Received 30 December 2020; | Revised 12 July 2021; | Accepted 5 September 2021

DOI: xxx/xxxx

RESEARCH

Adaptive Batching for Gaussian Process Surrogates with Application in Noisy Level Set Estimation

Xiong Lyu | Michael Ludkovski*

[1]Department of Statistics and Applied Probability, University of California at Santa Barbara, Santa Barbara, CA 93106-3110, USA Email: lyu@pstat.ucsb.edu

Correspondence
Michael Ludkovski, Department of Statistics and Applied Probability, University of California at Santa Barbara, Santa Barbara, CA 93106-3110, USA Email: ludkovski@pstat.ucsb.edu

We develop adaptive replicated designs for Gaussian process metamodels of stochastic experiments. Adaptive batching is a natural extension of sequential design heuristics with the benefit of replication growing as response features are learned, inputs concentrate, and the metamodeling overhead rises. Motivated by the problem of learning the level set of the mean simulator response, we develop five novel schemes: Multi-Level Batching (MLB), Ratchet Batching (RB), Adaptive Batched Stepwise Uncertainty Reduction (ABSUR), Adaptive Design with Stepwise Allocation (ADSA) and Deterministic Design with Stepwise Allocation (DDSA). Our algorithms simultaneously (MLB, RB and ABSUR) or sequentially (ADSA and DDSA) determine the sequential design inputs and the respective number of replicates. Illustrations using synthetic examples and an application in quantitative finance (Bermudan option pricing via Regression Monte Carlo) show that adaptive batching brings significant computational speed-ups with minimal loss of modeling fidelity.

KEYWORDS:
GP surrogates, level set estimation, stochastic simulation, design of experiments, stepwise uncertainty reduction

1 | INTRODUCTION

Metamodels offer a cheap statistical representation of complex and/or expensive stochastic simulators that arise in applications ranging from engineering to environmental science and finance [32]. Gaussian process (GP) frameworks have emerged as the leading family of metamodels thanks to their flexibility, analytical tractability and superior empirical performance. However, for GP metamodels to be fast, it is imperative to keep the respective design size $|\mathcal{A}|$ manageable. In particular, unless the simulator is truly expensive or the input domain is vast, the typical recommendation is to restrict to hundreds of inputs, $|\mathcal{A}| \ll 10^3$. This creates a major tension as frequently the stochastic simulator has low signal-to-noise ratio or a complex noise structure. A prototypical example is where the simulator $Y(x) = F(X_{[0,\Delta t]})|_{X_0=x}$ involves functionals of a continuous-time Markov chain or stochastic differential equation solution (X_t), whereby the stochasticity tends to dominate the trend/drift term for short Δt, and moreover simulation noise is non-Gaussian and state-dependent (heteroskedastic).

A natural solution is to employ *batching*, known in the stochastic simulation community as nested Monte Carlo. Re-using the same input to generate multiple outputs allows for a Law of Large Numbers (LLN) averaging which can be analytically combined with the GP predictive equations to keep the computational complexity as a function of k (number of unique inputs) rather than of the capital-N (number of simulator calls). The seminal technique of *stochastic kriging* [1] shows that these computational savings are exact assuming the GP hyperparameters, in particular the noise variance τ^2, are known. Such batching becomes

critical in the use of GP models in our motivating application of solving optimal stopping problems via Regression Monte Carlo, where tens of thousands of simulations are called for.

In the classical setup, the metamodeling objective is to learn the mean response over the entire domain [21, 22, 11], whereby, modulo heteroskedastic noise, one expects to utilize the same batching level across all inputs, i.e. splitting the total budget $N = k \times r$ into k batches of r replicates at locations $\bar{x}_1, \dots, \bar{x}_k$. See [1] for a discussion of how to pick k for a given budget N, as well as some proposals for handling non-constant $\tau^2(x)$. We are interested in more targeted objectives, where the picture is much less clear. As two canonical examples we recall Bayesian Optimization (finding the maximum mean response) and Level Set Estimation (determining the input sub-domain where the mean response exceeds a given threshold). In both settings GP metamodels have been shown to especially shine, not least because they organically match the sequential adaptive designs typically utilized; the respective Expected Improvement schemes form a major feature of the GP ecosystem. Since these objectives imply preferentially sampling a small portion of the input space—the neighborhood of the maximum, or the neighborhood of the desired contour—the exploration-exploitation paradigm leads to increasingly concentrated designs. Such concentration suggests to adaptively determine the amount of batching. Intuitively, replication should be low for more exploratory sites and should rise in the neighborhood of interest, where we replicate to achieve computational savings. Indeed, the intrinsic cost of replication is linked to the variability of the response at the respective inputs, which will be minimal if the inputs are very close together. From a different perspective, replication trades off costly, precise outputs (large r) vis-a-vis cheap outputs with low signal-to-noise ratio (low r).

The above motivates *adaptively batched* designs, where r is input-dependent. While this idea was investigated for Bayesian Optimization [20, 30] and for Integrated Mean Squared Error (IMSE) minimization [1, 8], neither of these fully reveal the underlying tension between exploration (replicate less, larger metamodel overhead) and exploitation (replicate more, generate computational savings). In this article we propose several schemes that explicitly focus on this issue. To evaluate them we concentrate on the problem of level set estimation where the contour is adaptively learned through the sequential design but retains a spatial structure (unlike Bayesian Optimization where convergence to the single input yielding the global maximum is desired). Consequently, we expect a complex interaction between the selection of inputs and the respective replication amounts. In this context, our main contribution is to extend the paradigm of Expected Improvement to include sequential selection of both the input locations x_n and the replication counts r_n. We benchmark the proposed algorithms and show that they provide significant savings compared to the naive fixed-batching approach. In particular, we are able to obtain schemes that reduce $N \simeq 10^5$ simulations to efficient replicated designs of just a few hundred unique inputs.

Beyond benchmarking the developed algorithms on several synthetic examples, we also implement and extend them to heteroskedastic modeling for the motivating application of valuation of Bermudan options. In the latter context, the Regression Monte Carlo (RMC) paradigm is used to provide a simulation-based algorithm that hinges on recursive estimation of certain level sets that correspond to the so-called stopping boundaries. Building upon the successful use of GP surrogates for RMC [24, 27], we demonstrate that adaptive batching significantly speeds up this approach, making it more scalable and efficient. In particular while in [24] sequential design was typically too slow to be useful, adaptively batched models beat basic implementation on both speed and memory requirements. We note that there are other important applications of level set estimation, from quantifying the reliability of a system or its failure probability [5], to ranking pay-offs from several available actions in dynamic programming [16].

The rest of the paper is organized as follows. Section 2 formalizes the GP model and the contour-learning objective. Section 3 develops heuristics for sequential designs that jointly optimize over the new input and replication level. Section 4 takes a different tack and explores dynamic replication through allocating new simulations to existing inputs. Section 5 benchmarks the proposed schemes on three synthetic case studies and Section 6 on two more examples from Bermudan option pricing. Section 7 concludes.

2 | STATISTICAL MODEL

Consider a latent $f : D \to \mathbb{R}$ which is a continuous function over a d-dimensional input space $D \subseteq \mathbb{R}^d$. We wish to identify the contour ∂S, where, without loss of generality, S is the zero level set

$$S = \{x \in D : f(x) \geq 0\}. \tag{1}$$

Thus, our metamodeling objective is equivalent to learning the sign of $f(x)$ for any $x \in D$. For any $x_i \in D$, we have access to a simulator $Y(x_i)$ that generates noisy outputs of $f(x_i)$:

$$Y(x_i) = f(x_i) + \epsilon_i, \tag{2}$$

where ϵ_i's are realizations of independent, mean zero random variables with variance τ^2. To describe replication, we distinguish between simulation inputs x_i, some of which may be identical, and unique inputs $\bar{x}_i$. Let $y_i^{(j)}$ be the j^{th} output of $r_i \geq 1$ replicates observed at $\bar{x}_i$ and let $\bar{y}_i := \frac{1}{r_i}\sum_{j=1}^{r_i} y_i^{(j)}$ be the average over these replicates. This notation follows the "unique-n/full-N" formulation in [8].

The inference of ∂S proceeds by building a metamodel $\hat{f}$, which induces $\hat{S} = \{x \in D : \hat{f}(x) > 0\}$, and evaluating its *error rate* $\mathcal{ER}$, i.e. the integral over the symmetric difference between $\hat{S}$ and true S weighted by a given measure $\mu(\cdot)$:

$$\mathcal{ER}(S, \hat{S}) = \int_{x \in D} \mathbb{I}(\text{sign}\, \hat{f}(x) \neq \text{sign}\, f(x))\mu(dx) = \mu(S \Delta \hat{S}), \tag{3}$$

where $S\Delta\hat{S} := (S \cap \hat{S}^c) \bigcup (S^c \cap \hat{S})$. S can also be defined using Vorob'ev expectation [12] or conservative probability estimate [9, 3].

Reconstructing S via a metamodel can be divided into two aspects: the construction of the response model $x \mapsto Y(x)$, and the development of the design of experiments (DoE) for efficiently selecting the inputs $\bar{x}_1, \bar{x}_2, \ldots$. To account for the second aspect, we use n to denote the rounds of sequential DoE, k_n to denote the number of unique inputs $\bar{x}$'s sampled by step n and $N_n = \sum_{i=1}^{k_n} r_i^{(n)}$ the respective number of simulator calls made. The superscript on r_i allows the replicate counts to evolve over n as well, see Section 4. The metamodel training set by step n consists of $\mathcal{A}_n = \left\{(\bar{x}_i, r_i^{(n)}, \bar{y}_i), 1 \leq i \leq k_n\right\}$.

The Gaussian process paradigm treats f as a random function whose posterior distribution is determined from its prior and the training set(s) $\mathcal{A}_n$. We view $f(\cdot) \sim GP(m(\cdot), K(\cdot,\cdot))$ as a realization of a Gaussian process specified by its mean function $m(x) := \mathbb{E}[f(x)]$ and covariance function $K(x, x') := \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))]$. The noise distribution is $\epsilon \sim \mathcal{N}(0, \tau^2)$; and thus the observation $\bar{y}$ also follows a normal distribution. For simplicity we take $m(x) = 0$. The conditional distribution $f|\mathcal{A}_n$ is another Gaussian process, with posterior mean $\hat{f}^{(n)}(x_*)$ and covariance $v^{(n)}(x_*, x'_*)$ at arbitrary inputs x_*, x'_* given by

$$\hat{f}^{(n)}(x_*) = \mathbf{k}(x_*)[\mathbf{K} + \tau^2\mathbf{R}^{(n)}]^{-1}\bar{\mathbf{y}}_{1:k_n}, \tag{4}$$

$$v^{(n)}(x_*, x'_*) = K(x_*, x'_*) - \mathbf{k}(x_*)[\mathbf{K} + \tau^2\mathbf{R}^{(n)}]^{-1}\mathbf{k}(x'_*)^T, \tag{5}$$

with the $1 \times k_n$ vector $\mathbf{k}(x_*) = K(x_*, \bar{\mathbf{x}}_{1:k_n})$, the $k_n \times 1$ vector $\bar{\mathbf{y}}_{1:k_n} = \{\bar{y}_i, 1 \leq i \leq k_n\}$, the $k_n \times k_n$ matrix $\mathbf{K}$ given by $\mathbf{K}_{ij} = K(\bar{x}_i, \bar{x}_j)$, and the $k_n \times k_n$ diagonal matrix $\mathbf{R}^{(n)}$ given by $\mathbf{R}^{(n)}_{ii} := \frac{1}{r_i^{(n)}}$. The posterior mean $\hat{f}^{(n)}(x_*)$ is treated as a point estimate of $f(x_*)$, and the posterior standard deviation $s^{(n)}(x_*) := \sqrt{v^{(n)}(x_*, x_*)}$ as the uncertainty of this surrogate.

Remark 1. It is also common in practice that the simulators exhibit input-dependent noise, calling for a heteroskedastic metamodel. Given the noise distribution $\epsilon_i \sim \mathcal{N}(0, \tau(x_i)^2)$ with a known $\tau(\cdot)$, the conditional distribution $f|\mathcal{A}_n$ is given by

$$\hat{f}^{(n)}(x_*) = \mathbf{k}(x_*)[\mathbf{K} + \tilde{\mathbf{R}}^{(n)}]^{-1}\bar{\mathbf{y}}_{1:k_n},$$
$$v^{(n)}(x_*, x'_*) = K(x_*, x'_*) - \mathbf{k}(x_*)[\mathbf{K} + \tilde{\mathbf{R}}^{(n)}]^{-1}\mathbf{k}(x'_*)^T,$$

with the diagonal matrix $\tilde{\mathbf{R}}^{(n)}$ given by $\tilde{\mathbf{R}}^{(n)}_{ii} := \frac{\tau(x_i)^2}{r_i^{(n)}}$. All the batching algorithms proposed in Section 3 and 4 naturally extend to the heteroskedastic context if we replace $\tau^2\mathbf{R}^{(n)}$ with $\tilde{\mathbf{R}}^{(n)}$. The main challenge is then to handle estimation of the unknown conditional variance $\tau(\cdot)$, see e.g. [1, 7]. The algorithms proposed below have been ported to work with the `R hetGP` library [7] that provides an efficient way to jointly learn the mean and variance response surfaces under replicated designs.

3 | ADAPTIVE DESIGNS

3.1 | Level Set Estimation

An adaptive DoE approach is needed to select $\bar{x}_1, \bar{x}_2, \ldots$ sequentially since the level-set S is defined in terms of the unknown f. The standard framework of DoE is to add new inputs one-by-one at each round, using an acquisition function $\mathcal{I}_n(x)$ to pick $\bar{x}_{n+1}$. The acquisition function quantifies the value of information from running a new simulation at x conditional on an existing

training set $\mathcal{A}_n$, and picks $\bar{x}_{n+1}$ as the myopic maximizer of $\mathcal{I}_n$:

$$\bar{x}_{n+1} = \arg\sup_{x \in D} \mathcal{I}_n(x). \tag{6}$$

Building upon the seminal Expected Improvement criterion [18], various level-set sampling criteria were proposed by Bichon et al. [6], Picheny et al. [29], Bect et al. [5] and Ranjan et al. [31]. Further instances of $\mathcal{I}(x)$ can be found in Chevalier et al. [13 12], Azzimonti et al. [2 4], and Bolin and Lindgren [9]. The basic idea in sequential level-set estimation is to assess the *information gain* from new simulations, targeting the learning of the contour. Most of the above criteria were originally proposed for deterministic experiments with no simulation noise, or cases with known τ^2. We refer to Lyu et al. [27] for a summary of level set estimation in stochastic experiments with heteroskedastic $\tau^2(x)$, which can be seen as the counterpart of the earlier study in Jalali et al. [17] for Bayesian Optimization with stochastic simulators.

In this section we construct a sequential batched DoE to jointly select $(\bar{x}_{n+1}, r_{n+1})$. At each DoE round we pick a *new* input $\bar{x}_{n+1}$ and the associated replication amount r_{n+1}; thus by round n there are n unique inputs. In our first proposal, we formulate this task as balancing the trade-off between accuracy and cost. A small number of replicates is cheap to simulate but yields inaccurate information about ∂S; querying with many replicates is expensive but accurate. An imperfect analogy can be made to multi-fidelity Bayesian Optimization [19, 30]. As a second proposal, we relate replication to simulation and model fitting overhead costs, leading to maximization of the information gain $\mathcal{I}(x, r)$ per unit cost [20, 28].

Remark 2. Another meaning of batched DoE refers to selecting multiple new inputs $\bar{x}_k$ in parallel, see Chevalier et al. [12]. In this article, batching always refers to using replicates; we add (at most) one new input at each DoE round.

