LEARNING-ACCELERATED ALGORITHMS FOR SIMULATION AND OPTIMIZATION

BY

CHENCHAO SHOU

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Mechanical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

ABSTRACT

Simulation and optimization are fundamental building blocks for many computational methods in science and engineering. In this work, we explore the use of machine learning techniques to accelerate compute-intensive tasks in both simulation and optimization. Specifically, two algorithms are developed: (1) a variance reduction algorithm for Monte Carlo simulations of mean-field particle systems, and (2) a global optimization algorithm for noisy expensive functions.

For the variance reduction algorithm, we develop an adaptive-control-variates technique for a class of simulations, where many particles interact via common mean fields. Due to the presence of a large number of particles and highly nonlinear dynamics, simulating these mean-field particle models is often time-consuming. Our algorithm treats the body of particles in the system as a source of training data, then uses machine learning to automatically build a model for the underlying particle dynamics, and finally constructs control variates with the learned model. We prove that the mean estimators from our algorithm are unbiased. More importantly, we show that, for a system with sufficiently many particles, our algorithm asymptotically produces more efficient estimators than naive Monte Carlo under certain regularity conditions. We applied our variance reduction algorithm to an aerosol particle simulation and found that the resulting simulation is about 7 times faster.

The second algorithm is a parallel surrogate optimization algorithm, known as ProSRS, for noisy expensive black-box functions. Within this algorithm, we develop an efficient weighted-radial-basis regression procedure for constructing the surrogates. Furthermore, we introduce a novel tree-based technique, called the “zoom strategy”, to further improve optimization efficiency. We prove that if ProSRS is run for sufficiently long, with probability converging to one there will be at least one sample among all the evaluations that will be arbitrarily close to the global minimum. We compared ProSRS to several state-of-the-art Bayesian optimization algorithms on a suite of standard benchmark functions and two real machine-learning hyperparameter-tuning problems. We found that our algorithm not only achieves significantly faster optimization convergence, but is also orders of magnitude cheaper in computational cost. We also applied ProSRS to the problem of characterizing and validating a complex aerosol model against experimental measurements, where twelve simulation parameters must be optimized. This case illustrates the use of ProSRS for general global optimization problems.

To my parents, for their love, support and encouragement.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Professor Matthew West for his patient guidance and constant support throughout my PhD study. Without many insightful suggestions from him, my research would simply not achieve what has been achieved so far. I would like to thank my committee Professor Rayadurgam Srikant, Professor Prashant Mehta, Professor Alexandra Chronopoulou and Professor Niao He for their helpful feedback on my work. I also want to thank Professor Nicole Riemer and her PhD student Jeffrey Curtis for offering help from time to time. Finally I want to acknowledge funding under DOE DE-SC0006689 and NSF CMMI-1150490, as well as support from Blue Waters. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

TABLE OF CONTENTS

LIST OF TABLES viii

LIST OF FIGURES ix

LIST OF ABBREVIATIONS xiii

CHAPTER 1 INTRODUCTION 1
1.1 Motivation 1
1.2 Outline 3

CHAPTER 2 VARIANCE REDUCTION FOR MEAN-FIELD PARTICLE SIMULATIONS 4
2.1 Introduction 4
2.2 Problem definition 6
2.3 Algorithm 8
2.4 Analytical results 13
2.5 Numerical study 1: 1D demonstration 24
2.6 Numerical study 2: a linear Gaussian example 28
2.7 Numerical study 3: aerosol particle simulation 32
2.8 Conclusions 38

CHAPTER 3 PARALLEL SURROGATE OPTIMIZATION FOR NOISY EXPENSIVE FUNCTIONS 40
3.1 Introduction 40
3.2 ProSRS algorithm 44
3.3 Convergence 52
3.4 Numerical results 56
3.5 Application of ProSRS to a general problem 67
3.6 Conclusions 75

CHAPTER 4 CONCLUSIONS 76

APPENDIX A ASSUMPTION VALIDATION FOR NUMERICAL STUDY 1 77

APPENDIX B MEAN FIELD CONVERGENCE FOR AEROSOL PARTICLE MODEL 80

APPENDIX C ALGORITHMS IN NUMERICAL STUDY 2 87

APPENDIX D COMPUTATION OF CONTROL VARIATES MEAN FOR NUMERICAL STUDY 2 89

APPENDIX E OPTIMIZATION BENCHMARK FUNCTIONS . . . 91

REFERENCES 96

LIST OF TABLES

3.1 Parameter values for the ProSRS algorithm 57
3.2 Experiment conditions for optimization benchmark problems (the last numeric figure in the function name is the problem dimension) 57
3.3 Unknown parameters in PartMC simulation 68
3.4 Optimization results 72

LIST OF FIGURES

2.1 A schematic diagram of adaptive control variates algorithm (Alg. 1). 9
2.2 Convergence of regression coefficients for the least squares method, showing the mean square error between regression coefficients from the least squares and the coefficients of the projection for time $t = 5$. 26
2.3 Performance of a cubic polynomial regression model for time $t = 5$ with $R_1 = 100$. The cubic model (red) trained with the least square method gives good prediction for the particle property (green) in the evaluation phase. 26
2.4 Variance reduction performance of adaptive control variates algorithm for the one-dimensional mean-field system in Eq. 2.47. The number of sample paths in the learning phase is $R_1 = 100$. The number of independent learning phases R_3 (see Eq. 2.10) for estimating variance reduction is 20. 27
2.5 Variance reduction performances for the 1D linear Gaussian example with different number of particles. 31
2.6 Variance reduction performance for the 1D linear Gaussian example when $R_1 = 100$ and $N = 100$. 31
2.7 Variance reduction performance of our adaptive control variates algorithm for the mean estimation of the two bulk properties: total mass concentration (blue) and scattering coefficient (red). The number of independent learning phases R_3 (see Eq. 2.10) for estimating variance reduction is 20. 37

2.8 Statistical error and computation time of our adaptive control variates algorithm and naive Monte Carlo for the mean estimation of two bulk properties as the number of sample paths in the evaluation phase varies from 10 to 500. The error is measured by the L_2 norm of the standard deviation of estimators at different time steps. Comparing to naive Monte Carlo (green), the estimator error of our algorithm (red) is reduced by $\sim 67\%$ for fixed computation time, and the required computation time is reduced by $\sim 85\%$ for the same precision. 38

3.1 A schematic diagram of a general master-worker framework for a parallel surrogate optimization algorithm. In each iteration, the algorithm (master) constructs a surrogate based on the available evaluations, proposes multiple points based on the surrogate, and distributes these points to different processes (workers) for parallel evaluations. The evaluated points are then fed back into the loop to update the surrogate in the next iteration. 43

3.2 Illustration of the zoom strategy on a 1-D parabola. The red curve shows the surrogate fit to all the noisy evaluations (green dots) of the objective function (black curve). The blue curve shows the surrogate fit using only the local evaluation data in the zoomed-in domain. The local fit is likely to agree well with the global fit on the restricted domain, and is much cheaper to construct. 45

3.3 Illustration of the tree structure of ProSRS algorithm on a 2-D problem. The black box on the left shows the domain of a root node. The two red boxes and one blue box show two children and one grandchild of the root node. 46

3.4 Optimization curves for the benchmark functions. The error bar shows the standard deviation of 20 independent runs. 60

3.5 Optimization curves for the hyperparameter-tuning problems. The error bar shows the standard deviation of 20 independent runs. The number of parallel cores is 8 for both problems. The expected error (objective F) is estimated by averaging 5 independent samples. 61

3.6 Compare the modeling capability of RBF regression as used in ProSRS (dark blue lines) and GP regression with kernels: Matern1.5, Matern2.5 and RBF (green, red and light blue lines respectively) on 12 optimization benchmark functions. The y axis is the relative error in terms of the L_2 norm of the difference between a model g and the underlying true function $\mathbb{E}[f]$ over the function domain. The error bar shows the standard deviation of 10 independent runs. 62
3.7 Computational costs of different algorithms for the twelve optimization benchmark problems. The plots show the mean and standard deviation of 20 independent runs. The x axis is the number of iterations in actual optimization excluding the initial DOE iteration. The y axis is the actual time that was consumed by an algorithm in each iteration, and does not include the time of parallel function evaluations. 65
3.8 Computational costs of different algorithms for the two hyperparameter tuning problems. The plots show the mean and standard deviation of 20 independent runs. The x axis is the number of iterations in actual optimization excluding the initial DOE iteration. For different algorithms, the y axis is the actual time that was consumed by the algorithm in each iteration, and does not include the time of parallel function evaluations. The time for training and evaluating the machine learning models is shown in black. 66
3.9 Optimization efficiency of different algorithms on the two hyperparameter-tuning problems. Total time on the horizontal axis is the actual elapsed time including both algorithm running time and time of evaluating expensive functions. The shaded areas show the standard deviation of 20 independent runs. 66
3.10 A general workflow for solving a noisy expensive optimization problem with ProSRS algorithm. 70
3.11 Optimization curve for the PartMC characterization problem. The y axis is the noisy function value of the ProSRS algorithm output. 71
3.12 Computational costs of ProSRS algorithms (blue curve) and the evaluations of error function ϵ (black curve). The x axis is the number of iterations in the optimization, excluding the initial DOE iteration. 72
3.13 The outputs of the optimized PartMC model (with the optimal parameters in Table 3.4) versus SMPS measurements. The shaded area shows the standard deviation of 10 independent runs. 73

3.14 The outputs of the optimized PartMC model (with the optimal parameters in Table 3.4) versus SP2 measurements. The shaded area shows the standard deviation of 10 independent runs. 74

E.1 Surface plots for benchmark functions Ackley, Alpine, Griewank, Levy, SumPower, and SixHumpCamel. 94
E.2 Surface plots for benchmark functions Schaffer, Dropwave, Goldstein-Price and Rastrigin. 95

LIST OF ABBREVIATIONS

MD	Molecular dynamics
PartMC	Particle-resolved Monte Carlo
MARS	Multivariate adaptive regression splines
ProSRS	Progressive Stochastic Response Surface
SRS	Stochastic Response Surface
RBF	Radial basis function
DOE	Design of experiments
GP	Gaussian process
LP	Local penalization
EI	Expected improvement
UCB	Upper confidence bound
LCB	Lower confidence bound
MCMC	Markov Chain Monte Carlo
SGD	Stochastic gradient descent
AS	Ammonium sulfate
RB	Regal black

CHAPTER 1

INTRODUCTION

1.1 Motivation

Many scientific and engineering applications are computationally intensive. Examples include cosmological simulations, molecular dynamics simulations, atmospheric climate simulations and deep learning. The time to complete these tasks can be anywhere from several hours to multiple days or weeks [1, 2].

Machine learning, a field of computer science, has recently gained increasing popularity in the community due to its success in many areas including natural language processing [3], image recognition [4], game playing [5] and anomaly detection [6]. From an abstract perspective, machine learning generally deals with a collection of methods that are able to automatically extract knowledge from data. The knowledge could be in the form of a mapping between two sets of variables, as is often the case in supervised learning, or may be some representation of the underlying structure of the data as in unsupervised learning.

In this work, we explore the use of machine learning for accelerating compute-intensive applications. The idea of using machine learning to accelerate numerical computation is not new, and has been exploited by numerous researchers in various fields. For example, Gao and Kitchin [7] constructed a neural network potential energy function to improve time-consuming molecular dynamics (MD) simulations. The traditional MD simulations are based on density-functional theory, and can be run with only short time scales and a small scale of atoms due to the high computational cost. With the use of machine learning, they showed that MD simulations can be accelerated by orders of magnitude without sacrificing accuracy, thus enabling simulations at much larger scales. Hughes et al. [8] trained gradient-boosted regression trees to predict the global distribution of the aerosol mixing state index. The conventional method of accurately capturing global distributions of this quantity involves running large-scale detailed aerosol models, the cost of which is often prohibitively high. Silver et al. [5] developed a chess-playing program called AlphaGo using value and policy neural networks. The deep neural networks were trained via supervised learning from human expert moves and by reinforcement learning from self-play. The trained neural networks significantly accelerates Monte Carlo tree search, making AlphaGo the first program to defeat a world champion of Go. Rasp et al. [9] used deep learning for climate model parameterization, resulting in a significantly faster simulation compared to a cloud-resolving model and producing more accurate results than the traditional subgrid-processes method.

Machine learning has also been used to reduce the computational cost to meet stringent requirements of time-critical applications. For example, Ladický et al. [10] formulated physics-based fluid simulation as a regression problem and trained a regression forest to approximate the behaviors of particles obtained with a traditional solver. The resultant data-driven simulation is highly efficient, and can be used for high-quality real-time animation and computer graphics. Tompson et al. [11] also accelerated fluid simulations via machine learning but with convolutional networks. Chen et al. [12] used basis function methods to efficiently solve harmonic Maxwells equation for real-time controls and uncertainty quantification.

In this dissertation, we develop two specific learning-accelerated algorithms: (1) a variance reduction algorithm for Monte Carlo simulations of mean-field particle systems, and (2) a global optimization algorithm for noisy expensive black-box functions. In terms of methodology, both algorithms exploit one big idea: the expense of a task allows us to embed more "wisdom" into the computation. Because the task is computationally expensive, we can invest some extra time, which is a negligible fraction of the task, to make a better decision. This extra investment in the context here involves the training and the evaluation of machine learning models. Both algorithms then utilize the learned knowledge to achieve an efficiency improvement.

1.2 Outline

The remainder of this dissertation is organized as follows. The two algorithms will be presented separately in Chapter 2 and Chapter 3. Each Chapter will have its own introduction, algorithm and numerical result section. We will conclude in Chapter 4.

CHAPTER 2

VARIANCE REDUCTION FOR MEAN-FIELD PARTICLE SIMULATIONS

2.1 Introduction

Direct simulation of a large population of interacting particles or agents has been shown to be an effective and powerful way to understand complex physical and biological phenomena such as the draping behavior of woven cloth [13], the mixing state of soot particles [14] and biological aggregation behavior [15]. Compared to a continuum model, a particle-based model, in many cases, can yield more accurate results [14, 16–18]. Because the particles are modeled explicitly, a particle model can also reveal some features of the system that would be otherwise hidden in a continuum model.

The outputs of these particle simulations are often stochastic in nature due to random inputs and random interactions. To precisely measure an output, the statistical error of the output needs to be appropriately handled. Indeed, the statistical error can often be the dominant source of error for a particle simulation [19]. Monte Carlo methods are typically used to reduce these statistical errors. The most crude form of Monte Carlo, known as naive Monte Carlo, reduces the error by taking the average of samples from many independent sample paths. The statistical error of naive Monte Carlo decreases at the rate of $R^{-\frac{1}{2}}$, where R is the number of sample paths. Because particle models are often very expensive to run due to a large number of particles and highly complex dynamics, one would desire to draw a minimum number of sample paths to achieve the desired precision.

Variance reduction techniques are very useful for reducing the number of sample paths needed. Specifically, they reduce the variance of the output for a given number of sample paths, and in this way the efficiency of Monte Carlo simulation is improved. Some popular variance reduction techniques are importance sampling, control variates, antithetic sampling and stratified sampling [20, Chap. 9]. Among these, the method of control variates is one of the most effective techniques [21, Chap. 4]. The control variates method reduces the variance by constructing a new statistic (with known mean) that strongly correlates with the output. This technique has been successfully applied to a number of problems in machine learning. For instance, in reinforcement learning the efficiency of a policy gradient estimator can be enhanced with the use of control variates [22, Sec. 13.4]. In this case, the control variate is constructed from a baseline function that depends only on the state (not the action). A good heuristic for the baseline is the value function, which leads to a policy gradient estimator using the advantage function. Because the control variate depends on a baseline function, this variance reduction technique is commonly known as the "baseline shift" method in reinforcement learning. As another related example, Ranganath et al. [23] have used control variates to improve the efficiency of variational inference in machine learning.

The effectiveness of the control variates method largely depends on the quality of the constructed statistic. However, for a general problem it is often not obvious how to construct a good statistic, especially when the dynamics of the system is highly complex.

The method of adaptive control variates [24] is a way to deal with such difficulty. It works by first parameterizing the statistics with some unknown parameters, and then learning these parameters adaptively from the data. Adaptive control variates has been previously successfully applied to many financial problems. Ehrlichman and Henderson [24] applied adaptive control variates to a multivariate American option pricing problem, in which significant variance reduction was demonstrated even in a high-dimensional setting. Henderson and Simon [25] developed simulation schemes using adaptive control variates and showed that under certain conditions, the convergence rate of the estimator could be exponential, faster than the canonical rate of Monte Carlo simulation. Kim and Henderson [26] introduced two adaptive procedures for general parametrization of control variates and demonstrated their effectiveness on the problem of barrier options. A comprehensive review of adaptive control variates is available in [27].

In this work, we develop a new adaptive control variates algorithm for a class of particle simulations, in which many particles interact via common mean fields. Previous works [28–30] have demonstrated the use of the antithetic sampling technique to reduce the variance of particle simulations. However, their works mainly focused on the particle systems where the dynamics is dominated by the change in number of particles, whereas in this work we focus on the cases where the dynamics is dominated by particle transformations. It is worth mentioning that although our method is different from that in the previous works, the two approaches are complementary and can be applied simultaneously to the same particle systems.

Compared to the existing adaptive control variates methods [24–26], our algorithm proposes a novel scheme for learning the parameters in the control variates. Conventionally the training data is collected from Monte Carlo simulations with only one training sample per sample path. Because the particle system is often very high-dimensional (due to a large number of particles), one may need to draw many sample paths in order to accumulate enough data for training. As the simulations are usually expensive, the conventional adaptive control variates method could lead to prohibitive overhead. Our new method, in contrast, does not formulate the learning problem with respect to the system observable directly. Rather, we treat each individual particle as one training sample. In this way, the amount of the training data per sample path is significantly increased and the dimension of the learning problem is greatly reduced, thereby making it feasible to apply the control variates method to particle simulations.

2.2 Problem definition

We consider a class of discrete-time stochastic processes in which a large number of particles interact via common mean fields. Here we allow the mean fields to be dependent on the full history of the states. Specifically,

$$\begin{aligned} x_{t+1,i} &= f_t(\phi_0, \phi_1, \ldots, \phi_t, X_{t,i}), \\ \phi_t &= l_t(X_{t,1}, X_{t,2}, \ldots, X_{t,N}), \\ X_{t,i} &= \left[x_{0,i}^T, x_{1,i}^T, \ldots, x_{t,i}^T\right]^T, \quad i = 1, 2, \ldots, N, \quad t \in \mathbb{N}, \end{aligned} \tag{2.1}$$

where $x_{t,i}$ is a random vector representing the state of particle i at time t, and vector ϕ_t denotes random mean fields of the system at time t with the property that as the number of particles N approaches infinity, the mean fields ϕ_t converge to some deterministic constants. The random vector $X_{t,i}$ summarizes all the states of particle i up to time t. The mean fields are dependent on the states of all the particles, and are symmetric about particles (i.e., function l_t in Eq. 2.1 is symmetric about its N arguments). The initial particles $x_{0,i}$ are i.i.d. sampled from some probability distribution.

