with a merged-dimensional setting, to showcase the effectiveness of cGP in environments characterized by low signal-to-noise ratios, a scenario very common in Bayesian optimization for HPC tuning. The inherent variability in the performance of black-box functions, particularly evident in HPC applications, stems from the intricacies of communication and machine configurations within these systems.

To comment this further, the current literature in Bayesian optimization does not take refined examination of signal-to-noise ratios in black-box functions. One reason is that the noise part is usually confounded with the estimated additive noise in the (GP) surrogate model (Luo et al. 2022). The other reason is that most targeted applications did not involve very high and heterogeneous noise (i.e., difference magnitudes of noise at different parameter configurations) like we have observed in HPC and scientific applications such as SuperLU and hypre (see Figures 3 and 5). In the scenario where we do not know the actual degree of smoothness nor the signal-to-noise ratio of the black-box function, empirically cGP shows reasonable results and its cluster-classify mechanism can adapt well.

4.2.4 High-dimensional tuning of SuperLU_DIST. We acknowledge that there are different definitions for high dimensionality. When we conduct a grid search on a one- or two-dimensional domain, the grid size (i.e., the number of grid points) is usually of the magnitude of 10^3 to 10^6. In the tuning context, we usually consider 3- to 32- dimensional domains as high-dimensional tuning domains. When the dimensionality exceeds 32, even a binary enumeration vector would take more than 2^{32} bytes=32Gb to assign. This already exceeds the usual memory size of current computational machines, and makes the grid search infeasible.

The high-dimensional application we are studying is the tuning problem arising in the distributed-memory parallel sparse LU factorization of SuperLU_DIST (Li et al. 2023) SuperLU_DIST is a numerical software package developed for parallel LU factorization of large sparse matrices. We provide additional results for two well-known high-dimensional industrial simulation examples in Appendix F where we have studied both maximum and minimum of a high-dimensional function.

We use 8 Haswell nodes with 32 cores on each node, and 256 MPI processess in total. We tested the following set of matrices: Si2, SiH4, SiNa, benzene, Na5, Si5H12, Si10H16, SiO, H2O, GaAsH5, and Ga3As3H12 (the details of these matrices are available at `https://sparse.tamu.edu/PARSEC`). This tuning problem has 4 tunable parameters, LOOKAHEAD (number of lookahead panels used to overlap the communication with computation), nprow (number of row processes out of 256 MPI processes), NSUP and NREL (parameters defining maximum and relaxed sizes of the supernode representation). There is another categorical parameter COLPERM representing different ways of permuting the columns, that we encode and treat as an integer variable for this application. We can compare the tuned execution time shown in 7 and observe that the range and variation of execution time is large, and the randomly selected configurations produce execution times with big variations (see Figure 19) in Table 6.

Note that cGP outperforms GP for most matrices when a proper exploration rate is chosen (see Table 6 and Figure 19).

All of these parameters are integers, and so we need to round them to the nearest integer when fitting the surrogate model over a continuous domain. The black-box objective function f in this tuning problem is the running time of LU factorization for a given matrix; our examples arise in chemistry (see Appendix E). In this application, we want to find an optimal configuration of the SuperLU_DIST application to minimize the running time $f_{\min}$. $f_{\max}$ is not relevant in this application.

The following figures are organized in the same format: In the top panel, we show the absolute SuperLU_DIST running time ($f_{\min}$) obtained by each specific surrogate model with different number of sequential samples and 10 pilot samples in one run. In the bottom panel, we show the relative ratio ($f_{\min}$ obtained by cGP models divided by the $f_{\min}$ obtained by simple GP) of SuperLU_DIST running time obtained by each specific surrogate model against the one using the simple GP surrogate model, ratios that are less than 1 means better performance.

In Figure 7 with sufficient samples, we show the performance of GP and cGP surrogate models. In this application, the choice of the maximal number of components affects the result. The cGP provides comparable and usually better optimal configurations for SuperLU_DIST, especially for matrices Si2, SiH4, SiNa, benzene, Na5, Si5H12. For these matrices, we can see that cGP with exploration rates and clustering provide the configurations of SuperLU_DIST that are faster than using the GP surrogate. For the other matrices, the cGP (with at most 3 clusters) is usually not much worse than GP except for the Na5 matrix, at least for some selection of exploration rates and clustering.

For most matrices under consideration cGP is at least as good as GP, therefore, we can use cGP in place of GP without much worry about getting a much worse $f_{\min}$. For matrices Si10H16, SiO, H2O, GaAsH5, Ga3As3H12 (i.e., larger than 10Mb in Table 6) , the noise in the running time is negligible, and we observe that cGP is generally producing a better factorization time with only a few exceptions. The improvement can be as much as 7.3% (for GaAsH6 matrix). Again, we expect that when cGP is giving a worse result (e.g., Si10H16 matrix), it is within a 7% margin and for a better choice of exploration rate (1.0, the number of components $k = 2$), we can achieve a 5% improvement. While the absolute improvement is not as significant as in the synthetic functions or those in Appendix D, such

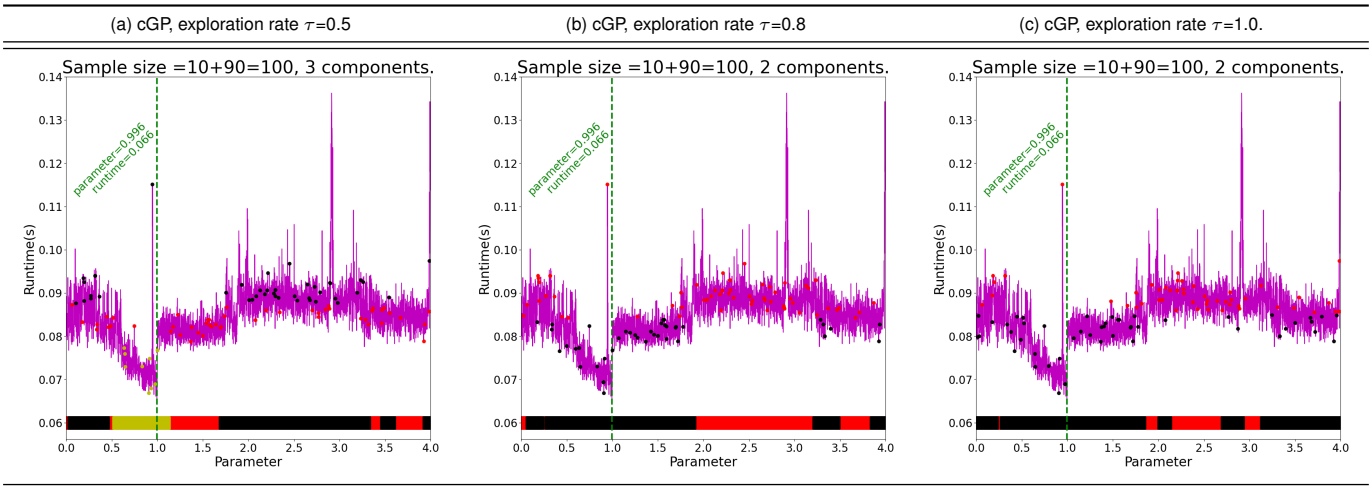


Figure 5. The computational time function of merged-dimensional hypre and partitions found by using maximum number of clusters equal to 3 and (a) τ = 0.5 (b) τ = 0.8 (c) τ = 1.0, from fitted cGP surrogate models. We also use magenta solid lines to overlay the truth for comparison purposes.

Table 4. Optimal BLIS benchmark execution time found by different surrogate models among 50 different random seeds.

#	cGP (τ=0.5)	cGP (τ=0.8)	cGP (τ=1.0)	Forest	GBRT	SMAC	GP
10	**0.298965**	**0.298965**	**0.298965**	0.313409	0.354673	0.395923	0.313409
30	0.298386	0.298353	**0.298335**	0.299198	0.299122	0.298886	0.299193
50	0.298200	0.298118	**0.297978**	0.298978	0.298860	0.298477	0.298936
70	0.298055	0.297977	**0.297854**	0.298722	0.298582	0.298316	0.298841
90	0.297842	0.297780	**0.297676**	0.298627	0.298403	0.298247	0.298807

Table 5. Optimal hypre solver execution time found by different surrogate models among 100 different random seeds.

#	cGP (τ=0.5)	cGP (τ=0.8)	cGP (τ=1.0)	Forest	GBRT	SMAC	GP
10	0.0724	**0.0719**	0.0722	0.0745	0.0733	0.0750	0.0731
30	0.0704	**0.0703**	0.0704	0.0745	0.0727	0.0734	0.0704
50	**0.0697**	**0.0697**	**0.0697**	0.0744	0.0721	0.0726	0.0699
70	**0.0692**	0.0694	**0.0692**	0.0743	0.0718	0.0721	0.0694
90	**0.0689**	**0.0689**	0.0690	0.0742	0.0715	0.0717	0.0690

an improvement is quite important when we often need to conduct LU factorization multiple times.

The geometries of the objective function in SuperLU_DIST applications are not known to us like that of matmul. However, we show the power of cGP by simply choosing a few different cGP surrogate models as the hypre application essentially shows the importance of identifying the correct non-smoothness when we attempt to tune scientific applications.

In the SuperLU_DIST application, we can see that $f_{\min}$ gets improved over GP results frequently, and would not lose by a large margin.

To consider both maximum and minimum simultaneously, in the piston application (see Appendix F), we can see that $f_{\max}$ is much improved. The $f_{\min}$ is also well approximated using this surrogate, although our goal is the maximum. In the same Appendix F, we also provide robot arm tuning experiments to show that cGP also performs well in a relatively smooth black-box function problem, whilst other competitors capable of handling non-smoothness (e.g., Forest, GBRT) perform relatively poorly even compared to GP. Both piston (7-dimensional) and robot arms (8-dimensional) are consider expensive scientific simulations, broadening our focus from HPC applications to further scientific applications.

Table 6. The fixed matrices (stored in raw text files with the size indicated in bytes) considered in the SuperLU_DIST application on 8 cori Haswell nodes, with 32 cores used on each node. We consider those matrices smaller than 10Mb as "small" while the others "large".

