Closed-Form Active Learning using Expected Variance Reduction of Gaussian Process Surrogates for Adaptive Sampling

George P. Kontoudis and Michael Otte

***Abstract*— Adaptive sampling of latent fields remains a challenging task, especially in high-dimensional input spaces. In this paper, we propose an active learning method of expected variance reduction with Gaussian process (GP) surrogates using a closed-form gradient. The use of closed-form gradient leads the optimization to find better solutions with reduced computations. We derive the closed-form gradient for active learning Cohn (ALC) using GP surrogates that are formed with the separable squared exponential covariance function. Moreover, we provide algorithmic details for the execution of the closed-form ALC (cALC). Numerical experiments with multiple input space dimensions illustrate the efficacy of our method.**

I. Introduction

Autonomous systems have shown substantial improvements in recent years, due to enhanced computational capabilities, development of software tools, and novel machine learning and control system techniques. Big data is a ubiquitous area of research, yet regression in high-dimensional input spaces remains an open research topic, even with smaller dataset size. A powerful technique for regression is the non-parametric Gaussian process (GP), but with high computational complexity requirements [1]. An approach to circumvent the complexity of GPs is to carefully select the input points so that similar prediction accuracy is achieved with much smaller dataset size. To this end, active learning of GPs is used to facilitate sequential growing of datasets and perform adaptive sampling [2]. However, when there are no closed-form solutions to the optimization of active learning, the use of approximation methods (e.g., finite difference method (FDM)) for gradient computation are promoted. For low-dimensional input fields of $D \leq 2$, gradient approximation methods are efficient, yet in high input space dimensions the approximation performs poorly.

Our aim in this work is to derive a closed-form solution for the optimization of active learning Cohn (ALC) [3] to reduce the iterations of active learning optimization, alleviate the computations, and improve the prediction accuracy in high-dimensional latent fields. Applications for realistic model learning include at least 6D input spaces such as helicopters [4], underwater vehicles [5], and model identification of pH process [6] to name few.

Related work: Although GPs produce accurate predictions, they scale poorly with the dataset size n. In particular, the training requires cubic computations $\mathcal{O}(n^3)$ to invert an $n \times n$ covariance matrix multiple times during the optimization and the prediction entails quadratic computations $\mathcal{O}(n^2)$. Although there are many efforts to reduce the computational complexity [7], [8], the workload still remains high.

To avoid the use of large datasets with GPs, sequential active learning with GP surrogates is an alternative approach [9, Chapter 6]. Seo *et al.* [3] introduced the first active learning methods using GP surrogates. In particular, the authors proposed active learning MacKay (ALM) which seeks to maximize the expected Shannon information gain by selecting as next input the location of maximum variance of the GP surrogate, and active learning Cohn (ALC) which aims to maximize the integrated deduced variance by selecting as next input the location that if sampled will reduce the overall variance of the GP surrogate model. Active learning for adaptive sampling is applied to other areas, such as computer experiments (or surrogates) [9], robotics [10], manufacturing [11], and multi-agent systems [12].

Another criterion for active learning with GPs exploits the expected Fisher information (ALFIE) [13]. ALFIE proposes next input locations that improve the GP surrogate model accuracy of hyperparameters, but not necessarily the accuracy of predictions. Liu *et al.* [14] introduced an adaptive sampling strategy that maximizes the expected prediction error by taking into account the variance and bias. Recently [2] surveyed adaptive sampling techniques with GP surrogates.

Bayesian optimization (BO) works similar to active learning (AL) methods by performing sequential adaptive sampling, but has a different objective. In particular BO aims to address global optimization problems [9], [15]. In other words, BO focuses on sampling strategies that identify the global minimum of unknown functions, while AL techniques focus on improving the prediction accuracy or reducing the uncertainty. This means that BO strategies may leave complete areas of the input space unexplored which is highly unlikely for most AL strategies. A data-driven methodology for adaptive sampling by minimizing the cumulative regret is presented in [10]. The methodology is evaluated with real world experiments using an autonomous underwater vehicle that collected water samples for marine ecosystem monitoring. In [16] the authors employ the GP upper confidence bound (GP-UCB) [17] to implement adaptive sampling with GP surrogate models. The methodology is evaluated with an unmanned surface vehicle that surveyed a region with an altimeter to create a bathymetric map. A multi-agent adaptive sampling methodology with mixture of GP experts for environmental monitoring is presented in [12]. The authors used the GP-UCB along with expected minimization and proposed

G. P. Kontoudis and M. Otte are with the Maryland Robotics Center and the Department of Aerospace Engineering, University of Maryland, College Park, MD, USA. E-mails: {kont,otte}@umd.edu.

This work was supported by the MRC Postdoctoral Fellowship Program and the Office of Naval Research via grant N00014-20-1-2712.

a centralized algorithm for a team of agents. In [18], an active learning methodology is discussed using sparse GP models and the online deterministic annealing (ODA). ODA allows pseudo-inputs to adjust their locations at each learning step.

Contribution: The contribution of this paper is the derivation of the closed-form gradient for active learning of GP surrogates using the separable squared exponential (SSE) covariance function. Numerical experiments validate the efficiency of the optimization with closed-form gradient for adaptive sampling in high-dimensional input spaces.

II. Active Learning with Gaussian Processes

In this section, we review Gaussian processes, active learning, and state the problem.

A. Gaussian Processes

The observation model is described by,

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

where $f(\boldsymbol{x}) \sim \mathcal{GP}(0, k_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}'))$ is a zero-mean GP with covariance function $k_{\boldsymbol{\theta}} : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$, $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^D$ is the input vector of D dimension, and $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$ is the i.i.d. noise with variance σ_ϵ^2. We select the general version of the most common covariance function, the separable squared exponential (SSE) which follows,

$$k_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp\left\{ -\sum_{d=1}^{D} \frac{(x_d - x'_d)^2}{l_d^2} \right\}, \tag{1}$$

where σ_f^2 is the signal variance and l_d the length-scale hyperparameter for the d-th direction of the input space. The main difference of the SSE covariance function as compared to the most commonly used squared exponential (SE) is that a separate length-scale hyperparameter is assigned for each input dimension, i.e., SSE has $l_1, \dots, l_D$ length-scale hyperparameters while SE has just one length-scale hyperparameter l. Thus, a GP with SSE covariance function can more accurately predict anisotropic latent fields. The overall goal is to predict a latent field f given data $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$ with $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^n$ the inputs, $\boldsymbol{y} = \{y_i\}_{i=1}^n$ the outputs, and n the number of observations. This is two-step probabilistic learning, where first the hyperparameters $\boldsymbol{\theta}$ are estimated and then a prediction takes place at unknown inputs $\boldsymbol{x}_*$.