Equation 2.1 describes the dynamics of a particle system, in which particles are not only self-driven but also subject to common interactions. Such processes have appeared in a wide array of applications, including computer communication systems [31], mean field games [32], ferromagnetics [33] and aerosol condensation [34]. For the condensation process, as a concrete example, Eq. 2.1 can be used to describe the changes of the sizes of aerosol particles due to their interactions with water vapor. In this case, $x_{t,i}$ is the diameter of an aerosol particle and the mean field ϕ_t is water vapor concentration. The set of equations specifically describes the following condensation process: as water vapor condenses onto an aerosol particle, the diameter of this particle increases due to added liquid water. Each aerosol particle in the atmosphere takes away some amount of water vapor through condensation so that water vapor in the atmosphere becomes less, leading to a decrease in vapor concentration.

In this work, we are interested in the expectation of the observable in the form of:

$$y_t = \frac{1}{N}\sum_{i=1}^{N} g(x_{t,i}), \quad t = 1, 2, \ldots, T, \tag{2.2}$$

where the observable y_t is some bulk property of the particle system, and function g represents some property of an individual particle.

The mean estimator of naive Monte Carlo is given by

$$\hat{y}_t^{\text{naive}} = \frac{1}{R}\sum_{r=1}^{R}\left[\frac{1}{N}\sum_{i=1}^{N} g(x_{t,i}^{(r)})\right], \tag{2.3}$$

where the superscript '(r)' denotes an i.i.d. sample from the r^{th} sample path, and R is the total number of sample paths for naive Monte Carlo.

The objective of this work is to design an algorithm that would produce a more efficient mean estimator than the naive one in Eq. 2.3.

2.3 Algorithm

2.3.1 Algorithm description

Algorithm 1 Adaptive control variates

```
    **Phase 1 (learning phase):**
1:  Generate $R_1$ sample paths $\{x_{t,i}^{(r)} \mid r = 1, 2, \ldots, R_1\}$ from the process in Eq. 2.1
2:  Solve regression $\beta_t = \operatorname{argmin}_{\beta \in \Theta} \sum_{r=1}^{R_1} \sum_{i=1}^{N} \left( g(x_{t,i}^{(r)}) - h(x_{0,i}^{(r)}; \beta) \right)^2$ for $t = 1, 2, \ldots, T$
3:  Compute $E_t = \mathbb{E}[h(x_{0,1}^{(1)}; \beta_t) \mid \beta_t]$ for $t = 1, 2, \ldots, T$
    **Phase 2 (evaluation phase):**
4:  Generate another $R_2$ sample paths $\{x_{t,i}^{(r)} \mid r = 1, 2, \ldots, R_2\}$
5:  Construct estimator $\hat{y}_t^{\mathrm{cv}} = \frac{1}{R_2} \sum_{r=1}^{R_2} \left[ \frac{1}{N} \sum_{i=1}^{N} \left( g(x_{t,i}^{(r)}) - h(x_{0,i}^{(r)}; \beta_t) \right) \right] + E_t$ for $t = 1, 2, \ldots, T$
```

Our adaptive control variates algorithm is presented in Alg. 1. It consists of two phases. The first phase (Lines 1–3) is a learning phase, where the control variates strategy is learned from the simulation data. Line 2 builds a model for the particle property $g(x_{t,i})$ in terms of the initial state of the particle $x_{0,i}$. Here we formulate it as a least-squares fitting problem, where $h(\bullet; \beta)$ is a generic representation of parametric regression with β being regression coefficients in some space Θ. However, it should be noted that the least squares formulation here is only for notational convenience. The algorithm does not impose any constraint on the way a model is constructed. In fact, the model can be built with any machine learning algorithm, including non-parametric regression and regression forests. With this broad understanding, $h(\bullet; \beta_t)$ in Lines 3 and 5 should be thought as evaluating a model rather than necessarily substituting some known coefficients β_t into the expression $h(\bullet; \beta)$. The training data of the model is all the particles from different sample paths in the learning phase. Line 3 computes the conditional expectation given the learned model. We refer to this conditional expectation as the *mean of control variates*. Since the initial particles are identically distributed, the expectation does not depend on the choice of a particle. For simplicity, here we choose the first particle from the first sample path.

The second phase (Lines 4 and 5) is an evaluation phase, where we apply

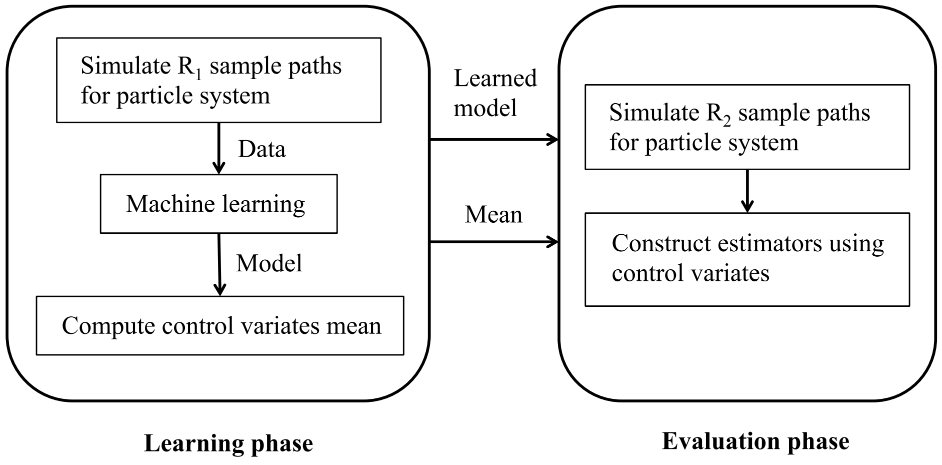
Simulate R_1 sample paths for particle system
Data
Machine learning
Model
Compute control variates mean
Learned model
Mean
Simulate R_2 sample paths for particle system
Construct estimators using control variates
Learning phase
Evaluation phase

Figure 2.1: A schematic diagram of adaptive control variates algorithm (Alg. 1).

the learned strategy to the mean estimation of the system observable. The sample paths drawn in this phase (Line 4) must be independent of those in the learning phase (Line 1). Line 5 constructs a control variates estimator using the model and the control variates mean obtained in the learning phase. Figure 2.1 shows a schematic diagram for the entire algorithm.

2.3.2 Remarks on the algorithm

Suppose the model (Line 2) is "perfect" so that $g(x_{t,i}^{(r)}) = h(x_{0,i}^{(r)}; \beta_t)$. Then the variance of the estimator $\hat{y}_t^{\text{cv}}$ is simply zero. Indeed, the variance reduction performance of Alg. 1 is largely determined by how well a model predicts the particle property $g(x_{t,i})$ based on its initial state $x_{0,i}$.

From the expression of the function f_t in Eq. 2.1, it is easy to see that the state of the particle i at time t (i.e., $x_{t,i}$) depends on its initial state $x_{0,i}$ and the mean fields $\phi_0, \phi_1, \ldots, \phi_{t-1}$. As a result, the particle property $g(x_{t,i})$ can be expressed as

$$g(x_{t,i}) = \Psi_t(\phi_0, \phi_1, \ldots, \phi_{t-1}, x_{0,i}). \tag{2.4}$$

Hence, the regression in Line 2 is essentially to find a model h that approximates the function Ψ_t:

$$\Psi_t(\phi_0, \phi_1, \ldots, \phi_{t-1}, x_{0,i}) \approx h(x_{0,i}; \beta_t). \tag{2.5}$$

Because the mean fields have low variance in the presence of a large number of particles, the function Ψ_t should be reasonably well approximated by some function that only depends on $x_{0,i}$. This is the main reason why we build a model that only depends on the initial state of the particle. Another reason is that the initial particles are sampled from some known distribution so that in most cases the mean of control variates (Line 3) can be computed cheaply and accurately without resorting to Monte Carlo methods.

In many applications, the dynamics of the particles is so complex that the function Ψ_t in Eq. 2.4 is almost intractable analytically. This intractability hinders the application of approximation methods that heavily rely on the closed-form expression of the function, such as derivative-based methods [35]. Our method of function approximation is data-driven, thus not requiring any prior knowledge about the system. The method essentially treats the particle dynamics as a black box, and uses machine learning to build an approximate model directly from the data.

Our algorithm has two phases: a learning phase and an evaluation phase. The reason for not having only one phase (i.e., not using the same data for learning and evaluation) is that the control variates mean is often very difficult to compute analytically in a one-phase setting as the coefficients of the learnt model are usually very complex functions of the random simulation data. By having two separate and independent phases, one can usually compute the control variates mean easily because the coefficients are treated as constants when computing the conditional expectation. Another way of viewing this is that if we were computing the control variates mean as the conditional expectation in Line 3 but with only one phase, then the resultant control variates estimator would be biased.

Due to the presence of the learning phase, there is some computational overhead associated with adaptive control variates algorithm compared to naive Monte Carlo. The hope is that the gain from variance reduction as a result of the constructed control variates outweighs the loss due to fewer sample paths to be generated because of the computational overhead. In general, there is no guarantee that the gain wins over the loss. However, there are several conducive factors that make this likely to happen for a system under investigation. The factors are:

(a) **Large number of particles in the system**. This gives a large amount of training data since our algorithm treats every particle as one training sample. Combining this large volume of data with the use of machine learning, it is likely to produce a good model for the dynamics without requiring too many sample paths for the learning phase.

(b) **Expensive simulations of particle dynamics**. In many applications, the complex dynamics of the particle system together with a large number of particles lead to long simulation times. As a result, the computational overhead of training regression models (Line 2) and computing the means of control variates (Line 3) can be negligible compared to the cost of generating sample paths in the evaluation phase.

(c) **Convergence property of the mean-field system**. The mean-field system often exhibits a certain form of convergence in the limit of a large number of particles [14, 36]. As a result, if we simulate the same particle system but with fewer particles, the dynamics of this reduced-size particle system would be similar to that of the original system as long as the number of particles is not too small. Therefore, in the learning phase we may generate sample paths from a reduced-size system instead of from the original system for the sake of minimizing the computational overhead. This technique is exploited in the second numerical example, and is demonstrated in Section 2.7.2.

2.3.3 Characterization of variance reduction

The variance reduction performance of the algorithm is measured by the standard variance reduction factor VR [24, 26]. It is defined as the ratio of the variance of the naive estimator divided by that of the control variates estimator given that the number of sample paths of the naive Monte Carlo is equal to that of the evaluation phase of Alg. 1:

$$\mathrm{VR}_t := \left. \frac{\mathrm{Var}\left(\hat{y}_t^{\text{naive}}\right)}{\mathrm{Var}\left(\hat{y}_t^{\text{cv}}\right)} \right|_{R=R_2} . \tag{2.6}$$

From Eq. 2.6, it is clear that the variance of the mean estimator is reduced if and only if VR is greater than one.

The variances of the naive estimator and the control variates estimator in Eq. 2.6 are generally unknown for a given problem. As a result, they need to estimated through random samples drawn from the simulations. For the naive estimator, the variance estimation is straightforward as we can apply the standard sample variance formula. To estimate the variance of the adaptive control variates estimator for our algorithm, we will use the following analytical result.

Lemma 2.1 *The variance of an adaptive control variates estimator is equal to the expectation of the variance given the learned model. That is*

$$Var\big(\hat{y}_t^{cv}\big) = \mathbb{E}[\,Var\big(\hat{y}_t^{cv} \mid \beta_t\big)].$$

Proof. By the law of total variance,

$$\mathrm{Var}\big(\hat{y}_t^{\mathrm{cv}}\big) = \mathbb{E}[\mathrm{Var}\big(\hat{y}_t^{\mathrm{cv}} \mid \beta_t\big)] + \mathrm{Var}\big(\mathbb{E}[\hat{y}_t^{\mathrm{cv}} \mid \beta_t]\big). \tag{2.7}$$

Note that

$$\mathbb{E}[\hat{y}_t^{\mathrm{cv}} \mid \beta_t] = \mathbb{E}\big[\frac{1}{N}\sum_{i=1}^{N} g(x_{t,i}^{(1)}) \mid \beta_t\big] + \mathbb{E}\big[-\frac{1}{N}\sum_{i=1}^{N} h(x_{0,i}^{(1)};\beta_t) + E_t \mid \beta_t\big]. \tag{2.8}$$

By Eq. 2.12, the second expectation on the right hand side of Eq. 2.8 is zero. Since the learning phase and the evaluation phase are independent, the first expectation on the right hand side of Eq. 2.8 is equal to $\mathbb{E}[y_t]$. As a result, $\mathbb{E}[\hat{y}_t^{\mathrm{cv}} \mid \beta_t] = E[y_t]$. Hence,

$$\mathrm{Var}\big(\mathbb{E}[\hat{y}_t^{\mathrm{cv}} \mid \beta_t]\big) = 0. \tag{2.9}$$

Substituting Eq. 2.9 into Eq. 2.7 yields the desired result. □

Based on Lemma 2.1, $\mathrm{Var}\big(\hat{y}_t^{\mathrm{cv}}\big)$ is estimated using the following procedure: (i) first run several independent learning phases, then (ii) given the model in each learning phase, estimate the variance of the control variates estimator using the samples in the corresponding evaluation phase, and finally (iii) take the average of all these variance estimators from different learning phases. Mathematically,

$$\widehat{\mathrm{Var}}\big(\hat{y}_t^{\mathrm{cv}}\big) = \frac{1}{R_3}\sum_{r=1}^{R_3} \widehat{\mathrm{Var}}\big(\hat{y}_t^{\mathrm{cv}} \mid \beta_t^{(r)}\big), \tag{2.10}$$

where R_3 is the number of independent learning phases. The expression $\widehat{\mathrm{Var}}\big(\hat{y}_t^{\mathrm{cv}} \mid \beta_t^{(r)}\big)$ denotes the estimated variance of the estimator given the model in the learning phase. Once the variance estimators for both algorithms are obtained, substituting them into Eq. 2.6 gives an estimator for the variance reduction factor.

2.4 Analytical results

We prove that the mean estimators from Alg. 1 are unbiased (Theorem 2.1). That is, the mean of an adaptive control variates estimator is identical to the true mean of the observable.

Theorem 2.1 *The mean estimators in Line 5 of Alg. 1 are unbiased. That is,* $\mathbb{E}[\hat{y}_t^{cv}] = \mathbb{E}[y_t]$.

Proof. Computing the expected control variates estimator gives

$$\begin{aligned}\mathbb{E}[\hat{y}_t^{\mathrm{cv}}] &= \mathbb{E}\big[\frac{1}{N}\sum_{i=1}^{N} g(x_{t,i}^{(1)})\big] + \mathbb{E}\big[-\frac{1}{N}\sum_{i=1}^{N} h(x_{0,i}^{(1)};\beta_t) + E_t\big] \\ &= \mathbb{E}[y_t] + \mathbb{E}\Big[\mathbb{E}\big[-\frac{1}{N}\sum_{i=1}^{N} h(x_{0,i}^{(1)};\beta_t) + E_t \mid \beta_t\big]\Big].\end{aligned} \tag{2.11}$$

Note that $\{h(x_{0,i}^{(1)};\beta_t) \mid i = 1, 2, \ldots, N\}$ are i.i.d. conditioned on the learned model because the learning phase and the evaluation phase are independent. Hence,

$$\mathbb{E}\big[-\frac{1}{N}\sum_{i=1}^{N} h(x_{0,i}^{(1)};\beta_t) + E_t \mid \beta_t\big] = -\mathbb{E}\big[h(x_{0,1}^{(1)};\beta_t) \mid \beta_t\big] + E_t = 0. \tag{2.12}$$

Substituting Eq. 2.12 into Eq. 2.11 yields the desired result. □

Next, we give the main result of our adaptive control variates algorithm (Theorem 2.2). Specifically, we prove that for the system with sufficiently many particles, our algorithm asymptotically produces more efficient estimators than naive Monte Carlo provided that some conditions are satisfied. This analytical result is quite general in that it does not assume any specific type of regression method.

We make several assumptions (Assumptions 2.1–2.4) for the main theorem. For the ease of conveying these assumptions, we first define two functions with respect to regression coefficients β:

$$V(\beta) \coloneqq \mathrm{Var}\big(g(x_{t,1}) - h(x_{0,1};\beta)\big), \tag{2.13}$$

$$C(\beta) \coloneqq \mathrm{Cov}\big(g(x_{t,2}), h(x_{0,1};\beta)\big), \tag{2.14}$$

where function g is defined in Eq. 2.2, representing the property of a particle, and function h is a generic form of regression (Line 2 of Alg. 1).

The first assumption (Assumption 2.1) is a mild regularity condition to ensure that we work with finite variances and covariances.

Assumption 2.1 (Moment regularity) *Assume random variables $g(x_{t,1})$ and $h(x_{0,1};\beta)$ have finite second moment*[1]*.*

The second assumption (Assumption 2.2) mainly assumes that the learned regression coefficients "stabilize" as more and more training data become available.

Assumption 2.2 (Regression convergence) *Assume that for sufficiently large N fixed,*

(i) Learned regression coefficients β_t (Line 2 of Alg. 1) converge in probability to a constant vector β_t' as the number of learning sample paths R_1 approaches infinity (i.e., $\beta_t \xrightarrow[R_1\to\infty]{p.} \beta_t'$).

(ii) The two functions defined in Eq. 2.13 and Eq. 2.14 converge in mean: $V(\beta_t) \xrightarrow[R_1\to\infty]{\mathcal{L}^1} V(\beta_t')$ and $C(\beta_t) \xrightarrow[R_1\to\infty]{\mathcal{L}^1} C(\beta_t')$.

As the number of particles becomes sufficiently large, the mean field ϕ_t is almost a constant vector so that function $g(x_{t,2})$ is almost a function that only depends on $x_{0,2}$ (see Eq. 2.4). Consequently, with some mild regularity conditions, it can be shown that the covariance $C(\beta_t')$ would converge to zero as $N \to \infty$ (recall that $x_{0,1}$ and $x_{0,2}$ are independent). Assumption 2.3

[1]Here we assume that $h(x_{0,1};\beta) \in \mathcal{L}^2$ for *any* fixed $\beta \in \mathbb{R}^m$, where m is the dimension of β. We also assume $g(x_{t,1}) \in \mathcal{L}^2$ for *all* time $t = 1, 2, \dots, T$. We use $\mathcal{L}^2$ to denote the space of the random variables that have finite second moment. In general, whenever there is a t-dependent variable in the assumption, it should be understood that it is assumed that the condition holds true "for all time $t = 1, 2, \dots, T$". This convention is used in the remaining assumptions as well as in Assumption 2.5–2.9.

essentially imposes conditions on this convergence. One sufficient condition for this assumption to be true is that the rate of convergence is faster than N^{-1} (in this case, lim inf is zero).

Assumption 2.3 (Asymptotic uncorrelation) *Assume* $\liminf_{N\to\infty} C(\beta'_t)N \geq 0$.

The limiting regression model $h(x_{0,1};\beta'_t)$ often achieves some form of optimality within the model space. As a result, it is reasonable to expect that the variance with this optimal model (i.e., $V(\beta'_t)$) is less than that without any model. The last assumption (Assumption 2.4) requires this variance difference to be bounded away from zero by some positive margin.

Assumption 2.4 (Variance optimality) *Assume* $\liminf_{N\to\infty}\Big(Var\big(g(x_{t,1})\big) - V(\beta'_t)\Big) > 0$.

Now we give our main theorem (Theorem 2.2) as follows.