Size of raw data file	d	Number of nonzeros entries	Matrix name	GP optimized factorization time (seconds)	cGP optimized factorization time (best among all cGP configurations, seconds)
224Kb	769	17801	Si2	0.145	0.140
2.0Mb	5041	171903	SiH4	0.283	0.272
2.3Mb	5743	198787	SiNa	0.311	0.282
2.9Mb	8219	242669	benzene	0.362	0.326
3.6Mb	5832	305630	Na5	0.285	0.283
8.6Mb	19896	738598	Si5H12	1.136	1.110
11Mb	17077	875923	Si10H16	0.698	0.663
16Mb	33401	1317655	SiO	1.992	1.859
26Mb	67024	2216736	H2O	5.686	5.232
39Mb	61349	3381809	GaAsH6	5.207	4.825
36Mb	61349	5970947	Ga3As3H12	5.382	5.313

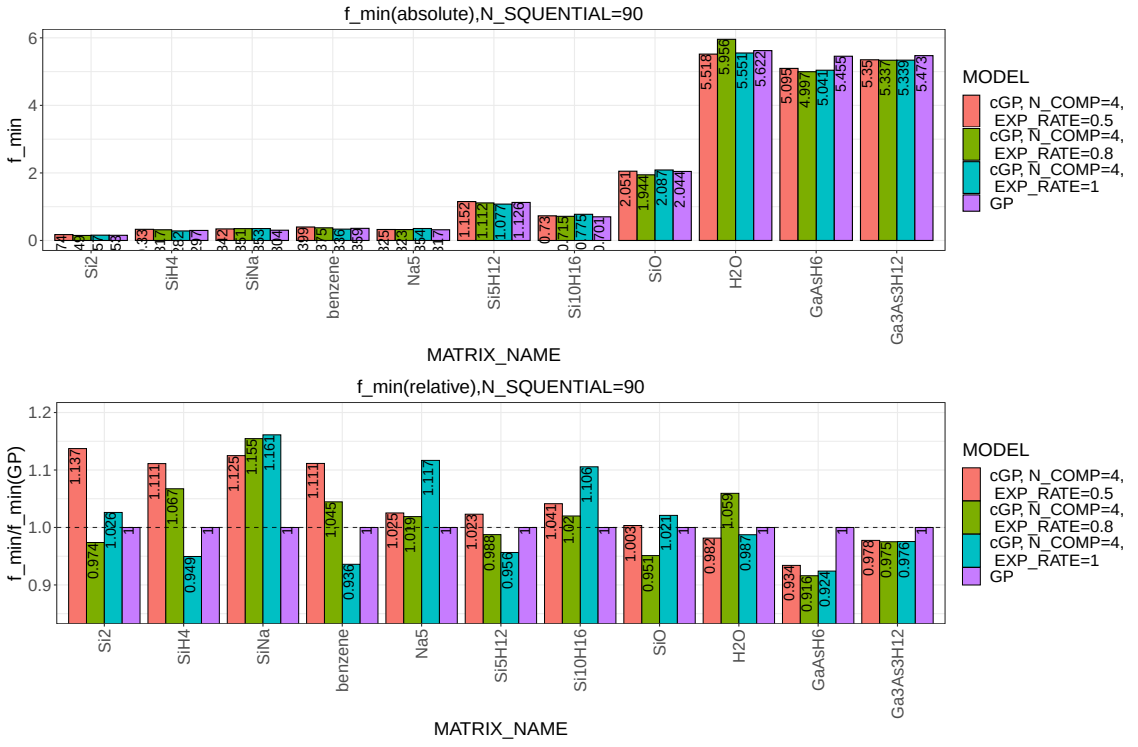


Figure 6. In the top panel, we show the absolute SuperLU_DIST running time ($f_{\min}$) obtained by each specific surrogate model with 90 sequential samples and 10 pilot samples in one run. In the bottom panel, we show the relative ratio ($f_{\min}$ obtained by certain cGP models divided by the $f_{\min}$ obtained by simple GP) of SuperLU_DIST running time obtained by each specific surrogate model against the one using simple GP surrogate model, ratios that are less than 1 means better performance.

5 Discussion

5.1 Contributions.

This paper deals with the non-smoothness problems in the tuning context, with a focus on numerical algorithm tuning. For tuning problems, the non-smoothness problem is more challenging when we have limited samples, relatively high dimensionality, and non-restrictive partition shape (of the support) induced by non-smoothness. This specific problem has not been treated nor explored to the best of our knowledge, but arises often in the tuning context.

To address this potential problem, we propose a new partition-based cGP surrogate that allows sequential surrogate modeling for non-smooth black-box objective

functions, identifies the non-smoothness and improves the computational efficiency, which extends to high-dimensional domains without much increase in complexity. In addition, the proposed cGP model is a new approach to create a dynamic local GP model compared to existing local GPs.

Our unique approach of cluster-classification step also proves to be effective for different kinds of non-smoothness. To our best knowledge, it is also a novel attempt to introduce dynamic kernels in an online supervised setting (Luo et al. 2024). When we have knowledge of the underlying geometry of the problem, this knowledge can usually help us to choose the parameters of cGP surrogate models to attain better results (e.g., Bukin N.6 in Figure 2). However, even when we do not know the objective functions for certain applications, cGP may help us to identify the geometry (e.g., matmul in Section 4.2.1 and 4.2.3). Even if we do not care about the geometry of the objective function, we can usually get an improvement in our tuning task (e.g., large matrices in SuperLU_DIST in Section 4.2.4). In the least favorable situation, using cGP would not lose much in tuning tasks compared to GP (e.g., small matrices in SuperLU_DIST).

The cGP model usually has a computational advantage over GP since it fits multiple smaller GP components instead of one large GP as discussed in Section 4.1.4. In addition, cGP can be considered as a generalization of the GP model, when we choose the maximal number of components to be 1, cGP reduces to the GP model. In this sense, cGP can be a standalone modeling method that incorporates the change-surface detection within a Bayesian framework as well.

We offer a parallel implementation available in our repository, which not only accelerates the fitting as detailed in Figure 11 but also underscores that cGP is primarily designed for parallel applications.

5.2 Future work.

It would be natural to introduce a prior mechanism to modify the current cGP model into a fully Bayesian framework which exceeds the scope of the current paper. The cGP model could also be extended to multiple-output scenarios, where we may learn the partition from multiple similar runs as well as cross-correlation between partitions.

Replacing the simple GP model with cGP raises the question of how the trade-off between the number of model parameters and the number of sequential samples can be balanced when we have limited samples for modeling.

A natural future work is to examine whether the optimization results are sensitive to this choice of the clustering and classification algorithms, and their hyper-parameters. We should explore optimal strategies for setting ξ in non-smooth functions, like matmul, to effectively capture or avoid specific discontinuities with apriori information instead of mere experimentation. To fully address model parameter selection, an algorithmic multi-level approach might be needed to choose different acquisition functions and exploration rates for different partition components adaptively. Furthermore, we also intend to develop new clustering criteria based on graph metrics that fit into the cGP framework, where the cluster-classification mechanism could leverage (graph-based) convex clustering (Sun et al. 2021).

We are motivated by non-existence of the first-order gradients of black-box functions. Non-smoothness motivated by higher order gradients is more complicated and so will be efforts to model that kind of non-smoothness (Luo and Strait 2021; Luo et al. 2020). To summarize, we have identified a rarely explored practical problem, and proposed and implemented a novel algorithm that generalizes the classical GP surrogate model. This cGP model has the ability to handle non-smoothness and complex geometry of the objective black-box function with limited online samples. We provide evidence how this model outperforms the simple GP surrogate model for high and low dimensional functions, and discuss its flexibility and possible extensions.

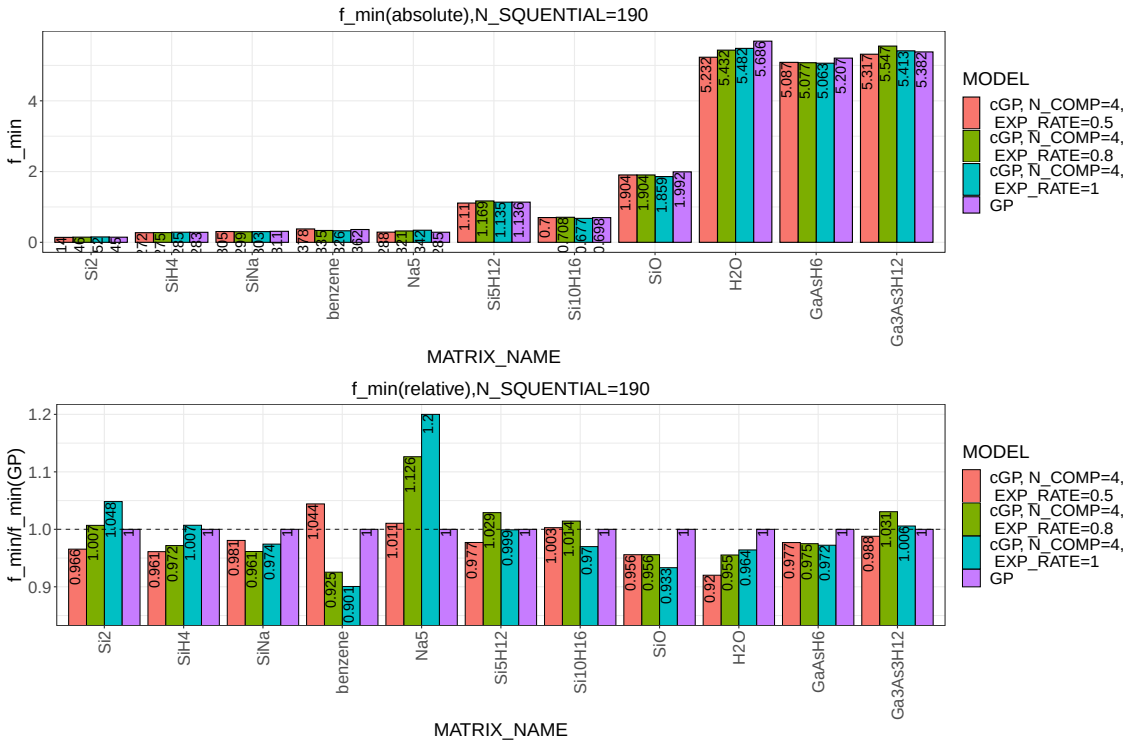


Figure 7. In the top panel, we show the absolute SuperLU_DIST running time ($f_{\min}$) obtained by each specific surrogate model with 190 sequential samples and 10 pilot samples in one run. In the bottom panel, we show the relative ratio ($f_{\min}$ obtained by certain cGP models divided by the $f_{\min}$ obtained by simple GP) of SuperLU_DIST running time obtained by each specific surrogate model against the one using simple GP surrogate model, ratios that are less than 1 means better performance.

Copyright statement

Acknowledgements

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. We used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231. We stored our code at https://github.com/gptune/cGP. Younghyun Cho was with University of California, Berkeley, when he worked on this paper. He is now with Santa Clara University.

References

Adams RP and MacKay DJC (2007) Bayesian Online Changepoint Detection. *arXiv:0710.3742* .

Alonso P, Catalán S, Igual FD, Mayo R, Rodríguez-Sánchez R and Quintana-Ortí ES (2015) Time and Energy Modeling of High-performance Level-3 BLAS on x86 Architectures. *Simulation Modelling Practice and Theory* 55: 77–94.

An J and Owen A (2001) Quasi-regression. *Journal of complexity* 17(4): 588–607.

Barry D and Hartigan JA (1993) A Bayesian Analysis for Change Point Problems. *Journal of the American Statistical Association* 88(421): 309–319.

Basseville M and Nikiforov IV (1993) *Detection of Abrupt Changes: Theory and Application*. Prentice Hall Information and System Sciences Series. Englewood Cliffs, N.J: Prentice Hall.

Benner P, Ezzatti P, Quintana-Ortí ES, Remón A and Silva JP (2016) Tuning the Blocksize for Dense Linear Algebra Factorization Routines with the Roofline Model : 18–29.

Bergstra J and Bengio Y (2012) Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* .

Bhatia N et al. (2010) Survey of Nearest Neighbor Techniques. *arXiv:1007.0085* .

Bilmes J, Asanovic K, Chin CW and Demmel J (1997) Optimizing Matrix Multiply Using PHiPAC: A Portable, High-performance, ANSI C Coding Methodology. In: *ACM International Conference on Supercomputing 25th Anniversary Volume*. pp. 253–260.

Blackford LS, Petitet A, Pozo R, Remington K, Whaley RC, Demmel J, Dongarra J, Duff I, Hammarling S, Henry G et al. (2002) An Updated Set of Basic Linear Algebra Subprograms (BLAS). *ACM Transactions on Mathematical Software* 28(2): 135–151.

Booker AJ, Dennis JE, Frank PD, Serafini DB, Torczon V and Trosset MW (1999) A Rigorous Framework for Optimization of Expensive Functions by Surrogates. *Structural Optimization* 17(1): 1–13.

Broderick T and Gramacy RB (2010) Treed Gaussian Process Models for Classification. In: *Classification as a Tool for Research*. Springer, pp. 101–108.

Bull AD (2011) Convergence Rates of Efficient Global Optimization Algorithms. *Journal of Machine Learning Research* 12(10).