1) Training: The first step is to train the hyperparameters $\boldsymbol{\theta} = (l_1, \dots, l_D, \sigma_f, \sigma_\epsilon)^\intercal \in \Theta \subset \mathbb{R}^{D+2}$ by maximizing,

$$\log p(\boldsymbol{y} \mid \boldsymbol{X}) = -\frac{1}{2}\left(\boldsymbol{y}^\intercal \boldsymbol{C}_n^{-1} \boldsymbol{y} + \log|\boldsymbol{C}_n| + n \log 2\pi\right),$$

where $\boldsymbol{C}_n = \boldsymbol{K} + \sigma_\epsilon^2 I_n$ is the covariance matrix with $\boldsymbol{K} = k_{\boldsymbol{\theta}}(\boldsymbol{X}, \boldsymbol{X}) \succeq 0 \in \mathbb{R}^{n \times n}$ the semi-positive correlation matrix.

2) Prediction: After training the hyperparameters $\hat{\boldsymbol{\theta}}$, the posterior distribution of the location of interest $\boldsymbol{x}_* \in \mathbb{R}^D$ yields a multi-variate normal distribution $p(y_* \mid \mathcal{D}, \boldsymbol{x}_*) \sim \mathcal{N}(\mu(\boldsymbol{x}_*), \sigma^2(\boldsymbol{x}_*))$ with prediction mean and variance,

$$\mu_{\text{full}}(\boldsymbol{x}_*) = \boldsymbol{k}_*^\intercal \boldsymbol{C}_n^{-1} \boldsymbol{y}, \tag{2}$$

$$\sigma_{\text{full}}^2(\boldsymbol{x}_*) = k_{**} - \boldsymbol{k}_*^\intercal \boldsymbol{C}_n^{-1} \boldsymbol{k}_*, \tag{3}$$

where $\boldsymbol{k}_* = k_{\boldsymbol{\theta}}(\boldsymbol{X}, \boldsymbol{x}_*) \in \mathbb{R}^n$ and $k_{**} = k_{\boldsymbol{\theta}}(\boldsymbol{x}_*, \boldsymbol{x}_*) \in \mathbb{R}$.

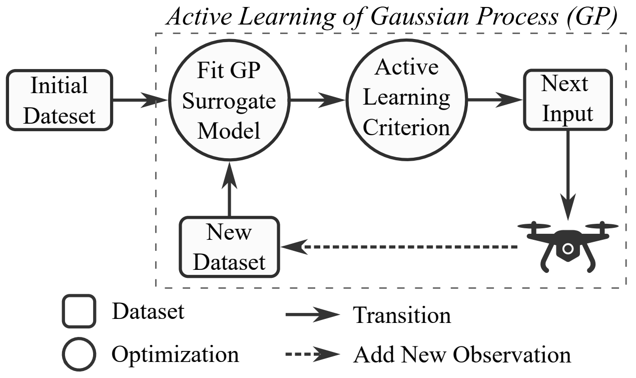

Fig. 1. Block diagram of active learning using GP surrogate models.

3) Complexity: The training step requires $\mathcal{O}(n^3)$ computations for the inverse of the covariance matrix $\boldsymbol{C}_n^{-1}$. Note that for every iteration of the training optimization a new inverse of the covariance matrix has to be computed, because the covariance matrix is a function of the optimizing parameter $\boldsymbol{\theta}$. The prediction mean (2) entails $\mathcal{O}(n)$ computations, while the variance (3) yields and $\mathcal{O}(n^2)$ computations.

B. Active Learning of Gaussian Processes

Since the training complexity of GPs is considerably high $\mathcal{O}(n^3)$, an alternative strategy is to carefully select the observation inputs $\boldsymbol{X}$ by sequentially growing the size n of the dataset $\mathcal{D}$ based on GP surrogate models. To this end, we can achieve similar model estimation and prediction accuracy with a significantly smaller dataset. We employ an active learning strategy for the decision making of new sampling inputs, as shown in Fig 1. We start with a small initial dataset $\mathcal{D}_n$ to build the first GP surrogate model with hyperparameters $\hat{\boldsymbol{\theta}}_n$. Next, we use an active learning criterion to find the next input $\boldsymbol{x}_{n+1}$. Then, the autonomous system is assigned to take an observation from the selected input $y(\boldsymbol{x}_{n+1})$. After gathering the observation, a new dataset is formed $\mathcal{D}_{n+1} = \mathcal{D}_n \cup (y_{n+1}, \boldsymbol{x}_{n+1})$. Using $\mathcal{D}_{n+1}$, a new GP surrogate model is fitted which subsequently is used to find the next input. The method terminates either until a condition is met or after predetermined iterations.

We select the active learning Cohn (ALC) [3] criterion that produces accurate predictions among other criteria [9]. The ALC criterion is defined as the negative expected reduction in variance over the input space $J := -\int_{\mathcal{X}} \Delta\sigma^2(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$. In other words, the criterion looks for an input that if observed, the overall variance of the GP surrogate will be reduced. Since this integral does not admit an analytical solution, the ALC is approximated by a summation over a reference set,

$$\begin{aligned}
\boldsymbol{x}_{n+1} &= \arg\min_{\boldsymbol{x} \in \mathcal{X}} \left\{ -\int_{\mathcal{X}} \Delta\sigma^2(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \right\} \\
&\approx \arg\min_{\boldsymbol{x} \in \mathcal{X}} \left\{ -\frac{1}{N_{\text{ref}}} \sum_{t=1}^{N_{\text{ref}}} \Delta\sigma^2(\boldsymbol{x}, \boldsymbol{x}_t) \right\} \\
&= \arg\min_{\boldsymbol{x} \in \mathcal{X}} \left\{ -\frac{1}{N_{\text{ref}}} \sum_{t=1}^{N_{\text{ref}}} \sigma_n^2(\boldsymbol{x}_t) - \tilde{\sigma}_{n+1}^2(\boldsymbol{x}, \boldsymbol{x}_t) \right\}
\end{aligned} \tag{4}$$

where $\Delta\sigma^2(\boldsymbol{x}) = \sigma_n^2(\boldsymbol{x}_t) - \widetilde{\sigma}_{n+1}^2(\boldsymbol{x}, \boldsymbol{x}_t)$ is the deduced variance, $\widetilde{\sigma}_{n+1}^2(\boldsymbol{x}, \boldsymbol{x}_t)$ is the approximated variance using the estimated hyperparameters from the previous step $\hat{\theta}_n$ and assuming that the candidate location $\boldsymbol{x}$ is added in the dataset, $\boldsymbol{x}_t$ is a reference input from the reference set $\boldsymbol{X}_{\text{ref}} \in \mathbb{R}^{N_{\text{ref}} \times D}$. The elements of the deduced variance $\Delta\sigma^2(\boldsymbol{x}, \boldsymbol{x}_t)$ take the form of,

$$\sigma_n^2(\boldsymbol{x}_t) = k_n - \boldsymbol{k}_{n,t}^\intercal \boldsymbol{C}_n^{-1} \boldsymbol{k}_{n,t},$$
$$\widetilde{\sigma}_{n+1}^2(\boldsymbol{x}, \boldsymbol{x}_t) = k_n - \boldsymbol{k}_{n+1,t}^\intercal \boldsymbol{C}_{n+1}^{-1} \boldsymbol{k}_{n+1,t},$$