To begin, we repurpose two existing acquisition functions well suited to our needs. In our first proposal, we formulate the choice of input x_{n+1} and its replicate count r_{n+1} as two separate steps, which implies that $\mathcal{I}_n$ is only based on the existing information. The first acquisition function is Contour Upper Confidence Bound (cUCB) [27] which stems from the Upper Confidence Bound (UCB) strategies proposed by Srinivas et al. [33] for Bayesian Optimization. cUCB blends the minimization of $|\hat{f}^{(n)}(x)|$ (exploitation) with maximization of the posterior uncertainty $s^{(n)}(x)$ (exploration):

$$\mathcal{I}_n^{\mathrm{cUCB}}(x) := \left\{ -|\hat{f}^{(n)}(x)| + \rho^{(n)} s^{(n)}(x) \right\} \mu(x), \tag{7}$$

where $\rho^{(n)}$ is a sequence of UCB weights, and μ is a probability measure on the Borel σ-algebra $\mathcal{B}(D)$ (e.g., $\mu = \mathrm{Leb}_D$ the Lebesgue measure on D). Thus, cUCB targets inputs with high response uncertainty (large $s^{(n)}(x)$), and close to the contour $\partial\hat{S}$ (small $|\hat{f}^{(n)}(x)|$). See Lyu et al. [27] on the choice of the UCB weight sequence $\rho^{(n)}$. Maximizing $\mathcal{I}_n^{\mathrm{cUCB}}(\cdot)$ yields x_{n+1}; see Sections 3.2 and 3.3 on various ways to select the corresponding r_{n+1}.

In the second proposal, we jointly pick x_{n+1} and r_{n+1} in a single step, utilizing a look-ahead criterion. The gradient Stepwise Uncertainty Reduction (gSUR) criterion focuses on the local empirical error E_n defined by

$$E_n(x) := \Phi\left(-\frac{|\hat{f}^{(n)}(x)|}{s^{(n)}(x)} \right). \tag{8}$$

We interpret $E_n(x)$ as the local probability of misclassification of $\{x \in S\}$, see Bichon et al. [6], Echard et al. [15], Lyu et al. [27] and Ranjan et al. [31]. gSUR aims to select the input which produces the greatest *reduction* between the current $E_n(x)$ given $\mathcal{A}_n$ and the expected $E_{n+1}(x)$ conditional on the one-step-ahead design, $\mathcal{A}_{n+1} = \mathcal{A}_n \cup (\bar{x}_{n+1}, r_{n+1}, \bar{y}_{n+1})$. To do so, gSUR ties the selection of $\bar{x}_{n+1}$ to the look-ahead standard deviation $s^{(n+1)}(x, r)$ at x conditional on $\mathcal{A}_n$ and sampling r times at x. The latter is proportional to the current standard deviation $s^{(n)}(x)$ with the proportionality factor linked to r [14]:

$$\frac{s^{(n+1)}(x, r)^2}{s^{(n)}(x)^2} = \frac{\frac{\tau^2}{r}}{\frac{\tau^2}{r} + s^{(n)}(x)^2}, \tag{9}$$

since the replicated outputs $y_{n+1}^{(j)}$ are i.i.d.. Based on (9) and using the fact that $\mathbb{E}_{\bar{Y}(x)}[\hat{f}^{(n+1)}(x)] = \hat{f}^{(n)}(x)$, the gSUR metric approximates the effect of $\bar{Y}(x)$ on the look-ahead local empirical error $E_{n+1}(x)$:

$$\begin{aligned} \mathcal{I}_n^{\mathrm{gSUR}}(x, r) &:= \left\{ \Phi\left(-\frac{|\hat{f}^{(n)}(x)|}{s^{(n)}(x)} \right) - \Phi\left(-\frac{|\hat{f}^{(n)}(x)|}{s^{(n+1)}(x, r)} \right) \right\} \mu(x) \\ &\simeq \left\{ E_n(x) - \mathbb{E}_{\bar{Y}(x)}\left[E_{n+1}(x) \right] \right\} \mu(x). \end{aligned} \tag{10}$$

We note that $\mathcal{I}_n^{\mathrm{gSUR}}(x, r) = 0$ for $x \in \partial\hat{S}^{(n)}$ (i.e. when $\hat{f}^{(n)}(x) = 0$) so that the gSUR metric naturally enforces some exploration by sampling close to, but not exactly at, the estimated contour.

Algorithm 1 Multi-Level Batching (MLB)

Input: $\mathbf{r}_L$, η, k_0, r_0
$\mathcal{A}_{k_0} \leftarrow \{(\bar{x}_i, r_0, \bar{y}_i), 1 \le i \le k_0\}$, $(\hat{f}^{(k_0)}, s^{(k_0)}) \leftarrow f|\mathcal{A}_{k_0}$, $\gamma \leftarrow Ave(s^{(0)}(\bar{x}_{1:k_0}))$.
$N_{k_0} \leftarrow r_0 \times k_0$.
for $n = k_0, k_0+1, \dots$ **do**
  $\bar{x}_{n+1} \leftarrow \arg\max_{x \in D} \mathcal{I}_n^{cUCB}(x)$.
  while $s^{(n+1)}(\bar{x}_{n+1}, r^1) < \gamma$ ▷ Check if need to lower threshold **do**
    $\gamma \leftarrow \eta \times \gamma$.
  end while
  $r_{n+1} \leftarrow \max\{r \in \mathbf{r}_L : s^{(n+1)}(\bar{x}_{n+1}, r) \ge \gamma\}$.
  $\bar{y}_{n+1} \leftarrow \frac{1}{r_{n+1}} \sum_{j=1}^{r_{n+1}} y^{(j)}$.
  Update $\mathcal{A}_{n+1} \leftarrow \mathcal{A}_n \cup \{(\bar{x}_{n+1}, r_{n+1}, \bar{y}_{n+1})\}$.
  Obtain $(\hat{f}^{(n+1)}, s^{(n+1)}) \leftarrow f|\mathcal{A}_{n+1}$.
  $N_{n+1} \leftarrow N_n + r_{n+1}$.
end for

3.2 | Multi-Level Batching

Having determined $\bar{x}_{n+1}$ via the cUCB criterion $\mathcal{I}_n^{cUCB}$ (7), we turn to the task of picking r_{n+1}. The most basic batching strategy is Fixed Batching (FB):

$$r_{n+1} \equiv r_0$$

for some pre-specified batching level r_0. To improve upon FB, we select r_{n+1} from a discrete set $\mathbf{r}_L := \{r^1, \dots, r^L\}$, interpreted as representing L different *sampling levels*. Query at x on the ℓ-th level implies using r^ℓ replicates to generate observations $y^{(j)}, j = 1, \dots, r^\ell$ yielding the average $\bar{y}$. The cost of the ℓ-th level is proportional to r^ℓ.

In our context, we rely on the look-ahead standard deviation $s^{(n+1)}(\bar{x}_{n+1}, \cdot)$ in (9). Our Multi-Level Batching (MLB) Algorithm 1 aims to match $s^{(n+1)}(\bar{x}_{n+1}, r_{n+1})$ with a given threshold γ_n which acts as the target level for the next-step standard deviation. Intuitively, $1/\gamma_n$ controls the credibility of the model; it is progressively raised as the input space is explored. Recall that $r \mapsto s^{(n+1)}(\bar{x}_{n+1}, r)$ is monotone decreasing in (9); MLB chooses the *highest level* $r_{n+1} \in \mathbf{r}_L$ for which $s^{(n+1)}(\bar{x}_{n+1}, r_{n+1}) > \gamma_n$. If $s^{(n+1)}(\bar{x}_{n+1}, r) > \gamma_n$ for all $r \in \mathbf{r}_L$ then we use the highest $r_{n+1} = r^L$; if $s^{(n+1)}(\bar{x}_{n+1}, r) < \gamma_n$ for all $r \in \mathbf{r}_L$ then we lower the threshold by multiplying γ_n by a reduction factor $\eta < 1$, and try to identify r_{n+1} again, cf. [19].

3.3 | Ratchet Batching

By construction, the MLB Algorithm 1 will step back and forth between different replication levels r^ℓ. Since intuitively the design should concentrate as n grows, we expect r_n to grow over time which is achieved through the decreasing γ_n. By enforcing that $n \mapsto r_n$ is monotonically non-decreasing (in line with the intuition that replication becomes increasingly beneficial as n grows) we can simplify the choice of r_{n+1} and reduce algorithmic overhead. The resulting Ratchet Batching (RB) scheme picks r_{n+1} among just two replication levels (compared to L levels in MLB) and is summarized in Algorithm 2. Let $r_n^\uparrow = \min\{r \in \mathbf{r}_L : r > r_n\}$ be the next level. Then RB either keeps $r_{n+1} = r_n$ if $s^{(n+1)}(\bar{x}_{n+1}, r_n) \ge \gamma_n > s^{(n+1)}(\bar{x}_{n+1}, r_n^\uparrow)$ or increments to $r_{n+1} = r_n^\uparrow$ if $s^{(n+1)}(\bar{x}_{n+1}, r_n) > s^{(n+1)}(\bar{x}_{n+1}, r_n^\uparrow) \ge \gamma_n$. In the third case where $s^{(n+1)}(\bar{x}_{n+1}, r_n) < \gamma_n$ we lower the threshold γ_n as in MLB. For RB, the reduction factor η for γ should be close to 1, to avoid excessive ratcheting up. If η is not large enough, there is a risk to skip levels in $\mathbf{r}_L$ and to end up with excessive replication relative to number of simulation calls, leading to insufficient exploration.

3.4 | Adaptively Batched Stepwise Uncertainty Reduction

The FB, MLB and RB schemes all pick $\bar{x}_{n+1}$ first and then r_{n+1}. We next propose a procedure to pick both through a joint criterion optimization. The main idea is to tie the choice of r_{n+1} to *cost*, namely to maximize the ratio of the information gain and the cost of generating r outputs, plus the optimization overhead. The inclusion of the overhead in $\mathcal{I}_n$ comes from [35, 20, 28] in Bayesian Optimization problems, where the authors treated the total cost as the sum of query cost T_{sim} and the GP metamodeling overhead c_{ovh}. Stroh et al. [34] discussed estimating a probability of exceeding a threshold in a multi-fidelity stochastic simulator,

Algorithm 2 Ratchet Batching (RB)

Input: $\mathbf{r}_L$, η, k_0, r_0
$\mathcal{A}_{k_0} \leftarrow \{(\bar{x}_i, r_0, \bar{y}_i), 1 \le i \le k_0\}$, $(\hat{f}^{(k_0)}, s^{(k_0)}) \leftarrow f|\mathcal{A}_{k_0}$, $\gamma \leftarrow s^{(k_0)}$.
$N_{k_0} \leftarrow r_0 \times k_0$.
for $n = k_0, k_0+1, \dots$ **do**
  $\bar{x}_{n+1} \leftarrow \arg\max_{x \in D} \mathcal{I}_n^{cUCB}(x)$.
  while $s^{(n+1)}(\bar{x}_{n+1}, r_n) < \gamma$ **do** ▷ Check if need to lower threshold
    $\gamma \leftarrow \eta \times \gamma$.
  end while
  $r_n^{\uparrow} \leftarrow \min\{r \in \mathbf{r}_L : r > r_n\}$
  $r_{n+1} \leftarrow r_n \cdot 1_{\{s^{(n+1)}(\bar{x}_{n+1}, r_n^{\uparrow}) < \gamma\}} + r_n^{\uparrow} \cdot 1_{\{s^{(n+1)}(\bar{x}_{n+1}, r_n^{\uparrow}) \ge \gamma\}}$
  $\bar{y}_{n+1} \leftarrow \frac{1}{r_{n+1}} \sum_{j=1}^{r_{n+1}} y^{(j)}$.
  Update $\mathcal{A}_{n+1} \leftarrow \mathcal{A}_n \cup \{(\bar{x}_{n+1}, r_{n+1}, \bar{y}_{n+1})\}$.
  Obtain $(\hat{f}^{(n+1)}, s^{(n+1)}) \leftarrow f|\mathcal{A}_{n+1}$.
  $N_{n+1} \leftarrow N_n + r_{n+1}$.
end for

where the input $\bar{x}_{n+1}$ and the fidelity are estimated in a sequential way. We develop an analogue for level-set estimation via a gSUR-based acquisition function

$$\mathcal{I}_n^{ABSUR}(x, r) := \frac{\mathcal{I}_n^{gSUR}(x, r)}{c(r) + c_{ovh}(n)}, \tag{11}$$

where $c_{ovh}(n)$ is the overhead and $c(r) = r \cdot T_{sim}$ is the cost of r evaluations, linear in r. Combining (11) and (10), we obtain

$$\mathcal{I}_n^{ABSUR}(x, r) := \frac{\Phi\left(-\frac{|\hat{f}^{(n)}(x)|}{s^{(n)}(x)}\right) - \Phi\left(-\frac{|\hat{f}^{(n)}(x)|}{s^{(n)}(x)} \frac{\sqrt{r s^{(n)}(x)^2 + \tau^2}}{\tau}\right)}{r \cdot T_{sim} + c_{ovh}(n)}. \tag{12}$$

The resulting ABSUR Algorithm 3 myopically maximizes $\mathcal{I}^{ABSUR}$ over $x \in D$ and $r \in \mathcal{R} = [\underline{r}, \bar{r}]$. Intuitively, similar to the gSUR, ABSUR also targets the neighborhood of the zero contour ∂S and the value of r_{n+1} is controlled by $s^{(n)}(x)^2$ and $c_{ovh}(n)$; more replication results when $s^{(n)}(x)^2$ is small (neighborhood of the zero contour ∂S) or $c_{ovh}(n)$ is large (at a later stage of active learning). One could replace the numerator in (12) with other similar metrics that target reduction of contour uncertainty [27].

Algorithm 3 Adaptive Batched SUR (ABSUR)

Input: $\mathcal{R} = [\underline{r}, \bar{r}]$, k_0, r_0, T_{sim}, overhead cost function $n \mapsto c_{ovh}(n)$
$\mathcal{A}_{k_0} \leftarrow \{(\bar{x}_i, r_0, \bar{y}_i), 1 \le i \le k_0\}$, $(\hat{f}^{(k_0)}, s^{(k_0)}) \leftarrow f|\mathcal{A}_{k_0}$
$N_{k_0} \leftarrow r_0 \times k_0$
for $n = k_0, k_0+1, \dots$ **do**
  $(\bar{x}_{n+1}, r_{n+1}) \leftarrow \arg\sup_{x \in D, r \in \mathcal{R}} \mathcal{I}_n^{ABSUR}(x, r)$.
  $\bar{y}_{n+1} \leftarrow \frac{1}{r_{n+1}} \sum_{j=1}^{r_{n+1}} y^{(j)}$.
  Update $\mathcal{A}_{n+1} \leftarrow \mathcal{A}_n \cup \{(\bar{x}_{n+1}, r_{n+1}, \bar{y}_{n+1})\}$.
  Obtain $(\hat{f}^{(n+1)}, s^{(n+1)}) \leftarrow f|\mathcal{A}_{n+1}$.
  $N_{n+1} \leftarrow N_n + r_{n+1}$.
end for

There are four hyperparameters in ABSUR: the simulation cost T_{sim}, the overhead cost function $c_{ovh}(n)$ and the lower/upper bounds of replication $[\underline{r}, \bar{r}]$. For $c_{ovh}(n)$ we follow the recipe in [28], modeling it as a quadratic function of n to reflect the prediction complexity of GPs:

$$c_{ovh}(n; \boldsymbol{\theta}) = \theta_0 + \theta_1 n + \theta_2 n^2, \tag{13}$$

where $\boldsymbol{\theta}$ are fitted empirically. Alternatively Klein et al. [20] kept $c_{ovh}(n)$ as a constant. The constant T_{sim} represents the cost of obtaining each observation, measured in the same units as $c_{ovh}(n)$ (up to rescaling $\boldsymbol{\theta}$, we can assume $T_{sim} = 1$). If simulations are cheap, we would like to replicate more, and indeed lower T_{sim} leads to larger r_n's and therefore smaller designs. This feature implies that ceteris paribus T_{sim} should be set larger when input spaces are more voluminous, e.g. in higher-dimensional settings.

4 | ADAPTIVE DESIGN WITH STEPWISE ALLOCATION

The four strategies (FB, MLB, RB and ABSUR) discussed in Section 3 visit each input site $\bar{x}_{n+1}$ only once. Consequently, the respective replicate count r_{n+1} is determined at step $n+1$ and then remains the same throughout the latter steps. As an alternative, one can sequentially *allocate* new simulations across existing designs, thereby gradually growing $r_i^{(n)}$. Namely, the algorithm identifies existing "informative" inputs and augments their replicate counts, without changing the number of unique inputs k_n across the sequential design rounds n. In our context, we pair this augmentation with the option of expanding the design set itself. This choice is similar to the classical exploitation (do not change k_n) versus exploration (increase k_n). The resulting Adaptive Design with Stepwise Allocation (ADSA) approach resembles *Stepwise Approximate Optimal Design (SAO)*, an IMSE-based sequential design strategy proposed by Chen and Zhou [11] for mean response prediction.