Calotoiu A, Hoefler T, Poke M and Wolf F (2013) Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes. *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* : 1–12.

Chipman HA, George EI and McCulloch RE (1998) Bayesian CART Model Search. *Journal of the American Statistical Association* 93(443): 935–948.

Chipman HA, George EI and McCulloch RE (2010) BART: Bayesian Additive Regression Trees. *The Annals of Applied Statistics* 4(1): 266–298.

Cho Y, Demmel JW, Dereziński M, Li H, Luo H, Mahoney MW and Murray RJ (2023a) Surrogate-based autotuning for randomized sketching algorithms in regression problems. *arXiv preprint arXiv:2308.15720* .

Cho Y, Demmel JW, King J, Li XS, Liu Y and Luo H (2023b) Harnessing the crowd for autotuning high-performance computing applications. In: *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, pp. 635–645.

Cramér H and Leadbetter M (2013) *Stationary and Related Stochastic Processes: Sample Function Properties and Their Applications*. Dover Books on Mathematics. Dover Publications.

Devroye L, Györfi L and Lugosi G (2013) *A Probabilistic Theory of Pattern Recognition*, volume 31. Springer Science & Business Media.

Fabregat-Traver D, Ismail AE and Bientinesi P (2016) Accelerating Scientific Codes by Performance and Accuracy Modeling. *arXiv:1608.04694* .

Falgout RD, Jones JE and Yang UM (2005) Pursuing scalability for hypre's conceptual interfaces. *ACM Transactions on Mathematical Software (TOMS)* 31(3): 326–350.

Geurts P, Ernst D and Wehenkel L (2006) Extremely randomized trees. *Machine learning* 63: 3–42.

Gramacy RB (2020) *Surrogates: Gaussian Process Modeling, Design, and Optimization for the Applied Sciences*. Chapman and Hall/CRC.

Gramacy RB and Apley DW (2015) Local Gaussian Process Approximation for Large Computer Experiments. *Journal of Computational and Graphical Statistics* 24(2): 561–578.

Gramacy RB and Lee HKH (2008) Bayesian Treed Gaussian Process Models with an Application to Computer Modeling. *Journal of the American Statistical Association* 103(483): 1119–1130.

Gramacy RB and Ludkovski M (2015) Sequential Design for Optimal Stopping Problems. *SIAM Journal on Financial Mathematics* 6(1): 748–775.

Gramacy RB and Polson NG (2011) Particle Learning of Gaussian Process Models for Sequential Design and Optimization. *Journal of Computational and Graphical Statistics* 20(1): 102–118.

Head T, MechCoder GL, Shcherbatyi I et al. (2018) scikit-optimize/scikit-optimize: v0. 5.2. *Version v0* 5.

Herlands W, Wilson A, Nickisch H, Flaxman S, Neill D, Van Panhuis W and Xing E (2016) Scalable Gaussian Processes for Characterizing Multidimensional Change Surfaces. In: Gretton A and Robert CC (eds.) *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, *Proceedings of Machine Learning Research*, volume 51. PMLR, pp. 1013–1021.

Hernández-Lobato JM, Hoffman MW and Ghahramani Z (2014) Predictive Entropy Search for Efficient Global Optimization of Black-box Functions. *Advances in Neural Information Processing Systems* 27.

Hong JW and Kung HT (1981) I/O Complexity: The Red-Blue Pebble Game. In: *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing - STOC '81*. Milwaukee, Wisconsin, United States: ACM Press.

Igel C, Hansen N and Roth S (2007) Covariance Matrix Adaptation for Multi-objective Optimization. *Evolutionary Computation* .

Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, Ye Q and Liu TY (2017) Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30.

Kenett RS, Zacks S and Amberti D (2013) *Modern Industrial Statistics: With Applications in R, MINITAB and JMP*. John Wiley & Sons.

Klus J, Grunt P and Dobrovolný M (2021) Hyper-optimization with Gaussian Process and Differential Evolution Algorithm .

Krause A and Guestrin C (2007) Nonmyopic Active Learning of Gaussian Processes: An Exploration-Exploitation Approach. In: *Proceedings of the 24th International Conference on Machine Learning*. pp. 449–456.

Lamparth M, Bestehorn M and Märkisch B (2022) Gaussian Processes and Bayesian Optimization for High Precision Experiments. *arXiv preprint arXiv:2205.07625* .

Li XS, Lin P, Liu Y and Sao P (2023) Newly Released Capabilities in the Distributed-Memory SuperLU Sparse Direct Solver. *ACM Trans. Math. Softw.* 49(1). DOI:10.1145/3577197. URL https://doi.org/10.1145/3577197.

Lindauer M, Eggensperger K, Feurer M, Biedenkapp A, Deng D, Benjamins C, Ruhkopf T, Sass R and Hutter F (2022) Smac3: A versatile bayesian optimization package for hyperparameter optimization. *The Journal of Machine Learning Research* 23(1): 2475–2483.

Liu Y, Sid-Lakhdar WM, Marques O, Zhu X, Meng C, Demmel JW and Li XS (2021) Gptune: Multitask learning for autotuning exascale applications : 234–246.

Liu Z, Zhou L, Leung H and Shum HP (2015) Kinect Posture Reconstruction Based on a Local Mixture of Gaussian Process Models. *IEEE Transactions on Visualization and Computer Graphics* 22(11): 2437–2450.

Luo H, Cho Y, Demmel JW, Li X and Liu Y (2024) Hybrid Parameter Search and Dynamic Model Selection for Mixed-Variable Bayesian Optimization. *Journal of Computational and Graphical Statistics* 0(0): 1–14.

Luo H, MacEachern SN and Peruggia M (2020) Asymptotics of Lower Dimensional Zero-Density Regions. *arXiv:2006.02568* : 1–28.

Luo H, Nattino G and Pratola MT (2022) Sparse Additive Gaussian Process Regression. *Journal of Machine Learning Research* 23(1): 1–33.

Luo H and Strait J (2021) Combining Geometric and Topological Information in Image Segmentation. *2021 IEEE International Conference on Big Data (Big Data)* : 3841–3852.

Maddox WJ, Balandat M, Wilson AG and Bakshy E (2021) Bayesian Optimization with High-Dimensional Outputs. *Advances in Neural Information Processing Systems* 34: 19274–19287.

Martinez-Cantin R (2015) Locally-biased Bayesian Optimization using Nonstationary Gaussian Processes. In: *Neural Information Processing Systems (NIPS) workshop on Bayesian Optimization*, volume 7. p. 4.

NERSC (2024) Perlmutter Architecture. https://docs.nersc.gov/systems/perlmutter/architecture.

Nguyen-Tuong D, Seeger M and Peters J (2009) Model Learning with Local Gaussian Process Regression. *Advanced Robotics* 23(15): 2015–2034.

Noack MM, Luo H and Risser MD (2023) A unifying perspective on non-stationary kernels for deeper gaussian processes. *arXiv preprint arXiv:2309.10068* .

Owen AB, Dick J and Chen S (2014) Higher Order Sobol' Indices. *Information and Inference: A Journal of the IMA* 3(1): 59–81.

Paciorek CJ and Schervish MJ (2004) Nonstationary Covariance Functions for Gaussian Process Regression. In: *Advances in Neural Information Processing Systems 16*. MIT Press, pp. 273–280.

Park C, Huang JZ and Ding Y (2011) Domain Decomposition Approach for Fast Gaussian Process Regression of Large Spatial Data Sets. *Journal of Machine Learning Research* 11: 1697–1728.

Park S and Choi S (2010) Hierarchical Gaussian Process Regression. In: *Proceedings of 2nd Asian Conference on Machine Learning*, *Proceedings of Machine Learning Research*, volume 13. JMLR Workshop and Conference Proceedings, pp. 95–110.

Peise E, Fabregat-Traver D and Bientinesi P (2015) On the Performance Prediction of BLAS-based Tensor Contractions : 193–212.

Raponi E, Wang H, Bujny M, Boria S and Doerr C (2020) High Dimensional Bayesian Optimization Assisted by Principal Component Analysis .

Rasmussen CE and Williams CKI (2006) *Gaussian Processes for Machine Learning*. Cambridge, Mass: MIT Press.

Rios LM and Sahinidis NV (2013) Derivative-Free Optimization. *Journal of Global Optimization* .

Saatçi Y, Turner R and Rasmussen CE (2010) Gaussian Process Change Point Models. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10. pp. 927–934.

Schweidtmann AM, Bongartz D, Grothe D, Kerkenhoff T, Lin X, Najman J and Mitsos A (2021) Deterministic Global Optimization with Gaussian Processes Embedded. *Mathematical Programming Computation* 13(3): 553–581.

Shahriari B, Swersky K, Wang Z, Adams RP and de Freitas N (2016) Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE* 104(1): 148–175.

Slonim N, Aharoni E and Crammer K (2013) Hartigan's K-means vs. Lloyd's K means–is it time for a change? In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*.

Smith AFM (1975) A Bayesian Approach to Inference about a Change-Point in a Sequence of Random Variables. *Biometrika* 62(2): 407–416.

Snoek J, Larochelle H and Adams RP (2012) Practical Bayesian Optimization of Machine Learning Algorithms. *Advances in Neural Information Processing Systems* 25.

Snoek J, Swersky K, Zemel R and Adams R (2014) Input Warping for Bayesian Optimization of Non-stationary Functions. In: *International Conference on Machine Learning*. PMLR, pp. 1674–1682.

Solak E, Murray-smith R, Leithead W, Leith D and Rasmussen C (2003) Derivative Observations in Gaussian Process Models of Dynamic Systems. In: Becker S, Thrun S and Obermayer K (eds.) *Advances in Neural Information Processing Systems*, volume 15. MIT Press.

Stoyanov M (2018) Adaptive Sparse Grid Construction in a Context of Local Anisotropy and Multiple Hierarchical Parents. In: *Sparse Grids and Applications-Miami 2016*. Springer, pp. 175–199.

Stoyanov M, Seleson P and Webster C (2017) *A Surrogate Modeling Approach for Crack Pattern Prediction in Peridynamics*, chapter 0. American Institute of Aeronautics and Astronautics, pp. 1–11.

Sun D, Toh KC and Yuan Y (2021) Convex Clustering: Model, Theoretical Guarantee and Efficient Algorithm. *Journal of Machine Learning Research* 22: 1–30.

Sung CL, Haaland B, Hwang Y and Lu S (2019) A Clustered Gaussian Process Model for Computer Experiments. *arXiv:1911.04602* .

Teh YW et al. (2010) Dirichlet Process.

Whaley RC, Petitet A and Dongarra JJ (2001) Automated Empirical Optimizations of Software and the ATLAS Project. *Parallel Computing* 27(1-2): 3–35.

Wynne G, Briol FX and Girolami M (2021) Convergence Guarantees for Gaussian Process Means with Misspecified Likelihoods and Smoothness. *Journal of Machine Learning Research* 22(123): 1–40.

Zee FGV, Smith TM, Marker B, Low TM, Geijn RAVD, Igual FD, Smelyanskiy M, Zhang X, Kistler M, Austel V et al. (2016) The BLIS Framework: Experiments in Portability. *ACM Transactions on Mathematical Software (TOMS)* 42(2): 1–19.

Zilberstein S, Koehler J and Koenig S (1999) Adaptive Sampling for Black-Box Optimization. *Computer Science Department Faculty Publication Series* .