where $k_n = k_{\boldsymbol{\theta}}(\boldsymbol{x}_t, \boldsymbol{x}_t) \in \mathbb{R}$, $\boldsymbol{k}_{n,t} = k_{\boldsymbol{\theta}}(\boldsymbol{x}_t, \boldsymbol{X}_n) \in \mathbb{R}^n$, $\boldsymbol{k}_{n+1,t} = k_{\boldsymbol{\theta}}(\boldsymbol{x}_t, \boldsymbol{X}_{n+1}) \in \mathbb{R}^{n+1}$, $\boldsymbol{C}_n = k_{\boldsymbol{\theta}}(\boldsymbol{X}_n, \boldsymbol{X}_n) \in \mathbb{R}^{n \times n}$, $\boldsymbol{C}_{n+1} = k_{\boldsymbol{\theta}}(\boldsymbol{X}_{n+1}, \boldsymbol{X}_{n+1}) \in \mathbb{R}^{(n+1)\times(n+1)}$ with $\boldsymbol{X}_{n+1} = [\boldsymbol{X}_n^\intercal \quad \boldsymbol{x}] \in \mathbb{R}^{n+1}$. The covariance matrix $\boldsymbol{C}_{n+1}$ can be expressed as,

$$\boldsymbol{C}_{n+1} = \begin{bmatrix} \boldsymbol{C}_n & \boldsymbol{k}_{n+1} \\ \boldsymbol{k}_{n+1}^\intercal & k_{n+1,n+1} \end{bmatrix},$$

where $\boldsymbol{k}_{n+1} = k_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{X}_n) \in \mathbb{R}^n$ and $k_{n+1,n+1} = k_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}) \in \mathbb{R}$. The partitioned inverse identity is given as,

$$\boldsymbol{C}_{n+1}^{-1} = \begin{bmatrix} \left(\boldsymbol{C}_n^{-1} + \boldsymbol{o}_n \boldsymbol{o}_n^\intercal v_n\right) & \boldsymbol{o}_n \\ \boldsymbol{o}_n^\intercal & v_n^{-1} \end{bmatrix}, \tag{5}$$

where $\boldsymbol{o}_n = -v_n^{-1} \boldsymbol{C}_n^{-1} \boldsymbol{k}_{n+1} \in \mathbb{R}^n$ and $v_n = k_{n+1,n+1} - \boldsymbol{k}_{n+1}^\intercal \boldsymbol{C}_n^{-1} \boldsymbol{k}_{n+1} \in \mathbb{R}$.

Proposition 1: [3] The ALC criterion takes the form of,

$$J(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x}, \boldsymbol{x}_t) = -\frac{1}{N_{\text{ref}}} \sum_{t=1}^{N_{\text{ref}}} \frac{\left(\boldsymbol{k}_{n,t}^\intercal \boldsymbol{C}_n^{-1} \boldsymbol{k}_{n+1} - k_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}_t)\right)^2}{k_{n+1,n+1} - \boldsymbol{k}_{n+1}^\intercal \boldsymbol{C}_n^{-1} \boldsymbol{k}_{n+1}}. \tag{6}$$

Proof. Direct use of the partitioned inverse identity (5) to (4). A step-by-step proof is provided in [19, Appendix A.1]. □

A critical component of the ALC criterion (6) is the selection of the reference set. The traditional Latin hypercube sampling (LHS) is a space-filling sampling method that shares the properties of a random uniform distribution [20], [21]. Although there are multiple modern space-filling sampling methods, LHS remains competitive [9], [22]. A Latin hypercube is described by a matrix $\boldsymbol{L} \in \mathbb{R}^{q \times r}$, where each column $\text{col}_j\{\boldsymbol{L}\}$, $j = 1, \dots, r$ consist of q-equally levels of a permutation. The entries of the LHS matrix in the input space $[0,1]^D \subset \mathbb{R}^D$ are given by $[\boldsymbol{X}_{\text{LHS}}]_{ij} = ([\boldsymbol{L}]_{ij} + (q-1)/2 + [\boldsymbol{u}]_{ij})/q$, where $[\boldsymbol{u}]_{ij} \sim \mathcal{U}_{[0,1]}$. In our case, the reference set is designed as a LHS, $\boldsymbol{X}_{\text{ref}} = \boldsymbol{X}_{\text{LHS}}$.

To optimize the ALC criterion (6) we employ naive gradient descent,

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \gamma \left. \frac{\partial J(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x}, \boldsymbol{x}_t)}{\partial \boldsymbol{x}} \right|_{\boldsymbol{x} = \boldsymbol{x}^{(k)}}, \tag{7}$$

where $\partial J(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x}, \boldsymbol{x}_t)/\partial \boldsymbol{x} \in \mathbb{R}^D$ is the unknown gradient and k the iteration index. There are more sophisticated optimization algorithms that can be used, but our scope here is to illustrate the efficiency of the closed-form solution in simple terms. However, the proposed method can be used by any first-order method (e.g., ADMM [23], [24]).

A common approach for gradient approximation is to use the finite difference method (FDM) [25, Ch. 7]. To approximate the gradient at an input $\boldsymbol{x}$, the FDM evaluates the ALC criterion at $2D+1$ inputs on a collocated grid,

$$\frac{\partial J(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x}, \boldsymbol{x}_t)}{\partial [\boldsymbol{x}]_d} \approx \frac{J(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x} + \boldsymbol{s}_d, \boldsymbol{x}_t) - J(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x} - \boldsymbol{s}_d, \boldsymbol{x}_t)}{2s}$$

where $\boldsymbol{s}_d = [\boldsymbol{0}_{1:d-1}^\intercal \quad s \quad \boldsymbol{0}_{d+1:D}^\intercal]^\intercal \in \mathbb{R}^D$ is a vector of zeros except of the d-th entry that contains the spacing parameter s. For low-dimensional input spaces the approximation can be efficient, provided that the spacing parameter s is properly selected. However, for high-dimensional input spaces, FDM becomes inexact and computationally demanding with $2D+1$ evaluations. Thus, as the input dimension D increases, not only the optimization becomes inaccurate, but also the algorithm requires significant computations. To this end, our goal is to derive a closed-form solution of the ALC criterion (6).

Problem 1: Find the closed-form expression of the gradient for the ALC criterion $\partial J(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x}, \boldsymbol{x}_t)/\partial \boldsymbol{x}$ to perform closed-form optimization.

III. Closed-Form Active Learning

In this section, we discuss the derivation of the closed-form gradient for the optimization with the ALC criterion. In addition, we present the proposed algorithm that uses the closed-form solution for the active learning optimization. The algorithm makes use of LHS to determine the reference set and naive gradient descent for the optimization.