At each step n of the ADSA strategy we are given a budget of $\Delta r^{(n)}$ additional simulations, and the main decision is to determine whether we should choose a new input $\bar{x}_{k_n+1}$ that then receives all these $\Delta r^{(n)}$ replicates, or we should allocate the $\Delta r^{(n)}$ new simulator calls across the existing inputs $\bar{\mathbf{x}}_{1:k_n}$. In the latter case, we aim to minimize the global look-ahead integrated contour uncertainty $\mathcal{L}^{(n+1)}$ where the metric $\mathcal{L}^{(n)}$ is defined by

$$\mathcal{L}^{(n)} := \sum_{j=1}^{M} \omega_j^{(n)} \hat{f}^{(n)}(x_{j,*}) = (\boldsymbol{\omega}^{(n)})^T \mathbf{f}_*^{(n)} \simeq \int_D \Phi(-\hat{f}(x)/s^{(n)}(x)) \hat{f}^{(n)}(x) \mu(dx), \tag{14}$$

where $\mathbf{x}_* = x_{1,*}, \ldots, x_{M,*}$ is a test set of size M, $\mathbf{f}_*^{(n)} \equiv \hat{f}(\mathbf{x}_*)$ is the vector of predicted responses at $\mathbf{x}_*$, and $\omega_j^{(n)} \equiv \omega(x_{j,*})\mu(x_{j,*}) = \Phi(-\hat{f}^{(n)}(x_{j,*})/s^{(n)}(x_{j,*}))\mu(x_{j,*})$ are the weights that target the level-set region of interest (compare to the targeted integrated mean square error (tIMSE) criterion proposed by Picheny et al. [29]).

For allocation purposes, we approximate the look-ahead $\mathcal{L}^{(n+1)}$ as a linear combination of the M predictions $\hat{f}^{(n+1)}(x_{j,*})$ with *fixed* weights $\boldsymbol{\omega}^{(n)}$, whereby our goal is to minimize the variance of $(\boldsymbol{\omega}^{(n)})^T \mathbf{f}_*^{(n+1)}$ conditional on the extra allocations $\Delta r_i^{(n)}$ at each input $\bar{x}_i$. Since the covariance matrix of $\mathbf{f}_*^{(n+1)}$ given replication counts $\mathbf{R}^{(n+1)}$ is

$$\mathbf{C}^{(n+1)} = \mathbf{k}(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}(\mathbf{x}_*, \bar{\mathbf{x}}_{1:k_n})(\boldsymbol{K} + \tau^2 \mathbf{R}^{(n+1)})^{-1} \mathbf{k}(\mathbf{x}_*, \bar{\mathbf{x}}_{1:k_n})^T \tag{15}$$

the objective becomes the quadratic program that minimizes

$$\mathcal{I}_{SAO}((\Delta r_i)_{i=1}^{k_n}) = (\boldsymbol{\omega}^{(n)})^T \mathbf{C}^{(n+1)} \boldsymbol{\omega}^{(n)} \tag{16}$$

under the constraint $\sum_i \Delta r_i^{(n)} = \Delta r^{(n)}$.

Define the $k_n \times k_n$ matrix $\boldsymbol{\Sigma}^{(n)} = \mathbf{K} + \tau^2 \mathbf{R}^{(n)}$ and the $M \times k_n$ matrix $\mathbf{K}_* := \mathbf{K}(\mathbf{x}_*, \bar{\mathbf{x}}_{1:k_n})$. The next proposition, proven in Appendix A, explains how to pick $\Delta r_i^{(n)}$'s to minimize (16).

Proposition 4.1. Let $\Delta \mathbf{R}^{(n)} := \mathbf{R}^{(n)} - \mathbf{R}^{(n+1)}$ be a $k_n \times k_n$ diagonal matrix with elements $\Delta \mathbf{R}_{ii}^{(n)} = \frac{\Delta r_i^{(n)}}{(r_i^{(n)} + \Delta r_i^{(n)}) r_i^{(n)}} = [r_i^{(n)}]^{-1} - (r_i^{(n)} + \Delta r_i^{(n)})^{-1}$, $i = 1, \ldots, k_n$. Assume $\max_{i=1,\ldots,k_n} \Delta \mathbf{R}_{ii}^{(n)} \ll 1$. The optimal allocation rule that minimizes (16) is to assign $\Delta r_i^{(n)}$ to each $\bar{x}_i$ such that

$$r_i^{(n)} + \Delta r_i^{(n)} \propto \mathbf{U}_i^{(n)}, \tag{17}$$

where

$$\mathbf{U}^{(n)} = (\boldsymbol{\Sigma}^{(n)})^{-1} \mathbf{K}_*^T \boldsymbol{\omega}^{(n)}. \tag{18}$$

After obtaining the allocations $\Delta \mathbf{r}_{1,\ldots,k_n}^{(n)}$, we compute the resulting look-ahead tIMSE metric:

$$\mathcal{I}_{SAO}^{(n)-all} := \sum_{j=1}^{M} \tilde{s}^{(n+1)}(x_{j,*})^2 \omega_j^{(n)}, \tag{19}$$

Algorithm 4 Adaptive Design with Stepwise Allocation (ADSA)

Input: $\bar{\mathbf{x}}_*$, $\bar{\mathbf{x}}_{1:k_0}$, k_0, r_0, c_{bt}
$\mathcal{A}_{k_0} \leftarrow \{(\bar{x}_i, r_0, \bar{y}_i), i = 1, ..., k_0\}$. $(\hat{f}^{(k_0)}, s^{(k_0)}) \leftarrow f|\mathcal{A}_{k_0}$, $N_0 \leftarrow r_0 \times k_0$.
for $n = k_0, k_0 + 1, \dots$ **do**
  $\Delta r^{(n)} \leftarrow c_{bt}\sqrt{n}$.
  Calculate allocations $\Delta r_i^{(n)}, 1 \le i \le k_n$ with Algorithm 5 (see Appendix B).
  $\bar{x}_{k_n+1} \leftarrow \arg\max_{x\in D} \mathcal{I}_n^{cUCB}(x, \Delta r^{(n)})$.
  Calculate $\mathcal{I}_{SAO}^{(n)-all}$, $\mathcal{I}_{SAO}^{(n)-new}$ in (21) and (19).
  Case 1:
    New $\bar{y}_{k_n+1} \leftarrow \frac{1}{\Delta r^{(n)}} \sum_{j=1}^{\Delta r^{(n)}} y^j(\bar{x}_{k_n+1})$.
    Update $\mathcal{A}_{n+1} \leftarrow \mathcal{A}_n \cup \{(\bar{x}_{k_n+1}, \Delta r^{(n)}, \bar{y}_{k_n+1})\}$.
    $N_{n+1} \leftarrow N_n + \sum_i \Delta r_i^{(n)}$ (May not be exactly $\Delta r^{(n)}$).
    $k_{n+1} \leftarrow k_n + 1$.
  Case 2:
    For $i = 1, ..., k_n$, update $\bar{y}_i \leftarrow \frac{\bar{y}_i \times r_i^{(n)} + \sum_{j=1}^{\Delta r_i^{(n)}} y^j(\bar{x}_i)}{r_i^{(n)} + \Delta r_i^{(n)}}$, $r_i^{(n+1)} \leftarrow r_i^{(n)} + \Delta r_i^{(n)}$
    Update $\mathcal{A}_{n+1} \leftarrow \{(\bar{x}_i, r_i^{(n+1)}, \bar{y}_i)\}_{i=1,\dots,k_n}$.
    $N_{n+1} \leftarrow N_n + \sum_{i=1}^{k_n} \Delta r_i^{(n)}$
    $k_{n+1} \leftarrow k_n$
  Obtain $(\hat{f}^{(n+1)}, s^{(n+1)}) \leftarrow f|\mathcal{A}_{n+1}$.
  ADSA: **Do** Case **1 if** $\mathcal{I}_{SAO}^{(n)-all} > \mathcal{I}_{SAO}^{(n)-new}$, **otherwise do** Case **2**
  {FDSA variant:} **Do** Case **2**.
  {DDSA variant:} **Do** Case **1** if n is odd, Case **2** if n is even.
end for

where the look-ahead variance $\tilde{s}^{(n+1)}(\cdot)^2$ is based on the new replicate counts $r_i^{(n+1)} = r_i^{(n)} + \Delta r_i^{(n)}, i = 1, \dots, k_n$, see proof in [14, 16]:

$$\tilde{s}^{(n+1)}(\mathbf{x}_*)^2 = s^{(n)}(\mathbf{x}_*)^2 - \mathbf{k}_*(\mathbf{\Sigma}^{(n)})^{-1}\Delta\mathbf{R}^{(n)}(\mathbf{\Sigma}^{(n)})^{-1}\mathbf{k}_*^T. \tag{20}$$

The alternative to allocating over existing $\bar{\mathbf{x}}_{1:k_n}$ is to pick a new input x_{k_n+1} and assign it $\Delta r^{(n)}$ simulations. To do so, we use the cUCB criterion to make it consistent with FB, MLB and RB. (Other acquisition functions can also be used and experiments suggest that the algorithm is not sensitive to this choice.) Then we evaluate the resulting $\mathcal{I}_{SAO}^{(n)-new}$:

$$\mathcal{I}_{SAO}^{(n)-new} := \sum_{j=1}^{M} s^{(n+1)}(x_{j,*}, \Delta r^{(n)})^2 \omega_j^{(n)}, \tag{21}$$

$$s^{(n+1)}(x_{j,*}, \Delta r^{(n)})^2 = s^{(n)}(x_{j,*})^2 - \frac{v^{(n)}(x_{j,*}, \bar{x}_{k_n+1})^2}{\frac{\tau^2}{\Delta r^{(n)}} + s^{(n)}(\bar{x}_{k_n+1})^2}.$$

The sums in (19)-(21) are used as approximations of the underlying integrals over $x \in D$. Finally, we compare $\mathcal{I}_{SAO}^{(n)-new}$ and $\mathcal{I}_{SAO}^{(n)-all}$ to determine whether to sample at the new $\bar{x}_{k_n+1}$ or to allocate to existing $\mathbf{x}_{1:k_n}$, picking the maximum of the two tIMSE metrics.

For FB, MLB, RB and ABSUR, as we select one new input at each step, we have $k_n = n$. However, for ADSA we either select a new input or re-allocate, so that the resulting design size satisfies $k_n < n$. Thus, relative to the earlier schemes, in ADSA the size of $\mathcal{A}_n$ and the number of DoE rounds n are no longer deterministically linked and the number of unique inputs is endogenous to the particular algorithm run.

A major goal of all our schemes is for k_n to grow sub-linearly in n, i.e. new inputs are added less frequently as more simulations are run. There are two reasons for this: (1) As k_n grows, the input space is better explored and one should favor exploitation more and more; (2) the GP overhead increases in k_n so that each decision becomes more costly and therefore large batches are preferable. Put another way, $k_n \propto n$ and constant $\Delta r^{(n)}$ is equivalent to fixed batching $\bar{r} = N_n/k_n$ and we wish for r_n to grow

TABLE 1 Parameters for the 2-D modified *Branin-Hoo* and the 6-D modified *Hartman* experiments.

	PARAMETER	2-D *Branin-Hoo*	6-D *Hartman*
Simulation budget	N_T	2000	6000
Initial design size	k_0	20	60
Initial replicates	r_0	10	10
ADSA test set in (14)	M	500	1000
Replication levels	$\mathbf{r}_L$	$[5, 10, 15, 20, 30, 40, 50, 60, 80,$ $100, 140, 180, 240, 300]$	
ABSUR replication range	$\mathcal{R}$	$[5, 200]$	$[5, 300]$
ABSUR simulation cost	T_{sim}	0.01	0.05
ABSUR overhead cost in (13)	$c_{ovh}(n)$	$\boldsymbol{\theta} = [0.137, 8.15 \times 10^{-4}, 1.99 \times 10^{-6}]$	
ADSA batch factor	c_{bt}	10	3.33

(at least on average) in n. In ADSA, we organically prefer re-allocation over adding inputs as n grows. The user can further enhance this situation by making the batches $\Delta \boldsymbol{r}^{(n)}$ also grow in n. Specifically, we have found a good heuristic in taking $\Delta \boldsymbol{r}^{(n)}$ to be proportional to $\sqrt{n}$ (see proportionality constant c_{bt} in Algorithm 4), which is faster compared to constant batch sizes and more accurate than making $\Delta \boldsymbol{r}^{(n)}$ linear in n which is overly aggressive.

Deterministic DSA. In practice we observe that the ADSA scheme tends to alternate roughly equally between re-allocation and addition of new inputs. To save computational overhead, we consider the simplified *Deterministic Design with Stepwise Allocation* (DDSA) scheme that deterministically alternates between re-allocation and adding inputs, making $k_n = k_0 + \lceil (n - k_0)/2 \rceil$ also deterministic. Observe that DDSA no longer needs to evaluate the expensive $\mathcal{I}_{SAO}^{(n)-all}$ and $\mathcal{I}_{SAO}^{(n)-new}$.

5 | RESULTS

5.1 | Synthetic Experiments and Computational Implementation Details

In this section we benchmark the schemes on three synthetic case studies, employing rescaled *Branin-Hoo* ($d = 2$) and *Hartman* ($d = 6$) functions. We make linear transformations to the standard setups in order to rescale the output to $[-1, 1]$ and have the zero-contour "in the middle" of the input space. For the Branin-Hoo case, we further restrict and rescale the original domain to make f monotone along x^1 and to generate a single zero-contour curve. Full specifications are provided in the Online Supplement, see also [27]. The 2-D case studies with the Branin-Hoo response function employ two noise settings: (i) Gaussian $\epsilon \sim \mathcal{N}(0, 1)$; and (ii) heteroskedastic Student-t where the distribution of ϵ is input-dependent: $\epsilon(x) \sim t_{6-4x^1}(0, (0.4(4x^1 + 1))^2)$. The latter setting is to test the influence of noise mis-specification. The third case study is in 6-D using the Hartman response and noise $\epsilon \sim \mathcal{N}(0, 1)$.

The squared-exponential kernel

$$K_{\text{se}}(x, x') := \sigma_{\text{se}}^2 \exp\left(- \sum_{i=1}^{d} \frac{(x^i - x'^i)^2}{2\ell_i^2} \right)$$

is used throughout as the GP covariance function. The covariance hyperparameters $\boldsymbol{\vartheta} = \{\ell_1, \ldots, \ell_d, \sigma_{\text{se}}^2\}$ are estimated via MLE using the `fmincon` optimizer in `MATLAB`. We re-fit $\boldsymbol{\vartheta}$ every five DoE steps and otherwise treat it as fixed across n. The noise variance is taken to be known (i.e. $\tau = 1$) in the first and third case studies. It is fitted (as an unknown constant) along with $\boldsymbol{\vartheta}$ for the experiments with Student-t simulation noise.

For the 2-D case study the metrics $\mathcal{ER}$, $\mathcal{I}_{SAO}^{(n)-all}$, and $\mathcal{I}_{SAO}^{(n)-new}$ are computed as an equally weighted average over test points constructed using Latin Hypercube Sampling over the entire input space. In the 6-D case study we pick 80% of the test points from the region $\{x \in D : |f(x)| < 0.7\}$ that is close to the zero-contour and the remaining 20% from the rest of the input space; the respective weights to compute the metrics are based on the volume of the former region. The same setup was used in [27]; see also [12] for a detailed comparison between different sampling methods.

We use FB with batch size $r \equiv 10$ as a baseline, and compare the performance of MLB, RB, ABSUR, ADSA and DDSA. Performance is based on the error rate $\mathcal{ER}$ in (3), i.e. evaluating (numerically, using a test set of size M) the symmetric difference

between the true and estimated level set. This is done at a fixed simulation budget N_T, i.e. each scheme is run for k_T rounds until $N_{k_T} = N_T$ the budget is exhausted. Note that the resulting number of DoE rounds k_T will vary scheme-by-scheme and potentially run-by-run. We index N_n, k_n by the DoE sequential iterations, while N_T, k_T are indexed by total budget consumed. Table 1 provides further details about the parameters specific to each scheme. To optimize the various $\mathcal{I}$ acquisition functions we use a global, gradient-free, genetic optimization approach as implemented in the `ga` function in `MATLAB`, with tolerance of 10^{-3} and 200 generations.