Theorem 2.2 (Variance reduction theorem) *Suppose Assumptions 2.1–2.4 are satisified. Then there exists* $N_0 \in \mathbb{N}$ *such that for any system (Eq. 2.1) with the number of particles greater than* N_0*, the variance reduction factor (Eq. 2.6) is greater than one for all time* t *as long as the number of sample paths in the learning phase is sufficiently large.*

Proof. The goal is to show that there exists $N' \in \mathbb{N}$ such that for any $N > N'$, there exists $R' \in \mathbb{N}$ so that for any $R_1 > R'$ and given $R = R_2$,

$$\mathrm{Var}\big(\hat{y}_t^{\mathrm{cv}}\big) < \mathrm{Var}\big(\hat{y}_t^{\mathrm{naive}}\big), \quad \text{for all } t = 1, 2, \ldots, T. \tag{2.15}$$

Without loss of generality, for the remainder of the proof we assume $R = R_2 = 1$.

For the clarity of showing the dependence on different variables, we introduce the notation

$$\mathrm{Var}^{\mathrm{cv}}(N, R_1, t) := \mathrm{Var}\big(\hat{y}_t^{\mathrm{cv}} \mid \beta_t\big), \quad \mathrm{Var}^{\mathrm{naive}}(N, t) := \mathrm{Var}\big(\hat{y}_t^{\mathrm{naive}}\big).$$

By Lemma 2.1, $\mathrm{Var}\big(\hat{y}_t^{\mathrm{cv}}\big) = \mathbb{E}\big[\mathrm{Var}^{\mathrm{cv}}(N, R_1, t)\big]$. As a result, to show the statement in Eq. 2.15 it suffices to show that for any $t \in \{1, 2, \ldots, T\}$ fixed, there exists $N'' \in \mathbb{N}$ such that for any $N > N''$, there exists $R'' \in \mathbb{N}$ so that for all $R_1 > R''$,

$$\mathbb{E}\big[\mathrm{Var}^{\mathrm{cv}}(N, R_1, t)\big] < \mathrm{Var}^{\mathrm{naive}}(N, t). \tag{2.16}$$

Now we fix time t. Conditioned on the learned (random) model, E_t in Line 5 of Alg. 1 is a constant. Therefore, the variance of the control variates estimator given the learned model is

$$\mathrm{Var}^{\mathrm{cv}}(N, R_1, t) = \mathrm{Var}\Big(\frac{1}{N}\sum_{i=1}^{N}\big(g(x_{t,i}) - h(x_{0,i}; \beta_t)\big) \mid \beta_t\Big). \tag{2.17}$$

Since particles are initially i.i.d. and the system is symmetric under particle relabeling,

$$\begin{aligned}\mathrm{Var}^{\mathrm{naive}}(N, t) &= \mathrm{Var}\big(\frac{1}{N}\sum_{i=1}^{N} g(x_{t,i})\big)\\ &= \frac{1}{N}\mathrm{Var}\big(g(x_{t,1})\big) + \frac{N-1}{N}\mathrm{Cov}\big(g(x_{t,1}), g(x_{t,2})\big).\end{aligned} \tag{2.18}$$

Similarly,

$$\begin{aligned}\mathrm{Var}^{\mathrm{cv}}(N, R_1, t) = &\frac{1}{N}\mathrm{Var}\big(g(x_{t,1}) - h(x_{0,1}; \beta_t) \mid \beta_t\big)\\ &+ \frac{N-1}{N}\mathrm{Cov}\big(g(x_{t,1}) - h(x_{0,1}; \beta_t), g(x_{t,2}) - h(x_{0,2}; \beta_t) \mid \beta_t\big).\end{aligned} \tag{2.19}$$

Because (i) the learning phase and the evaluation phase are independent, (ii) the particles are symmetric, and (iii) initial particles are independent of each other conditioned on the learned model, we have

$$\begin{aligned}&\mathrm{Cov}\big(g(x_{t,1}) - h(x_{0,1}; \beta_t), g(x_{t,2}) - h(x_{0,2}; \beta_t) \mid \beta_t\big)\\ &\qquad = \mathrm{Cov}\big(g(x_{t,1}), g(x_{t,2})\big) - 2\mathrm{Cov}\big(g(x_{t,2}), h(x_{0,1}; \beta_t) \mid \beta_t\big).\end{aligned} \tag{2.20}$$

By Eqs. 2.18, 2.19 and 2.20, and using the functions defined in Eqs. 2.13 and

2.14, we have

$$\begin{aligned}
&\mathrm{Var}^{\mathrm{naive}}(N,t) - \mathrm{Var}^{\mathrm{cv}}(N,R_1,t)\\
&\qquad = \Big(\mathrm{Var}\big(g(x_{t,1})\big) - V(\beta_t) + 2(N-1)C(\beta_t)\Big)N^{-1}\\
&\qquad = \Big(\mathrm{Var}\big(g(x_{t,1})\big) - V(\beta_t') + 2(N-1)C(\beta_t')\Big)N^{-1}\\
&\qquad + \big(V(\beta_t') - V(\beta_t)\big)N^{-1} + 2(1-N^{-1})\big(C(\beta_t) - C(\beta_t')\big). \quad (2.21)
\end{aligned}$$

Taking expectations on both sides of Eq. 2.21, we get

$$\begin{aligned}
&\mathrm{Var}^{\mathrm{naive}}(N,t) - \mathbb{E}\big[\mathrm{Var}^{\mathrm{cv}}(N,R_1,t)\big]\\
&\qquad = \Big(\mathrm{Var}\big(g(x_{t,1})\big) - V(\beta_t') + 2(N-1)C(\beta_t')\Big)N^{-1}\\
&\qquad + \mathbb{E}\big[V(\beta_t') - V(\beta_t)\big]N^{-1} + 2(1-N^{-1})\mathbb{E}\big[C(\beta_t) - C(\beta_t')\big]. \quad (2.22)
\end{aligned}$$

Denote the positive lim inf[1] in Assumption 2.4 to be A. Then by definition of the lim inf, there exists $N_1 \in \mathbb{N}$ such that for all $N > N_1$,

$$\mathrm{Var}\big(g(x_{t,1})\big) - V(\beta_t') > \frac{A}{2}. \quad (2.23)$$

By Assumption 2.3 and using the definition of the lim inf, there exists $N_2 \in \mathbb{N}$ such that for all $N > N_2$,

$$2(N-1)C(\beta_t') = 2\big(NC(\beta_t')\big)\left(\frac{N-1}{N}\right) > 2\left(-\frac{A}{4}\right)\left(\frac{N-1}{N}\right) > -\frac{A}{2}. \quad (2.24)$$

Let $N_3 = \max(N_1, N_2)$. Then adding the inequalties in Eq. 2.23 and 2.24, we have for all $N > N_3$,

$$\Big(\mathrm{Var}\big(g(x_{t,1})\big) - V(\beta_t') + 2(N-1)C(\beta_t')\Big)N^{-1} > \left(\frac{A}{2} - \frac{A}{2}\right)N^{-1} = 0. \quad (2.25)$$

Now take N'' to be the maximum of N_3 and the minimum integer for the conditions in Assumption 2.2 to be satisfied, and fix N to be an arbitrary integer that is larger than N''. By $\mathcal{L}^1$ convergence in Assumption 2.2, we

[1]Here we only discuss the case when the lim inf is finite (i.e., $0 < A < \infty$). If $A = \infty$, the inequality in Eq. 2.25 trivially holds for sufficiently large N.

have

$$\mathbb{E}\big[V(\beta_t') - V(\beta_t)\big]N^{-1} + 2(1-N^{-1})\mathbb{E}\big[C(\beta_t) - C(\beta_t')\big] \xrightarrow{R_1\to\infty} 0. \tag{2.26}$$

By the definition of the limit, there exists $R'' \in \mathbb{N}$ such that for all $R_1 > R''$,

$$\begin{aligned}\mathbb{E}\big[V(\beta_t') - V(\beta_t)\big]N^{-1} + 2(1-N^{-1})\mathbb{E}\big[C(\beta_t) - C(\beta_t')\big] >\\ -\Big(\mathrm{Var}\big(g(x_{t,1})\big) - V(\beta_t') + 2(N-1)C(\beta_t')\Big)N^{-1},\end{aligned} \tag{2.27}$$

which implies $\mathrm{Var}^{\text{naive}}(N,t) > \mathbb{E}\big[\mathrm{Var}^{\text{cv}}(N,R_1,t)\big]$ (this is clear from Eq. 2.22).

□

The next theorem (Theorem 2.3) is a specialized version of the previous theorem, where we restrict the regression method to the class of basis function regression. This theorem shows how the assumptions made in the previous general theorem can be satisfied in a concrete setting. It is worth noting that the class of basis function models is quite broad as the basis functions can be any polynomials, radial basis functions and so forth. In particular, it contains the class of the classical linear regression models.

Just like the main theorem, we make a few assumptions (Assumptions 2.5–2.9). The first assumption (Assumption 2.5) sets up a classical least-squares regression problem. The compact set Θ in the assumption can be considered as the finite range of numbers that a computer can represent.

Assumption 2.5 *Assume the regression coefficients β_t (Line 2 of Alg. 1) minimize the least squares error, $\sum_{r=1}^{R_1}\sum_{i=1}^{N}\big(g(x_{t,i}^{(r)}) - \beta^T\eta(x_{0,i}^{(r)})\big)^2$, over the domain Θ, where the vector-valued function $\eta : \mathbb{R}^k \to \mathbb{R}^m$ denotes a set of predetermined basis functions, and Θ is a compact subset of $\mathbb{R}^m$.*

The next assumption (Assumption 2.6) impose some regularity on the regression model space. Here the condition $1 \in \nu$ essentially requires that the basis function regression model has an intercept term. The non-singularity condition of correlation matrix ensures that there is a unique β that corresponds to the projection to the subspace ν. In the case of classical linear regression, this condition is equivalent to that the covariance matrix of predictors (i.e., $\mathrm{Cov}(x_{0,1})$) is non-singular. More generally, a sufficient condition for this to be true is that all the predictors are independent and have positive variances (in this case, covariance matrix is simply diagonal thus trivially non-singular).

Assumption 2.6 *Assume* $1 \in \nu := \{\beta^T\eta(x_{0,1}) \mid \beta \in \mathbb{R}^m\} \subseteq \mathcal{L}^2 := \{X \mid \mathbb{E}[X^2] < \infty\}$ *and the correlation matrix* $\mathbb{E}\Big[\eta(x_{0,1})\big(\eta(x_{0,1})\big)^T\Big]$ *is non-singular.*

Assumption 2.7 *Assume* $\Psi_t(\phi_0, \phi_1, \ldots, \phi_{t-1}, x_{0,1})$ *converges in mean square to* $\Psi_t(c_0, c_1, \ldots, c_{t-1}, x_{0,1})$ *as* N *goes to infinity, where the function* Ψ_t *is defined in Eq. 2.4, and* c_t *is the constant limit of the mean field* ϕ_t *as* $N \to \infty$.

As we will show in the proof, there is a unique coefficient vector in $\mathbb{R}^m$ that corresponds to the projection $\Pi_\nu\big(g(x_{t,1})\big)$. The next assumption (Assumption 2.8) requires that this unique vector is also in the set Θ. Here the condition on lim inf is speaking to Assumption 2.3 of Theorem 2.2.

Assumption 2.8 *Assume*[1] *there is a vector* $\beta_t' \in \Theta$ *such that* $(\beta_t')^T\eta(x_{0,1}) = \Pi_\nu\big(g(x_{t,1})\big)$ *and* $\liminf_{N\to\infty} Cov\big(g(x_{t,2}), (\beta_t')^T\eta(x_{0,1})\big)N \geq 0$.

Assumption 2.9 *Assume the variance of* $\Pi_\nu\big(\Psi_t(c_0, c_1, \ldots, c_{t-1}, x_{0,1})\big)$ *is positive*[2].

Now we state the specialized version of our variance reduction theorem as follows.

Theorem 2.3 *Suppose Assumptions 2.5–2.9 are satisified. Then there exists* $N_0 \in \mathbb{N}$ *such that for any system (Eq. 2.1) with the number of particles greater than* N_0*, the variance reduction factor (Eq. 2.6) is greater than one for all time* t *as long as the number of sample paths in the learning phase is sufficiently large.*

Proof. To prove this theorem, we need several Lemmas (2.2 – 2.5).

Lemma 2.2 *Let* g *be a function on* $\mathbb{R}^k \times \Theta$*, where* Θ *is a compact subset of a Euclidean space. Let* $g(x,\theta)$ *be a continuous function of* θ *for each* x *and a measurable function of* x *for each* θ*. Assume also that* $|g(x,\theta)| \leq h(x)$ *for all* x *and* θ*, where* h *is integrable with respect to a probability distribution*

[1]By Assumption 2.6, it is clear that ν is a closed subspace in $\mathcal{L}^2$. Here the operator $\Pi_\nu(\bullet)$ denotes the projection onto the subspace ν. By Assumption 2.7 and the definition of mean square convergence, $g(x_{t,1}) = \Psi_t(\phi_0, \phi_1, \ldots, \phi_{t-1}, x_{0,1}) \in \mathcal{L}^2$ so that the projection $\Pi_\nu\big(g(x_{t,1})\big)$ is unique and well-defined.

[2]Assumption 2.7 implies $\Psi_t(c_0, c_1, \ldots, c_{t-1}, x_{0,1}) \in \mathcal{L}^2$ so that the projection is well-defined.

function F on $\mathbb{R}^k$. If $x_1, x_2, \ldots, x_n$ are i.i.d. samples from F and $Q_n(\theta) = n^{-1}\sum_{t=1}^{n} g(x_t, \theta)$, then $Q_n(\theta)$ has weak uniform convergence over Θ:

$$\sup_{\theta\in\Theta} \left|Q_n(\theta) - \mathbb{E}[Q_n(\theta)]\right| \xrightarrow[n\to\infty]{p} 0. \tag{2.28}$$

Proof. See Theorem 2 in Jennrich [37]. □

Lemma 2.3 *Let $Q_R(\beta) = \frac{1}{R}\sum_{r=1}^{R} g(x_r, \beta)$, where $\{x_r \mid r = 1, 2, \ldots, R\}$ are random vectors defined on the same probability space, g is a real-valued function and β is a vector in a set $\Theta \subseteq \mathbb{R}^m$. Suppose (i) Θ is compact, (ii) $Q_R(\beta)$ enjoys weak uniform convergence over Θ (weak uniform convergence is defined in Lemma 2.2), (iii) $\mathbb{E}[Q_R(\beta)]$ is continuous with β on the domain Θ, and (iv) $\mathbb{E}[Q_R(\beta)]$ has a unique minimum at β_0 in the domain Θ (i.e., $\beta_0 = \operatorname{argmin}_{\beta\in\Theta} \mathbb{E}[Q_R(\beta)]$). Let β_R minimize $Q_R(\beta)$ on the domain Θ (i.e., $Q_R(\beta_R) = \inf_{\beta\in\Theta} Q_R(\beta)$). Then β_R converges to β_0 in probability as $R \to \infty$.*

Proof. See Theorem 4.1.1 in Amemiya [38, Chap. 4]. Here we give the proof for interested readers. Let $\epsilon = \inf_{\beta\in B^c\bigcap\Theta} \big(\mathbb{E}[Q_R(\beta)] - \mathbb{E}[Q_R(\beta_0)]\big)$, where $B \subseteq \Theta$ is an arbitrary open neighborhood of β_0. Since Θ is compact and $\mathbb{E}[Q_R(\beta)]$ is continuous in β, the infimum can be obtained in the compact set $B^c \bigcap \Theta$. Because $\mathbb{E}[Q_R(\beta)]$ admits a unique minimum, we have $\epsilon > 0$. Note that

$$\mathbb{E}[Q_R(\beta_R)] - Q_R(\beta_R) \le \sup_{\beta\in\Theta} \left|Q_R(\beta) - \mathbb{E}[Q_R(\beta)]\right| \tag{2.29}$$

and

$$Q_R(\beta_R) - \mathbb{E}[Q_R(\beta_0)] \le Q_R(\beta_0) - \mathbb{E}[Q_R(\beta_0)] \le \sup_{\beta\in\Theta} \left|Q_R(\beta) - \mathbb{E}[Q_R(\beta)]\right|. \tag{2.30}$$

Adding Eqs. 2.29 and 2.30 yields

$$\mathbb{E}[Q_R(\beta_R)] - \mathbb{E}[Q_R(\beta_0)] \le 2\sup_{\beta\in\Theta} \left|Q_R(\beta) - \mathbb{E}[Q_R(\beta)]\right|. \tag{2.31}$$

Hence, $\sup_{\beta\in\Theta} \left|Q_R(\beta) - \mathbb{E}[Q_R(\beta)]\right| < \frac{\epsilon}{2}$ implies $\mathbb{E}[Q_R(\beta_R)] - \mathbb{E}[Q_R(\beta_0)] < \epsilon$. By weak uniform convergence, we have $\mathrm{P}\big(\sup_{\beta\in\Theta} \left|Q_R(\beta) - \mathbb{E}[Q_R(\beta)]\right| < \frac{\epsilon}{2}\big) \to 1$ as $R \to \infty$. As a result, $\mathrm{P}\big(\mathbb{E}[Q_R(\beta_R)] - \mathbb{E}[Q_R(\beta_0)] < \epsilon\big) \to 1$. That is, as $R \to \infty$, $\mathrm{P}\Big(\mathbb{E}[Q_R(\beta_R)] - \mathbb{E}[Q_R(\beta_0)] < \inf_{\beta\in B^c\bigcap\Theta} \big(\mathbb{E}[Q_R(\beta)] - \mathbb{E}[Q_R(\beta_0)]\big)\Big) \to 1$. Note that $\mathbb{E}[Q_R(\beta_R)]-\mathbb{E}[Q_R(\beta_0)] < \inf_{\beta\in B^c\bigcap\Theta} \big(\mathbb{E}[Q_R(\beta)]-$

$\mathbb{E}[Q_R(\beta_0)]\big)$ implies $\beta_R \in B$. Therefore, $\mathrm{P}(\beta_R \in B) \to 1$. Since B is an arbitrary open neighborhood of β_0, $\beta_R \xrightarrow{p} \beta_0$ as claimed. $\square$

Lemma 2.4 *Let ν be some closed subspace $\subseteq \mathcal{L}^2$ with $1 \in \nu$ and suppose $Y \in \mathcal{L}^2$. If $X = \Pi_\nu(Y)$, then Var$(Y) =$ Var$(Y - X) +$ Var(X).*

Proof. By the orthogonality principle, $\mathbb{E}[Y^2] = \mathbb{E}[(Y-X)^2] + \mathbb{E}[X^2]$. Since $1 \in \nu$, $(Y - X) \perp 1$. So $\mathbb{E}[Y - X] = 0$. Hence,

$$\begin{aligned}\mathrm{Var}(Y) &= \mathbb{E}[Y^2] - \mathbb{E}[Y]^2 = \mathbb{E}[(Y-X)^2] + \mathbb{E}[X^2] - \mathbb{E}[Y]^2 \\ &= \big(\mathbb{E}[(Y-X)^2] - \mathbb{E}[Y-X]^2\big) + \big(\mathbb{E}[X^2] - \mathbb{E}[X]^2\big) + \big(\mathbb{E}[X]^2 - \mathbb{E}[Y]^2\big) \\ &= \mathrm{Var}(Y-X) + \mathrm{Var}(X) + \mathbb{E}[X-Y]\mathbb{E}[X+Y] \\ &= \mathrm{Var}(Y-X) + \mathrm{Var}(X). \end{aligned}$$

$\square$

Lemma 2.5 *Let $\nu \subseteq \mathcal{L}^2$ be some closed subspace. Let $\{Y_n\}_{n\in\mathbb{N}}$ be a sequence of random variables defined on the same probability space. Suppose $Y_n \to Y$ in m.s., $X_n = \Pi_\nu(Y_n)$ and $X = \Pi_\nu(Y)$, then $X_n \to X$ in m.s. and Var$(X_n) \to$ Var(X).*

Proof. By linearity of the projection operator, $X_n - X = \Pi_\nu(Y_n - Y)$. By the orthogonality principle, $\mathbb{E}[(Y_n - Y)^2] = \mathbb{E}[(X_n - X)^2] + \mathbb{E}[(Y_n - Y - X_n + X)^2]$. In particular, we have $\mathbb{E}[(Y_n - Y)^2] \geq \mathbb{E}[(X_n - X)^2]$. Since $Y_n \to Y$ in m.s., $\mathbb{E}[(Y_n - Y)^2] \to 0$. Hence, $\mathbb{E}[(X_n - X)^2] \to 0$. By definition, $X_n \to X$ in m.s.. So $\mathbb{E}[X_n^2] \to \mathbb{E}[X^2]$ and $\mathbb{E}[X_n] \to \mathbb{E}[X]$. Hence, $\mathrm{Var}(X_n) = \mathbb{E}[X_n^2] - \mathbb{E}[X_n]^2 \to \mathbb{E}[X^2] - \mathbb{E}[X]^2 = \mathrm{Var}(X)$ as claimed. $\square$

Now we prove the theorem as follows.