A. Active Learning Optimization

We first consider the case of deriving the closed-form gradient for any covariance function and then we calculate the analytical solution for the SSE covariance function (1) which is our main contribution. The following Lemma provides the general form of the gradient regardless the selected covariance function.

Lemma 1: The gradient of the ALC criterion $J(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x}, \boldsymbol{x}_t)$ (6) for all covariance functions yields,

$$\frac{\partial J(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x}, \boldsymbol{x}_t)}{\partial \boldsymbol{x}} = \frac{\nabla_{\boldsymbol{x}} h(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x}) \sum_{t=1}^{N_{\text{ref}}} g_t(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x}, \boldsymbol{x}_t)}{N_{\text{ref}} h(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x})^2} - \frac{h(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x}) \sum_{t=1}^{N_{\text{ref}}} \nabla_{\boldsymbol{x}} g_t(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x}, \boldsymbol{x}_t)}{N_{\text{ref}} h(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x})^2}, \tag{8}$$

where the elements h, g, $\nabla_{\boldsymbol{x}} h$, $\nabla_{\boldsymbol{x}} g_t$ are given by,

$$h(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x}) = k_{n+1,n+1} - \boldsymbol{k}_{n+1}^\intercal \boldsymbol{C}_n^{-1} \boldsymbol{k}_{n+1}, \tag{9a}$$

$$\frac{\partial h(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x})}{\partial \boldsymbol{x}} = -2 \left(\frac{\partial}{\partial \boldsymbol{x}} k_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{X}_n)\right)^\intercal \boldsymbol{C}_n^{-1} \boldsymbol{k}_{n+1}, \tag{9b}$$

$$g_t(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x}, \boldsymbol{x}_t) = \left(\boldsymbol{k}_{n,t}^\intercal \boldsymbol{C}_n^{-1} \boldsymbol{k}_{n+1} - k_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}_t)\right)^2, \tag{9c}$$

$$\frac{\partial g_t(\hat{\boldsymbol{\theta}}_n; \boldsymbol{x}, \boldsymbol{x}_t)}{\partial \boldsymbol{x}} = 2\left(\boldsymbol{k}_{n,t}^\intercal \boldsymbol{C}_n^{-1} \boldsymbol{k}_{n+1} - k_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}_t)\right) \times \left(\frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{k}_{n,t}^\intercal \boldsymbol{C}_n^{-1} \boldsymbol{k}_{n+1} - \frac{\partial}{\partial \boldsymbol{x}} k_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{x}_t)\right). \tag{9d}$$

Proof. The objective function takes the form of,

$$J(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x},\boldsymbol{x}_t) = -\frac{1}{N_{\text{ref}}}\sum_{t=1}^{N_{\text{ref}}}\frac{\left(\boldsymbol{k}_{n,t}^{\intercal}\boldsymbol{C}_n^{-1}\boldsymbol{k}_{n+1} - k_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{x}_t)\right)^2}{k_{n+1,n+1} - \boldsymbol{k}_{n+1}^{\intercal}\boldsymbol{C}_n^{-1}\boldsymbol{k}_{n+1}}$$
$$= \frac{\sum_{t=1}^{N_{\text{ref}}}\left(\boldsymbol{k}_{n,t}^{\intercal}\boldsymbol{C}_n^{-1}\boldsymbol{k}_{n+1} - k_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{x}_t)\right)^2}{N_{\text{ref}}\left(k_{n+1,n+1} - \boldsymbol{k}_{n+1}^{\intercal}\boldsymbol{C}_n^{-1}\boldsymbol{k}_{n+1}\right)}$$
$$= -\frac{\sum_{t=1}^{N_{\text{ref}}} g_t(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x},\boldsymbol{x}_t)}{N_{\text{ref}}h(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x})}. \tag{10}$$

Next, we apply the quotient rule to obtain (8). Finally, (9a) and (9c) follow simple gradient rules to result in (9b), (9d) respectively. ■

After obtaining the general form of the gradient we derive the closed-form solution of the ALC gradient, by using the SSE covariance function in the following Proposition.

Proposition 2: The closed-form gradient of the ALC criterion $J(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x},\boldsymbol{x}_t)$ (6) for the SSE covariance function (1) to perform exact optimization using (7) yields,

$$\frac{\partial J(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x},\boldsymbol{x}_t)}{\partial \boldsymbol{x}} = \frac{\nabla_{\boldsymbol{x}}h(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x})\sum_{t=1}^{N_{\text{ref}}} g_t(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x},\boldsymbol{x}_t)}{N_{\text{ref}}h(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x})^2} - \frac{h(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x})\sum_{t=1}^{N_{\text{ref}}}\nabla_{\boldsymbol{x}}g_t(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x},\boldsymbol{x}_t)}{N_{\text{ref}}h(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x})^2}, \tag{11}$$

where the elements h, g, $\nabla_{\boldsymbol{x}}h$, $\nabla_{\boldsymbol{x}}g_t$ result in,

$$h(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x}) = k_{n+1,n+1} - \boldsymbol{k}_{n+1}^{\intercal}\boldsymbol{C}_n^{-1}\boldsymbol{k}_{n+1}, \tag{12a}$$
$$\frac{\partial h(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x})}{\partial \boldsymbol{x}} = 4\left((\mathbf{1}_D\boldsymbol{k}_{n+1}^{\intercal})\odot(\Lambda^{-1}\Delta\boldsymbol{X}^{\intercal})\right)\boldsymbol{C}_n^{-1}\boldsymbol{k}_{n+1}, \tag{12b}$$
$$g_t(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x},\boldsymbol{x}_t) = \left(\boldsymbol{k}_{n,t}^{\intercal}\boldsymbol{C}_n^{-1}\boldsymbol{k}_{n+1} - k_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{x}_t)\right)^2, \tag{12c}$$
$$\frac{\partial g_t(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x},\boldsymbol{x}_t)}{\partial \boldsymbol{x}} = -4\left(\boldsymbol{k}_{n,t}^{\intercal}\boldsymbol{C}_n^{-1}\boldsymbol{k}_{n+1} - k_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{x}_t)\right)\Lambda^{-1}\times \Big(\Delta\boldsymbol{X}^{\intercal}\left((\boldsymbol{k}_{n,t}^{\intercal}\boldsymbol{C}_n^{-1})^{\intercal}\odot\boldsymbol{k}_{n+1}\right) - k_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{x}_t)(\boldsymbol{x}-\boldsymbol{x}_t)\Big). \tag{12d}$$