We fit all the Gaussian Process surrogates using the `GPstuff` suite in `MATLAB` [36]. For easier reproducibility, our supplementary material contains `R` code, including the adaptive batching heuristics, to reproduce Figure 6 below. We are happy to provide the `MATLAB` codes upon request.

The proposed adaptive batching strategies are not limited to the vanilla GP setup. Other metamodels can be straightforwardly substituted as long as they allow to efficiently evaluate the $\mathcal{I}_n$ criteria and the batch look-ahead variance $s^{(n+1)}(x, r)$. As an illustration, motivated by the non-Gaussian simulation noise in the second case study and the option pricing application in Section 6, we implement a GP metamodel with Student-t observation noise (henceforth t-GP). In the t-GP formulation ϵ_i in (2) is taken to be t-distributed with variance τ^2 and $\nu > 2$ degrees of freedom. Lyu et al. [27] showed that t-GP is a good choice in the face of noise misspecification. Appendix C provides details of using a t-GP metamodel via a Laplace approximation approach. Our schemes are moreover ported to work with the `hetGP` [7] in `R`, see Table 3 below.

5.1.1 | Algorithm Tuning Parameters

In this section we briefly describe the various tuning parameters associated with the proposed algorithms. For the UCB weight sequence $\rho^{(n)}$ in cUCB, we follow the recipe in [27] and set $\rho^{(n)} = IQR(\hat{f}^{(n)})/3Ave(s^{(n)})$ which keeps both terms in (7) approximately stable as n changes. For MLB, we initialize γ as the average standard deviation $Ave(s^{(k_0)}(\bar{x}_{1:k_0}))$ and take the reduction factor $\eta = 0.5$. For RB we use the same initial γ but decrement it slower, $\eta = 0.8$. Higher η increases the overall design size k_T and therefore computation time. For MLB, $\eta \in [0.5, 0.7]$ leads to the lowest error rate $\mathcal{ER}$; for RB, we recommend $\eta \in [0.7, 0.9]$. For the replication levels $\mathbf{r}_L$ used in MLB and RB, we manually construct a "ladder" of r^ℓ's with spacing that increases roughly proportionally. In our experience, the choice of spacings (i.e. number of levels L) does not play a major role, with the most important parameter of $\mathbf{r}_L$ being its upper bound r^L. If r^L is too low, the gains from replication are limited; if r^L is too high we observe over-exploitation with a design that does not have enough unique inputs.

For ABSUR, we recommend minimal replication level $\underline{r}$ of 5 or 10, and maximum replication of $\bar{r} = 0.05N_T$, i.e. 5% of the total budget N_T. Table D1 in Appendix D shows the impact of varying $\bar{r}$ from 1% to 100% of N_T. Unsurprisingly, increasing $\bar{r}$ decreases the design size k_T and computation cost t. Note that because the scheme tries to optimize actual r_n in the interval $[\underline{r}, \bar{r}]$, for very large $\bar{r}$ that constraint is not binding and so the impact is minimal, see last few rows in Table D1 . In the middle of its range, the role of $\bar{r}$ is similar to that of r^L for MLB and RB.

The coefficients $\boldsymbol{\theta}$ in the quadratic overhead function $c_{ovh}(n)$ in (13), as well as the simulation cost T_{sim} are pre-tuned via a linear least squares regression with the given simulator and hardware setup. Thus, they are not really tuning parameters, but reflect the relative computational effort between regression and simulation. Nevertheless, to provide some intuition, the right panel of Table D1 shows the impact of changing T_{sim} for one of our experimental setups. Higher T_{sim} encourages exploration. Thus, to avoid too much exploitation and very high r_n's we recommend not to make T_{sim} too small; in our experiments this translates to $T_{sim} \in [0.01, 1]$.

For the batch factor in ADSA and DDSA we take $c_{bt} = 20/d$, which favors more exploration in higher-dimensional problems with larger input domains. Table D2 in Appendix D shows the effect of changing $c_{bt} \in [10/d, 80/d]$. For both algorithms the design size k_T decreases as c_{bt} increases. However, the change in k_T, as well as in the error rate $\mathcal{ER}$ for DDSA is more significant than for ADSA, especially when simulation noise is low. DDSA achieves lower $\mathcal{ER}$ with a smaller c_{bt}, while ADSA has a lower error rate with c_{bt} lower than $20/d$.

A benefit of working with simulation batches is that the related computation is trivially parallelizable. Like all sequential methods, our schemes cannot be run fully in parallel, since the choice of x_{n+1} must be done one-by-one. Nevertheless, assuming that most time is spent on simulation, distributing the generation of $y_i^{(j)}$, $j = 1, \ldots, r_n$ across computing cores will generate substantial savings that are not possible without batching. To maximally leverage this, one should set r_n to be a multiple of the available number of cores. In the examples below we do not employ any parallelization.

TABLE 2 Scheme performance across the two synthetic case studies. Results are means (± standard deviations) from 50 runs of each combination of a metamodel and batching scheme.

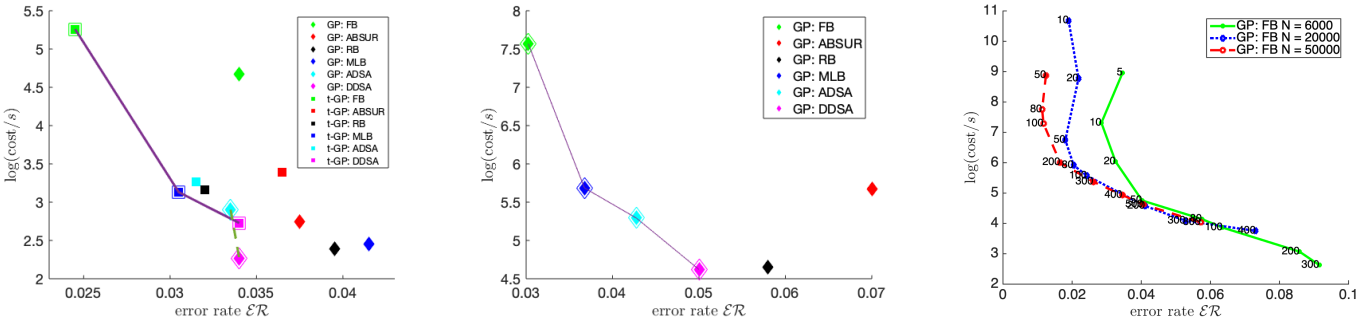
DESIGN	ERROR RATE $\mathcal{ER}_T$	TIME/S	AVE k_T
2-D Branin-Hoo WITH $\epsilon \sim \mathcal{N}(0,1)$			
FB	0.019 ± 0.005	118.89	200.00
ABSUR	0.021 ± 0.007	10.32	35.20
RB	0.021 ± 0.008	8.30	38.72
MLB	$\mathbf{0.018 \pm 0.008}$	8.63	38.44
ADSA	0.020 ± 0.008	14.11	34.42
DDSA	0.022 ± 0.007	**7.92**	37.00
6-D Hartman WITH $\epsilon \sim \mathcal{N}(0,1)$ AND $N_T = 6000$			
FB	0.030 ± 0.004	1934.51	600.00
ABSUR	0.070 ± 0.015	289.52	159.80
RB	0.058 ± 0.014	104.68	143.40
MLB	$\mathbf{0.037 \pm 0.008}$	294.49	240.62
ADSA	0.043 ± 0.007	198.82	171.74
DDSA	0.050 ± 0.009	**101.59**	142.00
6-D Hartman WITH $\epsilon \sim \mathcal{N}(0,1)$ AND $N_T = 30000$			
FB $r_n = 50$	0.015 ± 0.002	1654.32	600.00
FB $r_n = 100$	0.016 ± 0.002	461.57	330.00
FB $r_n = 200$	0.029 ± 0.006	**152.21**	195.00
ABSUR	0.022 ± 0.003	757.18	325.25
RB	0.024 ± 0.005	227.01	237.05
MLB	0.022 ± 0.006	240.61	242.95
ADSA	$\mathbf{0.016 \pm 0.002}$	995.57	373.80
DDSA	0.017 ± 0.002	522.00	350.00

5.2 | Algorithm Performance

Our main goal with adaptive batching is improved computational performance. Of course, a faster algorithm generally requires to sacrifice predictive accuracy. As such, direct comparison of schemes is not possible but must be considered through the above trade-off. Figure 1 and Table 2 show the link between the error rate $\mathcal{ER}$ from (3) and the running time across the proposed schemes. Since we desire fast and accurate schemes, there is a Pareto frontier going from top-left to bottom-right. In the 2-D case study (shown in the left panel in Figure 1), we see that the most accurate scheme is t-GP with FB, while the fastest is GP with DDSA. Another Pareto-efficient scheme is t-GP with MLB which is arguably the best (the second fastest among t-GPs, and the second most accurate). In 6-D ABSUR works poorly, probably due to under-performance of the gSUR criterion; see [27] who showed that cUCB appears to be empirically better for this 6-D Hartman function. Another reason is that gSUR converges in a slower rate, see the middle panel in Figure 2 : gSUR takes $N_T \simeq 30000$ simulations to achieve a comparably small error rate $\mathcal{ER}$. However, in Figure 1 , $N_T = 6000$ for 6-D experiments.

Looking at the running times, we see that there are major gains from adaptive batching; the baseline FB scheme takes almost 10 times longer to run than designs with adaptive r_i's. Fixed batching generally performs well in terms of $\mathcal{ER}$ (as it ends up being more exploratory) but practically those gains are crowded out by the huge cost in computational efficiency. Overall, among the five proposed schemes the recommended choice is MLB and ADSA which tend to produce low $\mathcal{ER}$ with a significant reduction in computational time.

As mentioned in the Introduction, the benefit of replication is inextricably tied to simulation noise. To this end, in Appendix D we investigate the role of the signal-to-noise ratio (SNR) on algorithm's performance by varying the noise variance τ^2 in the



2-D Branin-Hoo | 6-D Hartman | FB Comparison

FIGURE 1 Running time and ultimate error rate $\mathcal{ER}_T$ across different schemes. *Left panel:* 2-D Branin-Hoo with heteroskedastic noise and budget $N_T = 2000$. *Middle panel:* 6-D Hartman function with Gaussian noise and $N_T = 6000$. *Right panel:* 6-D Hartman function with Gaussian noise for FB with different values of r. The Pareto frontiers are highlighted for GP (solid line) and t-GP (dashed line).

2-D case study with Gaussian noise. Figure D1 shows that as τ^2 increases, designs become smaller (k_T decreases, except for ADSA). The performance metrics are reported in Tables D1 and D2 in Appendix D. As expected, lower SNR increases $\mathcal{ER}_T$ and algorithms should be tuned depending on the level of noise. For example, for ADSA and DDSA, one should increase c_{bl} if SNR is low; for ABSUR one should increase $\bar{r}$. Some intuition can also be gleaned from Table 2 and Table 3 : the second experiment with t-distributed noise has much lower SNR compared to the first one with $\epsilon \sim \mathcal{N}(0, 1)$. Lower simulation noise means that less replication is needed, which implies reducing $\bar{r}$ and r^L and tends to advantage MLB compared to ADSA and ABSUR. Consistent with conclusions in [27], t-GP performs better than plain GP in such a setup where noise is heavy-tailed.

To further investigate the impact of noise on different schemes, as well as to showcase the use of alternative GP metamodels, Table 3 shows results for the 2D Branin-Hoo experiment with heteroskedastic noise $\epsilon \sim t_{6-4x^1}(0, (0.4(4x^1 + 1))^2)$. In this experiment we test both the different batching schemes, as well as two other metamodel familiies: t-GP and hetGP. t-GP extends the GP paradigm to allow for t-distributed observations, see Appendix C. `hetGP`, implemented in the eponymous `R` library [7], non-parametrically learns not just the mean response but also the input-dependent observation noise surface $\tau^2(\cdot)$.

Using the `hetGP` library we further compare our adaptive batching to the cIMSPE algorithm described in Section 4.2 of [7]. cIMSPE is similar in spirit to ADSA except that it allocates simulations one-by-one. At each step, cIMSPE uses a criterion $\mathcal{I}_n$ to decide whether to add a new unique input, or increase by one the replicate count at an existing input. The comparison is based on the expected value of $\mathcal{I}_n$ and is replication-biased by comparing not just one-step-ahead but over a horizon of h. We use the cUCB criterion $\mathcal{I}^{cUCB}$ and a horizon of $h = 3$. While cIMSPE offers a strong motivation for sequential construction of replicated designs, it is extremely slow because it has as no intrinsic batching and therefore requires N sequential steps to allocate N simulations. Consequently, it is only feasible when N is small and takes orders of magnitudes more time in our setting with $N = 2000$. This limitation of the cIMSPE was one of the motivations for explicitly incorporating batching (rather than simply accommodating replication) in our approaches.

Several observations can be gleaned from Table 3 : (1) In terms of metamodels, t-GP and hetGP perform better than plain GP in this context with heteroskedastic noise. (2) In terms of adaptive batching schemes, their accuracy ($\mathcal{ER}$) is generally quite similar. DDSA runs the fastest and yields better-than-average error rates. (3) The comparator schemes yield similar error rates but are not competitive in terms of running times: cIMSPE is about 100 times slower and generates over a 1000 unique inputs compared to less than 50 for our schemes. FB is also slow ($\sim$ 6 times slower), although in combination with t-GP it does achieve the overall lowest error rate $\mathcal{ER}$.

To give some intuition about how the replication level should depend on the total budget N_T, the right panel of Figure 1 shows the performance of FB as we vary r and N_T. As expected, lower r generally leads to lower error rate $\mathcal{ER}$ but longer running time. This indicates the intrinsic necessity to explore the input space adequately which introduces a lower bound regarding the number of unique inputs $k_T = N_T/r$ for FB. However, for very low r (e.g. $r < 20$ for $N_T = 6000$) there is essentially no gain from additional exploration implying that one can safely agglomerate simulations into batches without sacrificing accuracy. The resulting J-shape in the Figure implies that there is an "optimal" $r^*(N)$ that minimizes $\mathcal{ER}$ without needless performance degradation: $r^*(6000) \simeq 10, r^*(2 \cdot 10^4) \simeq 50, r^*(5 \cdot 10^4) \simeq 100$. This feature showcases both the strength and the weakness of

TABLE 3 Scheme performance in the 2-D heteroskedastic synthetic case study with *2-D Branin-Hoo* response and noise $\epsilon(x^1, x^2) \sim t_{6-4x^1}(0, 0.16(4x^1+1)^2)$. Results are means ($\pm$ standard deviations) from 50 runs of each combination of a metamodel and batching scheme. Note that the running times for GP and t-GP, which are from MATLAB, and for hetGP, which is from R, are not comparable.

DESIGN	ERROR RATE $\mathcal{ER}_T$	TIME/S	AVE k_T
PLAIN GP IN MATLAB			
FB	0.034 ± 0.029	106.37	200.00
ABSUR	0.037 ± 0.039	15.50	39.14
RB	0.039 ± 0.035	10.93	39.92
MLB	0.041 ± 0.041	11.61	42.26
ADSA	$\mathbf{0.033 \pm 0.042}$	18.20	34.82
DDSA	0.034 ± 0.043	**9.67**	37.00
t-GP IN MATLAB			
FB	0.024 ± 0.010	192.44	200.00
ABSUR	0.036 ± 0.014	29.55	35.00
RB	0.032 ± 0.014	23.65	39.66
MLB	$\mathbf{0.030 \pm 0.018}$	22.88	39.72
ADSA	0.031 ± 0.013	26.26	30.68
DDSA	0.034 ± 0.018	**15.30**	37.00
HETGP IN R			
FB	0.035 ± 0.010	36.93	200.00
ABSUR	0.031 ± 0.011	5.38	46.40
RB	0.035 ± 0.010	1.45	48.10
MLB	0.034 ± 0.017	**1.31**	49.10
ADSA	0.035 ± 0.010	2.98	41.75
DDSA	$\mathbf{0.030 \pm 0.010}$	1.51	36.00
CIMSPE	0.032 ± 0.016	2.47 HRS	1028.20

fixed batching: in principle excellent performance is possible if $r \simeq r^*$ is fine-tuned; however such fine-tuning is very difficult and without it FB can be highly inefficient. The proposed adaptive batching schemes aim to automatically fine-tune r_n sequentially removing this limitation.