We will apply Theorem 2.2 for this proof. Let

$$h(x_{0,i}; \beta) = \beta^T \eta(x_{0,i}), \tag{2.32}$$

so that the variance function (Eq. 2.13) and the covariance function (Eq. 2.14) become

$$V(\beta) = \mathrm{Var}\big(g(x_{t,1}) - \beta^T \eta(x_{0,1})\big), \tag{2.33}$$

$$C(\beta) = \mathrm{Cov}\big(g(x_{t,2}), \beta^T \eta(x_{0,1})\big). \tag{2.34}$$

Now to prove the result, it suffices to check each assumption of Theorem 2.2, which we will show separately:

1. **Check Assumption 2.1**

Assumption 2.6 implies $h(x_{0,1};\beta) \in \mathcal{L}^2$. Assumption 2.7 implies $g(x_{t,1}) = \Psi_t(\phi_0, \phi_1, \ldots, \phi_{t-1}, x_{0,1}) \in \mathcal{L}^2$ (this is by definition of mean square convergence).

2. **Check Condition (i) in Assumption 2.2**

First define the function q:

$$q(Z_t^{(r)}, \beta) \coloneqq \sum_{i=1}^{N} \left(g(x_{t,i}^{(r)}) - \beta^T \eta(x_{0,i}^{(r)}) \right)^2, \tag{2.35}$$

where the random vector $Z_t^{(r)} = \left(x_{t,1}^{(r)}, x_{t,2}^{(r)}, \ldots, x_{t,N}^{(r)}, x_{0,1}^{(r)}, x_{0,2}^{(r)}, \ldots, x_{0,N}^{(r)} \right)$ and the coefficient vector β is in the domain Θ.

From Eq. 2.35 it is clear that $q(Z_t^{(r)}, \beta)$ is a continuous function with respect to β for any fixed $Z_t^{(r)}$. Moreover, we have the following inequalities:

$$\begin{aligned} q(Z_t^{(r)}, \beta) &\leq \sum_{i=1}^{N} \left[2\left(g(x_{t,i}^{(r)}) \right)^2 + 2\left(\beta^T \eta(x_{0,i}^{(r)}) \right)^2 \right] \\ &\leq \sum_{i=1}^{N} \left[2\left(g(x_{t,i}^{(r)}) \right)^2 + 2\|\beta\|^2 \|\eta(x_{0,i}^{(r)})\|^2 \right] \\ &\leq \sum_{i=1}^{N} \left[2\left(g(x_{t,i}^{(r)}) \right)^2 + 2C \|\eta(x_{0,i}^{(r)})\|^2 \right] \coloneqq r(Z_t^{(r)}). \end{aligned} \tag{2.36}$$

The second inequality above is a consequence of the Cauchy-Schwarz inequality. Since $\beta \in \Theta$ and Θ is compact, $\|\beta\|^2$ is upper bounded by some constant, which is denoted as C in the third inequality of Eq. 2.36. By Assumptions 2.6 and 2.7, we have that $\mathbb{E}[g^2(x_{t,i}^{(r)})]$ and $\mathbb{E}[\|\eta(x_{0,i}^{(r)})\|^2]$ are both finite. As a result, $r(Z_t^{(r)})$ in Eq. 2.36 is integrable. Now define

$$Q_{R_1,t}(\beta) \coloneqq \frac{1}{R_1} \sum_{r=1}^{R_1} q(Z_t^{(r)}, \beta). \tag{2.37}$$

Since $Z_t^{(1)}, Z_t^{(2)}, \ldots, Z_t^{(R_1)}$ are i.i.d., $Q_{R_1,t}(\beta)$ has the weak uniform convergence property by using Lemma 2.2:

$$\sup_{\beta \in \Theta} \left| Q_{R_1,t}(\beta) - \mathbb{E}[Q_{R_1,t}(\beta)] \right| \xrightarrow[R_1 \to \infty]{p} 0. \tag{2.38}$$

Note that

$$\begin{aligned}\mathbb{E}[Q_{R_1,t}(\beta)] &= \mathbb{E}\left[q(Z_t^{(1)},\beta)\right] \\ &= N\mathbb{E}\left[\big(g(x_{t,1}) - \beta^T\eta(x_{0,1})\big)^2\right] \\ &= N\Big(\mathbb{E}\big[g^2(x_{t,1})\big] - 2\mathbb{E}\Big[g(x_{t,1})\big(\eta(x_{0,1})\big)^T\Big]\beta \\ &\quad + \beta^T\mathbb{E}\Big[\eta(x_{0,1})\big(\eta(x_{0,1})\big)^T\Big]\beta\Big).\end{aligned} \tag{2.39}$$

Because the correlation matrix $\mathbb{E}\Big[\eta(x_{0,1})\big(\eta(x_{0,1})\big)^T\Big]$ is non-singular (Assumption 2.6), it is positive definite. As a result, $\mathbb{E}[Q_{R_1,t}(\beta)]$ is quadratic in β and is strictly convex. Hence, there is a unique vector in $\mathbb{R}^m$ such that $\mathbb{E}[Q_{R_1}(\beta)]$ is minimized. From Eq. 2.39, it is clear that this vector also uniquely minimizes $\mathbb{E}\left[\big(g(x_{t,1}) - \beta^T\eta(x_{0,1})\big)^2\right]$. By the orthogonality principle, this unique vector is exactly β_t' in Assumption 2.8. Since $\beta_t' \in \Theta \subseteq \mathbb{R}^m$, we have

$$\beta_t' = \operatorname*{argmin}_{\beta\in\Theta} \mathbb{E}[Q_{R_1,t}(\beta)]. \tag{2.40}$$

By Assumption 2.5, we have

$$Q_{R_1,t}(\beta_t) = \inf_{\beta\in\Theta} Q_{R_1,t}(\beta). \tag{2.41}$$

From Eq. 2.39, it is clear that $\mathbb{E}[Q_{R_1,t}(\beta)]$ is continuous with respect to β. By Lemma 2.3, we have

$$\beta_t \xrightarrow[R_1\to\infty]{p.} \beta_t'. \tag{2.42}$$

3. **Check Condition (ii) in Assumption 2.2**

Observe that $V(\beta)$ in Eq. 2.33 and $C(\beta)$ in Eq. 2.34 are respectively quadratic and linear in β. As a result, both $V(\beta)$ and $C(\beta)$ are continuous functions. Recall that β_t converges to β_t' in probability (Eq. 2.42). Hence, by the continuous mapping theorem, we have

$$V(\beta_t) \xrightarrow[R_1\to\infty]{p.} V(\beta_t') \quad \text{and} \quad C(\beta_t) \xrightarrow[R_1\to\infty]{p.} C(\beta_t'). \tag{2.43}$$

Since (i) $\beta_t \in \Theta$, (ii) Θ is compact, and (iii) $V(\beta)$ is a continuous function on Θ, the sequence of $|V(\beta_t)|$ indexed by R_1 is bounded by a finite positive constant. Similarly, the sequence of $|C(\beta_t)|$ is also bounded by some finite positive constant. Together with convergence in probability (Eq. 2.43), we have convergence in mean by the Lebesgue dominated convergence theorem:

$$V(\beta_t) \xrightarrow[R_1\to\infty]{\mathcal{L}^1} V(\beta_t') \quad \text{and} \quad C(\beta_t) \xrightarrow[R_1\to\infty]{\mathcal{L}^1} C(\beta_t'). \tag{2.44}$$

4. **Check Assumption 2.3**

This follows immediately from Assumption 2.8 and the definition of $C(\beta)$ in Eq. 2.34.

5. **Check Assumption 2.4**

Note that

$$\begin{aligned}
\mathrm{Var}\big(g(x_{t,1})\big) - V(\beta_t') &= \mathrm{Var}\big(g(x_{t,1})\big) - \mathrm{Var}\big(g(x_{t,1}) - (\beta_t')^T \eta(x_{0,1})\big) \\
&= \mathrm{Var}\big(g(x_{t,1})\big) - \mathrm{Var}\Big(g(x_{t,1}) - \Pi_\nu\big(g(x_{t,1})\big)\Big) \\
&= \mathrm{Var}\Big(\Pi_\nu\big(g(x_{t,1})\big)\Big) \\
&= \mathrm{Var}\Big(\Pi_\nu\big(\Psi_t(\phi_0,\phi_1,\dots,\phi_{t-1},x_{0,1})\big)\Big).
\end{aligned} \tag{2.45}$$

The third equality in Eq. 2.45 is a consequence of Lemma 2.4. By the mean square convergence in Assumption 2.7 and using Lemma 2.5, we have

$$\lim_{N\to\infty}\Big(\mathrm{Var}\big(g(x_{t,1})\big) - V(\beta_t')\Big) = \mathrm{Var}\Big(\Pi_\nu\big(\Psi_t(c_0,c_1,\dots,c_{t-1},x_{0,1})\big)\Big) > 0, \tag{2.46}$$

where the positiveness of the variance is due to Assumption 2.9. This implies the desired result since the limit and lim inf coincide.

□

2.5 Numerical study 1: 1D demonstration

In this section, we will show the performance of our adaptive control variates algorithm for a simple mean-field system, where the state of a particle is one-dimensional:

$$\begin{aligned}
x_{t+1,i} &= k\phi_t + e^{bx_{0,i}}, \\
\phi_t &= \frac{1}{N}\sum_{j=1}^{N} x_{t,j}, \quad i = 1,2,\dots,N, \quad t\in\mathbb{N},
\end{aligned} \tag{2.47}$$

where k and b are some deterministic constants. The initial particles are sampled i.i.d. from the standard Gaussian distribution (i.e., $x_{0,i} \sim \mathcal{N}(0, 1)$). The observable of this system is equal to the mean field:

$$y_t = \frac{1}{N} \sum_{i=1}^{N} x_{t,i}, \quad t = 1, 2, \ldots, T. \tag{2.48}$$

For this study, we set $k = 0.2$, $b = 1$, $N = 100$ and $T = 15$. We use the least squares method to build cubic polynomial regression models for the learning phase of the adaptive control variates algorithm. It can be checked that the system with these parameters satisfies all the assumptions of Theorem 2.3 (see Appendix A). It is worth noting that in this example the observable depends on the states of 100 particles (i.e., the particle system is 100-dimensional) whereas the regression problem is only 1-dimensional thanks to the new learning scheme of our algorithm.

We will first examine the convergence of regression coefficients for the least squares method. We observe from Fig. 2.2 that as the number of learning sample paths increases, the regression coefficients β_t from the least squares are approaching the coefficients β'_t of the projection (see Assumption 2.8 for definition of β'_t).

Figure 2.3 shows an example of a cubic polynomial model that is trained with the least squares method. We see that the trained model predicts well for the properties of the particles in the evaluation phase.

Figure 2.4 shows the variance reduction performance of our adaptive control variates algorithm. We observe that the variances of the mean estimators are reduced by an order of magnitude for all time steps. This significant variance reduction demonstrates the effectiveness of our adaptive control variates algorithm for the system under investigation. It also provides numerical validation for the assertion of Theorem 2.3.

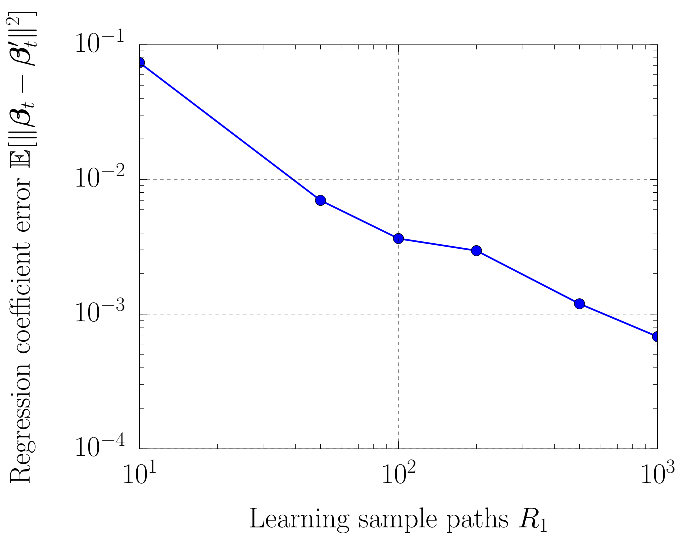

Figure 2.2: Convergence of regression coefficients for the least squares method, showing the mean square error between regression coefficients from the least squares and the coefficients of the projection for time $t = 5$.

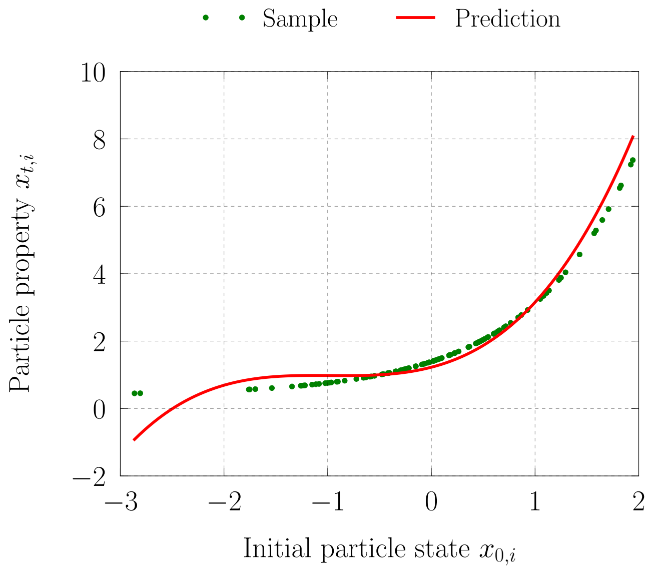

Figure 2.3: Performance of a cubic polynomial regression model for time $t = 5$ with $R_1 = 100$. The cubic model (red) trained with the least square method gives good prediction for the particle property (green) in the evaluation phase.

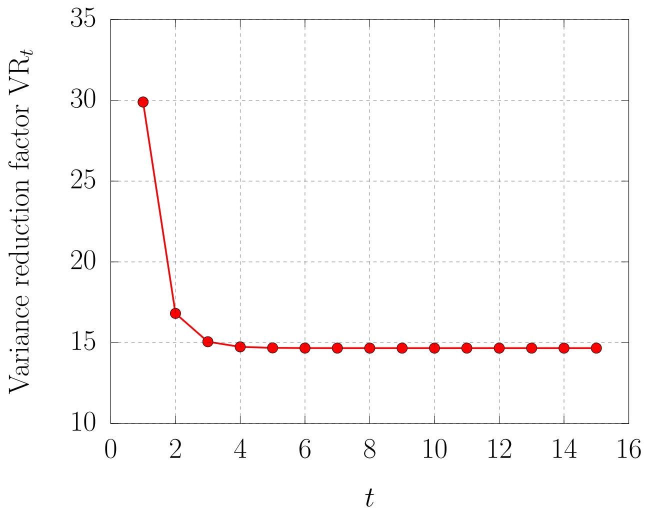

Figure 2.4: Variance reduction performance of adaptive control variates algorithm for the one-dimensional mean-field system in Eq. 2.47. The number of sample paths in the learning phase is $R_1 = 100$. The number of independent learning phases R_3 (see Eq. 2.10) for estimating variance reduction is 20.

2.6 Numerical study 2: a linear Gaussian example

In this section, we will demonstrate our adaptive control variates algorithm for a linear Gaussian mean-field system defined by

$$
\begin{aligned}
x_{t+1,i} &= \alpha x_{t,i} + \phi_t, \\
\phi_t &= \frac{1}{N}\sum_{j=1}^{N} x_{t,j}, \quad i = 1, 2, \ldots, N, \quad t \in \mathbb{N},
\end{aligned}
\tag{2.49}
$$

where $x_{t,i}$ is the state of a particle, ϕ_t is the mean field and $\alpha \neq 0$ is a constant. The initial particles are sampled i.i.d. from the standard Gaussian distribution (i.e., $x_{0,i} \sim \mathcal{N}(0,1)$). The observable y_t takes a quadratic form

$$
y_t = \frac{1}{N}\sum_{i=1}^{N} x_{t,i}^2, \quad t = 1, 2, \ldots, T. \tag{2.50}
$$

For this example, we can obtain an analytical expression for the particle state $x_{t,i}$ as a function of the initial states of particles. We assume that the particle state $x_{t,i}$ takes the form $x_{t,i} = \alpha^t x_{0,i} + \gamma_t \phi_0$ with γ_t being a constant to be determined. Substituting this assumed expression into Eq. 2.49, we find that γ_t must satisfy the following recursion

$$
\gamma_t - (1+2\alpha)\gamma_{t-1} + (\alpha^2 + \alpha)\gamma_{t-2} = 0, \tag{2.51}
$$

with $\gamma_0 = 0$ and $\gamma_1 = 1$. Using the technique of characteristic equations, we have

$$
\gamma_t = (\alpha+1)^t - \alpha^t. \tag{2.52}
$$

As a result, the particle state $x_{t,i}$ can be expressed analytically as

$$
x_{t,i} = \alpha^t x_{0,i} + \big((\alpha+1)^t - \alpha^t\big)\phi_0. \tag{2.53}
$$

Using the fact that the initial states are i.i.d. Gaussian and taking advantage of the derived analytical form (Eq. 2.53), we can compute the variance of the observable y_t as

$$
\mathrm{Var}(y_t) = 2\alpha^{4t} N^{-1} + 2\big((\alpha+1)^{4t} - \alpha^{4t}\big) N^{-2}. \tag{2.54}
$$

Hence, the variance of the naive estimator is given by

$$\mathrm{Var}(\hat{y}_t^{\text{naive}}) = \frac{\mathrm{Var}(y_t)}{R}, \tag{2.55}$$

where R is the number of sample paths.

For the adaptive control variates, we learn a simple quadratic model $ax_{0,i}^2 + b$ for the particle property $x_{t,i}^2$ using the least squares method. Given sufficiently many learning samples (i.e., R_1 is sufficiently large), the coefficients of the learned model would converge (see proof of Theorem 2.3). These limiting coefficients can be found using projection technique, which we will demonstrate next.