Proof. In addition to Lemma 1, we need to compute two gradients $\nabla_{\boldsymbol{x}}h(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x})$, $\nabla_{\boldsymbol{x}}g_t(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x},\boldsymbol{x}_t)$ with respect to (1). Thus, $\nabla_{\boldsymbol{x}}g_t(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x},\boldsymbol{x}_t)$ from (9d) yields,

$$\frac{\partial g_t(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x},\boldsymbol{x}_t)}{\partial \boldsymbol{x}} = 2\left(\boldsymbol{k}_{n,t}^{\intercal}\boldsymbol{C}_n^{-1}\boldsymbol{k}_{n+1} - k_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{x}_t)\right)\times\left(\frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{k}_{n,t}^{\intercal}\boldsymbol{C}_n^{-1}\boldsymbol{k}_{n+1} - \frac{\partial}{\partial \boldsymbol{x}}k_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{x}_t)\right)$$
$$= 2\left(\boldsymbol{k}_{n,t}^{\intercal}\boldsymbol{C}_n^{-1}\boldsymbol{k}_{n+1} - k_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{x}_t)\right)(\mathcal{T}_1 - \mathcal{T}_2). \tag{13}$$

Note that the SSE covariance function (1) can also be expressed in a vector form as,

$$k_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{x}_t) = \sigma_f^2\exp\left\{-\sum_{d=1}^{D}\frac{(x_d - x_{t,d})^2}{l_d^2}\right\}$$
$$= \sigma_f^2\exp\left\{-(\boldsymbol{x}-\boldsymbol{x}_t)^{\intercal}\Lambda^{-1}(\boldsymbol{x}-\boldsymbol{x}_t)\right\}, \tag{14}$$

where $\Lambda = \operatorname{diag}(l_1^2, l_2^2, \ldots, l_D^2)$.

We start by computing the second unknown term $\mathcal{T}_2$ because we shall use the result for the computation of the first unknown term $\mathcal{T}_1$. Hence, $\mathcal{T}_2$ using the vector form of SSE (14) yields,

$$\mathcal{T}_2 = \frac{\partial}{\partial \boldsymbol{x}}k_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{x}_t)$$
$$= \frac{\partial}{\partial \boldsymbol{x}}\sigma_f^2\exp\left\{-\sum_{d=1}^{D}\frac{(x_d - x_{t,d})^2}{l_d^2}\right\}$$
$$= \frac{\partial}{\partial \boldsymbol{x}}\sigma_f^2\exp\left\{-(\boldsymbol{x}-\boldsymbol{x}_t)^{\intercal}\Lambda^{-1}(\boldsymbol{x}-\boldsymbol{x}_t)\right\}$$
$$= \sigma_f^2\exp\left\{-(\boldsymbol{x}-\boldsymbol{x}_t)^{\intercal}\Lambda^{-1}(\boldsymbol{x}-\boldsymbol{x}_t)\right\}\times \frac{\partial}{\partial \boldsymbol{x}}(-(\boldsymbol{x}-\boldsymbol{x}_t)^{\intercal}\Lambda^{-1}(\boldsymbol{x}-\boldsymbol{x}_t))$$
$$= -2k_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{x}_t)\Lambda^{-1}(\boldsymbol{x}-\boldsymbol{x}_t). \tag{15}$$

The first unknown term $\mathcal{T}_1$ takes the form of,

$$\mathcal{T}_1 = \frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{k}_{n,t}^{\intercal}\boldsymbol{C}_n^{-1}\boldsymbol{k}_{n+1}$$
$$= \frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{\alpha}_t^{\intercal}k_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{X}_n)$$
$$= \frac{\partial}{\partial \boldsymbol{x}}\sum_{i=1}^{n}[\boldsymbol{\alpha}_t]_i k_{\boldsymbol{\theta}}(\boldsymbol{x},[\boldsymbol{X}_n]_i)$$
$$= \sum_{i=1}^{n}[\boldsymbol{\alpha}_t]_i\frac{\partial}{\partial \boldsymbol{x}}k_{\boldsymbol{\theta}}(\boldsymbol{x},[\boldsymbol{X}_n]_i), \tag{16}$$

where $\boldsymbol{\alpha}_t^{\intercal} = \boldsymbol{k}_{n,t}^{\intercal}\boldsymbol{C}_n^{-1} \in \mathbb{R}^n$. Next, we employ the result of the second unknown term $\mathcal{T}_2$ (15) as it appears in the resulting partial derivative of the first unknown term $\mathcal{T}_1$ (16) to obtain,

$$\mathcal{T}_1 = \sum_{i=1}^{n}[\boldsymbol{\alpha}_t]_i\frac{\partial}{\partial \boldsymbol{x}}k_{\boldsymbol{\theta}}(\boldsymbol{x},[\boldsymbol{X}_n]_i)$$
$$= \sum_{i=1}^{n}[\boldsymbol{\alpha}_t]_i(-2k_{\boldsymbol{\theta}}(\boldsymbol{x},[\boldsymbol{X}_n]_i)\Lambda^{-1}(\boldsymbol{x}-[\boldsymbol{X}_n]_i))$$
$$= -2\sum_{i=1}^{n}[\boldsymbol{\alpha}_t]_i k_{\boldsymbol{\theta}}(\boldsymbol{x},[\boldsymbol{X}_n]_i)\Lambda^{-1}\text{row}_i\{\Delta\boldsymbol{X}^{\intercal}\}$$
$$= -2\Lambda^{-1}\Delta\boldsymbol{X}^{\intercal}(\boldsymbol{\alpha}_t\odot k_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{X}_n)), \tag{17}$$

where $\Delta\boldsymbol{X} = [\boldsymbol{x}-[\boldsymbol{X}_n]_1,\ldots,\boldsymbol{x}-[\boldsymbol{X}_n]_n]^{\intercal} \in \mathbb{R}^{n\times D}$ and $\odot$ denotes the Hadamard product. Substitute $\mathcal{T}_2$ (15) and $\mathcal{T}_1$ (17) into (13) to obtain (12d).

The gradient $\nabla_{\boldsymbol{x}}h(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x})$ results in,

$$\frac{\partial h(\hat{\boldsymbol{\theta}}_n;\boldsymbol{x})}{\partial \boldsymbol{x}} = \frac{\partial}{\partial \boldsymbol{x}}\sigma_f^2 - 2\left(\frac{\partial}{\partial \boldsymbol{x}}k_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{X}_n)\right)^{\intercal}\boldsymbol{C}_n^{-1}\boldsymbol{k}_{n+1}$$
$$= -2\left(\frac{\partial}{\partial \boldsymbol{x}}k_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{X}_n)\right)^{\intercal}\boldsymbol{C}_n^{-1}\boldsymbol{k}_{n+1}$$
$$= 4[k_{\boldsymbol{\theta}}(\boldsymbol{x},[\boldsymbol{X}_n]_1)\Lambda^{-1}\text{row}_1\{\Delta\boldsymbol{X}^{\intercal}\},\ldots, k_{\boldsymbol{\theta}}(\boldsymbol{x},[\boldsymbol{X}_n]_n)\Lambda^{-1}\text{row}_n\{\Delta\boldsymbol{X}^{\intercal}\}]\boldsymbol{C}_n^{-1}\boldsymbol{k}_{n+1}$$
$$= 4\left((\mathbf{1}_D\boldsymbol{k}_{n+1}^{\intercal})\odot(\Lambda^{-1}\Delta\boldsymbol{X}^{\intercal})\right)\boldsymbol{C}_n^{-1}\boldsymbol{k}_{n+1}, \tag{18}$$

where $\mathbf{1}_D = [1, \dots, 1]^\intercal \in \mathbb{R}^D$. Since (18) is identical to (12d), this concludes the proof. ■

Lemma 1 is the general version of Proposition 2, where gradients (9b), (9d) are computed for the SSE covariance (1) as (12b), (12d).