Another goal of adaptive batching is to enable an organic way to grow designs as N_T changes (while for FB r necessarily must be pre-chosen in terms of N_T). A good algorithm is able to efficiently improve its accuracy as N_T grows, avoiding excessive exploration or exploitation. The left panel of Figure 2 shows the log error rate $\mathcal{ER}$ as a function of N_T for FB, ABSUR, RB, MLB, ADSA and DDSA for the 6-D *Hartman* experiments, respectively. For FB, we stopped at $N_T = 6000$ due to prohibitive running times for designs. We observe that while all schemes perform somewhat similarly, MLB reduces the error rate $\mathcal{ER}$ at the fastest rate when $N_n < 600$, and otherwise, ADSA is the fastest. ADSA shines in the later stage of sequential development of DoE, since it needs enough "candidate inputs" to calculate the allocation rule. In terms of computational efficiency, we are concerned not with $\mathcal{ER}$ in terms of N_T but in terms of running time—i.e. how much predictive accuracy can be achieved within a given time budget. The respective relationship is shown in the middle and right panels of Figure 2 where the x-axis is now in terms of t seconds. We observe that all the adaptive schemes reduce the error rate $\mathcal{ER}$ at a faster rate than a scheme with fixed replication level. In the early stage, RB and DDSA are the fastest, and ABSUR is the slowest. However, as N_T or t continues to rise, ADSA keeps reducing the error rate $\mathcal{ER}$ and eventually achieves a smaller $\mathcal{ER}$ than other algorithms. However, ADSA usually takes slightly longer time. In conclusion, ADSA is the most accurate algorithm given a large enough cost t or simulator calls N_T, and MLB is the most accurate algorithm when N_T is small. Results are consistent with those observed in Figure 1 .

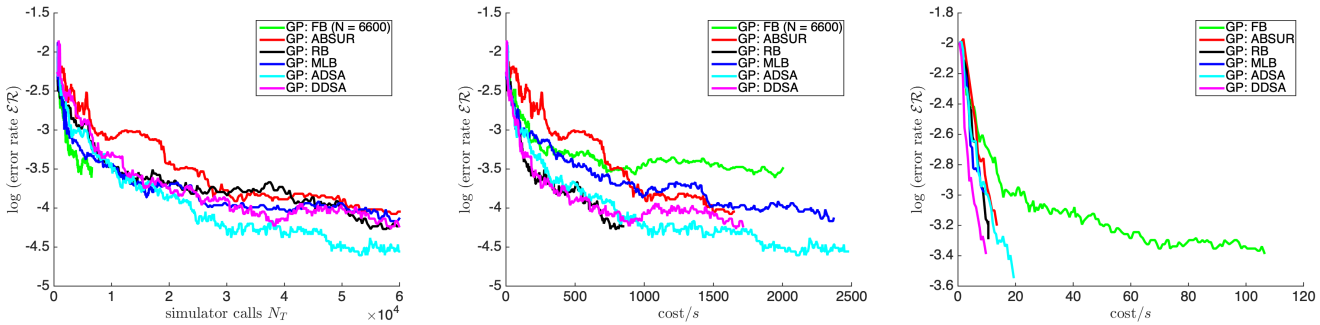


FIGURE 2 Log Error rate $\log \mathcal{ER}_t$ as a function of simulator calls N_t for FB ($r = 10$), ABSUR, RB, MLB, ADSA and DDSA and 6-D experiments (left panel). Log error rate $\log \mathcal{ER}_t$ as a function of running time t for the 6-D case study with Gaussian noise (middle panel) with $N_T = 60000$ and for the 2-D case study with heteroskedastic noise (right panel) with $N_T = 2000$. The FB algorithm is stopped at $N_t = 6000$ since computation is too slow.

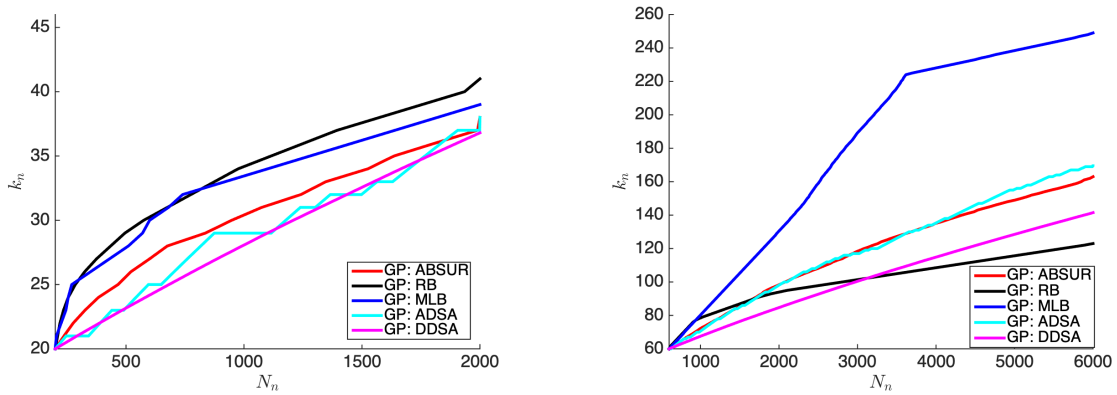


FIGURE 3 The design size k_n as a function of simulator calls N_n. Left: 2-D case study with heteroskedastic noise; Right: 6-D case study with Gaussian noise.

Recall that GP model fitting complexity is $\mathcal{O}(k_n^3)$ (driven by the matrix inversion $\mathbf{K}^{-1}$), so that the design size $k_n = |\mathcal{A}_n|$ is the primary driver of computational efficiency. In the baseline FB scheme, $r^{(n)} \equiv r$ is constant so that $k_n = N_n/r$ grows linearly in simulator budget N_n. This is precisely the reason that a constant r becomes impossible to maintain as N_n grows and why we had to abandon FB in the left panel of Figure 2 . A key aim of adaptive batching is to achieve *sub-linear* growth of k_n i.e. $k_n/N_n \to 0$ as n grows so that $r^{(n)}$ keeps getting larger as we develop the DoE. Figure 3 plots k_n as a function of N_n for 2-D and 6-D experiments. As desired, we observe a generally concave shape, which is approximately of square-root shape. The stair-case shape of k_n for ADSA is due to the adaptive re-allocation of new simulations which allow to increase N_n without changing k_n at some steps. We note that RB and ADSA achieve the most concave shape and hence would be the fastest for very large N_n which can be seen indirectly in Figure 2 as well.

5.3 | Comparing Designs

To drill down into the designs obtained from different approaches, Figure 4 visualizes the adaptively batched designs produced for the 2-D Branin-Hoo experiment with heteroskedastic Student-t noise. The left panel displays the resulting design size k_T with simulation budget of $N_T = 2000$. Recall that besides FB and DDSA, design sizes of all other schemes vary across algorithm runs (i.e. k_T depends on the particular realizations $y_{1:N_T}$), so that k_T is a random variable; in the plot we visualize its boxplot across 50 runs of each scheme. The smallest designs are obtained from ADSA (31-39 unique inputs). DDSA produces exactly $k_T = 37$ unique inputs. Recall that DDSA alternates between adding a new site and re-allocating to existing sites, while ADSA

does the same adaptively; in this case we find that slightly more than half the time re-allocation is preferred. The design size k_n for ABSUR is slightly larger at 34-42. The value of k_T for RB varies from 37 to 45, while for MLB has the greatest number of unique inputs, ranging from 34 to 50. Given $N_T = 2000$ the above implies that the schemes average about $Ave(r^{(n)})$ =40-60 replicates per site. The middle panel of Figure 4 shows the replication level $r^{(n)}$ as a function of design size k_n for a typical run of schemes from Section 3.4, illustrating how replication is increased sequentially. Methods that raise $r^{(n)}$ faster end up with smaller design size k_T. ABSUR increases $r^{(n)}$ the fastest, with MLB having a similar pattern. With RB $r^{(n)}$ grows slower, implying that RB builds designs with more unique inputs.

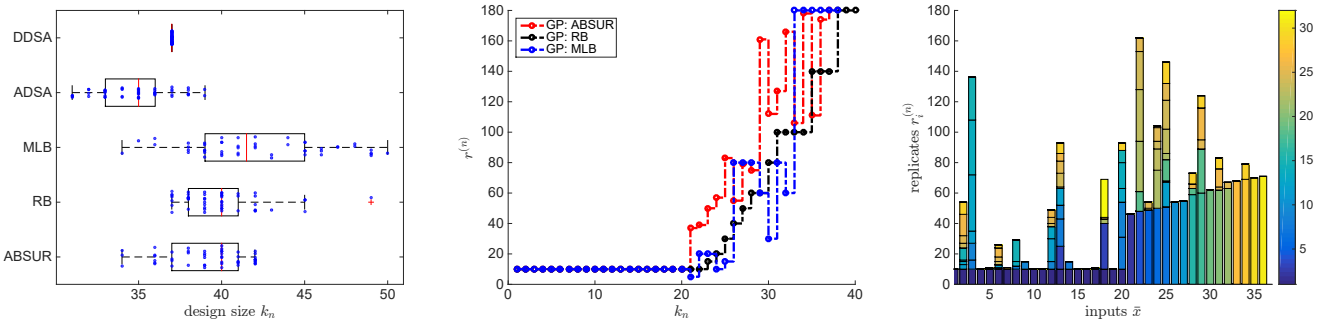


FIGURE 4 Visualizing adaptive batching for the 2-D case study with heteroskedastic noise. *Left panel:* distribution of design size k_T corresponding to $N_T = 2000$ across 50 algorithm runs. *Middle*: number of replicates $r^{(n)}$ as a function of algorithm step k_n for the schemes of Section 3. *Right:* evolution of $r_i^{(n)}$ for ADSA designs $\bar{\mathbf{x}}_{1:k_n}$. The total $r_i^{(N)}$ is decomposed into $\Delta r_i^{(n)}$ for $n = 1, \ldots, k_T$ with each Δr color-coded by round n.

The right panel of Figure 4 visualizes the replication of a representative ADSA run which has the option to add new inputs or re-allocate to existing ones. We show the sequential growth of $r_i^{(n)}$ through a stack histogram: the x-axis represents the unique inputs $\mathbf{x}_i$ as picked by the algorithm and the vertical stacks represent $\Delta r_i^{(n)}$, color-coded by the round n when they were added. We observe that only 10 out of the $n_0 = 20$ original inputs are revisited, and generally about half of the inputs are used in more than one round. At the same time, some inputs, such as $\bar{x}_{13}, \bar{x}_{20}, \bar{x}_{25}$ are visited in numerous rounds.

Figure 5 shows the estimated zero-contour $\partial\hat{S}$ with its 95% posterior credible band at $N_T = 2000$ in the 2-D test case with heteroskedastic noise. The volume of the credible band $\partial\hat{S}^{(\pm 0.95)}$, defined as

$$\partial\hat{S}^{(\pm 0.95)} = \left\{ x \in D : \left(\hat{f}^{(N_T)}(x) + 1.96 s^{(N_T)}(x)\right)\left(\hat{f}^{(N_T)}(x) - 1.96 s^{(N_T)}(x)\right) < 0 \right\}, \tag{22}$$

captures inputs x whose sign classification remains ambiguous and quantifies the uncertainty about the estimated zero-contour $\partial\hat{S}$. Numerically, we obtain $\partial\hat{S}^{(\pm 0.95)}$ and $\partial\hat{S}$ by predicting the GP surrogate on a fine 2D grid and then invoking the built-in contour-plotting commands. As expected, all schemes start by exploring the input space using a few replicates and then primarily sample in the target region around the level set, with increasing replication. Another feature that can be seen is that all methods favor the upper-left and bottom-right corners, which are regions that are simultaneously close to the edge of the input space (hence larger posterior $s_n(\cdot)$) and close to the zero contour. In particular, highest replication occurs in the upper-left region.

Comparing the first four plots, we find that the ABSUR is more efficient than RB and MLB, concentrating at the zero-contour faster and simultaneously ramping up $r^{(n)}$ quicker. In the plot, this happens already after just half-a-dozen steps. In contrast, RB takes about a dozen steps to explore with correspondingly low $r^{(n)}$'s. Although MLB also ramps up r_n quickly, it then steps back and forth between low and high replication levels, resulting in a slightly larger k_T than ABSUR. ADSA and DDSA perform similarly. One observation is that they select similar inputs to allocate the extra simulator calls. For example the initial inputs close to the left and top edge all get more replicates r_n via reallocation in ADSA and DDSA. Across the DoE rounds, ADSA chooses to reallocate budget in approximately 54% of them, so that $k_T = 0.54 N_T/\Delta r$. Therefore, the value of k_T is approximately the same for ADSA and DDSA.

Some of the design differences can be attributed to the different behavior of the underlying heuristics cUCB and gSUR. Indeed, cUCB tends to over-emphasize sampling around the zero-contour, while gSUR is more exploratory and tends to place a few

inputs right at the edge of the input domain (upper left corner and lower right corner in the plot with ABSUR). The aggressiveness of cUCB generates more accurate estimates $\partial\widehat{S}$ even if the posterior uncertainty is higher (wider CI band) sometimes.

In sum, it is possible in principle to fine-tune r_n such that the respective fixed-batch FB scheme performs as well as all the proposed algorithms. However, the "optimal" value of r_n is sensitive to the problem setting, and moreover depends on N_T. Adaptive batching designs resolve these practical challenges by automatically and sequentially picking r_n. Among the proposed schemes, DDSA and RB are the most efficient in terms of running time and producing compact designs. ADSA is a bit slower but tends to offer higher accuracy. Between ADSA and DDSA, the latter does better when k_T turns out to be large (e.g. large N_T or high SNR). ADSA is thus the recommended general-purpose scheme, being more robust to the choice of hyperparameters and offering a stable performance across all experiments.

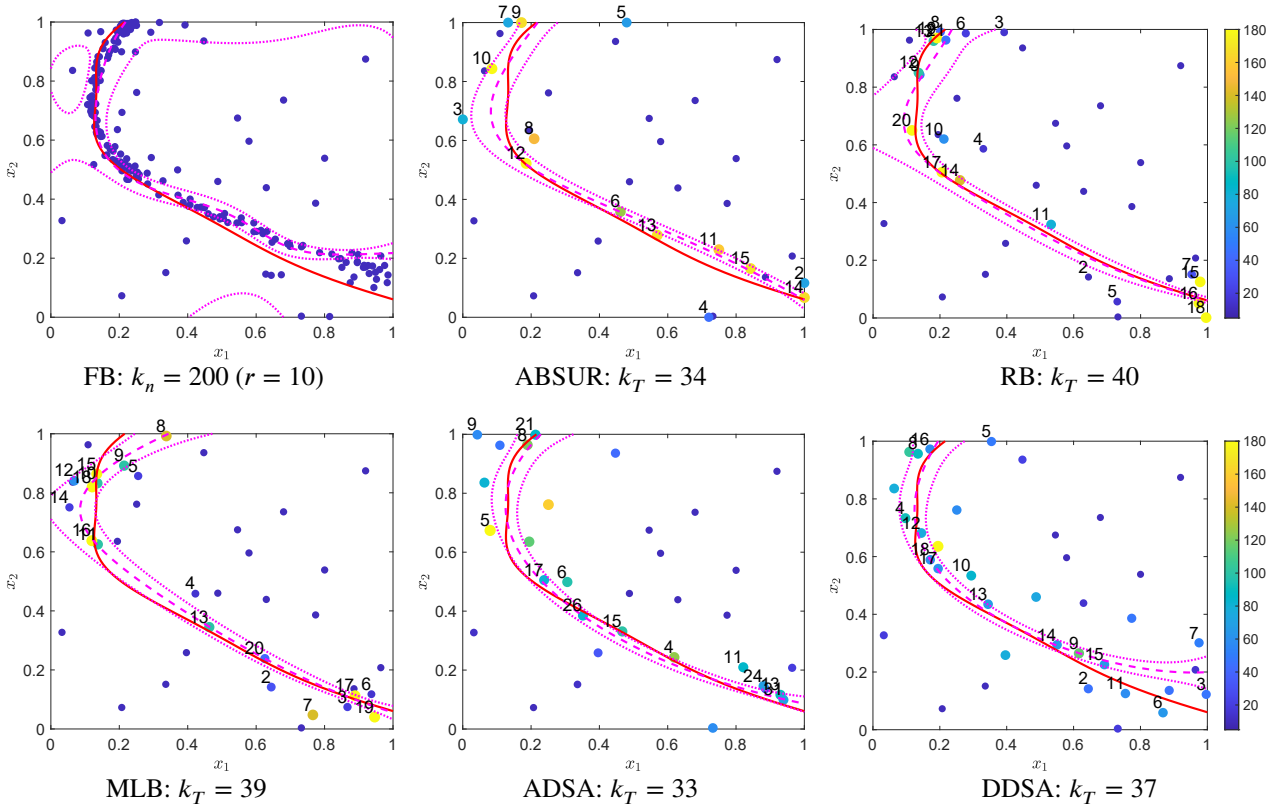


FIGURE 5 GP fits $f|\mathcal{A}_{k_T}$ and designs for 2-D case study with heteroskedastic noise. The dashed lines are the estimated posterior zero-contours $\{x : \hat{f}^{(N)}(x) = 0\}$ to be compared to the true contour (solid line). The dotted lines are the corresponding 95% credible intervals. The initial design (same across all panels) are the blue unlabelled dots. The labels indicate the order of the inputs $\bar{x}_i, i = 1, \ldots k_n$ and the respective color/size are proportional to the replication level $r^{(n)}$. Design sizes k_T vary across the schemes.