Appendices

A Synthetic Function Landscapes

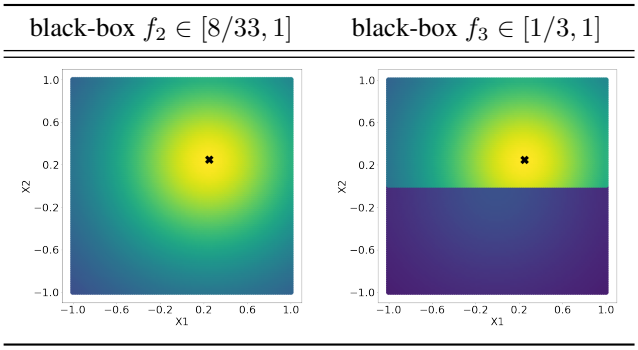


Figure 8. The smooth synthetic function f_2 and the non-smooth synthetic function f_3. We show exact minimal point by a black cross.

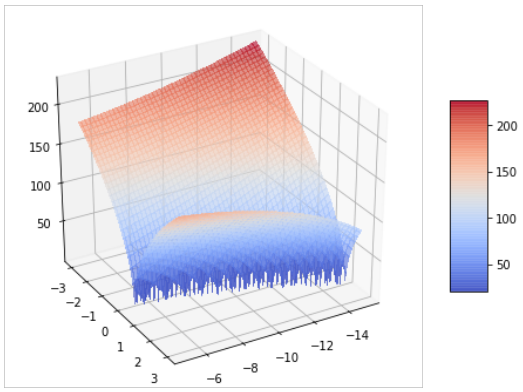


Figure 9. The Bukin N.6 function is defined by $f(x) = 100\sqrt{|x_2 - 0.01x_1^2|} + 0.01|x_1 + 10|$. The function has unique global minimum $f(x_*) = 0$ at $x_* = (-10, 1)$. We optimize the function to find its minimum over the domain of $(-15, 5) \times (-3, 3)$. The plot is based on evaluation of $f(x)$ on a 100 by 100 grid.

B Theoretic Results

Let us start with a simpler situation by supposing the partitions of components are fixed from the beginning of sequential sampling.

Assumption A. The number of components $k < \infty$ is known and cannot change.

To ensure the convergence of the surrogate model, we need to assume that the covariance kernel K satisfies:

Assumption B. The Fourier transform $\hat{K}(\boldsymbol{\xi}) := \int_{\mathbb{R}^d} 2^{-2\pi i \langle \boldsymbol{x}, \boldsymbol{\xi} \rangle} K(\boldsymbol{x}) d\boldsymbol{x}$ exists, is isotropic and non-increasing and satisfy either $\hat{K}(\boldsymbol{\xi}) = \Theta\left(\|\boldsymbol{\xi}\|^{-2\nu-d}\right)$ for some $\nu > 0$ or $\hat{K}(\boldsymbol{\xi}) = O\left(\|\boldsymbol{\xi}\|^{-2\nu-d}\right)$ for $\forall \nu > 0$ (and denote $\nu = \infty$). We call this ν the *smoothness parameter* of the covariance kernel K. (Assumption B is exactly the Assumptions 1,2 and 3 in Bull (2011))

Assumption C. The covariance kernel K is $\mathcal{C}^{\lceil 2\nu \rceil}$ and the $\lceil 2\nu \rceil$-th order Taylor approximation P_K satisfy $|K(\boldsymbol{x}) - P_K(\boldsymbol{x})| = O\left(\|\boldsymbol{x}\|^{2\nu} \left(-\log \|\boldsymbol{x}\|\right)^{2\alpha}\right)$ as $\boldsymbol{x} \to 0$ for some $\alpha \geq 0$. (Assumption 4 in Bull (2011))

Both Matérn family and exponential kernel satisfy Assumption 2 and 3 above. When the partitions are fixed, the crucial condition of the convergence of GP surrogate lies in the regularity of the component domains and the sample size. The regularity of the component domain can be described by

Definition 2. *(Lipschitz domain, Wynne et al. (2021)) An open set $\mathcal{X}_i \subset \mathbb{R}^d$ is called a (special) Lipschitz domain if there exists a rotation of $\mathcal{X}_i$, denoted by $\bar{\mathcal{X}}_i$, and a function $\psi : \mathbb{R}^{d-1} \to \mathbb{R}$ which satisfies the following:*
(1) $\mathcal{X}_i = \{(x, y) \in \mathbb{R}^d, y > \psi(x)\}$
(2) ψ is a Lipschitz function such that $|\psi(x) - \psi(x')| \leq M\|x - x'\|_2$ for any $x, x' \in \mathbb{R}^{d-1}$ and some $M > 0$.

For instance, the binary divisions adopted by Chipman et al. (2010) consist of Lipschitz domains with piecewise constant functions $\psi = c_j$.

Given a black-box function, it is essential to understand its properties to determine whether it can be approximated by Gaussian Process (GP) or clustered Gaussian Process (cGP) with a selected kernel. This understanding is rooted in the mathematical framework of Reproducing Kernel Hilbert Space (RKHS).

A function f is said to be in the RKHS if it can be represented as a linear combination of the kernel functions. If $f \notin \mathcal{H}_K$, the function cannot be approximated by the GP or cGP, leading to suboptimal optimization results. Proposition 1 emphasizes that it only applies to black-box functions that reside within the RKHS induced by well-behaved kernels.

The cGP model is noted for having an adaptive kernel that is "nicer" in the sense that it may also approximate non-smooth functions. This adaptability is a significant advantage in handling complex and non-smooth landscapes.

Proposition 1. *Suppose Assumptions A hold and that the domain $\mathcal{X}$ can be decomposed into Lipschitz domains $\mathcal{X}_1 \cup \cdots \cup \mathcal{X}_k = \mathcal{X}$ and each $\mathcal{X}_j$ is compact with an nonempty interior. In the reproducing kernel Hilbert space (RKHS) $\mathcal{H}_\theta(\mathcal{X})$ defined by the covariance kernel K_θ on $\mathcal{X}$, let us suppose that black-box functions $f\,|_{\mathcal{X}_j} \in \mathcal{H}_{\theta_U}(\mathcal{X}_j), j = 1, \cdots, k$ with $\theta \leq \theta_U < \infty$ satisfy Assumption B and C. Then there exists an integer $n_0 > k$ such that the interpolant optimum Bull (2011) $\boldsymbol{x}_n^*$ satisfy*

$$\sup_{\substack{\|f\|_{\mathcal{H}_{\theta_U}(\mathcal{X}_j)} \leq R \\ j = 1, \cdots, k}} \mathbb{E}_f \left| f(\boldsymbol{x}_n^*) - \min_{\mathcal{X}} f \right| = O\left(\left(\frac{n}{\log n}\right)^{-\nu/d} (\log n)^{\alpha}\right) \quad (3)$$

for some $R > 0$ and all sufficiently large sample size $n > n_0$. If $\nu = \infty$ then the statement holds for all $\nu < \infty$.

Proof. The idea of the proof is that the overall sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n$ of sequential sampling can be organized into sub-sequences for each component, which grow simultaneously due to our algorithm. Within each component we have an asymptotic scheme, the regularity of the component domains allow us to apply results in Bull (2011) obtained from the RKHS techniques for each component.

The sequential samples $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n$ are chosen from one of all k independent components in the cGP model. Therefore, if we use the super script to denote the component a sample point belongs to, we can reorganize the sample sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n$ into k sub-sequences where the sizes of sub-sequences satisfy $n_1 + \cdots + n_k = n$:

$$\begin{gathered}\boldsymbol{x}_1^{(1)}, \boldsymbol{x}_2^{(1)}, \cdots, \boldsymbol{x}_{n_1}^{(1)}, \\ \boldsymbol{x}_1^{(2)}, \boldsymbol{x}_2^{(2)}, \cdots, \boldsymbol{x}_{n_2}^{(2)}, \\ \cdots \\ \boldsymbol{x}_1^{(k)}, \boldsymbol{x}_2^{(k)}, \cdots, \boldsymbol{x}_{n_k}^{(k)}.\end{gathered}$$

Now we also need to show that all $n_1, n_2, \cdots, n_k > 1$ and tend to infinity, for otherwise, there would be components containing no sequential sample points at all. However, this cannot happen, due to the fact that overall sequential sampling strategy is to pick the maximizer of EI acquisition function $\mathrm{EI}_j(\boldsymbol{x})$ in each component weighted by component sample size n_j. If $n_1 = 1$ while $n_2, \cdots, n_k > 1$ then we can pick n_0 such that

$$\frac{\max_{j=2,\cdots,k} \max_{\boldsymbol{x}\in\mathcal{X}_j} \mathrm{EI}_j(\boldsymbol{x})}{\min(n_2, \cdots, n_k) + n_0/(k-1)} \leq \max_{\boldsymbol{x}\in\mathcal{X}_1} \mathrm{EI}_1(\boldsymbol{x})/1$$

The numerator is the maximal acquisition function among all $\mathrm{EI}_j(\boldsymbol{x}), j = 1, 2, \cdots, k$, which is non-increasing by the definition of EI function. The denominator $n_0/(k-1)$ can be intuitively understood as each sequential sample avoiding $\mathcal{X}_1$ would add $1/(k-1)$ sample to each of components $2, \cdots, k$.

Alternatively, assume that there exists some j_0 such that $n_{j_0} \not\to\infty$ and bounded by $n_{j_0} \leq N_0 < \infty$ as $n \to \infty$. When n is large enough, the following holds since the RHS is a constant and the LHS has its denominator tending to infinity while its numerator is bounded.

$$\frac{\max_{j\in A} \max_{\boldsymbol{x}\in\mathcal{X}_j} \mathrm{EI}_j(\boldsymbol{x})}{\min_{j\in A} n_j} \leq \frac{\max_{\boldsymbol{x}\in\mathcal{X}_{j_0}} \mathrm{EI}_{j_0}(\boldsymbol{x})}{N_0}$$

Therefore, we can increase the sample size $n > n_0$, then all weighted acquisition functions from components $2, \cdots, k$ would have maxima less than the maximum of $\mathrm{EI}_1(\boldsymbol{x})$. Therefore, the $(n+1)$-th sequential sample must be selected as the maximizer of $\mathrm{EI}_1(\boldsymbol{x})$ in $\mathcal{X}_1$. Following similar arguments on $n_j - \min_j n_j$, we can see that $\min_{j=1,\cdots,k} n_j \to \infty$ as $n \to \infty$.

When all $n_1, n_2, \cdots, n_k > 0$, by our algorithm, for the sub-sequence

$$\begin{gathered}\boldsymbol{x}_1^{(j)}, \boldsymbol{x}_2^{(j)}, \cdots, \boldsymbol{x}_{n_j}^{(j)}, \\ \boldsymbol{x}_J^{(j)} \coloneqq \arg\max_{\boldsymbol{x}\in\mathcal{X}_j} \mathrm{EI}_j(\boldsymbol{x})/J,\end{gathered}$$

for $j = 1, 2, \cdots, k$. It is straightforward to see that $\boldsymbol{x}_J^{(j)}$ is also the $\arg\max_{\boldsymbol{x}\in\mathcal{X}_j} \mathrm{EI}_j(\boldsymbol{x})$. Due to the fact that components are independent, each sub-sequence constitutes a sequential sampling within the component domain $\mathcal{X}_j$ with the same strategy. We use the assumption on $f\,|_{\mathcal{X}_j}$ and apply Theorem 5 of Bull (2011) to each of these k independent components ($j = 1, \cdots, k$) that

$$\sup_{\|f|_{\mathcal{X}_j}\|_{\mathcal{H}_{\theta^U}(\mathcal{X}_j)} \leq R_j} \mathbb{E}_f \left| f\,|_{\mathcal{X}_j}\,(\boldsymbol{x}_{n_j}^{(j)*}) - \min_{\mathcal{X}_j} f\,|_{\mathcal{X}_j} \right| =$$

$$O\left(\left(\frac{n_j}{\log n_j}\right)^{-\nu/d} (\log n_j)^{\alpha_j}\right),$$

and we take the maximum (since $k < \infty$) on the LHS to yield the stated result (3) with $\alpha = \max_j \alpha_j > 0$ and $R = \min_j R_j > 0$. □

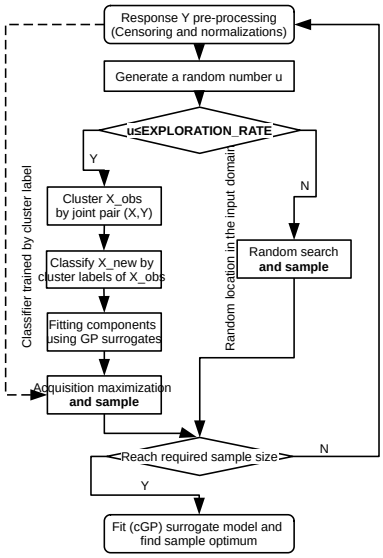
Our result is slightly weaker than Theorem 5 in Bull (2011) in the sense that the same claim of error in terms of the supremum norm holds for every sample size $n > 0$. Our results can be applied to quite general additive GP surrogate models with mutually independent components. One step further, Wynne et al. (2021) proved that even the ν are mis-specified, the convergence may still hold with a worse rate.