Without loss of generality, let us work with the first particle (i.e., $i = 1$). We define the subspace $\nu = \{ax_{0,1}^2 + b \mid a, b \in \mathbb{R}\}$. Then the limiting regression model is the projection onto the subspace ν:

$$\Pi_\nu(x_{t,1}^2) = \mathbb{E}[x_{t,1}^2] + \mathrm{Cov}(x_{t,1}^2, x_{0,1}^2)\mathrm{Var}(x_{0,1}^2)^{-1}(x_{0,1}^2 - \mathbb{E}[x_{0,1}^2]), \tag{2.56}$$

where Π is a projection operator. Using the analytical expression (Eq. 2.53) and Gaussian properties, we can compute the expectation, variance and covariance terms in Eq. 2.56, resulting in a quadratic model h:

$$h(x_{0,1}; a_t^*, b_t^*) = a_t^* x_{0,1}^2 + b_t^*, \tag{2.57}$$

where the coefficients a_t^* and b_t^* are given by

$$\begin{aligned} a_t^* &= \Big(\alpha^t + \big((\alpha+1)^t - \alpha^t\big)N^{-1}\Big)^2, \\ b_t^* &= \alpha^{2t} + N^{-1}\big((\alpha+1)^{2t} - \alpha^{2t}\big) - a_t^*. \end{aligned} \tag{2.58}$$

Hence, the mean of the control variates is given by

$$\mathbb{E}[h(x_{0,1}; a_t^*, b_t^*)] = a_t^* + b_t^*. \tag{2.59}$$

The control variates can be then constructed as

$$\begin{aligned} y_t^{\text{cv}} &= \frac{1}{N}\sum_{i=1}^{N}\left(x_{t,i}^2 - h(x_{0,i}; a_t^*, b_t^*)\right) + \mathbb{E}[h(x_{0,1}; a_t^*, b_t^*)] \\ &= y_t - \frac{1}{N}\sum_{i=1}^{N} h(x_{0,i}; a_t^*, b_t^*) + \mathbb{E}[h(x_{0,1}; a_t^*, b_t^*)], \end{aligned} \tag{2.60}$$

and the variance of the control variates estimator is given by

$$\mathrm{Var}(\hat{y}_t^{\text{cv}}) = \frac{\mathrm{Var}(y_t^{\text{cv}})}{R}. \tag{2.61}$$

After some algebra, the difference between the variance of y_t and the variance of y_t^{cv} is

$$\Delta := \mathrm{Var}(y_t) - \mathrm{Var}(y_t^{\text{cv}}) = 2a_t^* N^{-1}\Big(a_t^* + 2\big((\alpha+1)^t - \alpha^t\big)^2 N^{-1}(1 - N^{-1})\Big), \tag{2.62}$$

where a_t^* is given in Eq. 2.58. Observe that a_t^* is non-negative thus $\Delta \geq 0$, which implies that $\mathrm{Var}(\hat{y}_t^{\text{cv}}) \leq \mathrm{Var}(\hat{y}_t^{\text{naive}})$. This means that the variance of adaptive control variates estimator is no worse that that of naive one. Furthermore, as N goes to infinity, a_t^* converges to $\alpha^{2t} > 0$. As a result, Δ would become strictly positive for sufficiently large N (i.e., the variance is reduced for large N).

The observable y_t depends on the states of all the particles so the dimension of y_t is equal to the number of particles N. Having analytically found the variance of the naive estimator (Eq. 2.54) and the variance difference (Eq. 2.62), we can compute variance reduction for any N as

$$\mathrm{VR}_t = \frac{\mathrm{Var}(\hat{y}_t^{\text{naive}})}{\mathrm{Var}(\hat{y}_t^{\text{cv}})} = \frac{\alpha^{4t} + \mathcal{O}(N^{-1})}{\big((\alpha+1)^{2t} - \alpha^{2t}\big)^2 + \mathcal{O}(N^{-1})} N, \tag{2.63}$$

from which we see that the variance reduction factor is $\mathcal{O}(N)$.

To numerically verify this, we set $\alpha = -0.6$, $T = 10$ and vary N from 10 to 1000. The result is shown in Fig. 2.5. As we can see, the variance is reduced by 1–3 orders of magnitude for all time steps. The variance reduction is roughly proportional with the dimension N.

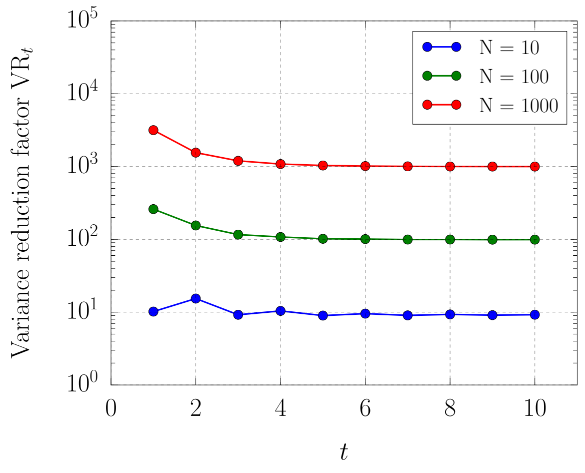

Figure 2.5: Variance reduction performances for the 1D linear Gaussian example with different number of particles.

The analysis above is for the limiting regression model (i.e., $R_1 \to \infty$). For the cases of finite learning samples, we set $R_1 = 100$, $N = 100$ and use least squares to train quadratic models. We find that the variance reduction is about 2 orders of magnitude as shown in Fig. 2.6.

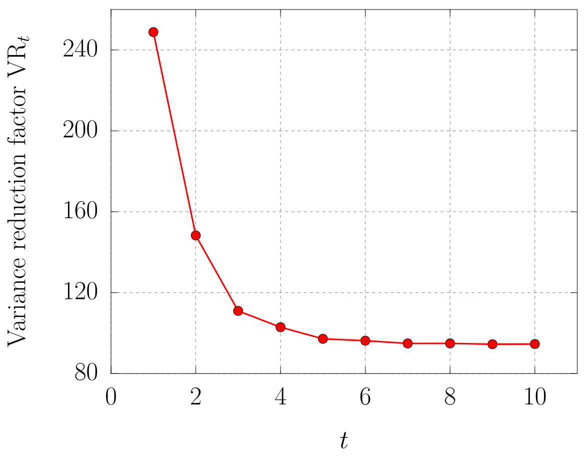

Figure 2.6: Variance reduction performance for the 1D linear Gaussian example when $R_1 = 100$ and $N = 100$.

2.7 Numerical study 3: aerosol particle simulation

2.7.1 Process description

In this study, we will demonstrate our algorithm for complex particle simulations of atmospheric aerosol dynamics, using the model and the codes of Riemer et al. [14]. This aerosol model explicitly stores the composition of a large number of aerosol particles, and contains many physicochemical processes, including emission, dilution, gas chemistry and gas-particle interactions. This particle-resolved model circumvents the combinatorial explosion that occurs in traditional methods when attempting to resolve high-dimensional aerosol distributions [14], and has been successfully applied to many aerosol problems, such as soot mixing state [14], cloud condensation nuclei [34] and black carbon [39, 40].

For brevity, here we only give a succinct description of the processes in the aerosol model. The details of each process have been elaborated elsewhere [14, 34]. The model is implemented in the software package PartMC[1].

This model considers a large number of aerosol particles (about 10^5) within a well-mixed computational volume in the air. The state of each particle is described by its composition of 20 different chemical species. The changes in the particle compositions are modeled explicitly as the particles evolve through different processes but their locations within the volume are not kept track of.

To start with, N_i particles are randomly sampled from some initial sources:

$$N_\mathrm{i} \sim \mathrm{Pois}(N_\mathrm{init}), \tag{2.64}$$

$$V_{0,i} \sim \ln\mathcal{N}(\mu_\mathrm{init}, \sigma^2_\mathrm{init}), \quad x_{0,i} = r_\mathrm{init} V_{0,i}, \quad i = 1, 2, \ldots, N_\mathrm{i}, \tag{2.65}$$

where N_init, μ_init and σ_init are parameters of probability distributions. The variable $V_{0,i}$ represents the initial volume of particle i while r_init is the volumetric fractions of the chemical species of a particle. The vector $x_{0,i}$ is the initial state of particle i.

The computational volume of an air box is initialized to be proportional

[1]It is available under the GNU General Public License (GPL) at `http://lagrange.mechse.illinois.edu/partmc/`.

to the initial (mean) number of particles:

$$V_{\mathrm{box}} \sim M_{\mathrm{init}} \coloneqq \sum_{\text{initial sources}} N_{\mathrm{init}}. \tag{2.66}$$

At each time t, $N_{\mathrm{e},t}$ particles are randomly sampled from every source of emission:

$$N_{\mathrm{e},t} \sim \mathrm{Pois}(\lambda_{\mathrm{emit}}(t) V_{\mathrm{box}}), \tag{2.67}$$

$$V_{t,i} \sim \ln\mathcal{N}(\mu_{\mathrm{emit}}, \sigma^2_{\mathrm{emit}}), \quad x_{t,i} = r_{\mathrm{emit}} V_{t,i}, \quad i = 1, 2, \ldots, N_{\mathrm{e},t}, \tag{2.68}$$

and $N_{\mathrm{d},t}$ particles are randomly sampled from every source of dilution:

$$N_{\mathrm{d},t} \sim \mathrm{Pois}(\lambda_{\mathrm{dil}}(t) V_{\mathrm{box}}), \tag{2.69}$$

$$V_{t,i} \sim \ln\mathcal{N}(\mu_{\mathrm{dil}}, \sigma^2_{\mathrm{dil}}), \quad x_{t,i} = r_{\mathrm{dil}} V_{t,i}, \quad i = 1, 2, \ldots, N_{\mathrm{d},t}. \tag{2.70}$$

Here the vector $x_{t,i}$ denotes the state of particle i at time t. Parameters $\lambda_{\mathrm{emit}}(t)$ and $\lambda_{\mathrm{dil}}(t)$ are respectively the emission rate and the dilution rate at time t.

In addition, $N_{\mathrm{r},t}$ particles are randomly removed from the air box for the dilution-out process:

$$N_{\mathrm{r},t} \sim \mathrm{Binom}(N_{\mathrm{tot},t}, p_{\mathrm{dil},t}), \tag{2.71}$$

where $N_{\mathrm{tot},t}$ is the total number of particles in the box at time t. The parameter $p_{\mathrm{dil},t}$ is the probability of a particle diluting out of the box.

Moreover, aerosol particles interact with gas species via gas-particle partitioning so that both the particle state and the concentrations of gas species change for every time step. During the partitioning process, the concentrations of the gas species are influenced by the average contribution of all the particles. As a result, the gas concentrations are the mean fields for this aerosol model. Indeed, we analytically prove in a simplified setting that the gas concentration converges to a constant as the number of particles becomes sufficiently large (see Appendix B).

The observables are total mass concentration and optical scattering coefficient [34], both taking the form of

$$y_t = \frac{1}{(V_{\text{box}})_t} \sum_{i=1}^{N_t} g(x_{t,i}). \tag{2.72}$$

Specifically, for the total mass concentration, function g represents the mass of an aerosol particle. For the optical scattering coefficient, function g is the scattering cross section, an optical property of an aerosol particle.

An illustration of the entire aerosol model is given in Appendix C.

2.7.2 MARS and regression techniques

We refer to the time a particle is first added to the box as the creation time of that particle. For a given output time, the creation times of all the particles in the system are generally not the same. Particles with different creation times experience different gas-particle partitioning processes due to their different histories. As a result, we build a model with respect to not only the initial state of the particle but also the creation time of that particle. Mathematically,

$$g(x_{t,i}) \approx h(x_{\tau_i,i}, \tau_i; \beta_t), \quad i = 1, 2, \ldots, N_t, \tag{2.73}$$

where τ_i denotes the creation time of particle i, and g is the particle mass or the scattering cross section of the particle.

The machine learning algorithm we use to train the model h in Eq. 2.73 is multivariate adaptive regression splines (MARS), a non-parametric regression method introduced by Friedman [41]. MARS is an adaptive procedure for regression, and is well suited for high-dimensional problems [42, Chap. 9]. A MARS model is quite flexible and enjoys a good bias-variance trade-off. Moreover, the model is efficient to build and fast to evaluate so that the computational overhead is well limited.

In addition to MARS, we utilize three techniques to further improve the performance of the model. The first technique is to split training samples by particle creation times. For a given output time t, the creation times can only assume values of $0, 1, \ldots, t$. Particles with creation times close to each other experience similar gas-particle partitioning processes. This motivates us to partition the training data into smaller chunks based on the creation time, and then build MARS models separately for each chunk of data. The resulting model is essentially a function piecewise in the creation time with each piece being a MARS model. Compared to the MARS model without splitting the data, this piecewise model is more flexible and possesses more predictive power. Because the computation time of a MARS algorithm scales linearly with the number of samples, this splitting strategy does not increase the total training time. To ensure sufficiently many samples for each partition, we use a simple greedy algorithm (Alg. 6 in Appendix C).

It is possible that a particle in the evaluation phase is beyond the range of the training data (e.g., a particularly large particle) so that the model may not generalize well to this particle. The second technique attempts to mitigate this generalization problem by imposing some regularization beyond the boundary of the model. Here we define the boundary of the model in terms of the range of the particle volume in the training data. The regularization of the upper bound is carried out via a simple "clamping" function: if the volume of a particle in the evaluation phase exceeds the maximum particle volume in the training data, the prediction of this particle is simply made equal to that of the particle with the maximum volume. The regularization of the lower bound is done using a similar clamping function. Given the piecewise MARS model resulted from the first technique, one can regularize only the upper bound or only the lower bound or both or neither of the bounds. To automatically determine which of the four regularization methods is the most appropriate, we use 5-fold cross validation.

The third technique is to build models from a scaled-down system. That is, we make the initial mean number of particles M_{init} (defined in Eq. 2.66) in the learning phase less than that of the evaluation phase:

$$(M_{\text{init}})_{\text{learn}} < (M_{\text{init}})_{\text{eval}}. \tag{2.74}$$

Since the computational cost of aerosol simulation is roughly proportional to the number of particles, by decreasing M_{init} the computational overhead during the learning phase can be significantly reduced. Yet the prediction performance of the model from this scaled-down system is comparable to that from the original system as long as $(M_{\text{init}})_{\text{learn}}$ is not too small. This is because the evolution of this type of process often exhibits convergence in the limit of a large number of particles so that models learned from a downsized system may still work reasonably well for the full system.

2.7.3 Algorithm performance for estimating aerosol properties

We applied our adaptive control variates algorithm to the aerosol simulations of an idealized urban plume scenario over a period of 24 h with the time step $\Delta t = 1$ min. The total mass concentration and the scattering coefficient were outputted every hour. The number of chemical species in a particle was 20. The initial mean number of particles $M_{\text{init}} = (M_{\text{init}})_{\text{eval}} = 10^5$. Other process parameters such as distributions of different sources are described in Zaveri et al. [34]. We used the Earth package [43] for the implementation of MARS. The partition parameter (of Alg. 6) $N_{\text{p}} = 5000$. We set $R_1 = 5000$ and $(M_{\text{init}})_{\text{learn}} = 100$. With the number of particles being about 10^5 and each particle consisting of 20 species, the dimension of the particle system is roughly 2×10^6, which is prohibitively high for the conventional adaptive control variates method.

Figure 2.7 shows the variance reduction performance of our adaptive control variates algorithm. We observe that the variances of the mean estimators are reduced by a factor of about 1–3 orders of magnitude for both bulk properties. The variance reduction factor of total mass concentration decreases quite sharply initially and then gradually increases after about the tenth step, whereas that of the scattering coefficient decreases gradually.

Figure 2.8 compares the statistical error of mean estimation and computation time of the two algorithms: our adaptive control variates algorithm and naive Monte Carlo. The statistical error is measured by the L_2 norm of standard deviations of mean estimators at different time steps.

First, we observe that our adaptive control variates algorithm is more efficient than naive Monte Carlo for both bulk properties. Specifically, for a fixed amount of computation time, the estimation error of our algorithm is about one third of that of naive Monte Carlo. For the same precision of mean estimators, our algorithm requires only about 15% the time of naive Monte Carlo.

Second, we see that the statistical error of naive Monte Carlo falls at the rate of $-\frac{1}{2}$ as expected. Due to the computational overhead of the learning

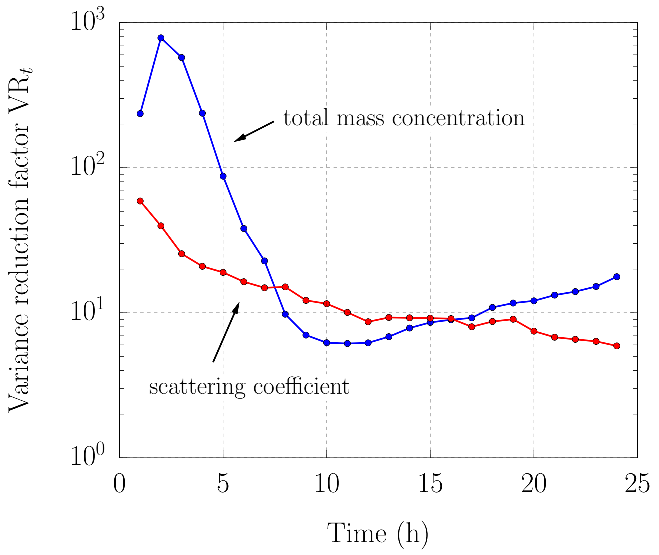

Figure 2.7: Variance reduction performance of our adaptive control variates algorithm for the mean estimation of the two bulk properties: total mass concentration (blue) and scattering coefficient (red). The number of independent learning phases R_3 (see Eq. 2.10) for estimating variance reduction is 20.

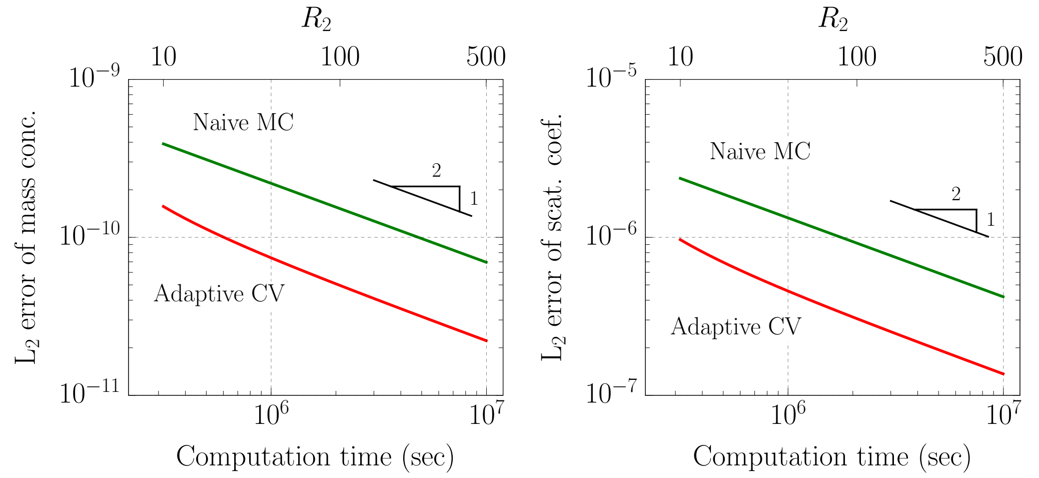

Figure 2.8: Statistical error and computation time of our adaptive control variates algorithm and naive Monte Carlo for the mean estimation of two bulk properties as the number of sample paths in the evaluation phase varies from 10 to 500. The error is measured by the L$_2$ norm of the standard deviation of estimators at different time steps. Comparing to naive Monte Carlo (green), the estimator error of our algorithm (red) is reduced by $\sim 67\%$ for fixed computation time, and the required computation time is reduced by $\sim 85\%$ for the same precision.

phase, the statistical error of adaptive control variates algorithm decreases at a slightly faster rate initially but as the number of sample paths in the evaluation phase increases, the computational overhead becomes more and more negligible so that the error rate approaches $-\frac{1}{2}$ eventually.