6 | APPLICATION TO OPTIMAL STOPPING

As a fourth and final case study, we consider an application of contour finding for determining the optimal exercise policy of a Bermudan financial derivative [24]. The underlying simulator is based on a d-dimensional geometric Brownian motion

TABLE 4 Parameters for the 2-D Basket Put Option and 3-D Max Call Option.

	2-D Basket Put	3-D Max-Call
Option	$\mathcal{K} = 40, \Delta t = 0.04, T = 1$	$\mathcal{K} = 100, \Delta t = 1/3, T = 3$
Parameters	$r = 0.06, \sigma = 0.2, X_0 = [40, 40]$	$r = 0.05, \sigma = 0.2, X_0 = [90, 90, 90]$
Budget	$N_T = 2000$, $k_0 = 20, r_0 = 20$	$N_T = 30,000$, $k_0 = 300, r_0 = 30$
FB	$r = 20$	$r = 30$
MLB/RB	$\mathbf{r}_L = \{20, 30, 40, 50, 60, 80, 120, 160\}$	$\mathbf{r}_L = \{20, 30, 40, 50, 80, 160, 240, 320, 480, 640\}$
ABSUR	$\mathcal{R} = [20, 160], T_{sim} = 0.01$	$\mathcal{R} = [20, 640], T_{sim} = 0.01$
ADSA	$c_{bt} = 10$	$c_{bt} = 6.67$

$(\boldsymbol{Z}_t) = (z_t^1, \dots, z_t^d)$ that represents prices of d assets and follows the log-normal dynamics

$$\boldsymbol{Z}_{t+\Delta t} = \boldsymbol{Z}_t \exp\left((r - \frac{1}{2}\operatorname{diag}\boldsymbol{\Xi})\Delta t + \sqrt{\Delta t}\boldsymbol{\Xi}\Delta \boldsymbol{W}_t\right), \tag{23}$$

where r is the interest rate, $\boldsymbol{\Xi}$ is the $d \times d$ covariance matrix and $\Delta \boldsymbol{W}_t \sim \mathcal{N}(0, \boldsymbol{I}_d)$ are the Gaussian stochastic stocks. Let $h(t, z)$ be the option payoff from exercising when $\boldsymbol{Z}_t = \boldsymbol{z}$. We assume that exercising is allowed every Δt time units, up to the option maturity T. The overall goal is to determine the stopping regions $\{S_t : t = \Delta t, 2\Delta t, \dots, T - \Delta t\}$ to maximize $\mathbb{E}[h(\tau, \boldsymbol{Z}_\tau)]$, where $\tau = \min\{t : \boldsymbol{Z}_t \in S_t\}$ is the exercise strategy. The dynamic programming principle implies that S_t can be recursively computed as the zero level set of the timing function $z \mapsto f(t, \boldsymbol{z}) = h(t, \boldsymbol{z}) - \mathbb{E}[h(\tau_t, \boldsymbol{Z}_{\tau_t})]$ where the latter term is the continuation value based on the exercise strategy from the forward-looking $\{S_s, s > t\}$. Numerically, this yields a simulator of $f(t, \boldsymbol{z})$ through pathwise reward over one-step-ahead simulations of $\boldsymbol{Z}_{t+\Delta t}$.

In this setting, the underlying distribution of $\boldsymbol{Z}_t$ at time t is log-normal since $\log \boldsymbol{Z}_t$ is multivariate normal. To reflect this fact which dictates the importance of correctly identifying whether $x \in S_t$ or not (since option exercising decisions are made *along* trajectories of $\boldsymbol{Z}$, conditional on the given initial value $\boldsymbol{Z}_0 = z_0$), we employ log-normal weights $\mu(dz) = p_{\boldsymbol{Z}_t}(\cdot|\boldsymbol{z}_0)$ in (3). We further use μ to weigh the respective $\mathcal{I}_n$ criteria when optimizing for new inputs. In line with the problem context, we assess performance using the ultimate estimated option value. The latter is evaluated via an out-of-sample Monte Carlo simulation that averages realized payoffs along a database of $M' = 10^5$ forward paths $\boldsymbol{z}_{0:T}^{1:M'}$:

$$\hat{V}(0, \boldsymbol{z}_0) = \frac{1}{M'}\sum_{m=1}^{M'} h(\tau_0^m, \boldsymbol{z}_{\tau_0^m}^{(m)}), \tag{24}$$

with $\tau_0^m := \min\{t : z_t^{(m)} \in \widehat{S}_t\} \wedge T$. Since our goal is to find the *best* exercise value, higher $\hat{V}$'s indicate a better approximation of $\{S_t\}$. To allow a direct comparison, we set parameters matching the test cases in [24] (cf. Table 4):

$$\text{2-D average Put option:} \qquad h_{Put}(t, \boldsymbol{z}) = e^{-rt}(\mathcal{K} - z^1 - z^2)_+;$$
$$\text{3-D Max-Call option:} \qquad h_{Call}(t, \boldsymbol{z}) = e^{-rt}(\max(z^1, z^2, z^3) - \mathcal{K})_+.$$

These settings have very low signal-to-noise ratio, and non-Gaussian heteroskedastic noise, so $N_T \gg 10^3$ is imperative. We use plain GP and t-GP metamodels (refitted every ten steps) with a constant noise variance τ^2 to model the timing function $f(t, \boldsymbol{z})$. All the adaptive algorithms combined with homoskedastic and heteroskedastic GP (t-GP) surrogates are publicly available as part of the `mlOSP` library in `R` [25].

Table 5 shows the performance of different designs/models. In the 2-D setting the best performing scheme is DDSA. We obtain savings of 80% in computation time compared to the baseline FB scheme. For the 3-D Max Call, DDSA achieves the highest payoff, and at a fraction ($\sim 1/20$th) of time. RB and MLB lead to slightly smaller payoff than DDSA, but with a saving of 60% in computation cost. ADSA leads to basically the same payoff as DDSA and takes approximately twice as much time compared with DDSA. ABSUR takes half the time of ADSA, leading to a lower payoff. In both 2-D and 3-D settings, ADSA and DDSA lead to a higher payoff and have a more stable performance than the other adaptive batch designs. In terms of design size k_T, ABSUR yields the largest k_T, while DDSA yields the most compact designs.

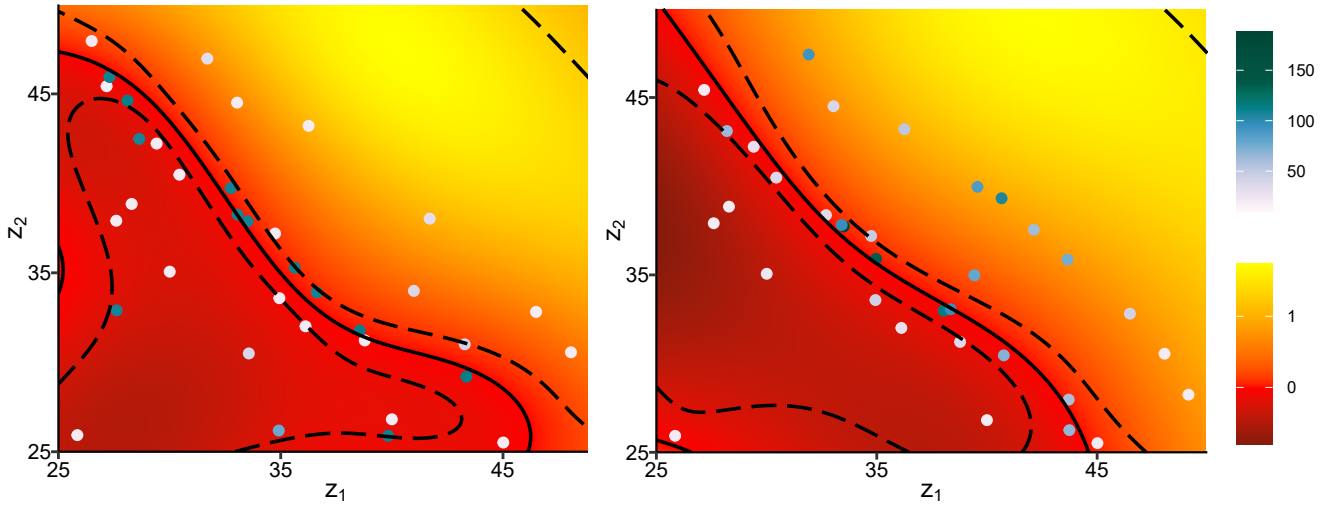
TABLE 5 Performance of GP metamodels with FB, MLB, RB, ABSUR, ADSA and DDSA designs in the 2-D Average Put and 3-D Max Call examples. Results are averages from 20 runs of each scheme.

DESIGN	MODEL	PAYOFF	TIME/S T	INPUTS k_T
2-D AVERAGE PUT				
FB	GP	1.451± 0.002	29.82	100.00
RB	GP	1.443± 0.004	5.42	35.85
MLB	GP	**1.440± 0.004**	**4.92**	33.97
ABSUR	GP	1.446± 0.004	11.40	53.80
ADSA	GP	1.445 ± 0.003	11.76	32.87
DDSA	GP	1.445 ± 0.003	5.42	34.00
FB	t-GP	1.449 ± 0.002	63.11	100.00
RB	t-GP	1.445 ± 0.004	11.36	36.39
MLB	t-GP	**1.443 ± 0.004**	**10.52**	35.35
ABSUR	t-GP	1.443 ± 0.004	26.13	49.79
ADSA	t-GP	1.447 ± 0.003	19.00	44.83
DDSA	t-GP	1.446 ± 0.003	11.31	34.00
3-D MAX CALL				
FB	GP	11.26 ± 0.01	2239.10	1000.00
RB	GP	**11.23 ± 0.01**	**37.42**	342.39
MLB	GP	11.24 ± 0.01	38.17	342.07
ABSUR	GP	11.23 ± 0.01	109.81	407.90
ADSA	GP	11.25 ± 0.01	194.05	460.33
DDSA	GP	11.26 ± 0.01	94.58	381.00

Figure 6 shows the GP fits $\hat{f}(t, \boldsymbol{z})$ for ABSUR and ADSA for the 2-D Put case study at $t = 0.6$. The desired zero-level contour goes from NW to SE and due to the chosen setting should be symmetric about the $z^1 = z^2$ line. We see that both strategies select inputs around the contour; consistent with the results shown in Figure 5 , ABSUR is somewhat more exploratory and yields wider credible intervals for the exercise boundary $\{\hat{f}^{(k_T)} = 0\}$ in regions close to the edge of the input space, especially at the NW and SE corners. ABSUR uses slightly more design sites $k_T(ABSUR) = 40 > k_T(ADSA) = 37$ and has a flatter distribution of replication counts. In contrast, ADSA uses up to $\max_n r^{(n)} = 188$ replicates. We also observe that several initial designs repeatedly receive more replications (up to 50 counts) in ADSA.

7 | CONCLUSION

We have proposed and investigated five different schemes for adaptive batching in metamodeling of stochastic experiments. All schemes explicitly address the shifting exploration-exploitation trade-off by capturing the intuition of increasingly beneficial replication as sequential design is constructed. Our presentation focused on the plain Gaussian Process paradigm but as shown are straightforwardly extended to alternatives, such as t-GP and `hetGP`. The key step is to construct an approximation of the batch look-ahead variance $s^{(n+1)}(x, r)$. Our results demonstrate that adaptive batching offers a simple mechanism to extract significant computational gains through building more compact designs and taking advantage of the symbiotic relationship between GPs and replication. Thus, compared with using a constant value for replicates r over all inputs like in FB, we are able to gain more than an order-of-magnitude speed-up with minimal loss of metamodeling fidelity. Among the proposed adaptive batching schemes, we advocate the use of ADSA and DDSA (the latter being essentially a faster heuristic). While they lead to similar results in lower dimensional experiments, ADSA is observed to be more accurate in complex settings, such as higher dimension or low signal-to-noise ratio.



ABSUR: $k_T = 40$ ADSA: $k_T = 37$

FIGURE 6 GP fits $f^{(k_T)}(t, \cdot)$ and designs $\mathcal{A}$ for 2-D average put option example at $t = 0.6$ and $N_T = 2000$. *Left panel:* ABSUR; *right:* ADSA. The solid lines are the estimated exercise boundary $\hat{f}^{(k_T)}(t, \boldsymbol{z}) = 0$ and the dashed lines are the corresponding 95% credible intervals. The scatter plot is the design $\mathcal{A}_{k_T}$ color-coded by replicate counts $r_i, i = 1, \ldots, k_T$.

Our focus has been on adaptive batching in the context of level-set estimation. Related problems such as evaluating the probability of failure, or evaluating a tail risk measure, would benefit from the same ideas and will be investigated in follow-up projects. Another extension is to tackle ϵ-softened optimization, i.e., target the region of ϵ-optimal inputs for a given $\epsilon > 0$. Such objective might be desirable to practitioners who simultaneously optimize over several (potentially non-quantitative) factors. This entails replacing the zero level set with $f(x) = 0$ with $f(x) = M_n$ where M_n is an estimator for $\max_x f(x)|\mathcal{A}_n$. For instance, one could obtain M_n similar to the computation of the Expected Improvement criterion in Bayesian Optimization. Another important problem that is beyond the scope of the present work is theoretical analysis about the asymptotic complexity of the proposed schemes such as ADSA, for example to establish the long-run growth rate of k_n in order to quantify the asymptotic complexity of the GP metamodel as $N_n \to \infty$.

Acknowledgements. We thank the anonymous reviewers for their helpful comments that helped to improve on earlier versionss of the manuscript; we are also grateful to Mickael Binois for useful discussions and help in porting our algorithms from MATLAB to R. Both authors were partially supported by NSF DMS-1521743. ML is additionally supported by NSF DMS-1821240.