This discussion reveals the effect and importance of re-weighting acquisition functions in the cGP algorithm. Assumptions B and C focus on the smoothness for f within each component. This convergence would not hold if EI is not weighted, regardless the distribution assumption of $\boldsymbol{x}_i's$. However, we shall point out that our Algorithm 1 would update the partition induced by the cluster-classification step for each sampling step. Therefore, this proposition does not cover exactly cGP, but we provide a justification for weighting acquisition functions when we use partition-based models in a Bayesian optimization setting. Besides, we do observe empirically that the partition usually becomes stable after sufficiently many samples are collected, so the simplification is not groundless.

That being said, we still need to rigorously prove clustering and classification consistency (in sense that the decision boundaries coincide with non-smooth borders with an appropriately chosen ξ). This remains an unsolved theoretic question in the current paper, but usually a reasonable choice of ξ or even $\xi = 1$ can lead to empirically satisfying optimization results.

C Flowchart and Configuration Parameters of cGP Algorithm 1

In this section, we provide a visual comparison between the complexities of cGP and GP, as a supplment to the discussion of algorithmic complexity in the Section 4.1.4.



Parameters of cGP

Sec.	Parameter	Description
Sec. 2.2.1	Clustering method	Choose a clustering method: { Bayesian Dirichlet GP mixture (DGM), K-means }
	Number of maximal clusters (if DGM is chosen)	Maximal possible number of clusters (provided by Bayesian-Dirichlet mixture) in the cGP surrogate model (default=3).
	Number of clusters (if K-means is chosen)	Number of clusters given by the user (default=3).
	Number of neighbors	Parameter for k-NN (k-nearest neighbors) to assign/classify new locations to fitted clusters (default=3)
Sec. 2.2.3	Exploration rate	Probability that the next sequential example would be sampled according to the acquisition maximization [0, 1]

Figure 10. The algorithm flowchart for the cGP surrogate model. The dashed line indicates the usual GP surrogate model, which does not include a partition scheme. The detailed psuedo-code is available in Algorithm 1. Note that in each step, the clustering and the classifiers are re-trained with the newly sampled observation.

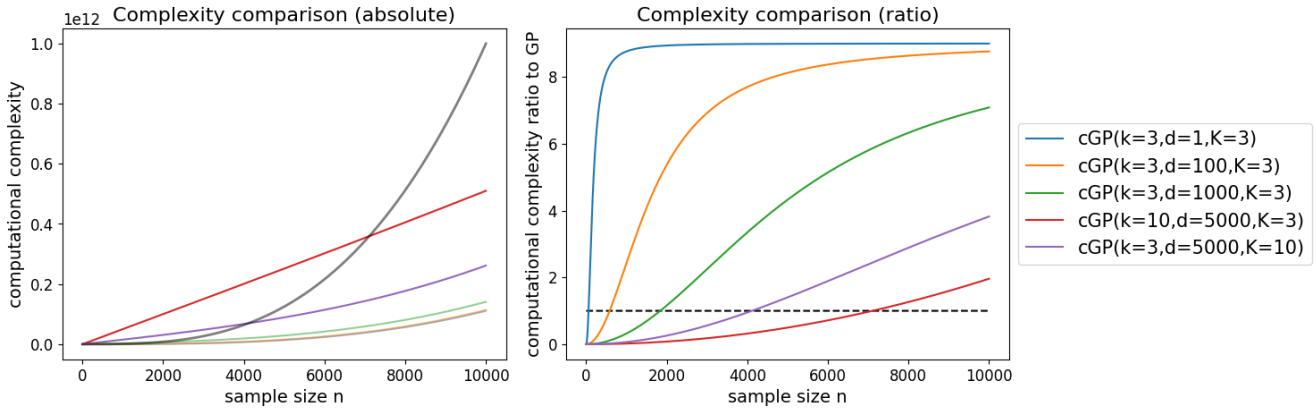


Figure 11. The complexity comparison plots of GP and cGP (with k-means and K-NN). The left panel shows the absolute complexity of GP (the black solid line) and cGP. the right panel shows the ratios between GP complexity and the corresponding cGP complexity (ratios greater than 1 (above the black dashed line) mean that cGP is more efficient).

D Additional Results of Experiments in Section 4.2.1

D.1 Recorded matmul data from HPC

We provide the pseudo-code for the 6-loop block matmul application in Algorithm 15. The following Figure 12 displays the actual computational speed of the matmul obtained from an HPC Cori Haswell computing node, with an L1 cache size of 32KB, L2 cache size of 256KB and L3 cache size of 40960KB. All cache-line sizes are 64 bytes.

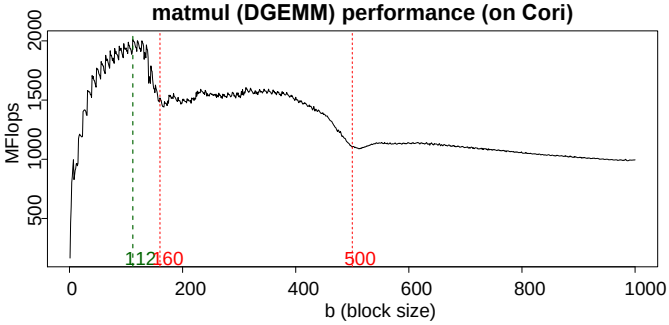


Figure 12. The computational speed for matrices of size $n = 1000$. This is the true black-block function obtained from a Haswell node. We also use a green dashed line to indicate the actual maximum $(\boldsymbol{x}_{\max}, f_{\max}) = (112, 2010.702)$; and red dashed lines for the regime cutoff $\boldsymbol{x} = b = 160, 500$ discussed in the text.

D.2 Selection of exploration rates

We first investigate the effect of exploration rates on the cGP model for matmul application in this section as shown in Figure 13.

In Figure 13, we show how the cGP fit changes with different exploration rates, which are defined in the flowchart (see Figure 10) and affects the random sampling procedure. If the exploration rate is 0.6, then with a probability 0.6

Algorithm 2: Blocked Matrix Multiplication (naive matmul)

Data: Matrices A and B, block sizes bn, bm, bl, timeout, matrix dimensions n, m, l
Result: Matrix C = A*B

```
Initialize matrices A, B, and C;
for in = 0 to n/bn do
    for kl = 0 to l/bl do
        for jm = 0 to m/bm do
            for x = in × bn to min((in + 1) × bn, n)
             do
                for z = kl × bl to
                 min((kl + 1) × bl, l) do
                    for y = jm × bm to
                     min((jm + 1) × bm, m) do
                        C[x][z] =
                          C[x][z] + A[x][y] × B[y][z];
                    end
                end
            end
        end
    end
end
Output: Matrix C, Mflops/s rate, and summary;
```

we choose the next sequential sample with acquisition maximization; and with probability 0.4, we randomly choose the next sample from the input domain (i.e., a random block size b).

It is not difficult to observe from Figure 13 that, with a lower exploration rate, the partition components remain similar, where for all exploration rates the drops at $b = 160$ and $b = 500$ are clearly indicated by the change of components. These two changes of regimes are caused by the overflow of 3 $b \times b$ matrices A, B and C from L1 and L2 cache. However, with a lower exploration rate, we also observe that the sample locations are more spread over the input domain. More investigations for the effect of exploration rates and comparison against non-stationary kernels are presented in Appendix D.

D.3 Supplementary experiments for block sizes

In Table 3, we also provide the percentage (among 100 runs with different shared random seeds) of runs that cGP outperforms simple GP surrogate models, as a quantitative supplement to the illustration in Figure 15, which supports the claim that our cGP surrogate model with the setups above (and different exploration rates) would consistently behave similarly like a simple GP surrogate model, while providing the non-smooth information of the black-box function in terms of both partitions and number of

As a supplement to the Figure 3, we also include exemplar results when we use Matérn 3/2 kernel with an additive dot product kernel. This creates a standard non-stationary kernel GP surrogate. Yet we still observe over-smoothing when there are few samples, and the sudden increase of uncertainty near the second drop in Figure 14.

Besides, the non-stationary GP surrogate does not provide any partitioning information. Below, we present the repeated experiment summaries for the matmul experiment in Section 4.2.1 in Figure 15 and Table 3.

D.4 Experiments for varying matrix sizes

Besides the problem setting of changing block size in the matmul application, we can also fix and optimize our blocking strategy but test the optimized blocking strategy with different matrix sizes. In this experiment, when the matrix size exceeds the memory cache size, the computational speed would experience an immediate drop, as we would observe in Figure 16 below. The black-box function f in this application is still computational speed, but the tuning variable $\boldsymbol{x}$ is the matrix size.

All of their cache-line sizes are 64 bytes, equivalent to 8 double precision floating point numbers. Based on this consideration, we can sample the block sizes b that are multiples of 8 in this example.

To eliminate the randomness caused by a single run, we repeat cGP (and GP) for 100 different random seeds with 10 pilot random samples for each run. In Figure 17, we compare the performance of a simple GP surrogate model searching over any integer $[16, 4096] \cap \mathbb{Z}$ against our cGP model. Our cGP model searches over integers that are multiples of 8 over $[16, 4096]$. Qualitatively, it is not hard to see that cGP performs better than the GP model under almost all sample sizes.

In this matrix multiplication example with varying matrix sizes, the cause of non-smooth points (i.e., reduction of communication; overflow of fast cache) and different kinds of behavior are clear and observed in the recorded dataset as shown in Figure 16. Our optimized strategy would attain the best performance on the matrix with matrix size $\boldsymbol{x}_{\max} = 256$. In the summary Figure 17, cGP surrogates behave similarly to the GP model when there are few samples; but when there are enough samples, the cGP model clearly identifies different partition regimes much better with a reasonable accuracy improvement. Therefore, the cGP model searches the optimum $f_{\max}$ and $\boldsymbol{x}_{\max}$ more efficiently compared to a simple GP surrogate, with more evidence shown in Table 7.

D.5 Hardware analysis

To better understand the performance behavior, we provide cache-miss profiling results of the matrix multiplication for varying matrix sizes. We used Linux's `Perf` to collect the cache miss rates. The profiling results show that the cache miss rate at each cache level has a non-smoothness point when exceeding a certain matrix size. We show the peak performance and cache traffic for only one execution of the matmul application. The execution is done on the same node type with the Haswell architecture as mentioned in Section 4.2.1. We obtain the data read/load traffic of L1 (Data cache, 32KB), L2 (Unified cache, 256KB) and L3 (Unified cache, 40960KB; also known as LLC, last layer cache) cache.