B. Closed-Form ALC (cALC) Algorithm

The main routine of cALC is provided in Alg. 1. The method starts by computing the space-filling inputs using LHS and then samples the corresponding inputs to obtain the initial dataset $\mathcal{D} = \{\boldsymbol{X}_n, \boldsymbol{y}_n\}$. Using $\mathcal{D}$, a GP surrogate model is fitted to train the hyperparameters $\hat{\boldsymbol{\theta}}_n$. Next, the algorithm proceeds to the active learning part. The first steps of cALC is to design a reference set $\boldsymbol{X}_{\text{ref}}$ and a multi-start location dataset $\boldsymbol{X}_{\text{m-s}}$. Note that for both datasets the `conditionedLHS` routine is used which operates similarly to `LHS`, but conditioned on the inputs of the previous LHS designs. The multi-start location dataset $\boldsymbol{X}_{\text{m-s}}$ is computed to start the gradient descent from multiple locations, as the objective function (6) is non-convex on $\boldsymbol{x}$ and local minima need to be avoided. Next, our contribution is highlighted in brown font. More specifically, the closed-form gradient is computed according to Proposition 2 (Alg. 1-[lines 10–14]). Then, a gradient step is taken with step γ following (7). If the L-D norm of the difference between the new input $\boldsymbol{x}_i^{(k+1)}$ and the previous input $\boldsymbol{x}_i^{(k)}$ is below the η threshold, then the optimization is terminated and the value of the ALC criterion for each multi-start location $J_{n+1,i}$ is computed. Otherwise, the algorithm iterates for a predetermined number of iterations $N_{\text{GD-max}}$. Next, we compare the minima we found from all the multi-start locations $\{J_{n+1,i}\}_{i=1}^{N_{\text{m-s}}}$ to identify the minimum with the lowest objective and we assign the corresponding element to the next input $\boldsymbol{x}_{n+1}$ (Alg. 1-[line 23]). Subsequently, the next input is sampled to obtained the new observation y_{n+1} and form the new dataset. Using the new dataset a new GP surrogate model is fitted to obtain the new trained hyperparameters. The closed-from ALC (Alg. 1-[lines 4–26]) iterates for a predetermined number of iterations $n_{\max}$ to output the hyperparameters of the GP surrogate model $\hat{\boldsymbol{\theta}}_{n_{\max}}$ and the final dataset $\mathcal{D}_{n_{\max}}$.

Algorithm 1 Closed-Form ALC (cALC)

```
Input: $D$, $k_{\boldsymbol{\theta}}$, $N_{\text{ref}}$, $N_{\text{m-s}}$, $N_{\text{GD-max}}$, $n_{\text{init}}$, $n_{\max}$, $\gamma$, $\eta$
Output: $\hat{\boldsymbol{\theta}}_{n_{\max}}$, $\mathcal{D}_{n_{\max}}$
 1: $\boldsymbol{X}_n \leftarrow$ LHS($n_{\text{init}}, D$)                          ▷ Initial Dataset
 2: $\boldsymbol{y}_n \leftarrow$ sample($\boldsymbol{X}_n$)
 3: $\hat{\boldsymbol{\theta}}_n \leftarrow$ GPtraining($\boldsymbol{X}_n, \boldsymbol{y}_n, k_{\boldsymbol{\theta}}$)                  ▷ Fit GP Surrogate
 4: repeat                                                   ▷ Closed-Form ALC
 5:     $\boldsymbol{X}_{\text{ref}} \leftarrow$ conditionedLHS($N_{\text{ref}}, D, \boldsymbol{X}_n$)
 6:     $\boldsymbol{X}_{\text{m-s}} \leftarrow$ conditionedLHS($N_{\text{m-s}}, D, \boldsymbol{X}_n, \boldsymbol{X}_{\text{ref}}$)
 7:     $\{\boldsymbol{x}_i^{(1)}\}_{i=1}^{N_{\text{m-s}}} = \boldsymbol{X}_{\text{m-s}}$
 8:     for $i = 1$ to $N_{\text{m-s}}$ do                ▷ Multiple Starting Locations
 9:         repeat                                      ▷ Compute ALC Gradient
10:             $h; \nabla h \leftarrow$ hGradH($k_{\boldsymbol{\theta}}, \hat{\boldsymbol{\theta}}_n, \boldsymbol{x}_i, \boldsymbol{X}_n$) (12a), (12b)
11:             for $t = 1$ to $N_{\text{ref}}$ do
12:                 $g_t; \nabla g_t \leftarrow$ gGrG($k_{\boldsymbol{\theta}}, \hat{\boldsymbol{\theta}}_n, \boldsymbol{x}_i, \boldsymbol{X}_n, \boldsymbol{X}_{\text{ref}}$) (12c), (12d)
13:             end for
14:             $\nabla J \leftarrow$ gradJ($h, \nabla h, \sum_{t=1}^{N_{\text{ref}}} g_t, \sum_{t=1}^{N_{\text{ref}}} \nabla g_t, N_{\text{ref}}$) (11)
15:             $\boldsymbol{x}_i^{(k+1)} \leftarrow$ gradientDescent($\boldsymbol{x}_i^{(k)}, \nabla J, \gamma$) (7)
16:             if $\|\boldsymbol{x}_i^{(k+1)} - \boldsymbol{x}_i^{(k)}\|_D < \eta$ then
17:                 $J_{n+1,i} \leftarrow$ objJ($\boldsymbol{x}_i^{(k+1)}, \{g_t\}_{t=1}^{N_{\text{ref}}}, h, N_{\text{ref}}$) (10)
18:                 $\boldsymbol{x}_{n+1,i} = \boldsymbol{x}_i^{(k+1)}$
19:                 break
20:             end if
21:         until $N_{\text{GD-max}}$
22:     end for
23:     $\boldsymbol{x}_{n+1} \leftarrow$ minJ($\{\boldsymbol{x}_{n+1,i}\}_{i=1}^{N_{\text{m-s}}}, \{J_{n+1,i}\}_{i=1}^{N_{\text{m-s}}}$)        ▷ Next Input
24:     $y_{n+1} \leftarrow$ sample($\boldsymbol{x}_{n+1}$)
25:     $\boldsymbol{y}_n = \boldsymbol{y}_n \cup y_{n+1}$; $\boldsymbol{X}_n = \boldsymbol{X}_n \cup \boldsymbol{x}_{n+1}$              ▷ New Dataset
26:     $\hat{\boldsymbol{\theta}}_n \leftarrow$ GPtraining($\boldsymbol{X}_n, \boldsymbol{y}_n, k_{\boldsymbol{\theta}}$)
27: until $n_{\max}$
28: return $\hat{\boldsymbol{\theta}}_{n_{\max}}, \mathcal{D}_{n_{\max}}$
```