□

APPENDIX

A ALLOCATION RULE

Proof of Proposition 4.1. Because the unique inputs are unchanged during the allocation step, comparing $\mathbf{C}^{(n+1)} = \mathbf{K}(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{K}_*(\boldsymbol{\Sigma}^{(n+1)})^{-1}\mathbf{K}_*^T$ to $\mathbf{C}^{(n)} = \mathbf{K}(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{K}_*(\boldsymbol{\Sigma}^{(n)})^{-1}\mathbf{K}_*^T$, the only term that changes is $\boldsymbol{\Sigma}^{(n+1)}$. Minimizing eq. (16) therefore reduces to maximizing

$$(\boldsymbol{\omega}^{(n)})^T \mathbf{K}_*(\mathbf{K} + \tau^2 \mathbf{R}^{(n+1)})^{-1}\mathbf{K}_*^T \boldsymbol{\omega}^{(n)} \tag{A1}$$

Decompose $\Delta\mathbf{R}^{(n)} =: \mathbf{B}^{(n)}\mathbf{B}^{(n)}$. Using the Woodbury Identity,

$$(\boldsymbol{\Sigma}^{(n+1)})^{-1} = (\mathbf{K} + \tau^2(\mathbf{R}^{(n)} - \Delta\mathbf{R}^{(n)}))^{-1} \simeq (\boldsymbol{\Sigma}^{(n)})^{-1} + \tau^2(\boldsymbol{\Sigma}^{(n)})^{-1}\Delta\mathbf{R}^{(n)}(\boldsymbol{\Sigma}^{(n)})^{-1}, \tag{A2}$$

where the last expression is obtained by dropping the term $\mathbf{B}^{(n)}[\mathbf{K}+\tau^2\mathbf{R}^{(n)}]^{-1}\mathbf{B}^{(n)} \simeq \mathbf{0}$ due to $\max_i \Delta\mathbf{R}_{ii}^{(n)} \ll 1$. Therefore, maximizing (A1) subject to $\sum_{i=1}^{k_n} \Delta r_i^{(n)} = \Delta r^{(n)}$ is equivalent to maximizing

$$\tilde{\mathcal{I}}_{SAO}(\Delta\mathbf{R}) = \tau^2 \cdot (\boldsymbol{\omega}^{(n)})^T \mathbf{K}_*(\boldsymbol{\Sigma}^{(n)})^{-1}\Delta\mathbf{R}^{(n)}(\boldsymbol{\Sigma}^{(n)})^{-1}\mathbf{K}_*^T\boldsymbol{\omega}^{(n)} + \lambda\left(\Delta r^{(n)} - \sum_{i=1}^{k_n}\Delta r_i^{(n)}\right), \tag{A3}$$

where λ is a Lagrange multiplier. The first-order optimality conditions are

$$\frac{\partial\tilde{\mathcal{I}}_{SAO}}{\partial\Delta r_i^{(n)}} = -\frac{\tau^2 \cdot (\boldsymbol{\omega}^{(n)})^T \mathbf{K}_*(\boldsymbol{\Sigma}^{(n)})^{-1}(\boldsymbol{\Sigma}^{(n)})^{-1}\mathbf{K}_*^T\boldsymbol{\omega}^{(n)}}{(r_i^{(n)}+\Delta r_i^{(n)})^2} - \lambda = 0 \tag{A4}$$

which leads to $r_i^{(n)} + \Delta r_i^{(n)} \propto [(\boldsymbol{\Sigma}^{(n)})^{-1}\mathbf{K}_*^T\boldsymbol{\omega}^{(n)}]_i$, $1 \le i \le k_n$ as in (18). □

Following Liu and Staum [23], we use a pegging procedure [10] to obtain integer-valued $\Delta r_i^{(n)}$, see Algorithm 5 in the Appendix. Note that due to the rounding, the added number of replicates $\sum_{i=1}^{k_n}\Delta r_i^{(n)}$ is not exactly $\Delta r^{(n)}$. Moreover, there are several approximations in Proposition 4.1 that render $\Delta r_i^{(n)}$ and (17) suboptimal: (1) we assume that $\max_{i=1,\dots,k_n}\Delta\mathbf{R}_{ii}^{(n)} \ll 1$; (2) we freeze the weights in (16) rather than using $\boldsymbol{\omega}^{(n+1)}$; (3) we round off to integer $\Delta r_i^{(n)}$.

Remark 3. Similar results about minimizing the look-ahead GP variance of a linear combination $\boldsymbol{\omega}^T\mathbf{f}$ appear in [1, 11, 23, 26]. Relative to Ankenman et al. [1] and Chen and Zhou [11], we get rid of all integrals, making (17) computationally efficient. The algorithm proposed by Ludkovski and Risk [26] relied on in-sample test set $\mathbf{x}_* = \bar{\mathbf{x}}_{1:k_n}$ while our test set is different from the existing inputs.

Proposition 4.1 can be extended to the heteroskedastic setting by replacing the constant value τ^2 in equations (A1), (A2), (A3) and (A4) by a diagonal matrix $\mathbf{S}$ where $\mathbf{S}_{ii} = \tau^2(x_i)$, $1 \le i \le k_n$. Solving eq. (A4) leads to $r_i^{(n)} + \Delta r_i^{(n)} \propto \tau^2(x_i)\mathbf{U}_i^{(n)}$, $1 \le i \le k_n$.

B PEGGING ALGORITHM FOR ADSA

Algorithm 5 Pegging Algorithm

Input: $I_0 = \{1,\dots,k_n\}$, $r = \sum_{i=1}^{k_n} r_i^{(n)}$, $\mathbf{U}^{(n)}$ from eq. (18)
$j \leftarrow 0$.
for all $i \in I_j$ **do**
  $\Delta r_i^{(n)} \leftarrow \frac{\mathbf{U}_i^{(n)}}{\sum_{j=1}^{k_n}\mathbf{U}_j^{(n)}} \times r - r_i^{(n)}$
  if $\Delta r_i^{(n)} \ge 0$ for all $i \in I_j$ **then**
    break
  else
    $I_{j+1} \leftarrow \{i \in I_j : \Delta r_i^{(n)} > 0\}$
    $\Delta r_i^{(n)} = 0$ for $i \notin I_{j+1}$
    $r \leftarrow r - \sum_{i\in I_j, i\notin I_{j+1}} r_i^{(n)}$
    $j \leftarrow j+1$
  end if
end for
Round all $\Delta r_i^{(n)}$, $i = 1,..,k_n$ to the nearest integer.
(If $\sum_{i=1}^{k_n}\Delta r_i^{(n)} = 0$, round $\max_{i=1}^{k_n}\Delta r_i^{(n)}$ up to the next integer)

C GP WITH STUDENT T-NOISE

The marginal likelihood of $\bar{\mathbf{y}}_{1:k_n}$ with a t-GP is (with $\mathbf{f} := f_{1:k_n} = (f(\mathbf{x}_1), \dots, f(\mathbf{x}_{k_n}))$)

$$p_{t\mathrm{GP}}\left(\bar{\mathbf{y}}_{1:k_n} \middle| \bar{\mathbf{x}}_{1:k_n}, \mathbf{r}^{(n)}_{1:k_n}, \mathbf{f}\right) = \prod_{i=1}^{k_n} \frac{\Gamma((\nu+1)/2)\sqrt{r_i^{(n)}}}{\Gamma(\nu/2)\sqrt{\nu\pi}\tau} \left(1 + \frac{r_i^{(n)}(y_i - f_i)^2}{\nu\tau^2}\right)^{-(\nu+1)/2}, \tag{C5}$$

where $\Gamma(\cdot)$ is the incomplete Gamma function. To integrate (C5) against the Gaussian prior $p(f|\boldsymbol{\vartheta})$ we use Laplace approximation [37]. Specifically, we use a second-order Taylor expansion of the log-likelihood around its mode, $\tilde{\mathbf{f}}^{(n)}_{t\mathrm{GP}} := \arg\max_{\mathbf{f}} p_{t\mathrm{GP}}(\mathbf{f}|\bar{\mathbf{x}}_{1:k_n}, \bar{\mathbf{y}}_{1:k_n})$, to obtain a Gaussian approximation to the posterior $f(x_*)|\mathcal{A}_n \sim \mathcal{N}(\hat{f}^{(n)}_{t\mathrm{GP}}(x_*), s^{(n)}_{t\mathrm{GP}}(x_*)^2)$ with

$$\hat{f}^{(n)}_{t\mathrm{GP}}(x_*) = \mathbf{k}(x_*)\mathbf{K}^{-1}\tilde{\mathbf{f}}^{(n)}_{t\mathrm{GP}}, \tag{C6}$$

$$\begin{aligned} v^{(n)}_{t\mathrm{GP}}(x_*, x'_*) &= K(x_*, x'_*) - \mathbf{k}(x_*)\left(\mathbf{K} + (\mathbf{W}^{(n)}_{t\mathrm{GP}})^{-1}\right)^{-1}\mathbf{k}(x'_*), \\ &= K(x_*, x'_*) - \mathbf{k}(x_*)(\boldsymbol{\Sigma}^{(n)}_{t\mathrm{GP}})^{-1}\mathbf{k}(x'_*) \end{aligned} \tag{C7}$$

where $\mathbf{W}^{(n)}_{t\mathrm{GP}}$ is diagonal with

$$W^{(n)}_{t\mathrm{GP},ii} = -\nabla^2 \log p_{t\mathrm{GP}}(\bar{y}_i|\tilde{f}^{(n)}_i, \bar{x}_i) = (\nu+1)\frac{\nu\frac{\tau^2}{r_i^{(n)}} - (\bar{y}_i - \tilde{f}^{(n)}_i)^2}{(\nu\frac{\tau^2}{r_i^{(n)}} + (\bar{y}_i - \tilde{f}^{(n)}_i)^2)^2}, \tag{C8}$$

since the likelihood factorizes over observations. Note that ν is treated as part of the GP hyperparameters and fitted via MLE.

Lyu et al. [27] then calculated the approximate step-ahead variance of t-GP:

$$s^{(n+1)}_{t\mathrm{GP}}(x_{k_n+1}, r^{(n)}_{k_n+1})^2 \simeq s^{(n)}_{t\mathrm{GP}}(x_{k_n+1})^2 \cdot \frac{\frac{\tau^2}{r^{(n)}_{k_n+1}}\frac{\nu+1}{\nu-1}}{\frac{\tau^2}{r^{(n)}_{k_n+1}}\frac{\nu+1}{\nu-1} + s^{(n)}_{t\mathrm{GP}}(x_{k_n+1})^2}. \tag{C9}$$

We replace Eq. (9) with (C9) to obtain the acquisition functions for t-GP.

Allocation Rule for t-GP: To implement ADSA and DDSA for t-GP we need (i) the analogue of Proposition 4.1 for the allocation rule $\Delta\mathbf{r}^{(n)}_{1:k_n}$ over the existing inputs $\bar{\mathbf{x}}_{1:k_n}$; (ii) the look-ahead variance $s^{(n+1),new}(x_*)$ conditional on adding a new input; (iii) look-ahead variance $s^{(n+1),all}(x_*)$ conditional on allocating $\Delta\mathbf{r}^{(n)}_{1:k_n}$. For all these tasks, the non-Gaussian likelihood (C5) underlying t-GP calls for further approximations provided in the following three Lemmas.

Lemma 1 (Allocation Rule)**.** The allocation $\Delta\mathbf{r}^{(n)}_{1:k_n}$ is like in Proposition 4.1 but relies on

$$\widetilde{\mathbf{U}}^{(n)}_{t\mathrm{GP}} = (\widetilde{\boldsymbol{\Sigma}}^{(n)}_{t\mathrm{GP}})^{-1}\mathbf{K}_*^T\boldsymbol{\omega}^{(n)}, \qquad \text{with} \quad \widetilde{\boldsymbol{\Sigma}}^{(n)}_{t\mathrm{GP}} := \left(\mathbf{K} + \frac{\nu+1}{\nu-1}\tau^2\mathbf{R}^{(n)}\right). \tag{C10}$$

Proof of Lemma 1. For t-GP, the noise matrix $\tau^2\mathbf{R}^{(n)}$ in eq. (5) is replaced with $(\boldsymbol{W}^{(n)}_{t\mathrm{GP}})^{-1}$. To calculate the ADSA/DDSA allocation rule with a t-GP metamodel we substitute $(\bar{y}_i - \tilde{f}^{(n)}_{t\mathrm{GP}}(\bar{x}_i))^2 \simeq \frac{\tau^2}{r_i^{(n)}}$ and $\tilde{f}^{(n)}_{t\mathrm{GP}}(\bar{x}_i) \simeq \tilde{f}^{(n+1)}_{t\mathrm{GP}}(\bar{x}_i)$ in eq. (C8) to obtain (cf. Lyu et al. 27)

$$\begin{aligned} W^{(n)}_{ii} &= (\nu+1)\frac{\nu\frac{\tau^2}{r_i^{(n)}} - (\bar{y}_i - \tilde{f}^{(n)}_i)^2}{(\nu\frac{\tau^2}{r_i^{(n)}} + (\bar{y}_i - \tilde{f}^{(n)}_i)^2)^2} \\ &\simeq (\nu+1)\frac{\nu\frac{\tau^2}{r_i^{(n)}} - \frac{\tau^2}{r_i^{(n)}}}{\left(\frac{\tau^2}{r_i^{(n)}} + \nu\frac{\tau^2}{r_i^{(n)}}\right)^2} = \frac{(\nu-1)r_i^{(n)}}{(\nu+1)\tau^2} := \widetilde{W}^{(n)}_{ii}. \end{aligned}$$

Hence, $(\boldsymbol{W}_{t\mathrm{GP}}^{(n)})^{-1} \simeq (\widetilde{\boldsymbol{W}}_{t\mathrm{GP}}^{(n)})^{-1} = \frac{\nu+1}{\nu-1}\tau^2\mathbf{R}^{(n)}$ and the covariance matrix $\mathbf{C}_{t\mathrm{GP}}^{(n)}$ of $f(\mathbf{x}_*)$ is approximated as

$$\begin{aligned}\mathbf{C}_{t\mathrm{GP}}^{(n)} &= \mathbf{K}(\mathbf{x}_*,\mathbf{x}_*) - \mathbf{K}_*\left(\mathbf{K} + (\mathbf{W}_{t\mathrm{GP}}^{(n)})^{-1}\right)^{-1}\mathbf{K}_*^T \\ &\simeq \mathbf{k}(\bar{\mathbf{x}}_*,\bar{\mathbf{x}}_*) - \mathbf{k}_*\left(\mathbf{K} + \frac{\nu+1}{\nu-1}\tau^2\mathbf{R}^{(n)}\right)^{-1}\mathbf{k}_*^T \\ &\simeq \mathbf{K}(\bar{\mathbf{x}}_*,\bar{\mathbf{x}}_*) - \mathbf{K}_*(\widetilde{\boldsymbol{\Sigma}}_{t\mathrm{GP}}^{(n)})^{-1}\mathbf{K}_*^T, \end{aligned} \tag{C11}$$

where $\widetilde{\boldsymbol{\Sigma}}_{t\mathrm{GP}}^{(n)}$ matches eq. (C10). The rest of the proof proceeds exactly like for the regular GP model in Proposition 4.1, after boosting τ^2 up by a constant ratio to $(\nu+1)/(\nu-1)\tau^2$. Then we obtain $\widetilde{\boldsymbol{U}}_{t\mathrm{GP}}^{(n)}$ as defined in (C10). □

Next, we need to approximate the next-step $\boldsymbol{W}_{t\mathrm{GP}}^{(n+1)}$. Unlike in the Gaussian case where $\boldsymbol{\Sigma}^{(n+1)}$ depends only on $\mathbf{R}^{(n+1)}$, for t-GP $\boldsymbol{W}_{t\mathrm{GP}}^{(n+1)}$ depends on $\bar{\mathbf{y}}_{1:k_n}$ (because it depends on $\tilde{\mathbf{f}}_{t\mathrm{GP}}$). We therefore need an approximation $\widehat{\boldsymbol{W}}_{t\mathrm{GP}}^{(n+1)}$ (the notation is to emphasize that it is different from the previous approximation $\widetilde{\boldsymbol{W}}_{t\mathrm{GP}}^{(n)}$ to $\boldsymbol{W}_{t\mathrm{GP}}^{(n)}$).