2.8 Conclusions

In this chapter we presented a novel adaptive control variates algorithm for a class of stochastic simulations, in which a large number of particles interact via common mean fields. Because these particle systems are often very high-dimensional, the conventional adaptive control variates methods are normally not applicable to these simulations. To deal with this difficulty, we proposed a new learning scheme that treats all the particles as training samples. Compared to the conventional methods, the amount of the training data per sample path of our algorithm is significantly enhanced and the dimension of the learning problem is greatly reduced.

We proved that the mean estimators from our algorithm are unbiased (Theorem 2.1). We also showed that for the system with sufficiently many particles, our algorithm will asymptotically produce more efficient estimators than naive Monte Carlo provided that some conditions are satisfied (Theorems 2.2 and 2.3). A numerical study on a simple one-dimensional mean-field system validated our theoretical claims. We applied our algorithm to a complex aerosol particle simulation, and found that the stochastic error of the mean estimator was reduced by about 67% and the required computation time was reduced by about 85%.

CHAPTER 3

PARALLEL SURROGATE OPTIMIZATION FOR NOISY EXPENSIVE FUNCTIONS

3.1 Introduction

Noisy optimization refers to a class of optimization problems, where the objective function is corrupted with random noise. The randomness in the objective may come from stochasticity in numerical computations (e.g., Monte Carlo simulations) or random measurement errors of physical experiments.

In this work, we consider a general global noisy optimization problem:

$$\begin{aligned} &\text{minimize} \quad F(x), \quad \text{where } F(x) := \mathbb{E}_\omega[f(x,\omega)], \\ &\text{s.t.} \quad x \in \mathcal{D} = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d] \subseteq \mathbb{R}^d, \end{aligned} \tag{3.1}$$

where f is an expensive black-box function, ω captures noise (randomness) in the function evaluation and the dimension d is low to medium (up to tens of dimensions). We assume that only noisy evaluations f are observed and the underlying objective function F is unknown.

The problem in Eq. 3.1 is a standard optimization problem [44, 45] that appears in many applications including operations [46], engineering designs [47], biology [48, 49], transportation [50, 51] and machine learning [52, 53]. Here we give two concrete examples of how this optimization problem can show up in real world. The first example is characterization of an expensive stochastic model. In this case, a model, with some parameters that cannot be measured or identified precisely due to physical limitations, needs to be calibrated against some experimental observations before it can be further used for predictions. Determining these unsure parameters can be formally cast into a problem in Eq. 3.1, where the objective function in this case is the expected discrepancy between the simulated outputs of the model and the experimental observations. The second example considers the scenario where the noisy function (i.e., f in Eq. 3.1) is a physical measurement. For instance, a chemist may have a few tunable process parameters (e.g., temperature), and would like to find the optimal value that gives the maximum reaction yield. Here the objective could be minus expected yield (minus is due to a maximization problem here). The random noise may arise from the yield-measuring process. Obtaining one yield value requires conducting a chemical experiment so the function f is often expensive to evaluate.

Another relevant problem is a stochastic bandit with infinitely many arms [54, 55]. In this type of problem, the goal is to find the optimal strategy within a continuous space so that the expected cumulative reward is maximized. An important theoretical result shown by Bubeck et al. [55] is that if the mean payoff function in a bandit is locally Lipschitz, then the rate of growth of the regret can be independent of the dimension of the space. The key difference between a bandit problem and the problem we consider in this work is that the objective function in a bandit problem is usually not expensive. As a result, the solution strategy for a bandit problem is often somewhat different from that for an expensive optimization problem.

Next, we will review the methods of solving the expensive optimization problem in Eq. 3.1. There are two main classes of methods for this: gradient-based methods and derivative-free methods. The gradient-based methods attempt to robustly estimate the unknown gradients of the function using noisy evaluations. The classic technique for gradient estimation is through stochastic approximation [56]. The technique of stochastic approximation has led to the development of several optimization algorithms [57–59]. Unfortunately these gradient-based algorithms are usually guaranteed to converge only to local optima. Moreover, because of the constraints on step sizes, these algorithms tend to make relatively slow progress towards an optimum. Hence, they are typically not suitable for expensive optimization problems [51].

Within the class of derivative-free methods, there are heuristic algorithms and surrogate-based algorithms. Here we use the term "heuristic algorithms" to generally refer to nature-inspired or simplex-based or direct search algorithms, such as particle swarm optimization [60], Nelder-Mead algorithm [61], simulated annealing [62], differential evolution [63] and direct search [64]. Despite the success in many applications, one drawback of these algorithms is that the trend of the underlying objective function is not well exploited. As a result, they often require a larger number of function evaluations compared to surrogate-based algorithms [65, 66].

Surrogate-based methods, also known as response surface methodology or metamodel methods, are a class of global optimization algorithms that efficiently search the domain with the assist of surrogates. The method starts with a space-filling experiment design. Next, in each iteration, a surrogate function that approximates the objective function is first constructed using available evaluations, and then a new set of point(s) is carefully proposed for the next iteration based on the surrogate. Because the function evaluation is expensive, spending extra computation in determining which points to evaluate is often worthwhile. For the noisy optimization problems, the surrogates are essentially regression models so the surrogate-based algorithms inherently have close connection with the field of machine learning.

In the family of surrogate-based methods, parallel surrogate optimization algorithms propose multiple points in each iteration, and the expensive evaluations of these points are performed in parallel [67]. These algorithms are often configured in a master-worker framework as illustrated in Fig. 3.1. Compared to the serial counterpart, parallel surrogate optimization uses the parallel cores of a machine more efficiently, thereby achieving better progress per unit wall time.

A popular method for noisy parallel surrogate optimization is Bayesian optimization [52, 53, 68–72]. Bayesian optimization typically works by assuming a Gaussian process prior over the objective function, constructing a Gaussian process (GP) surrogate [73] with the evaluations, and proposing new points through optimizing an acquisition function. Common acquisition functions are expected improvement (EI) [52], upper confidence bound (UCB) or lower confidence bound (LCB) [68, 69], and information-theoretic based [53, 70].

One issue with Bayesian methods is the high computational cost. Typically, training a GP surrogate requires solving a maximum likelihood problem, for which operations of complexity proportional to the cube of the number of evaluations are performed for many times [74]. To propose new points, Bayesian optimization usually requires the solution of sub-optimization problems (e.g., maximizing expected improvement) with the possible use of Monte Carlo procedures [52, 71]. When many parallel cores are used, so that the number of evaluations accumulates quickly with the number of iterations, Bayesian optimization algorithm itself can be even more expensive than the evaluation of the function f, and this is indeed observed in real

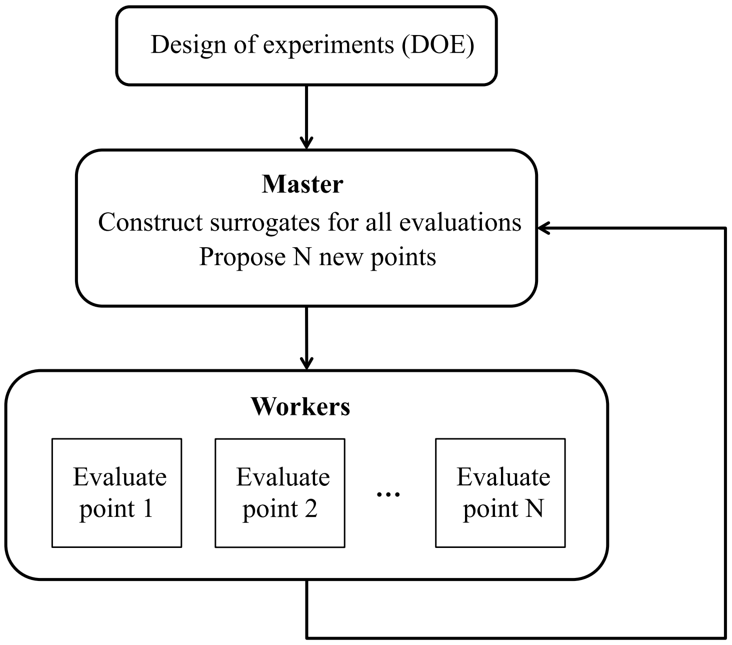

Figure 3.1: A schematic diagram of a general master-worker framework for a parallel surrogate optimization algorithm. In each iteration, the algorithm (master) constructs a surrogate based on the available evaluations, proposes multiple points based on the surrogate, and distributes these points to different processes (workers) for parallel evaluations. The evaluated points are then fed back into the loop to update the surrogate in the next iteration.

hyperparameter-tuning problems (see Section 3.4.3).

In this work, we develop a novel algorithm called ProSRS for noisy parallel surrogate optimization. Unlike Bayesian optimization that uses a GP model, our algorithm uses a radial basis function (RBF), which is more efficient computationally. We adopt an efficient framework, known as stochastic response surface (SRS) method [66, 75], for proposing new points in each iteration. The sub-optimization problems in the SRS method are discrete minimization problems. Compared to the original parallel SRS work [75], our work: (1) introduces a new tree-based technique, known as the "zoom strategy", for efficiency improvement, (2) extends the original work to the noisy setting (i.e., an objective function corrupted with random noise) through the development of a radial basis regression procedure, (3) introduces weighting to the regression to enhance exploitation, (4) implements a new SRS combining the two types of candidate points that were originally proposed in SRS [66]. We compare our algorithm to three well-established parallel Bayesian optimization algorithms. We find that our algorithm shows superior optimization performance on both benchmark problems and real hyperparameter-tuning problems, and yet its cost is orders of magnitude lower. The fact that our algorithm is significantly cheaper means that our algorithm is suitable for a wider range of optimization problems, not just very expensive ones.

3.2 ProSRS algorithm

Conventional surrogate optimization algorithms use all the expensive function evaluations from past iterations to construct the surrogate. As the number of evaluations grows over iterations, the cost of conventional methods thus increases. Indeed, the cost can increase rather quickly with the number of iterations, especially when a large number of parallel cores are used and so many points are evaluated per iteration.

To overcome this limitation, we develop a novel algorithm that does not necessarily use all the past evaluations while still being able to achieve good optimization performance. The key intuition here is that once an optimal region is approximately located, progress can be made by focusing on the evaluations within this region. This idea is illustrated in Fig. 3.2, where the red curve is a surrogate built with all the evaluations. Now suppose we

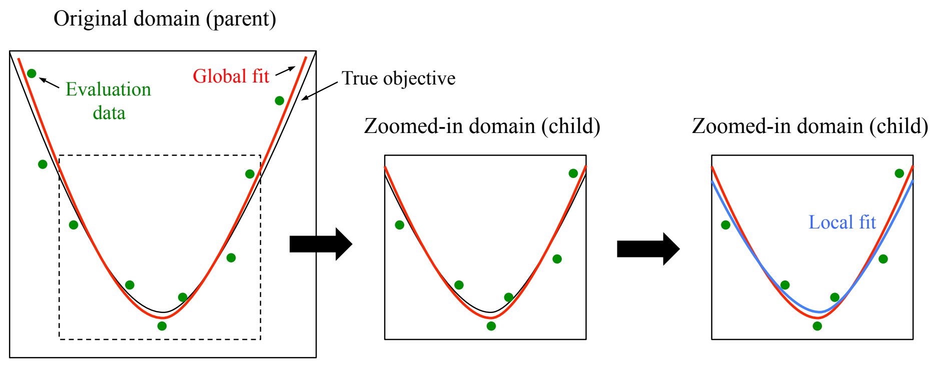

Figure 3.2: Illustration of the zoom strategy on a 1-D parabola. The red curve shows the surrogate fit to all the noisy evaluations (green dots) of the objective function (black curve). The blue curve shows the surrogate fit using only the local evaluation data in the zoomed-in domain. The local fit is likely to agree well with the global fit on the restricted domain, and is much cheaper to construct.

restrict the domain to a smaller region as indicated by the dashed black box and only fit the evaluation data within that region. We still obtain a good surrogate (blue curve) around the optimum, and it is cheaper as we are using fewer evaluations to do so. We now proceed with our optimization, treating the restricted region as our new domain and the local fit as our surrogate for optimization. This idea of recursively optimizing over hierarchical domains lies at the heart of our algorithm. In this work, we call this technique the "zoom strategy". Because it requires less evaluation data to build a local surrogate than to build a global one, the zoom strategy can significantly reduce the cost of the algorithm.

For ease of describing the relationships between different domains, we introduce the notion of a node. A node consists of a domain together with all the information needed by an optimization algorithm to make progress for that domain. We call the process of restricting the domain to a smaller domain the "zoom-in" process, in which case the node associated with the original domain is a "parent" node and the node for the restricted domain is a "child" node. The reverse process of zooming in is referred to as the "zoom-out" process (i.e., the transition from a child node to its parent node). See Fig. 3.3 for an illustration of this structure.

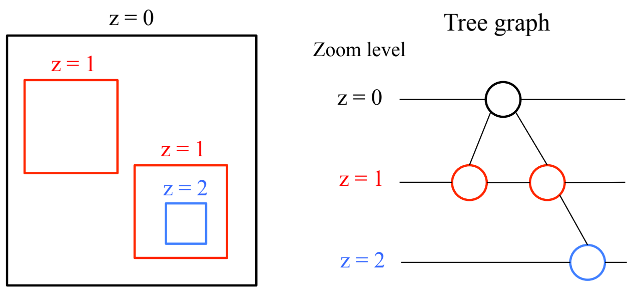

Figure 3.3: Illustration of the tree structure of ProSRS algorithm on a 2-D problem. The black box on the left shows the domain of a root node. The two red boxes and one blue box show two children and one grandchild of the root node.

3.2.1 Algorithm overview

We now present our algorithm, namely Progressive Stochastic Response Surface (ProSRS)[1], in Alg. 2. Like most surrogate optimization algorithms, ProSRS starts with a space-filling design of experiments (DOE). Here we use Latin hypercube sampling with maximin criterion for the initial design. In our algorithm, a node $\mathcal{N}$ is formally defined by a quadruplet:

$$\mathcal{N} = (D, \Omega, S, \beta), \tag{3.2}$$

where D is the evaluation data in the domain Ω. The variable S characterizes the exploitation (versus exploration) strength of ProSRS. Mathematically, it is a tuple:

$$S = (\gamma, p, \sigma), \tag{3.3}$$

where γ is a radial basis regression parameter (see Section 3.2.2) and p, σ are two parameters in the step of proposing new points (see Section 3.2.3). The variable β in Eq. 3.2 is the zoom-out probability.

For each iteration, we first construct a radial-basis surrogate using the evaluation data D (Line 7), followed by the step of proposing new points for parallel evaluation (Line 8). The proposed points must not only exploit the optimal locations of a surrogate, but also explore the untapped regions in the domain to improve the quality of the surrogate. Indeed, achieving the appropriate balance between exploitation and exploration is the key to the

[1]Code is publicly available at `https://github.com/compdyn/ProSRS`.

Algorithm 2 Progressive Stochastic Response Surface (ProSRS)

```
1: **Inputs:** $m$, $\beta_{\text{init}}$, $S_{\text{init}}$ and $N$
2: Generate $m$ Latin hypercube samples: $X = (x_1, x_2, \ldots, x_m)$
3: Evaluate samples $X$ in parallel to give $Y = (y_1, y_2, \ldots, y_m)$
4: Initialize the current node $= (D, \Omega, \beta, S)$ with evaluation data $D =$
   $(X, Y)$, domain $\Omega$ = optimization domain $\mathcal{D}$, zoom-out probability
   $\beta = \beta_{\text{init}}$ and variable $S = S_{\text{init}}$
5: **for** iteration $= 1, 2, \ldots, N$ **do**
6:     Obtain $D, \Omega, \beta, S$ from the current node
7:     $g \leftarrow \text{RBF}(D, S)$                          ▷ Build surrogate (Sect. 3.2.2)
8:     $X_{\text{new}} \leftarrow \text{SRS}(D, \Omega, S, g)$                 ▷ Propose points (Sect. 3.2.3)
9:     $Y_{\text{new}} \leftarrow$ evaluate samples $X_{\text{new}}$ in parallel
10:    Augment evaluation data $D$ with proposed points $(X_{\text{new}}, Y_{\text{new}})$
11:    Update the variable $S$ of current node                  ▷ see Sect. 3.2.4
12:    **if** $S$ reaches the critical value **then**
13:        **if** restart condition is met **then**
14:            Restart from DOE
15:        **else**
16:            Create or update a child node            ▷ Zoom in (Sect. 3.2.5)
17:            Reset variable $S$ of current node
18:            Set the child node to be the current node
19:        **end if**
20:    **end if**
21:    **if** no restart **and** the parent of current node exists **then**
22:        With probability $\beta$, set its parent node to be the current node  ▷
   Zoom out
23:    **end if**
24: **end for**
25: **return** $x_{\text{best}}$ = the sample with the lowest $y$ value
```

success of a surrogate optimization algorithm. For this, we use an efficient procedure, known as Stochastic Response Surface (SRS) method, that was first developed by Regis and Shoemaker [66] and later extended to the parallel setting in their subsequent work [75].

After performing expensive evaluations in parallel, we update the exploitation strength variable S (Line 11) so that for a specific node, the exploitation strength progressively increases with the number of iterations (see Section 3.2.4 for the update rule). The purpose of this step is to help locate the optimal region for zooming in. Once the exploitation strength reaches some prescribed threshold (Line 12; see Section 3.2.5 for details), the algorithm will decide to zoom in (Line 16) by setting a child to be the current node (neglecting the restart step in Line 14 for now). The updating of the variable S and the zoom-in mechanism generally make ProSRS "greedier" as the number of iterations increases. To balance out this increasing greediness over iterations, we implement a simple ϵ-greedy policy by allowing the algorithm to zoom out with some small probability in each iteration (Line 22). Because of the mechanism of zooming in and out, ProSRS will generally form a "tree" during the optimization process, as illustrated in Fig. 3.3.

Finally we would like to address the restart steps (Line 13 and 14) in Alg. 2. We make the algorithm restart completely from scratch when it reaches some prescribed resolution after several rounds of zooming in. Specifically, to check whether to restart, we first perform the step of creating or updating a child node like the normal zoom-in process (Line 16). Suppose the resulted child node has n points in its domain $\Omega \subseteq \mathbb{R}^d$, then ProSRS will restart if for all $i = 1, 2, \ldots, d$,

$$n^{-\frac{1}{d}} \ell_i(\Omega) < r(b_i - a_i), \tag{3.4}$$

where $r \in (0, 1)$ is a prescribed resolution parameter, $\ell_i(\Omega)$ denotes the length of the domain Ω in the i^{th} dimension, a_i and b_i are the bounds for the optimization domain $\mathcal{D}$ (Eq. 3.1). The reason for restarting from a DOE is to avoid the new runs being biased by the old runs so that the algorithm has a better chance to discover other potentially optimal regions. Indeed, extensive study [66, 75, 76] has shown that restarting from the initial DOE is better than continuing the algorithm with past evaluations.