IV. Numerical Experiments

In this section, we perform numerical experiments to illustrate the efficiency of the proposed method. We use synthetic functions of different dimensions: i) generative GP with $\boldsymbol{\theta} = (l_1 = 1.9, l_2 = 0.6, \sigma_f = 2.1, \sigma_\epsilon = 0.1)^\intercal$ as hyperparameters and $D = 2$; ii) Hartmann function with $D = 3$; and iii) Hartmann function with $D = 6$. For (i) the initial dataset has $n_{\text{init}} = 45$ size and $N_{\text{m-s}} = 3$ multi-start locations; for (ii) the initial dataset has $n_{\text{init}} = 60$ size and $N_{\text{m-s}} = 5$ multi-start locations; and for (iii) the initial dataset has $n_{\text{init}} = 90$ size and $N_{\text{m-s}} = 6$ multi-start locations. We perform $n_{\max} = 100$ active learning iterations for all test functions. We compare four methods: a) GP with random assignment of inputs that follow a uniform distribution (GP-random); b) GP with LHS for input selection (GP-LHS); c) ALC with finite difference method for gradient approximation (fALC); and d) ALC with closed-form computation of gradient (cALC) which is the proposed method. For methods (a) and (b) the size of the dataset is $n = 145$ on test function (i), $n = 160$ on test function (ii), and $n = 190$ on test function (iii). For methods (c) and (d) the size of the datasets are identical, but are progressively growing (i.e., $n = n_{\text{init}} + n_{\max}$), and the size of the reference set is $N_{\text{ref}} = 30$.

All inputs are normalized to unit dimensions $[0, 1]^D$. We add iid noise that corresponds to 10% of the standard deviation of the ground truth output values, i.e., $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$ with $\sigma_\epsilon = 0.1\,\text{std}(f(\boldsymbol{X}))$. We set the gradient descent step to $\gamma = 5$ and the convergence tolerance to $\eta = 10^{-4}$. For every test function, we perform 100 Monte Carlo replications to remove the effect of random assignment of data. We assess the proposed method with three metrics, the prediction RMSE, the average iterations for one ALC loop, and the average required time for one ALC loop. The prediction RMSE follows $\text{RMSE} = [\sum_{s=1}^{n_S} (\mu(\boldsymbol{x}_{s,*}) - y(\boldsymbol{x}_{s,*}))^2]^{1/2}$,with n_S the number of test points. We conduct all numerical experiments in MATLAB using GPML [26] on an Intel Core i9-10900K CPU at 3.70 GHz with 64.0 GB memory RAM.

Best prediction accuracy (2D): In Fig. 2, we present the evaluation of the four methods for the generative GP function of $D = 2$. Since GP-random is a batch method, it is not evaluated with respect to iterations and time. The proposed method cALC outperforms all other methods on prediction RMSE, revealing that the closed-form gradient leads to more accurate predictions. The fALC is competitive,

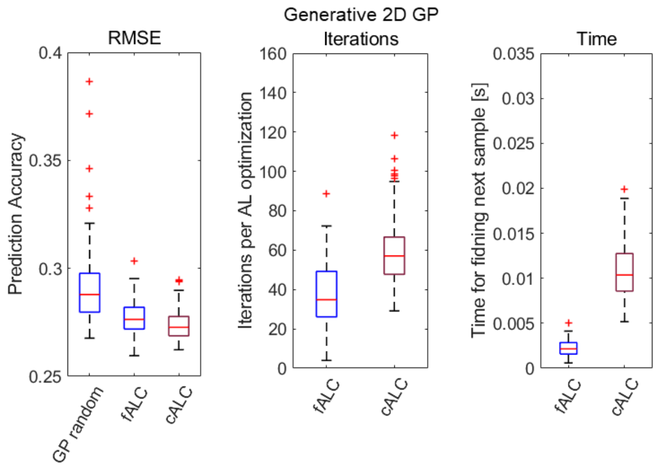

Fig. 2. Evaluation of generative GP function with input dimension $D = 2$.

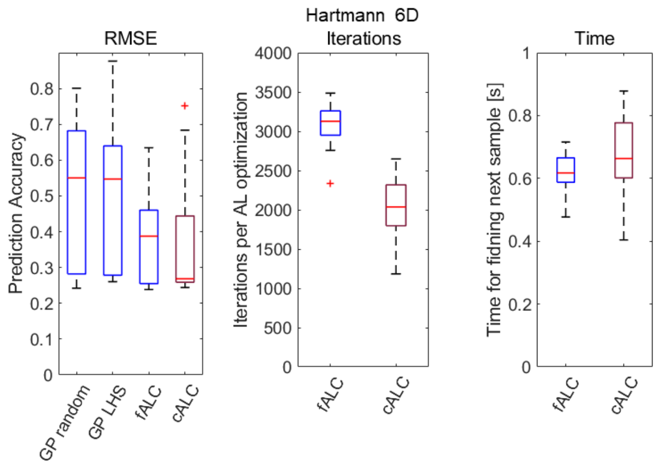

Fig. 4. Evaluation of Hartmann function with input dimension $D = 6$.

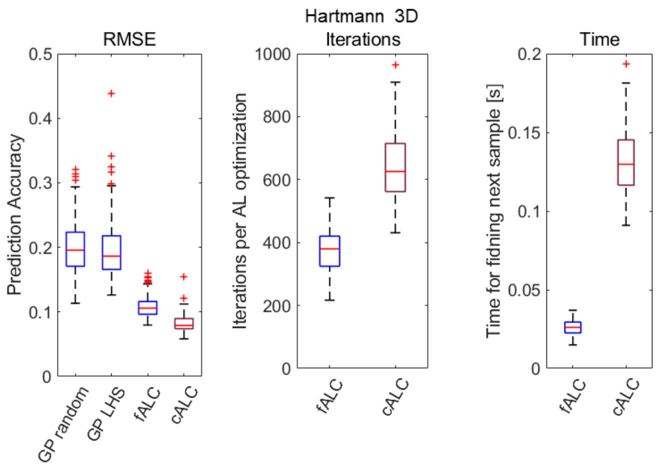

Fig. 3. Evaluation of Hartmann function with input dimension $D = 3$.

while GP-random produces higher RMSE values. To this end, sequential active learning methods (fALC and cALC) are efficient when compared to single shot GP methods (GP-random). Although cALC is more accurate than fALC, the latter requires less iterations and lower time for finding the next sample location. We will show in the next comparisons that this difference in iteration number and time-per-loop diminishes as we increase the input dimension.