Theoretical calculations concerning only the point whether matrices fit into storage show that, ideally, the matrix size that makes the matrices exceed the L1 (≈52), L2 (≈105) and L3 (≈1322) caches (using $\sqrt{\frac{1}{3}\text{cache bytes}/8}$

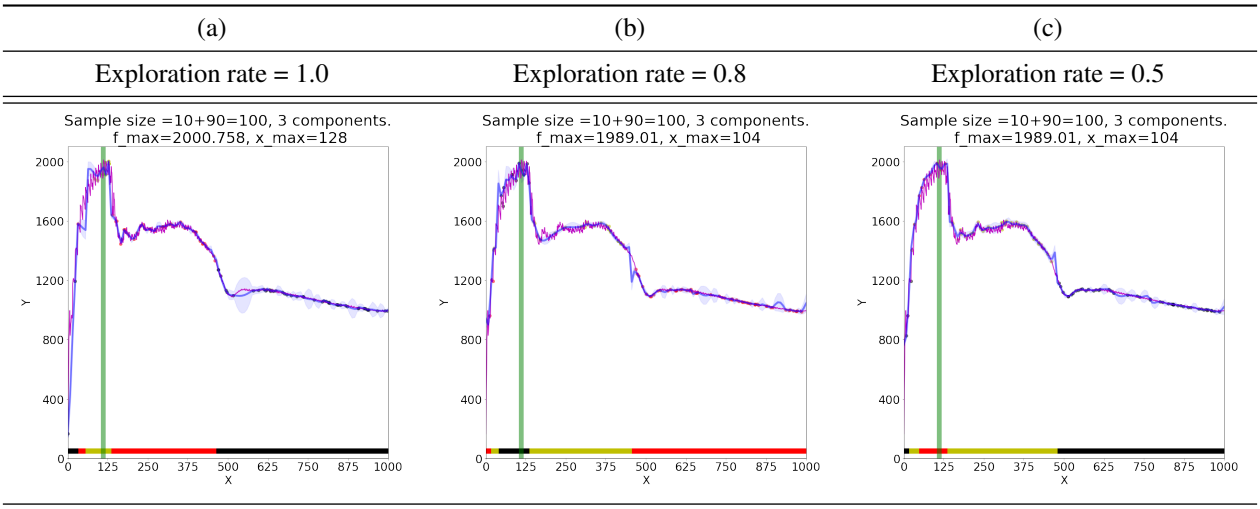


Figure 13. The computational speed function and fitted surrogate models. The fit from GP surrogate model with 60 sequential samples and exploration rate (a) 1. (b) 0.8. (c) 0.5.

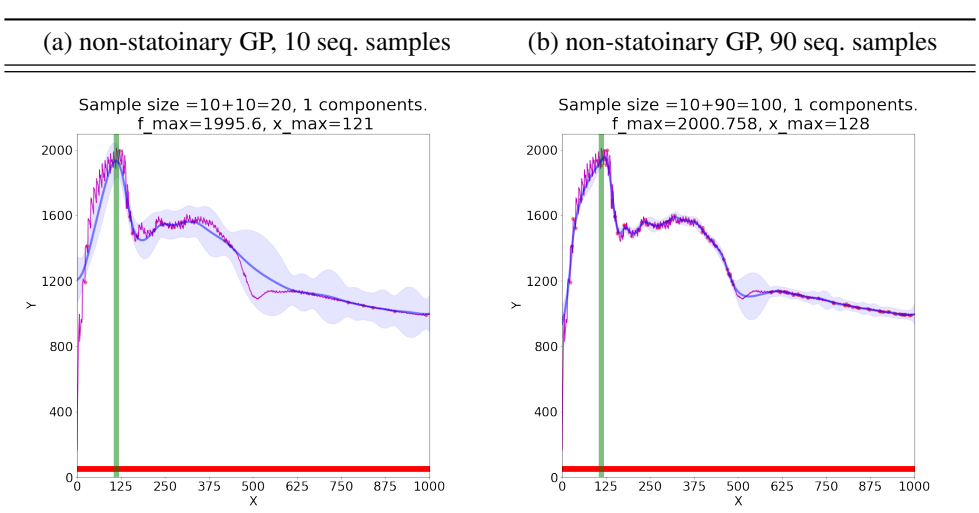


Figure 14. The computational speed function and fitted surrogate models. The dark blue solid line and light blue shaded area are the mean and variance of fitted prediction models. $(\boldsymbol{x}_{\max}, f_{\max})$ records the optimal block size and the actual optimal speed is illustrated by green vertical line on the figure. Bottom colored bars with different colors indicate to which cluster certain portions of the domain belong (but color has no semantic meaning). We also use magenta solid lines to overlay the truth from Figure 12 for comparison purposes. (The exploration rates of these experiments are all 1.)

Table 7. (a) The percentage of cGP fitted surrogates with optima equal or better than the baseline GP surrogate under different exploration rates. (b) The number of cGP fitted surrogates that attain the actual optimal matrix size $\boldsymbol{x}_{\max} = 256$ under different exploration rates. A simple GP surrogate model would only reach $\boldsymbol{x}_{\max} = 256$ **once** in 100 different random seeds. (c) The average number of components in the fitted cGP surrogate models among 100 different random seeds.

		(a)			(b)			(c)		
Exp. rate		1.0	0.8	0.5	1.0	0.8	0.5	1.0	0.8	0.5
Sample size	$n = 10$	0.57	0.69	0.78	2	2	2	1.79	1.85	1.83
	$n = 30$	0.65	0.74	0.82	3	3	5	2.08	2.10	2.20
	$n = 50$	0.67	0.74	0.85	4	5	9	2.15	2.20	2.27
	$n = 70$	0.62	0.74	0.82	4	6	11	2.24	2.31	2.38
	$n = 90$	0.61	0.70	0.80	4	7	12	2.30	2.37	2.38

for accessing three double precision square matrices in the matmul application.) These threshold matrix sizes are illustrated by blue vertical lines in the Figure 18. The statistic we monitor is the *cache miss rate* (Its definition is

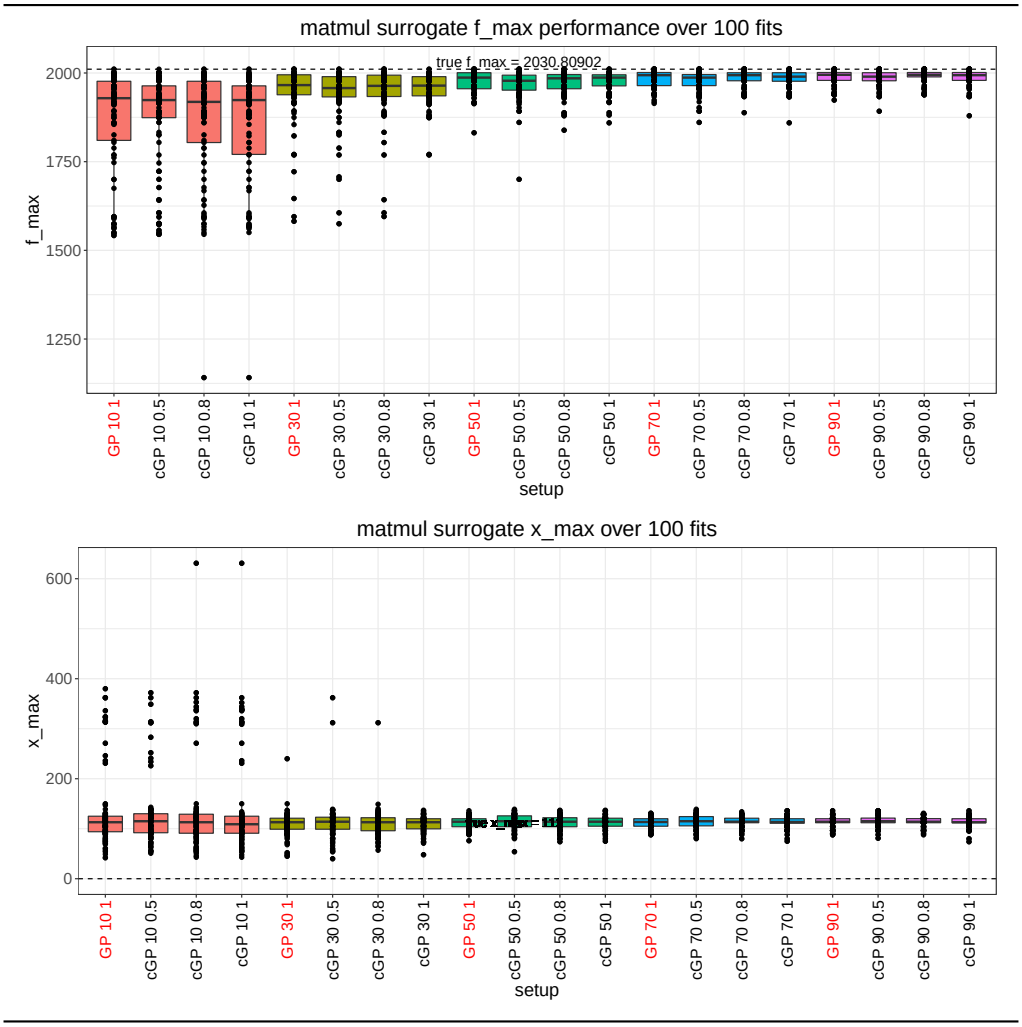


Figure 15. The box plot of $f_{\max}$ (top) and $\boldsymbol{x}_{\max}$ (bottom, log scale) elicited from sequential samples of 100 fitted surrogate models, each dot in the box plot represents the optimal point in each run (hence each box plot contains 100 points). The sequential sample size varies from 10 to 190 with 10 pilot samples. The exploration rates of the cGP model are chosen to be 0.5, 0.8 and 1 (x-axis format: model/sequential sample size/exploration rate).

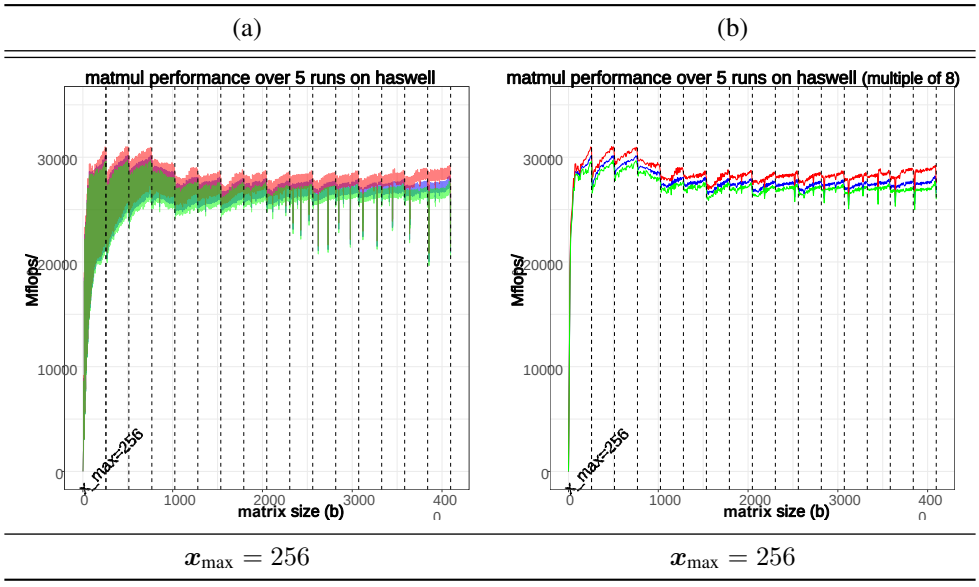


Figure 16. The computational speed for matrices of sizes varying from 16 to 4096. The red line shows the maximum among 5 runs; the green line shows the minimum among 5 runs; the blue line shows the average among 5 runs. (a) The matrix size varies from 16 to 4096, obtained from 5 different runs. (b) The matrix sizes are multiples of 8, varying from 16 to 4096. We also use dashed lines to indicate the matrix sizes that are multiples of 256 where we expect a drop by machine architecture.