3.2.2 Weighted radial basis regression

Given the evaluation data $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, a radial basis surrogate takes the form

$$g(x) = \sum_{i=1}^{n} c_i \phi(\|x - x_i\|), \quad x \in \mathbb{R}^d, \tag{3.5}$$

where the function ϕ is a radial basis function. In this work, we choose ϕ to be a multiquadric function. The radial basis coefficients c_i are obtained by minimizing the L_2-regularized weighted square loss:

$$\text{Loss} = \sum_{j=1}^{n} e^{\gamma \hat{y}_j} \big(y_j - g(x_j)\big)^2 + \lambda \sum_{j=1}^{n} c_j^2, \quad \text{with} \quad \hat{y}_j = \frac{y_j - \min y_k}{\max y_k - \min y_k}, \tag{3.6}$$

where γ is a non-positive weight parameter (one component of the variable S; see Eq. 3.3) and λ is a regularization constant determined automatically through cross validation. This loss function is quadratic in the coefficients c_i so that the minimization problem admits a unique solution and can be solved efficiently.

The term $e^{\gamma \hat{y}_j}$ in Eq. 3.6 represents the weight for the j^{th} sample, and $\hat{y}_j$ can be interpreted as the normalized y value with the understanding that $\hat{y}_j = 0$ if $\max y_k = \min y_k$. It is clear that $\gamma = 0$ disables the weighting in the RBF regression. When γ is negative, the samples with smaller y values gain more weight, so the RBF regression produces a better fit for the samples with low y values (the "best" samples). Consequently, smaller weight parameter γ values imply greater exploitation.

3.2.3 Stochastic response surface method

To propose new points for parallel evaluations, we use the general Stochastic Response Surface (SRS) framework [66, 75]. The first step of the stochastic response surface method is to randomly generate candidate points in the domain Ω. In the original SRS work [66], the authors introduced two types of candidate points and proposed one algorithm for each type. Here we consider the candidate points to be a mixture of both types.

Type I candidate points are sampled uniformly over the domain. Type II candidate points are generated by adding Gaussian perturbations around the current best point x^*, where x^* is the point in the evaluation data D with the lowest value of the RBF surrogate g. The covariance matrix for the Gaussian perturbation is a diagonal matrix with its diagonal being $\sigma^2 l_i^2(\Omega)$ $(i = 1, 2, \ldots, d)$, where σ is one component of the variable S (see Eq. 3.3) and $l_i(\Omega)$ is the length of the domain in the i^{th} dimension. Any generated point that lies outside the domain would be replaced by the nearest point in the domain so that all the Type II candidate points are within Ω. The proportion of these two types of candidate points is controlled by a parameter p, which is another component of the variable S. Specifically, we generate $1000d$ candidate points with a fraction of $\frac{1}{10}\lfloor 10p \rfloor$ points being Type I and the remainder being Type II.

The second step is to measure the quality of each candidate point using two criteria: the value of the response surface (RBF surrogate) and the minimum distance from previously evaluated points. The points with low response values are of high exploitation value, while the ones with large minimum distances are of high exploration value. In the SRS method, every candidate point is given a score on each of the two criteria, and a weight between 0 and 1 is used for trading off one criterion for the other. For our algorithm, we generate an array of weights that are equally-spaced in the interval $[0.3, 1]$ with the number of weights being equal to the number of parallel cores (if the number of cores is one, we alternate weights between 0.3 and 1 from iteration to iteration). This weight array, also known as the "weight pattern" in the original work [66], is used to balance between exploitation and exploration among the proposed points. The procedures of scoring the candidate points and selecting the proposed points from the candidate points based on the weight pattern are described in detail in Regis and Shoemaker [75].

3.2.4 Update procedure for variable S

After obtaining new evaluations, we update the variable S of the current node (Line 11 of Alg. 2). The goal of this updating step is to gradually increase the exploitation strength. As listed in Eq. 3.3, the variable S of a node consists of 3 parameters: (1) a weight parameter γ for radial basis regression, (2) a parameter p that controls the proportion of Type I candidate points in the

SRS method, and (3) a parameter σ that determines the spread of Type II candidate points. The exploitation strength will be enhanced by decreasing any of these 3 parameters.

Algorithm 3 Update p, σ and γ

```
if $p \geq 0.1$ then
    $p \leftarrow p n_{\text{eff}}^{-\frac{1}{d}}$
else if the counter for number of consecutive failed iterations = $C_{\text{fail}}$ then
    Reset the counter
    $\sigma \leftarrow \sigma/2$ and $\gamma \leftarrow \gamma - \Delta\gamma$
end if
```

The update rule is specified in Alg. 3, which can be understood as having two separate phases. The first phase is when there are still some Type I candidate points generated in the SRS method (i.e., $p \geq 0.1$). During this phase, the values of σ and γ are unchanged but the p value is decreased with each iteration. The rate of decrease is determined by $n_{\text{eff}}^{-1/d}$, where n_{eff} is the effective number of evaluations for the current iteration. The effective number of evaluations n_{eff} is computed by first uniformly partitioning the domain Ω into cells with the number of cells per dimension being equal to $\lceil n^{1/d} \rceil$, where n is the number of points in the evaluation data D. Then n_{eff} is number of cells that are occupied by at least one point. The quantity $n_{\text{eff}}^{1/d}$ can be viewed as a measurement of the density of the evaluated points in the domain Ω. Therefore, we essentially make the decreasing rate proportional to the evaluation density.

When the p value drops below 0.1, so that all the candidate points are Type II, we enter the second phase of the state transition, where the parameter p does not change but σ and γ are reduced. Just like in Regis and Shoemaker [66], we use the number of consecutive failures as the condition for deciding when to reduce the value of σ. Here an iteration is counted as a failure if the best y value of the proposed points for the current iteration does not improve the best y value of the evaluations prior to the proposing step. The counter is set to zero at the beginning of the algorithm, and starts to count the number of consecutive failures only when $p < 0.1$. Whenever the number of consecutive failures reaches some prescribed threshold C_{fail}, we reduce σ by half and decrease γ by $\Delta\gamma$.

3.2.5 Zoom Strategy

The updating of the variable S (Line 11) will make the parameter σ gradually decrease over iterations. Once σ drops below some critical value σ_{crit} (i.e., S reaches the critical value in Line 12) and the restart condition is not satisfied, the algorithm will zoom in by either creating a new child node or updating an existing child node. Specifically, we start the zoom-in process by finding the point that has the lowest fit value among the evaluation data D, which we will denote as x^*. Depending on the location of x^* and the locations of the children of the current node, there are two possible scenarios.

The first scenario is that x^* does not belong to the domain of any of the existing child nodes or there is no child for the current node. In this case a new child node is created. The domain Ω of this child node is generated by shrinking the domain of the current node with the center being at x^* and the length of each dimension being ρ-fractional of that of the current domain. The parameter $\rho \in (0, 1)$ is called the zoom-in factor, which is a constant set prior to the start of the algorithm. After shrinkage, any part of the new domain that is outside the current domain will be clipped off so that the domain of a child is always contained by that of its parent. Given the domain of the new child node, its evaluation data D are all the past evaluations that are within this domain. The zoom-out probability β and the variable S of this child node are set to the initial values β_{init} and S_{init} respectively.

The other possibility is that x^* belongs to at least one child of the current node. Among all children whose domains contain x^*, we select the child whose domain center is closest to x^*. The evaluation data D of this selected child node is updated by including all the past evaluations that are within its domain. Since the selected child node is being revisited, we reduce its zoom-out probability by $\beta \leftarrow \max(\beta/2, \beta_{\text{min}})$, where β_{min} is a constant lower bound for the zoom-out probability.

3.3 Convergence

In this section we state a convergence theorem for our ProSRS algorithm (Alg. 2). More specifically, if ProSRS is run for sufficiently long, with probability converging to one there will be at least one sample among all the evaluations that will be arbitrarily close to the global minimizer of the objective function. Because the point returned in each iteration is the one with the lowest noisy evaluation (not necessarily with the lowest expected value), as the underlying expectation function is generally unknown, this theoretical result does not immediately imply the convergence of our algorithm. However, in practice one may implement posterior Monte Carlo procedures for choosing the true best point from the evaluations (see Section 3.5.2).

Theorem 3.1 *Suppose the objective function F in Eq. 3.1 is continuous on the domain $\mathcal{D} \subseteq \mathbb{R}^d$ and x_{opt} is the unique minimizer of F, characterized by*[1] *$F(x_{opt}) = \inf_{x\in\mathcal{D}} F(x) \in (-\infty, +\infty)$ and $\inf_{x\in\mathcal{D}, \|x-x_{opt}\|\geq\eta} F(x) > F(x_{opt})$ for all $\eta > 0$. Let x_n be the sample with the minimum objective value among all the samples up to iteration n. Then $x_n \to x_{opt}$ almost surely as $n \to \infty$.*

Proof. We define the zoom level z to be zero for the root node and, whenever zooming in occurs, the zoom level of the child node is one plus that of its parent node so that every node in the tree is associated with a unique zoom level (see Fig. 3.3).

First, we argue that there is an upper bound on the zoom level for ProSRS algorithm. Since after each zoom-in step, the size of the domain is shrunk by at least the zoom-in factor $\rho \in (0,1)$, the domain length for a node of a zoom level $z \in \mathbb{N}$ is upper bounded by $\rho^z(b_i - a_i)$ for each dimension $i = 1, 2, \ldots, d$. Here a_i and b_i are the domain boundaries for the root node (Eq. 3.1). Now let us consider a node with zoom level $z^* = \lceil \log_\rho r \rceil + 1$, where $r \in (0,1)$ is the prescribed resolution parameter for the restart (see Eq. 3.4). We further denote the domain length of this node in each dimension to be ℓ_i and the number of evaluation points within its domain to be n, then we have for all $i = 1, 2, \ldots, d$,

$$n^{-\frac{1}{d}}\ell_i \leq \ell_i \leq \rho^{z^*}(b_i - a_i) = \rho^{\lceil \log_\rho r\rceil + 1}(b_i - a_i) < \rho^{\log_\rho r}(b_i - a_i) = r(b_i - a_i),$$

which would satisfy the restart condition (Eq. 3.4). This implies that the zoom level of ProSRS must be less than z^*. In other words, the zoom level is upper bounded by $z_{\max} = z^* - 1 = \lceil \log_\rho r \rceil$.

Now fix some $\epsilon > 0$ and define $\Delta := \max(z_{\max}, N_{\text{DOE}} + 1)$, where N_{DOE} is

[1] Here we adopt the convention that if $\{x \in \mathcal{D}, \|x - x_{\text{opt}}\| \geq \eta\} = \emptyset$, then $\inf_{x\in\mathcal{D}, \|x-x_{\text{opt}}\|\geq\eta} F(x) = +\infty$.

the number of iterations for the initial space-filling design. The main idea of the following proof is similar to that in the original SRS paper [66].

Since the objective function F is continuous at the unique minimizer x_{opt}, there exists $\delta(\epsilon) > 0$ so that whenever x is within the open ball $\mathcal{B}(x_{\text{opt}}, \delta(\epsilon))$, $f(x) < f(x_{\text{opt}}) + \epsilon$.

The probability that a candidate point generated in the root node (of either Type I or Type II) is located within the domain $\mathcal{B}(x_{\text{opt}}, \delta(\epsilon)) \cap \mathcal{D}$ can be shown to be bounded from below by some positive $\nu(\epsilon)$ (see Section 2 of Regis and Shoemaker [66]). Here $\mathcal{D}$ is the domain of the optimization problem (Eq. 3.1). Since all the candidate points are generated independently, the probability that all the candidate points are within $\mathcal{B}(x_{\text{opt}}, \delta(\epsilon)) \cap \mathcal{D}$ is greater than or equal to $L(\epsilon) := \nu(\epsilon)^t > 0$, where t is a constant denoting the number of candidate points.

Now we define a positive quantity $h(\epsilon) := L(\epsilon)(\beta_{\min})^{\Delta}$, where $\beta_{\min}$ is the minimum zoom-out probability (see Section 3.2.5). We further define the event

$$\begin{aligned} A_i := \{&\text{for each of the iterations } (i-1)\Delta + 1, (i-1)\Delta + 2, \ldots, i\Delta, \text{ there is} \\ &\text{at least one candidate point that lies outside the domain} \\ &\mathcal{B}(x_{\text{opt}}, \delta(\epsilon)) \cap \mathcal{D}\}, \quad i \in \mathbb{Z}^+. \end{aligned}$$

Let probability $P_i := P(A_i \mid A_1 \cap A_2 \cap \ldots \cap A_{i-1})$ with the understanding that $P_1 = P(A_1)$. Then

$$P(A_1 \cap A_2 \cap \ldots \cap A_k) = \prod_{i=1}^{k} P_i, \quad k \in \mathbb{Z}^+. \tag{3.7}$$

For now, let us assume $i > 1$. For the iteration $(i-1)\Delta$, there are 3 possible events that could happen when we are about to run Line 21 of the ProSRS algorithm (Alg. 2):

$$\begin{aligned} E_1 &= \{\text{decide to restart}\}, \\ E_2 &= \{\text{decide not to restart and the parent node exists}\}, \\ E_3 &= \{\text{decide not to restart and the parent node does not exist}\}. \end{aligned}$$

Let z_{i-1} be the zoom level of the current node at this moment. Then we have the following inequalities:

$$\begin{aligned}
&P(\overline{A_i} \mid A_1 \cap A_2 \cap \ldots \cap A_{i-1} \cap E_1) \\
&= P(\text{among iterations } (i-1)\Delta + 1, (i-1)\Delta + 2, \ldots, i\Delta, \text{there exists one} \\
&\qquad \text{iteration for which all the candidate points are within domain} \\
&\qquad \mathcal{B}(x_{\text{opt}}, \delta(\epsilon)) \cap \mathcal{D} \mid A_1 \cap A_2 \cap \ldots \cap A_{i-1} \cap E_1) \\
&\geq P\Big(\text{all the candidate points are within } \mathcal{B}(x_{\text{opt}}, \delta(\epsilon)) \cap \mathcal{D} \text{ for iteration} \\
&\qquad \big((i-1)\Delta + N_{\text{DOE}} + 1\big) \mid A_1 \cap A_2 \cap \ldots \cap A_{i-1} \cap E_1\Big) \geq L(\epsilon) \geq h(\epsilon) \\
&P(\overline{A_i} \mid A_1 \cap A_2 \cap \ldots \cap A_{i-1} \cap E_2) \\
&\geq P\Big(\text{decide to zoom out for iterations } (i-1)\Delta, (i-1)\Delta + 1, \ldots, \\
&\qquad (i-1)\Delta + z_{i-1} - 1 \text{ and all the candidate points are within} \\
&\qquad \mathcal{B}(x_{\text{opt}}, \delta(\epsilon)) \cap \mathcal{D} \text{ for the iteration } \big((i-1)\Delta + z_{i-1}\big) \\
&\qquad \mid A_1 \cap A_2 \cap \ldots \cap A_{i-1} \cap E_2\Big) \geq L(\epsilon)(\beta_{\min})^{z_{i-1}} \geq h(\epsilon) \\
&P(\overline{A_i} \mid A_1 \cap A_2 \cap \ldots \cap A_{i-1} \cap E_3) \\
&\geq P\Big(\text{all the candidate points are within } \mathcal{B}(x_{\text{opt}}, \delta(\epsilon)) \cap \mathcal{D} \text{ for the iteration} \\
&\qquad \big((i-1)\Delta + 1\big) \mid A_1 \cap A_2 \cap \ldots \cap A_{i-1} \cap E_3\Big) \geq L(\epsilon) \geq h(\epsilon).
\end{aligned}$$

That is, for any $i > 1$, $P(\overline{A_i} \mid A_1 \cap A_2 \cap \ldots \cap A_{i-1} \cap E_j) \geq h(\epsilon)$ for all $j = 1, 2, 3$. Hence, $P(\overline{A_i} \mid A_1 \cap A_2 \cap \ldots \cap A_{i-1}) \geq h(\epsilon)$, which implies $P_i \leq 1 - h(\epsilon)$ for any $i > 1$. Now if $i = 1$, again we have $P_1 = 1 - P(\overline{A_1}) \leq 1 - h(\epsilon)$ because the probability that all the candidates are within $\mathcal{B}(x_{\text{opt}}, \delta(\epsilon)) \cap \mathcal{D}$ for the iteration $(N_{\text{DOE}} + 1)$ is greater or equal to $h(\epsilon)$. Therefore, $P_i \leq 1 - h(\epsilon)$ holds true for all $i \in \mathbb{Z}^+$. Using Eq. 3.7, we have

$$P(A_1 \cap A_2 \cap \ldots \cap A_k) \leq \big(1 - h(\epsilon)\big)^k. \tag{3.8}$$

Since $h(\epsilon) \in (0, 1)$, $P(A_1 \cap A_2 \cap \ldots \cap A_k)$ converges to zero, or equivalently

$P(\overline{A_1 \cap A_2 \cap \ldots \cap A_k})$ converges to one as $k \to \infty$. Observe that

$$\begin{aligned}
&\overline{A_1 \cap A_2 \cap \ldots \cap A_k} \\
&= \{\text{among iterations } 1, 2, \ldots, k\Delta, \text{there is an iteration for which all the} \\
&\qquad \text{candidate points are within } \mathcal{B}(x_{\text{opt}}, \delta(\epsilon)) \cap \mathcal{D}\} \\
&\subseteq \{\text{among iterations } 1, 2, \ldots, k\Delta, \text{there is an evaluation sample } x \text{ within} \\
&\qquad \mathcal{B}(x_{\text{opt}}, \delta(\epsilon)) \cap \mathcal{D}\} \\
&\subseteq \{\text{among iterations } 1, 2, \ldots, k\Delta, \text{there is an evaluation sample } x \text{ such} \\
&\qquad \text{that } f(x) < f(x_{\text{opt}}) + \epsilon\} \\
&\subseteq \{f(x_{k\Delta}) < f(x_{\text{opt}}) + \epsilon\} = \{|f(x_{k\Delta}) - f(x_{\text{opt}})| < \epsilon\}.
\end{aligned}$$

Hence, $f(x_{k\Delta})$ converges to $f(x_{\text{opt}})$ in probability as $k \to \infty$. Therefore, there is a subsequence of $\big(f(x_{k\Delta})\big)_{k\in\mathbb{N}}$ which is also a subsequence of $\big(f(x_n)\big)_{n\in\mathbb{N}}$, that converges almost surely to $f(x_{\text{opt}})$. Because $f(x_n)$ is monotonically decreasing so that the limit always exists, $f(x_n)$ converges to $f(x_{\text{opt}})$ almost surely. Finally, by the uniqueness of the minimizer, x_n converges to x_{opt} almost surely. The arguments for the last two almost-sure convergences are essentially the same as those used in proving the convergence of a simple random search algorithm (see the proof of Theorem 2.1 in Spall [59]).

□

3.4 Numerical results

In this section, we compare our ProSRS algorithm to three state-of-the-art parallel Bayesian optimization algorithms: GP-EI-MCMC [52], GP-LP [72] with acquisitions LCB and EI. The parameter values of ProSRS algorithm are listed in Table 3.1, where d is optimization dimension and N_{core} is the number of parallel cores.