Lemma 2 (Look-Ahead t-GP Variance)**.** The look-ahead variance at x_* conditional on allocating $\Delta r^{(n)}$ simulations to a new input $\bar{x}_{k_n+1}$ is approximately given by

$$\tilde{s}_{t\mathrm{GP}}^{(n+1),new}(x_*)^2 \simeq s_{t\mathrm{GP}}^{(n)}(x_*)^2 - \frac{v_{t\mathrm{GP}}^{(n)}(x_*,\bar{x}_{k_n+1})^2}{\frac{(\nu+1)\tau^2}{(\nu-1)\Delta r^{(n)}} + s_{t\mathrm{GP}}^{(n)}(\bar{x}_{k_n+1})^2}. \tag{C12}$$

Finally, to obtain $\mathcal{I}_{SAO}^{(n),all}$ we define

$$\widehat{W}_{ii}^{(n+1)} := (\nu+1)\frac{\nu\frac{\tau^2}{r_i^{(n+1)}} - (\bar{y}_i^{(n)} - \tilde{f}_{t\mathrm{GP}}^{(n)}(\bar{x}_i))^2}{\left((\bar{y}_i^{(n)} - \tilde{f}_{t\mathrm{GP}}^{(n)}(\bar{x}_i))^2 + \nu\frac{\tau^2}{r_i^{(n+1)}}\right)^2}, \tag{C13}$$

based on the approximation $(\bar{y}_i^{(n+1)} - \tilde{f}_{t\mathrm{GP}}^{(n+1)}(x_i))^2 \simeq (\bar{y}_i^{(n)} - \tilde{f}_{t\mathrm{GP}}^{(n)}(x_i))^2$. This yields

Lemma 3 (Look-ahead t-GP variance after batch allocation)**.**

$$\tilde{s}_{t\mathrm{GP}}^{(n+1),all}(x_*) \simeq K(x_*,x_*) - \mathbf{K}_*\left(\mathbf{K} + (\widehat{\mathbf{W}}_{t\mathrm{GP}}^{(n+1)})^{-1}\right)^{-1}\mathbf{K}_*^T. \tag{C14}$$

D TUNING PARAMETERS FOR ABSUR AND ADSA

References

[1] Ankenman, B., B. L. Nelson, and J. Staum, 2010: Stochastic kriging for simulation metamodeling. *Operations research*, **58**, no. 2, 371–382.

[2] Azzimonti, D., J. Bect, C. Chevalier, and D. Ginsbourger, 2016: Quantifying uncertainties on excursion sets under a Gaussian random field prior. *SIAM/ASA Journal on Uncertainty Quantification*, **4**, no. 1, 850–874.

[3] Azzimonti, D. and D. Ginsbourger, 2018: Estimating orthant probabilities of high-dimensional Gaussian vectors with an application to set estimation. *Journal of Computational and Graphical Statistics*, **27**, no. 2, 255–267.

[4] Azzimonti, D., D. Ginsbourger, C. Chevalier, J. Bect, and Y. Richet, 2021: Adaptive design of experiments for conservative estimation of excursion sets. *Technometrics*, **63**, no. 1, 13–26.

[5] Bect, J., D. Ginsbourger, L. Li, V. Picheny, and E. Vazquez, 2012: Sequential design of computer experiments for the estimation of a probability of failure. *Statistics and Computing*, **22**, no. 3, 773–793.

[6] Bichon, B. J., M. S. Eldred, L. P. Swiler, S. Mahadevan, and J. M. McFarland, 2008: Efficient global reliability analysis for nonlinear implicit performance functions. *AIAA Journal*, **46**, no. 10, 2459–2468.

TABLE D1 Varying $\bar{r}$ (left panel) and T_{sim} (right panel) for ABSUR. We report the mean error rate $\mathcal{ER}_T$, running time t (in seconds) and the design size k_T for the 2-D synthetic case studies with Gaussian noise $\epsilon \sim \mathcal{N}(0, \tau^2)$ and budget $N_T = 2000$. All other hyperparameters are set as in Table 1 . Results are based on 20 macroreplications of each scheme.

$\bar{r}$	$\mathcal{ER}_T$	t	k_T	T_{sim}	$\mathcal{ER}_T$	t	k_T
$\tau^2 = 0.01$				$\tau^2 = 0.01$			
$0.01N_T$	0.21%	54.1	111.5	0.0001	2.16%	11.4	31.0
$0.025N_T$	0.24%	28.2	59.2	0.001	0.27%	12.5	31.9
$0.05N_T$	0.23%	20.9	43.5	0.01	0.30%	15.2	38.6
$0.1N_T$	0.30%	15.3	38.6	0.1	0.21%	23.6	60.4
$0.25N_T$	0.31%	13.7	36.0	1	0.19%	34.5	100.1
N_T	0.58%	9.0	30.1	10	0.23%	31.6	115.1
$\tau^2 = 0.25$				$\tau^2 = 0.25$			
$0.01N_T$	1.26%	48.0	110.9	0.0001	1.45%	9.6	30.1
$0.025N_T$	1.31%	22.0	57.6	0.001	1.44%	9.0	30.4
$0.05N_T$	1.18%	13.5	40.9	0.01	1.29%	10.1	34.7
$0.1N_T$	1.29%	9.7	34.7	0.1	1.38%	16.8	53.8
$0.25N_T$	1.41%	9.9	33.1	1	1.29%	31.6	97.7
N_T	1.64%	8.2	29.8	10	1.30%	37.2	128.6
$\tau^2 = 1$				$\tau^2 = 1$			
$0.01N_T$	2.05%	46.1	110.8	0.0001	2.27%	8.4	30.0
$0.025N_T$	2.01%	21.1	57.5	0.001	2.46%	8.8	30.4
$0.05N_T$	1.78%	12.4	40.8	0.01	1.93%	9.5	34.3
$0.1N_T$	1.93%	9.7	34.3	0.1	1.89%	16.5	53.9
$0.25N_T$	2.03%	9.2	32.9	1	1.98%	31.5	100.6
N_T	2.24%	9.2	30.8	10	2.10%	44.3	141.9

[7] Binois, M. and R. B. Gramacy, 2021: hetGP: Heteroskedastic Gaussian process modeling and sequential design in R. *Journal of Statistical Software*, **98**, no. 13, 1–44, doi:10.18637/jss.v098.i13.
URL https://www.jstatsoft.org/v098/i13

[8] Binois, M., J. Huang, R. B. Gramacy, and M. Ludkovski, 2019: Replication or exploration? Sequential design for stochastic simulation experiments. *Technometrics*, **61**, no. 1, 7–23.

[9] Bolin, D. and F. Lindgren, 2015: Excursion and contour uncertainty regions for latent Gaussian models. *Journal of the Royal Statistical Society: Series B: Statistical Methodology*, 85–106.

[10] Bretthauer, K. M., A. Ross, and B. Shetty, 1999: Nonlinear integer programming for optimal allocation in stratified sampling. *European Journal of Operational Research*, **116**, no. 3, 667–680.

[11] Chen, X. and Q. Zhou, 2017: Sequential design strategies for mean response surface metamodeling via stochastic kriging with adaptive exploration and exploitation. *European Journal of Operational Research*, **262**, no. 2, 575–585.

[12] Chevalier, C., J. Bect, D. Ginsbourger, E. Vazquez, V. Picheny, and Y. Richet, 2014: Fast parallel kriging-based stepwise uncertainty reduction with application to the identification of an excursion set. *Technometrics*, **56**, no. 4, 455–465.

[13] Chevalier, C., D. Ginsbourger, J. Bect, and I. Molchanov, 2013: Estimating and quantifying uncertainties on level sets using the Vorob'ev expectation and deviation with Gaussian process models. *mODa 10–Advances in Model-Oriented Design and Analysis*, Springer, 35–43.

TABLE D2 Mean error rate $\mathcal{ER}_T$, computation cost t (in seconds) and the design size k_T for ADSA and DDSA with variable c_{bt} for the 2-D synthetic case studies with Gaussian noise and budget $N_T = 2000$. All other hyperparameters are the same as in Table 1 . Results are based on 20 macroreplications of each scheme.

	ADSA			DDSA		
			$\tau^2 = 0.01$			
c_{bt}	$\mathcal{ER}_T$	t	k_T	$\mathcal{ER}_T$	t	k_T
0.5	0.54%	204.2	25.4	0.21%	139.0	226
1	0.67%	125.3	23.4	0.23%	58.0	133
2.5	0.57%	62.8	23.9	0.20%	24.1	73
5	0.72%	37.2	23.1	0.20%	13.8	51
10	0.83%	22.2	22.4	0.25%	7.6	37
20	1.03%	11.9	21.3	0.39%	4.1	30
40	1.07%	6.5	20.8	2.04%	2.3	25
80	1.42%	3.8	20.5	1.21%	1.3	23
			$\tau^2 = 0.25$			
c_{bt}	$\mathcal{ER}_T$	t	k_T	$\mathcal{ER}_T$	t	k_T
0.5	1.45%	211.9	29.6	1.20%	147.9	226
1	1.37%	125.7	26.3	1.21%	64.5	133
2.5	1.50%	66.3	23.9	1.26%	25.5	73
5	1.38%	38.7	23.3	1.19%	13.5	51
10	1.41%	22.5	22.8	1.32%	7.5	37
20	1.48%	12.8	22.2	1.43%	4.4	30
40	1.71%	6.8	21.7	1.55%	2.4	25
80	1.76%	3.7	21.0	1.76%	1.4	23
			$\tau^2 = 1$			
c_{bt}	$\mathcal{ER}_T$	t	k_T	$\mathcal{ER}_T$	t	k_T
0.5	1.94%	358.8	256.0	1.70%	146.9	226
1	1.94%	172.0	134.0	1.80%	63.7	133
2.5	1.91%	76.0	69.0	1.89%	27.0	73
5	1.95%	42.8	45.9	1.90%	15.6	51
10	1.97%	24.2	33.2	1.99%	8.0	37
20	2.04%	13.3	27.3	2.26%	4.5	29
40	2.03%	7.0	24.2	2.71%	2.3	25
80	2.63%	4.0	22.3	3.13%	1.3	23

[14] Chevalier, C., D. Ginsbourger, and X. Emery, 2014: Corrected kriging update formulae for batch-sequential data assimilation. *Mathematics of Planet Earth*, Springer, 119–122.

[15] Echard, B., N. Gayton, and M. Lemaire, 2010: Kriging based Monte Carlo simulation to compute the probability of failure efficiently: AK-MCS method. *6emes Journées Nationales de Fiabilité, 24–26 mars, Toulouse, France*.

[16] Hu, R. and M. Ludkovski, 2017: Sequential design for ranking response surfaces. *SIAM/ASA Journal on Uncertainty Quantification*, **5**, no. 1, 212–239.

[17] Jalali, H., I. Van Nieuwenhuyse, and V. Picheny, 2017: Comparison of Kriging-based algorithms for simulation optimization with heterogeneous noise. *European Journal of Operational Research*, **261**, no. 1, 279–301.

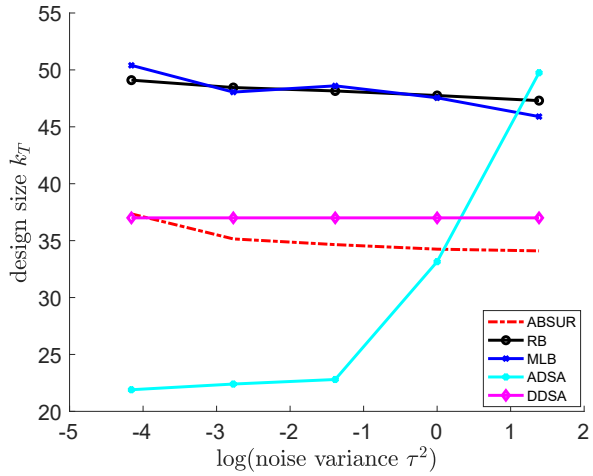


FIGURE D1 Design size k_T as a function of noise variance τ^2 at $\tau^2 = \{4^{-3}, 4^{-2}, 4^{-1}, 1, 4\}$ in the 2-D experiment with $\epsilon \sim \mathcal{N}(0, \tau^2)$ and budget $N_T = 2000$. Hyperparameters are set the same as in Table 1 .

[18] Jones, D. R., M. Schonlau, and W. J. Welch, 1998: Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, **13**, no. 4, 455–492.

[19] Kandasamy, K., G. Dasarathy, J. B. Oliva, J. Schneider, and B. Póczos, 2016: Gaussian process bandit optimisation with multi-fidelity evaluations. *Advances in Neural Information Processing Systems*, 992–1000.

[20] Klein, A., S. Falkner, S. Bartels, P. Hennig, and F. Hutter, 2017: Fast Bayesian optimization of machine learning hyperparameters on large datasets. *Artificial Intelligence and Statistics*, PMLR, 528–536.

[21] Koehler, J., A. Puhalskii, and B. Simon, 1998: Estimating functions evaluated by simulation: A Bayesian-analytic approach. *Annals of Applied Probability*, **8**, no. 4, 1184–1215.

[22] Le Gratiet, L. and J. Garnier, 2015: Asymptotic analysis of the learning curve for Gaussian process regression. *Machine Learning*, **98**, no. 3, 407–433.

[23] Liu, M. and J. Staum, 2010: Stochastic kriging for efficient nested simulation of expected shortfall. *Journal of Risk*, **12**, no. 3, 3.

[24] Ludkovski, M., 2018: Kriging metamodels and experimental design for Bermudan option pricing. *Journal of Computational Finance*, **22**, no. 1, 37–77.

[25] — 2020: mlOSP: Towards a unified implementation of regression Monte Carlo algorithms. *arXiv preprint arXiv:2012.00729*.

[26] Ludkovski, M. and J. Risk, 2018: Sequential design and spatial modeling for portfolio tail risk measurement. *SIAM Journal on Financial Mathematics*, **9**, no. 4, 1137–1174.

[27] Lyu, X., M. Binois, and M. Ludkovski, 2021: Evaluating Gaussian process metamodels and sequential designs for noisy level set estimation. *Statistics and Computing*, **31**, no. 4, 1–21.

[28] McLeod, M., M. A. Osborne, and S. J. Roberts, 2017: Practical Bayesian optimization for variable cost objectives. *arXiv preprint arXiv:1703.04335*.

[29] Picheny, V., D. Ginsbourger, O. Roustant, R. T. Haftka, and N.-H. Kim, 2010: Adaptive designs of experiments for accurate approximation of a target region. *Journal of Mechanical Design*, **132**, no. 7, 071008.

[30] Poloczek, M., J. Wang, and P. Frazier, 2017: Multi-information source optimization. *Advances in Neural Information Processing Systems*, 4288–4298.

Nomenclature

n Sequential design step, indexes most quantities below	$\hat{f}(\cdot)$ Posterior mean
$\mathcal{A}$ Design set	$v(\cdot)$ Posterior variance
D Input space	$s(\cdot)$ Posterior standard deviation
d Dimension of input space	$\mathcal{I}(\cdot)$ Acquisition function
$Y(\cdot)$ Response	ρ cUCB weight
X Design	$\mu(\cdot)$ Lebesgue measure
k Number of unique inputs	E Local empirical error
N Total budget	γ Standard deviation threshold
r Replicate count	η Reduction factor
$\bar{x}$ Design location	L Number of replication levels
τ Noise variance	c_{ovh} Optimization overhead
f Latent function	T_{sim} Computation time
S Level set	$\mathcal{L}$ Look-ahead integrated contour uncertainty
ϵ Noise	ω Level set contour weights
$\bar{y}$ Average response	c_{bt} Batch factor
$\mathcal{ER}$ Error rate	l Length-scale
$K(\cdot,\cdot)$ Covariance function	$\sigma_{\mathbf{se}}$ Function variance
	M Test set size

[31] Ranjan, P., D. Bingham, and G. Michailidis, 2008: Sequential experiment design for contour estimation from complex computer codes. *Technometrics*, **50**, no. 4, 527–541.

[32] Santner, T. J., W. I. Notz, and B. J. Williams, 2003: *The Design and Analysis of Computer Experiments*. Springer.

[33] Srinivas, N., A. Krause, S. M. Kakade, and M. W. Seeger, 2012: Information-theoretic regret bounds for Gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, **58**, no. 5, 3250–3265.

[34] Stroh, R., S. Demeyer, N. Fischer, J. Bect, and E. Vazquez, 2017: Sequential design of experiments to estimate a probability of exceeding a threshold in a multi-fidelity stochastic simulator. *61th World Statistics Congress of the International Statistical Institute (ISI 2017)*.

[35] Swersky, K., J. Snoek, and R. P. Adams, 2013: Multi-task Bayesian optimization. *Advances in neural information processing systems*, 2004–2012.

[36] Vanhatalo, J., J. Riihimäki, J. Hartikainen, P. Jylänki, V. Tolvanen, and A. Vehtari, 2013: GPstuff: Bayesian modeling with Gaussian processes. *Journal of Machine Learning Research*, **14**, no. Apr, 1175–1179.

[37] Williams, C. K. and D. Barber, 1998: Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20**, no. 12, 1342–1351.