Best prediction accuracy & improved computations (3D): In Fig. 3, we demonstrate the efficiency of the four methods for the Hartmann test function of $D = 3$. We observe that the proposed method (cALC) outperforms all other methods in terms of prediction accuracy (RMSE). In addition, the iterations of fALC are still less than that of cALC, but their difference is decreased. However, time-per-loop still remains more efficient with the fALC method with similar difference to the previous case of $D = 2$. Next, we show that the proposed cALC method improves the time and iteration requirements as we increase the input space dimension.

Best prediction accuracy with similar computations (6D): In Fig. 4, we show the four methods for the Hartmann test function of $D = 6$. Similarly to test functions with $D = 2, 3$ input dimensions, the proposed cALC outperforms all other methods in prediction accuracy (RMSE), but this time the median value difference is even higher between cALC and fALC. This reveals that the closed-form gradient (cALC) improves the prediction accuracy of active learning in higher dimensional input spaces when compared to approximated gradient methods with finite differences (fALC). In contrast to 2D and 3D input dimensions, the iteration number and time-per-loop perform similarly for fALC and cALC in the 6D input space. This means that with high input dimensions, fALC and cALC require similar time-per-loop and iterations. As a result, the proposed method (cALC) not only improves the prediction accuracy, but also requires similar iteration number and time-per-loop to fALC for 6D input spaces.

V. Conclusion

We propose an active learning method with Gaussian process surrogates using closed-form gradient (cALC). The closed-form gradient derivation is emphasized with a detailed proof. The proposed cALC produces the most accurate predictions among all compared methods for all input dimensions. As the input dimension increases to 6D, cALC not only produces more accurate results, but also requires similar computations. Ongoing work is emphasizing on new active learning methods that balances exploration and exploitation.

References

[1] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*, 2nd ed. Cambridge, MA, USA: MIT Press, 2006.

[2] H. Liu, Y.-S. Ong, and J. Cai, “A survey of adaptive sampling for global metamodeling in support of simulation-based complex engineering design,” *Structural and Multidisciplinary Optimization*, vol. 57, no. 1, pp. 393–416, 2018.

[3] S. Seo, M. Wallat, T. Graepel, and K. Obermayer, “Gaussian process regression: Active data selection and test point rejection,” in *IEEE-INNS-ENNS International Joint Conference on Neural Networks*, vol. 3, 2000, pp. 241–246.

[4] T. J. Koo and S. Sastry, “Output tracking control design of a helicopter model based on approximate linearization,” in *IEEE Conference on Decision and Control*, vol. 4, 1998, pp. 3635–3640.

[5] J. Petrich and D. J. Stilwell, “Robust control for an autonomous underwater vehicle that suppresses pitch and yaw coupling,” *Ocean Engineering*, vol. 38, no. 1, pp. 197–204, 2011.

[6] J. Kocijan, R. Murray-Smith, C. E. Rasmussen, and A. Girard, “Gaussian process model based predictive control,” in *American control conference*, vol. 3, 2004, pp. 2214–2219.

[7] H. Liu, Y.-S. Ong, X. Shen, and J. Cai, “When Gaussian process meets big data: A review of scalable GPs,” *IEEE Trans. on Neural Networks and Learn. Systems*, vol. 31, no. 11, pp. 4405–4423, 2020.

[8] G. P. Kontoudis and D. J. Stilwell, “Decentralized nested Gaussian processes for multi-robot systems,” in *IEEE International Conference on Robotics and Automation*, 2021, pp. 8881–8887.

[9] R. B. Gramacy, *Surrogates: Gaussian Process Modeling, Design and Optimization for the Applied Sciences*. Chapman Hall/CRC, 2020.

[10] J. Das, F. Py, J. B. Harvey, J. P. Ryan, A. Gellene, R. Graham, D. A. Caron, K. Rajan, and G. S. Sukhatme, “Data-driven robotic sampling for marine ecosystem monitoring,” *The International Journal of Robotics Research*, vol. 34, no. 12, pp. 1435–1452, 2015.

[11] C. Lee, K. Wang, J. Wu, W. Cai, and X. Yue, “Partitioned active learning for heterogeneous systems,” *Journal of Computing and Information Science in Engineering*, vol. 23, no. 4, p. 041009, 2023.

[12] W. Luo and K. Sycara, “Adaptive sampling and online learning in multi-robot sensor coverage with mixture of Gaussian processes,” in *Inter. Conf. on Robotics and Automation*, 2018, pp. 6359–6364.

[13] R. B. Gramacy and D. W. Apley, “Local Gaussian process approximation for large computer experiments,” *Journal of Computational and Graphical Statistics*, vol. 24, no. 2, pp. 561–578, 2015.

[14] H. Liu, S. Xu, Y. Ma, X. Chen, and X. Wang, “An adaptive Bayesian sequential sampling approach for global metamodeling,” *Journal of Mechanical Design*, vol. 138, no. 1, p. 011404, 2016.

[15] D. R. Jones, M. Schonlau, and W. J. Welch, “Efficient global optimization of expensive black-box functions,” *Journal of Global optimization*, vol. 13, no. 4, p. 455, 1998.

[16] Y. T. Tan, A. Kunapareddy, and M. Kobilarov, “Gaussian process adaptive sampling using the cross-entropy method for environmental sensing and monitoring,” in *International Conference on Robotics and Automation*, 2018, pp. 6220–6227.

[17] N. Srinivas, A. Krause, S. Kakade, and M. Seeger, “Gaussian process optimization in the bandit setting: No regret and experimental design,” in *Intern. Conference on Machine Learning*, 2010, pp. 1015–1022.

[18] C. N. Mavridis, G. P. Kontoudis, and J. S. Baras, “Sparse Gaussian process regression using progressively growing learning representations,” in *IEEE Conference on Decision and Control*, December 2022.

[19] R. B. Gramacy and H. K. Lee, “Adaptive design and analysis of supercomputer experiments,” *Technometrics*, vol. 51, no. 2, pp. 130–145, 2009.

[20] M. McKay, R. Beckman, and W. Conover, “Comparison the three methods for selecting values of input variable in the analysis of output from a computer code,” *Technometrics*, vol. 21, no. 2, 1979.

[21] M. Stein, “Large sample properties of simulations using latin hypercube sampling,” *Technometrics*, vol. 29, no. 2, pp. 143–151, 1987.

[22] F. A. Viana, “Things you wanted to know about the latin hypercube design and were afraid to ask,” in *World Congress on Structural and Multidisciplinary Optimization*, vol. 19, no. 24.05, 2013.

[23] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*, 2011, vol. 3, no. 1.

[24] G. P. Kontoudis and D. J. Stilwell, “Fully decentralized, scalable Gaussian processes for multi-agent federated learning,” *arXiv preprint arXiv: 2203.02865*, 2022.

[25] A. Sobester, A. Forrester, and A. Keane, *Engineering design via surrogate modelling: A practical guide*. John Wiley & Sons, 2008.

[26] C. E. Rasmussen and H. Nickisch, “Gaussian processes for machine learning (GPML) toolbox,” *Journal of Machine Learning Research*, vol. 11, pp. 3011–3015, 2010.