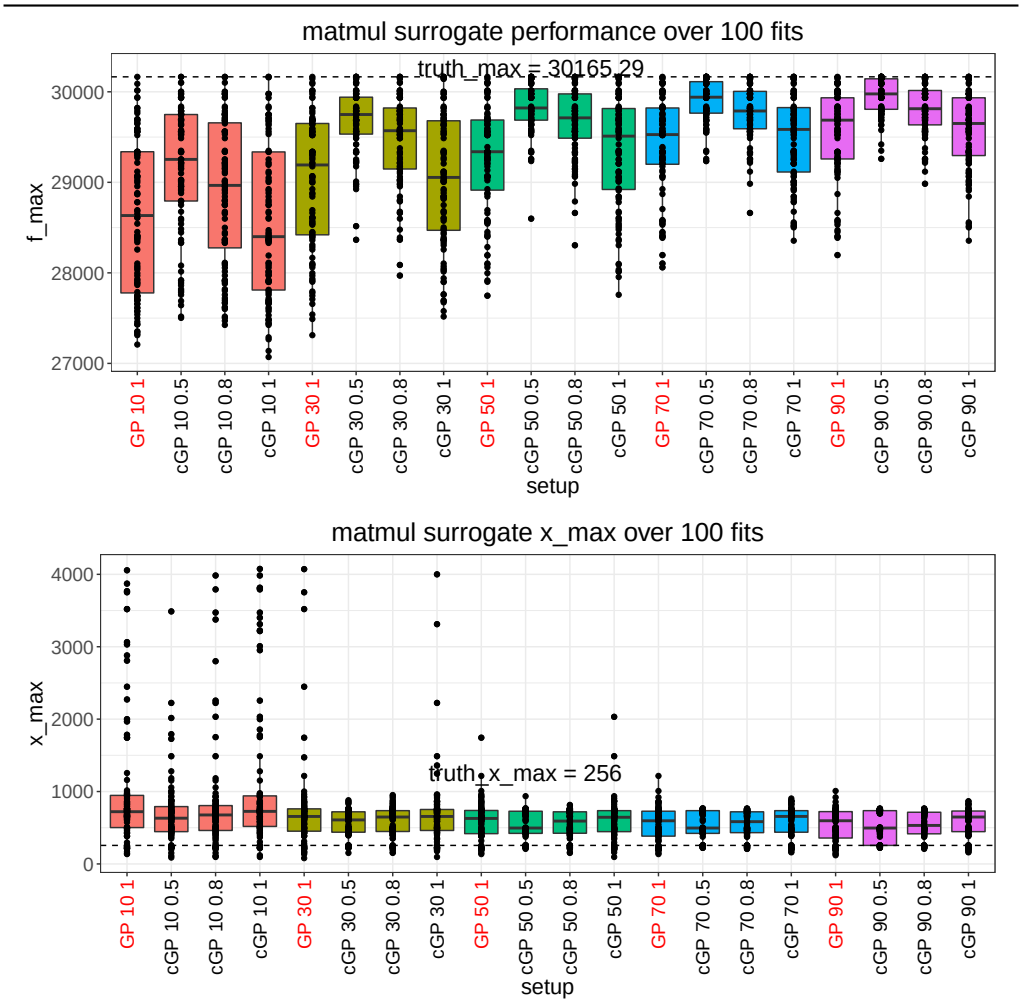


Figure 17. The box plot of $f_{\max}$ (top) and $\boldsymbol{x}_{\max}$ (bottom) elicited from sequential samples of 100 fitted surrogate models, each dot in the box plot represents the optimal point in each run (hence each box plot contains 100 points). The sequential sample size varies from 10 to 190 with 10 pilot samples. The exploration rates of the cGP model are chosen to be 0.5, 0.8 and 1.

slightly different for L1, L2 and L3 cache, but generally it is the number of cache misses among all cache read/write during a certain procedure.) For L1 cache, we monitor the data load miss rate; for L2 cache, we monitor the data demand miss rate and all demand miss rate; for L3 cache, we monitor the data load, store miss rates and their sum. These hardware measurements lend support to our claim that the black-box function f does have some non-smoothness and partially explain why the partition obtained by the cGP model is beneficial. By incorporating the non-smoothness in the tuning context, our model also induces informative partitions consistent with non-smoothness observed in the profiling.

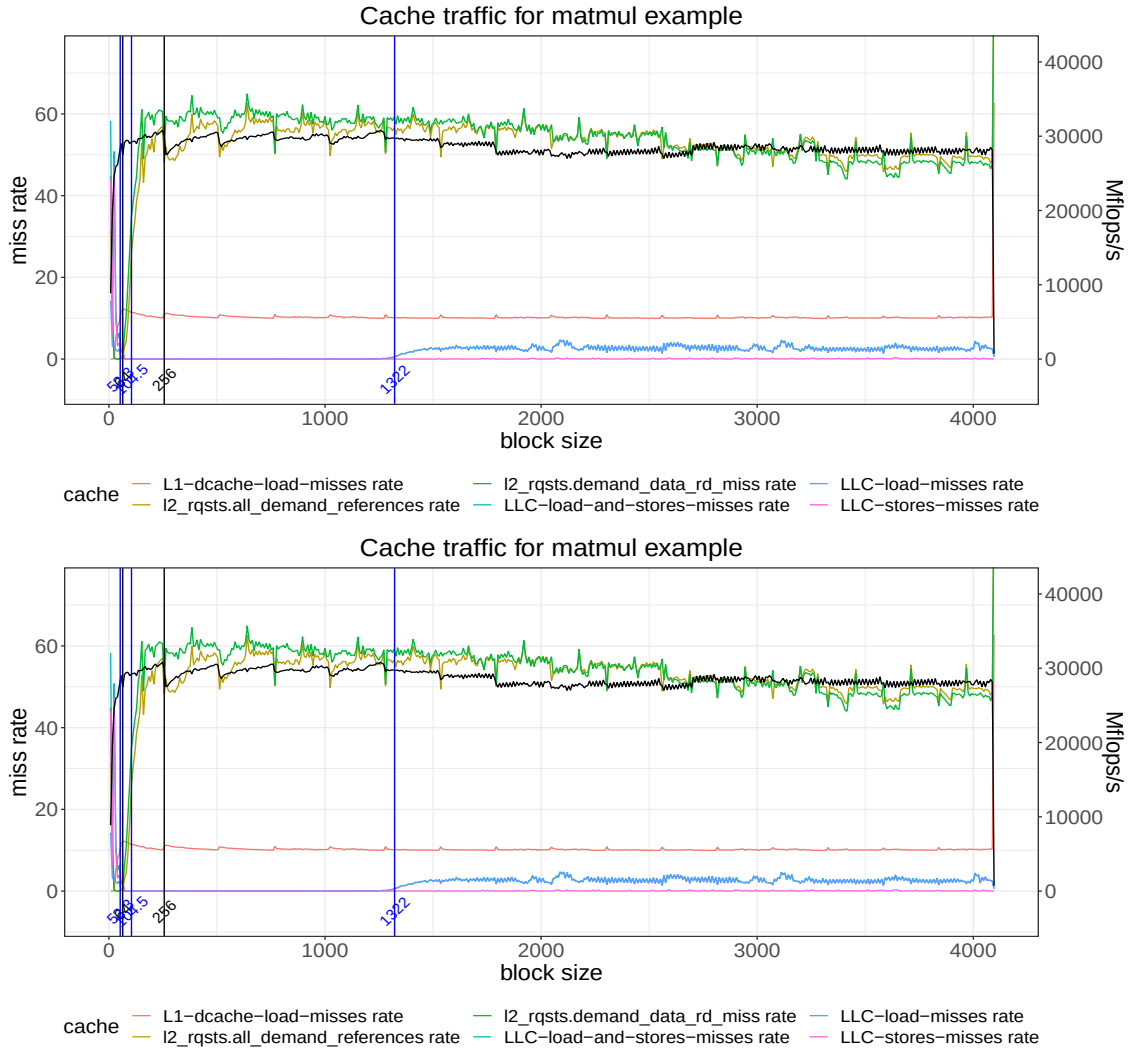


Figure 18. Cache traffic and computational performance for one batch of the matmul application. The top panel shows the traffic when the block size changes among multiples of 8 from 10 to 4096; the bottom panel shows a zoom-in version of the top panel focusing on the traffic for block size less than 512. The x-axis is the block size in the matmul application. The left y-axis is the percentage of miss rate for each type of cache. The right y-axis is the computational speed of the matmul application.

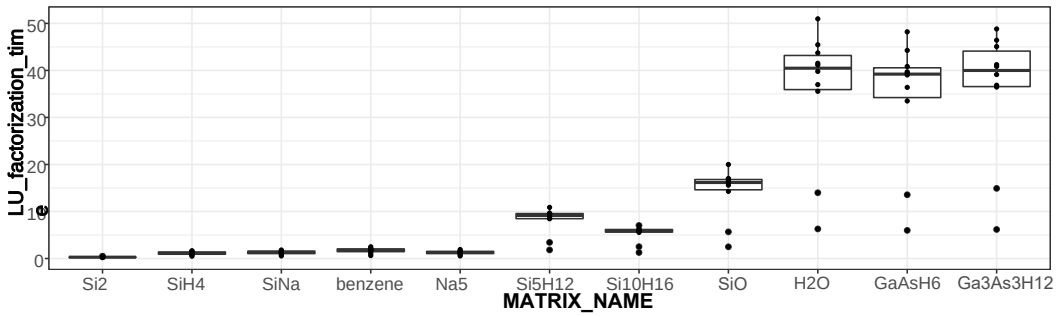


Figure 19. Boxplots for LU factorization time when 4 parameters in the SuperLU application are randomly chosen with the same random number generator. To reduce randomness, we repeat 10 times for each parameter configuration and use the average value.

E Summary of SuperLU Testing Matrices

As above, we fix the random seed to ensure the pilot samples are the same for each surrogate model. Even with the same random seed, the running time has some noise due to the working condition of each node. In our experiments, we use the DGM classifier with the maximal number of components k=2, 4 and exploration rates τ=0.5, 0.8, 1.0 for the cGP model and compare their performance against the simple GP surrogate model.

In Table 6 and Figure 19, we display the approximate time range for the LU factorization when cGP explores the 4-parameter space with different parameter configurations, the details of these matrices are available at https://sparse.tamu.edu/PARSEC. The matrix dimension d is

also displayed ($d \times d$ square matrices) and the optimized LU factorization time obtained by simple GP surrogate model (averaged over 10 times) is provided for comparison as well.

F Additional Result of Experiments using Simulation Functions

In this high-dimensional application we study the piston cycle time model which was proposed in Kenett et al. (2013) for quality control in industrial applications, and has been well-studied in Owen et al. (2014). It is important to tune the 7-dimensional variables (e.g., piston surface area, initial gas volume, etc.) of a piston to manufacture products whose minimum and maximum cycle time are within a certain range. This piston model function is a continuous function describing the cycle time of a piston, defined on a 7-dimensional domain with all of its tuning variables being continuous. Although the model function is continuous, empirically, it does produce dramatic changes that make it almost look "non-smooth", as we already observed in the Bukin N.6 function (Figure 9). In this specific application, both the maximum and minimum are of interest. Therefore, we expect that we need to fit a good surrogate to find all function optima.

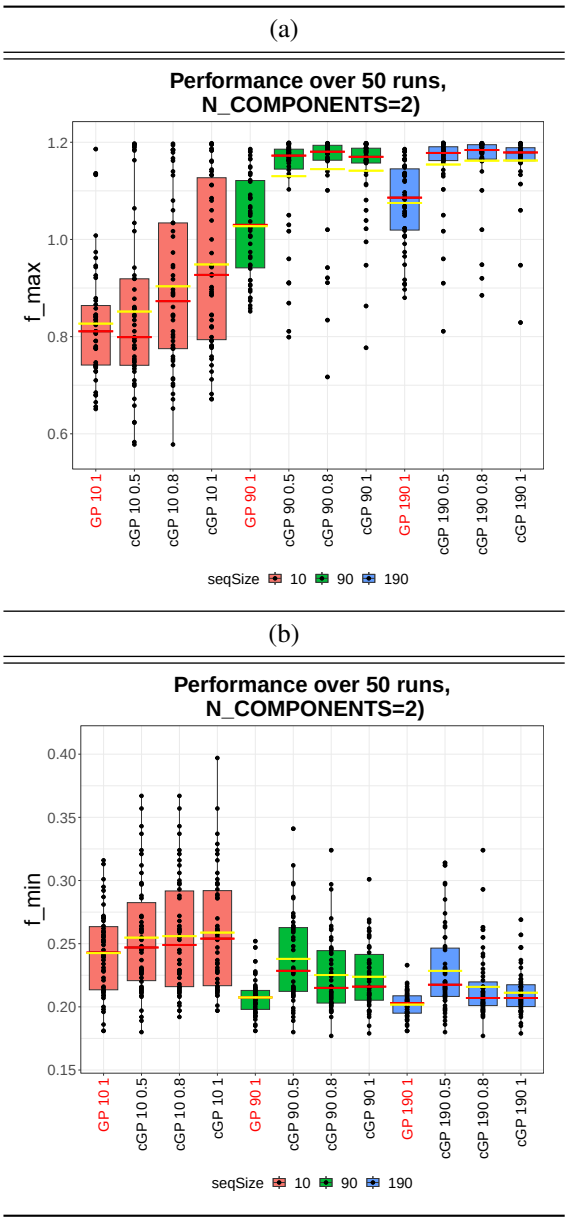


Figure 20. (a) The box plot of $f_{\max}$ with y-scale $(0.5, 1.2)$ and (b) $f_{\min}$ with y-scale $(0.1, 0.8)$ from sequential samples of 50 fitted surrogate models of piston simulation function, each dot in the box plot represents the optimal point in each run (hence each box plot contains 50 points). The sequential sample size varies from 10 to 190 with 10 pilot samples. Exploration rates of the cGP model are chosen to be 0.5, 0.8 and 1. We also show the sample mean (yellow horizontal bar) and median (red horizontal bar) among 100 optima points.

In Figure 20, we show the performance of GP and cGP surrogate models. We use the DGM classifier and set the maximal number of components to be 3 for cGP and compare against the GP with the same kernel. The choice of the number of components does not affect the results much for this application. However, we still calibrate the exploration rate and increase the allowed sequential sample size in our exploration, and show all boxplots on the same scale. We can see that cGP clearly outperforms GP when we search for $f_{\max}$. Panel (a) in Figure 20 shows that cGP systematically gives larger $f_{\max}$ and performs much better than GP surrogate when there is a lack of sequential samples. When we search for $f_{\min}$, panel (b) in Figure 20 shows that both GP and cGP produce similar $f_{\min}$. However, cGP still explores the high-dimensional domain more efficiently and gives the smallest $f_{\min}$ among GP surrogates. Note that in the $f_{\max}$ situation, cGP produces a roughly 0.10 in function value improvement on average (difference between sample means of $f_{\max}$); but in the $f_{\min}$ situation, cGP only loses less than 0.02 function value on average. Considering the fact that this function is defined on a 7-dimensional domain, we observed that cGP can approximate the underlying black-box function quite well with relatively few samples.

In the table at the bottom of Figure 20 we provide a similar quantitative comparison for GP and cGP surrogates. The fraction of out-performing batches shows that the cGP with a high exploration rate is preferred when looking for $f_{\max}$. Although the quantitative percentage does not favor cGP when searching for $f_{\min}$, by comparing the sample means and medians obtained from GP and cGP, the claim that cGP does not give a much worse $f_{\min}$ is supported. This kind of trade-off is common in choosing different surrogates. It is worth pointing out that cGP models can have a significant gain like this even if a real "non-smoothness" does not occur.

In Table 8, we show the results of tuning the four-segment robot arm cost function with 8-dimensional continuous input variables. The function is relatively smooth and possess symmetries; however, it will fail a couple of state-of-the-art ensemble methods when the regress is fitting an inherently non-smooth function (e.g., the gradient boosting regression tree) but favor a smoother surrogate model like GP. This conversely provides support of our cGP design not using fully non-smooth regressors like decision trees. Thanks to the flexibility from using DGM as partitioning models, we can adaptively determine the number of jumps and drops.

The values (0.5, 0.8, 1.0) next to cGP might indicate different configurations or parameters used within the cGP model for these experiments. The values in each cell represent the best values obtained by each surrogate model,

Table 8. Different competitors on minimization of the (un-normalized) four-segment robot arm cost function in An and Owen (2001) among 50 different random seeds with different sequential sample sizes. The GP would gives a perfect fit for the noiseless target function as a baseline surrogate, therefore we compare all the rest competing surrogate models. The best model among cGP (DGM with 3 clusters), Forest and GBRT are indicated by bold fonts.

	cGP, exp. rate =			Forest	GBRT	GP*
size	0.5	0.8	1.0			
10	0.3633	0.3633	0.3633	0.3199	**0.3061**	0.3633
30	0.1188	0.0902	**0.0684**	0.1520	0.1599	0.0713
50	0.0802	0.0439	**0.0356**	0.1012	0.1005	0.0344
70	0.0550	0.0325	**0.0266**	0.0884	0.0674	0.0190
90	0.0435	0.0235	**0.0175**	0.0716	0.0558	0.0059

lower values indicate better performance. We usually do not know the underlying black-box function is smooth and can use simple GP without any consideration. Neither of Forest and GBRT (nor existing surrogate models, to our best knowledge) surrogates are capable of handling smooth truth, therefore we observed that they are converging quite slowly. However, we can see that cGPs are consistently better than the Forest and GBRT. This practice of "not knowing smoothness but use a non-smooth surrogate" obviously harms the effectiveness of the surrogate-based optimization. Our cGP stands out in this situation that it (along with DGM that allows 3 maximum of clusters) reduces to only one component when it cannot detect any non-smoothness in the observed data. This also echoes the Figure 5 and column (c) in Table 3 that the cGP stabilize at the "correct" number of components. Actually, cGP produces only 1 component (i.e., reduces to GP, the perfect surrogate) about 50% of times in these runs. While GP (since it is the truth in this case) is better than cGP, but their difference decreases as the sampling size increases from 50 to 90.

G Key parameters in cGP Implementation

There are two classes of parameters we can set for the current cGP implementation summarized here and in Figure C.

G.1 Model specific parameters

These parameters control the behavior of the surrogate model and numerical stability, usually not data-dependent.

We can choose methods for fitting GP surrogate model. There are two fitting methods provided in the implementation. Frequentist method is usually faster yet it sometimes causes inaccurate estimates of kernel parameters. Bayesian method (MCMC or HMC) is slower yet it provides more robust estimates of kernel parameters and better overall reproducibility.

- *METHOD = 'FREQUENTIST' or 'BAYESIAN'*
  - 'FREQUENTIST' method means that the (hyper-)parameters of the surrogate model are estimated by maximum likelihood estimates using optimization (e.g. lbfgs). This would be usually faster than a fully Bayesian approach at the cost of less robustness and reproducibility.
  - 'BAYESIAN' method means that the (hyper-)parameters are estimated by putting a (Gamma) prior and obtain posterior via Bayes' theorem. The estimated value of (hyper-)parameters would be posterior sample mean. The Bayesian model is fitted by HMC or MCMC.
    Following 3 parameters are only for HMC Bayesian sampling, they would be in effect when METHOD = 'BAYESIAN'
    - *N_BURNIN*: The number of burn-in steps in posterior sampling, in each of the steps, only the likelihood surrogate model would be evaluated (therefore, it is not as expensive as evaluation of black-box function f).
    - *N_MCMCSAMPLES*: The number of posterior samples to be drawn after burn-in.
    - *N_INFERENCE*: The number of posterior samples that should be used for posterior estimate calculation.

We can determine how often the sequential sampling procedure should use a random sampling scheme by adjusting the probability of getting a random sampling.

The first trade-off in the cGP is as follows. If the black-box function is believed to be very oscillatory, a lower EXPLORATION_RATE would allow the sequential sampling exploring the domain faster instead of being trapped in a local optima.

Sometimes the random sampling, instead of surrogate model guided sampling (i.e. via acquisition function maximization), would accelerate the exploration of the domain. Random sampling allows faster exploration over the input domain, especially for wiggly black-box function f.

- *EXPLORATION_RATE = 1*
  Exploration rate is the probability (between 0 and 1) that the next step produced by maximizing acquisition function.
  - If EXPLORATION_RATE = 1, then all the sequential samples would be chosen by maximizing the acquisition function over the domain.
  - If EXPLORATION_RATE = 0, then all the sequential samples would be chosen by random uniform sampling over the domain.

To set up the clustering and classification method used for cGP surrogate model, the very first question we should ask is whether we believe there are different regimes in the black-box function. Although with sufficient samples cGP and GP shows little difference (since the clustering scheme would essentially be smoothed out by sufficient samples), when the samples are limited each component of cGP would require more parameters to be fitted. Balancing the number of maximum components and the number of samples belonging to each component introduces the second trade-off: sample size-component trade-off.

The second question is how do we assign new locations to each of existing cluster components. In the current

implementation, we use K-nearest neighbors to assign new locations to fitted clusters. Therefore, if the black-box function is expected to be smooth within each regime, then a larger N_NEIGHBORS would produce a better result.

- *N_COMPONENTS*: This is the maximal possible number of clusters (provided by Bayesian-Dirichlet mixture) in the cGP surrogate model. This option will be used only if NO_CLUSTER=False.
- *Y_AMPLIFY*: This parameter ξ is used only for XY-joint pair $(\boldsymbol{x}, \xi y)$ clustering. When doing clustering, we can multiply Y by a factor Y_AMPLIFY$=\xi$ to make sure that the importance of input X and response Y are both taken into consideration at appropriate scales in the clustering procedure. Be default this is 1, however, sometimes we may want to adjust this to make X and Y are roughly on the same scale. That is to say, we use the joint input-response pair $(\boldsymbol{x}, \xi y)$ for clustering algorithm. When $\xi = 1$, it is exactly the algorithm we described in the main text; when $\xi = 0$, it reduces to clustering based on input location $\boldsymbol{x}$ only.
- *N_NEIGHBORS*: When assigning predictive locations to different cluster components by K-nearest neighbors method, and when $K = 1$ this reduces to allocation to the nearest cluster.

G.2 Application specific parameters

These parameters shall be chosen depending on the specific data source or application, and would be data-dependent.

We can determine the number of pilot and sequential samples based on the budget and time limit. Under the limited budget, more pilot samples usually lead to a better fitted surrogate model. Fewer samples are needed if the signal-to-noise ratio of the black-box function is high; fewer samples are needed if the black-box function is smooth.

- *N_PILOT*: The number of pilot samples need to draw from the black-box function f.
- *N_SEQUENTIAL:* The number of sequential samples need to draw from the black-box function f.