For test problems, we first used a suite of standard optimization benchmark problems from the literature. Table 3.2 summarizes the conditions for all the benchmark experiments. For each benchmark problem, a Gaussian noise was added to the true underlying function. We tested with commonly-used optimization domains, and the standard deviation of the noise roughly matched the range of a function. The function expressions for these bench-

Table 3.1: Parameter values for the ProSRS algorithm

Parameter	Meaning	Value
m	number of DOE samples	$\lceil 3/N_{\text{core}} \rceil N_{\text{core}}$
S_{init}	initial value of variable S	(0, 1, 0.1)
σ_{crit}	critical σ value	0.025
β_{init}	initial zoom-out probability	0.02
β_{min}	minimum zoom-out probability	0.01
ρ	zoom-in factor	0.4
r	resolution parameter for restart	0.01
C_{fail}	critical number of consecutive failures	$\max(\lceil d/N_{\text{core}} \rceil, 2)$
$\Delta\gamma$	change of γ value	2

Table 3.2: Experiment conditions for optimization benchmark problems (the last numeric figure in the function name is the problem dimension)

Function	Optimization Domain	Std. of Gaussian noise
Ackley10	$[-32.768, 32.768]^{10}$	1
Alpine10	$[-10, 10]^{10}$	1
Griewank10	$[-600, 600]^{10}$	2
Levy10	$[-10, 10]^{10}$	1
SumPower10	$[-1, 1]^{10}$	0.05
SixHumpCamel2	$[-3, 3] \times [-2, 2]$	0.1
Schaffer2	$[-100, 100]^{2}$	0.02
Dropwave2	$[-5.12, 5.12]^{2}$	0.02
Goldstein-Price2	$[-2, 2]^{2}$	2
Rastrigin2	$[-5.12, 5.12]^{2}$	0.5
Hartmann6	$[0, 1]^{6}$	0.05
PowerSum4	$[0, 4]^{4}$	1

mark problems are given in detail in Appendix E.

Next, we test algorithms on real hyperparameter-tuning problems. The problem of tuning hyperparameters of a machine learning model can be viewed as an expensive optimization problem in Eq. 3.1. In this case, the function f is a validation or cross-validation error for a machine learning model and the vector x represents the hyperparameters to be tuned. The function f is typically expensive since one evaluation of f involves training and scoring one or multiple machine learning models. The noise associated with f may come from the fact that a machine learning algorithm (e.g., random forest) contains random elements or a stochastic optimization method (e.g., SGD) is invoked during the training process.

Specifically, two hyperparameter-tuning problems are considered: (1) tuning 5 hyperparameters of a random forest (2) tuning 7 hyperparameters of a deep neural network. For both problems, when tuning an integer-valued hyperparameter, we rounded the continuous output from an optimization algorithm to the nearest integer before feeding it to the machine learning algorithm. The next two paragraphs below give the details of the two hyperparameter-tuning problems:

Random forest. We tuned a random forest, one of the most widely used classification algorithms, on the well-known Adult dataset [77]. The dataset consists of 48842 instances with 14 attributes, and the task is to classify income based on census information. We tuned 5 hyperparameters: number of trees on $[1, 300]$, number of features on $[1, 14]$, maximum depth of a tree on $[1, 100]$, minimum number of samples for the node split on $[2, 1000]$ and minimum number of samples for a leaf node on $[1, 1000]$. We minimized the 5-fold cross-validation error.

Deep neural network. We tuned a feedforward deep neural network with 2 hidden layers on the popular MNIST dataset [78]. This tuning problem is also considered in [79]. We used the same training-validation data split as in the TensorFlow tutorial [80] with the training set having 55000 data points and the validation set having 5000 data points. We tuned 7 hyperparameters: number of units in each hidden layer on $[1, 100]$, L_1 and L_2 regularization constants, both on a log scale on $[10^{-8}, 10^0]$, learning rate on a log scale on $[10^{-4}, 10^0]$, batch size on $[50, 1000]$ and number of epochs on $[5, 50]$. We minimized the validation error.

3.4.1 Optimization performance versus iteration

The first results that we consider are the optimization result (function value) versus iteration number. All the algorithms are proposing and evaluating the same number of points per iteration, so these results measure the quality of these proposed points. As we will see, ProSRS does significantly better than the existing methods.

Figure 3.4 shows the optimization progress versus the number of iterations. The objective function on the y axis is the evaluation of the underlying true expectation function (not the noisy function) at the algorithm output. The error bar is the standard deviation of 20 independent runs. All algorithms are run with 12 parallel cores.

As we can see from Figure 3.4, our algorithm performs the best on almost all of the problems. In particular, ProSRS is significantly better on high-dimensional functions such as Ackley and Levy, as well as highly-complex functions such as Dropwave and Schaffer. Excellent performance on these benchmark problems shows that our algorithm can cope with various optimization landscape types.

Figure 3.5 shows optimization performance on the two hyperparameter-tuning problems. Here we include a random search algorithm as a baseline in addition to the optimization algorithms. First, we see that surrogate optimization algorithms are in general significantly better than the random search algorithm. This is no surprise as the surrogate optimization algorithm selects every evaluation point carefully in each iteration. Second, among the surrogate optimization algorithms, our ProSRS algorithm is better than the GP-EI-MCMC algorithm (particularly on the random forest tuning problem), and is much better than the two GP-LP algorithms.

3.4.2 Optimization performance analysis

In the previous section we demonstrated that our ProSRS algorithm generally achieved superior optimization performances compared to the Bayesian optimization algorithms. In this section, we give some insight into why our algorithm could be better. We performed the analysis with a numerical experiment that studied the modeling capability of RBF (as used in ProSRS) and GP models (as used in the Bayesian optimization methods).

More specifically, we investigated RBF and GP regression on the twelve optimization benchmark functions listed in Table 3.2, varying the number n of training data points from 10 to 100. For each test function and every n, we first randomly sampled n points $(X_1, X_2, \ldots, X_n)$ over the function domain using Latin hypercube sampling, and then evaluated these n samples to get noisy responses $(Y_1, Y_2, \ldots, Y_n)$. Then given the data (X_1, Y_1),

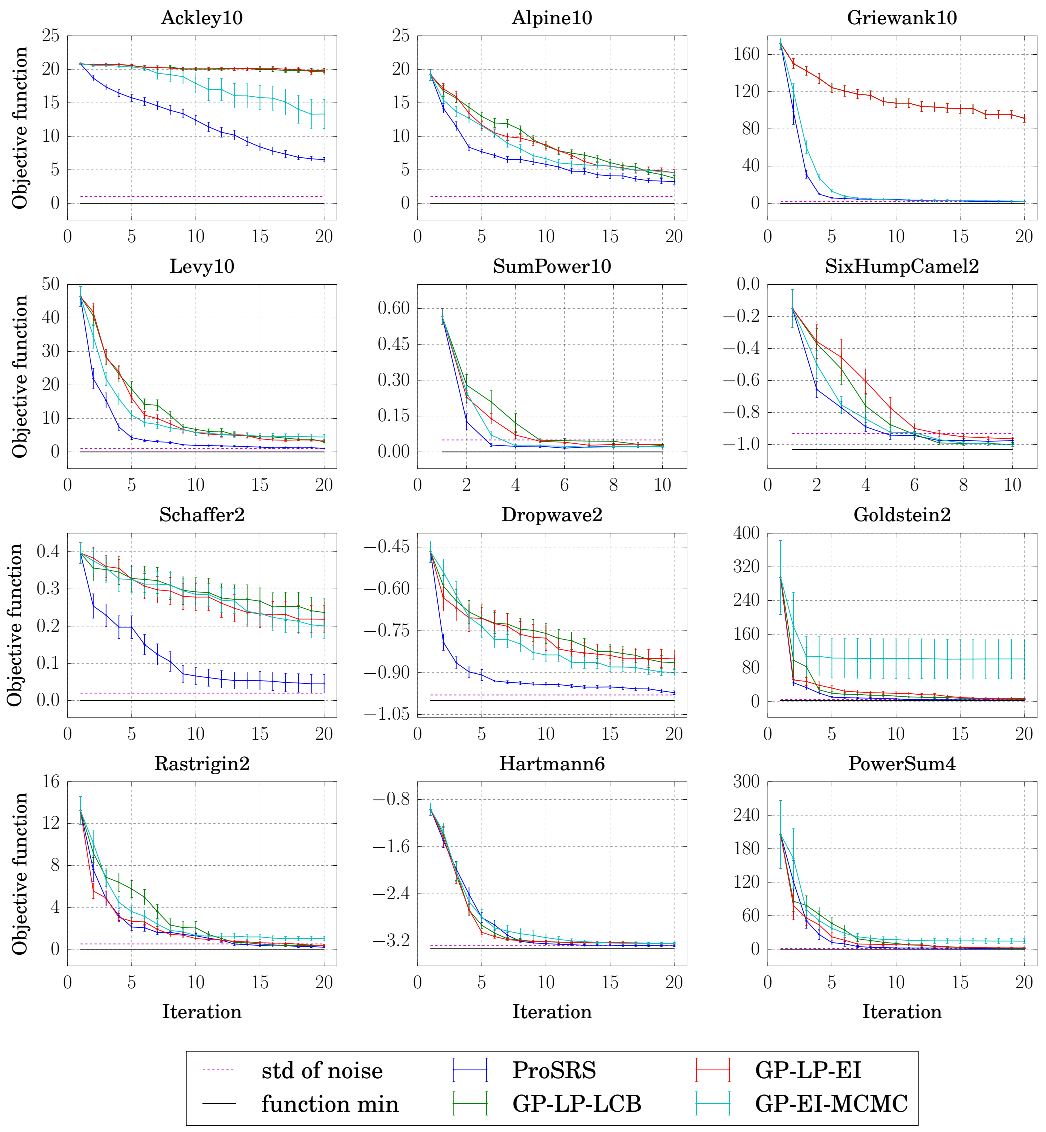

Figure 3.4: Optimization curves for the benchmark functions. The error bar shows the standard deviation of 20 independent runs.

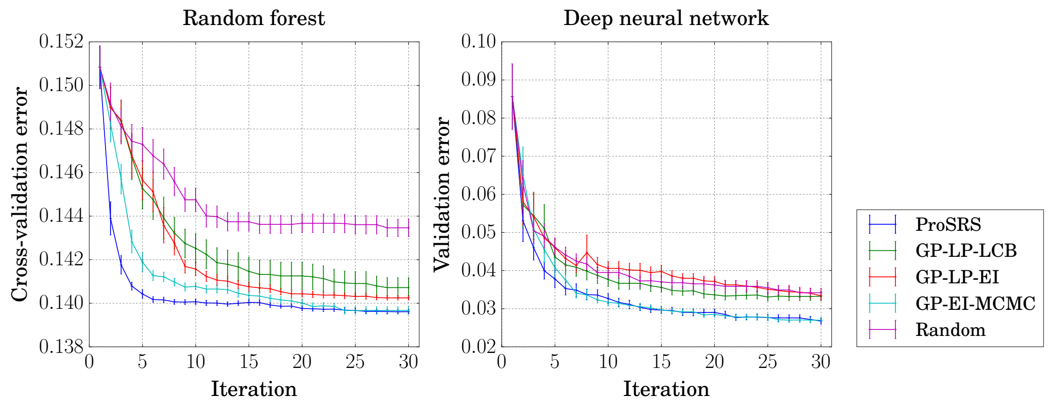

Figure 3.5: Optimization curves for the hyperparameter-tuning problems. The error bar shows the standard deviation of 20 independent runs. The number of parallel cores is 8 for both problems. The expected error (objective F) is estimated by averaging 5 independent samples.

$(X_2, Y_2), \ldots, (X_n, Y_n)$, we trained 4 models: a RBF model using the cross validation procedure developed in the ProSRS algorithm with no weighting, and 3 GP models with commonly used GP kernels: Matern1.5, Matern2.5 and RBF.

We used the Python scikit-learn package[1] for the implementations of GP regression. We set the number of restarts for the optimizer in GP regression to be 10. We evaluated each regression model by measuring the relative error in terms of the L_2 norm of the difference between a model g and the underlying true function $\mathbb{E}[f]$ over the function domain. We repeated the training and evaluation procedure for 10 times, and reported the mean and the standard deviation of the measured relative errors.

The results are shown in Figure 3.6. We can see that cross-validated RBF regression (as used in our ProSRS method) generally produces a better model than those from GP regression (as used in the Bayesian optimization methods). Specifically, the RBF model from ProSRS is significantly better for the test functions Griewank, Levy, Goldstein and PowerSum, and is on par with GP models for Schaffer, Dropwave and Hartmann.

From this numerical study, we can draw two conclusions. First, the ProSRS RBF models seem to be able to better capture the objective functions than GP regression models. One possible explanation for this is that the ProSRS

[1]Python package for Gaussian Processes: `http://scikit-learn.org/stable/modules/gaussian_process.html`.

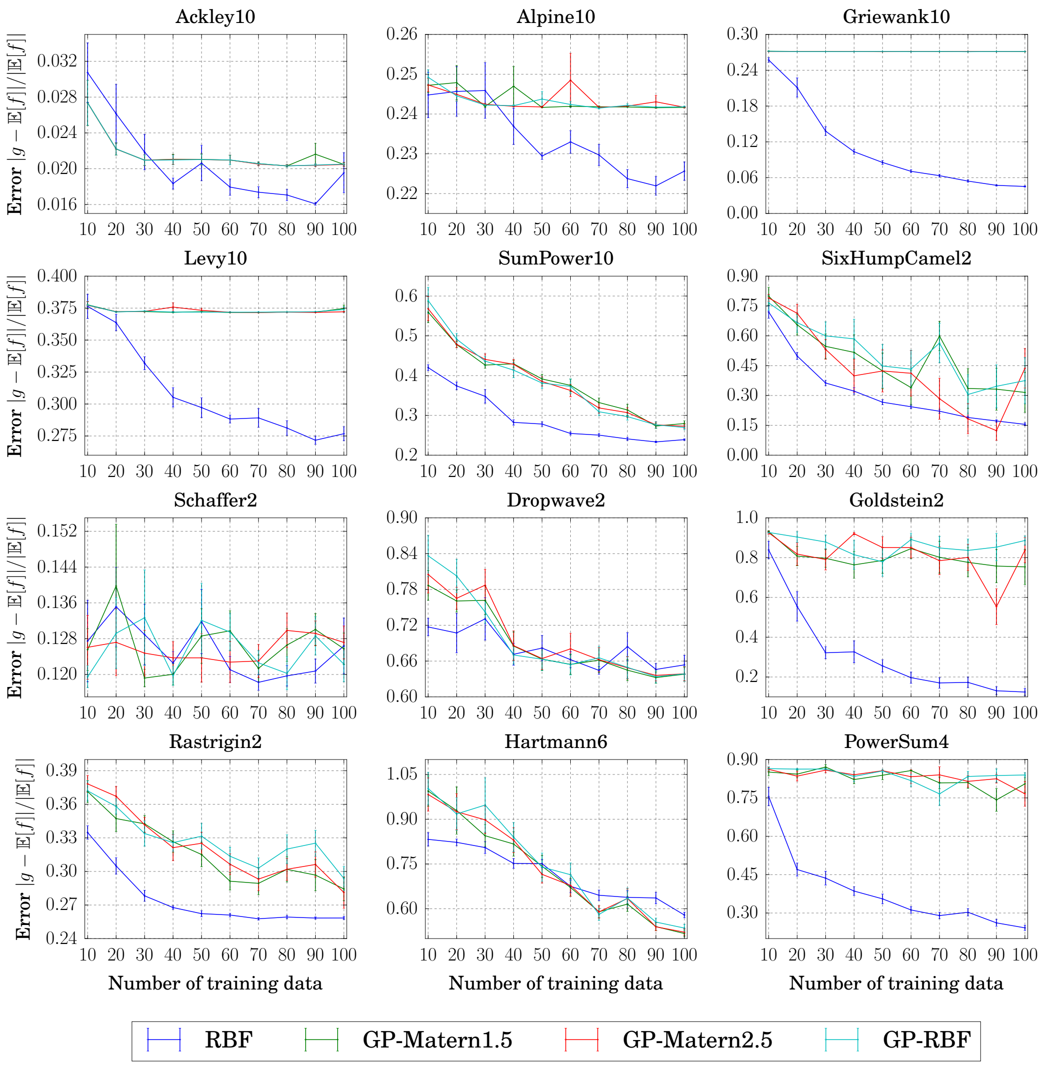

Figure 3.6: Compare the modeling capability of RBF regression as used in ProSRS (dark blue lines) and GP regression with kernels: Matern1.5, Matern2.5 and RBF (green, red and light blue lines respectively) on 12 optimization benchmark functions. The y axis is the relative error in terms of the L_2 norm of the difference between a model g and the underlying true function $\mathbb{E}[f]$ over the function domain. The error bar shows the standard deviation of 10 independent runs.

RBF regression uses a cross validation procedure so that the best model is selected directly according to the data, whereas GP regression builds models relying on the prior distributional assumptions about the data (i.e., Gaussian process with some kernel). Therefore, in a way the ProSRS regression procedure makes fewer assumptions about the data and is more "data-driven" than GP. Since the quality of a surrogate has a direct impact on how well the proposed points exploit the objective function, we believe that the superiority of the RBF models plays an important part in the success of our ProSRS algorithm over those Bayesian optimization algorithms.

Second, for those test functions where ProSRS RBF and GP have similar modeling performances (i.e., Schaffer, Dropwave and Hartmann), the optimization performance of ProSRS (using RBF) is nonetheless generally better than Bayesian optimization (using the GP models), as we can see from Figure 3.4. This suggests that with surrogate modeling performance being equal, the ProSRS sample selection strategy (i.e., SRS and zoom strategy) may still have an edge over the probablity-based selection criterion (e.g., EI-MCMC) of Bayesian optimization.

3.4.3 Algorithm cost

In Section 3.4.1 we saw that ProSRS achieved faster convergence per iteration, meaning that it was proposing better points to evaluate in each iteration. In this section we will compare the cost of the algorithms and show that ProSRS is in addition much cheaper per iteration. The main focus here is to compare the cost of the algorithm, not the cost of evaluating the function f since the function-evaluation cost is roughly the same among the algorithms.

Figure 3.7 and Figure 3.8 show the computational costs of running different algorithms for both optimization benchmark problems and the two hyperparameter-tuning problems. The time was benchmarked on Blue Waters[1] XE compute nodes. We observe that our ProSRS algorithm is generally about 1 to 4 orders of magnitude cheaper than the other algorithms. It is worth noting that for the hyperparameter-tuning problems, the cost of the GP-EI-MCMC algorithm is in fact consistently higher than that of the training and the evaluation of a machine learning model, and the cost gap becomes

[1]Blue Waters: `https://bluewaters.ncsa.illinois.edu`.

larger as the number of iterations increases.

From Fig. 3.8 we can see that the cost of our algorithm scales roughly $\sim \mathcal{O}(1)$ with the number of iterations in the long run (i.e., when the algorithm is run with a large number of iterations, the general trend of the cost stays flat with iterations). This scaling behavior is generally true for our algorithm, and is a consequence of the zoom strategy and the restart mechanism exploited by our algorithm.

3.4.4 Overall optimization efficiency

In this section, we will show the overall optimization efficiency for the two real hyperparameter-tuning problems, which takes into account not only the optimization performance per iteration but also the cost of the algorithm and the expensive function evaluations. From Fig. 3.9, we can see that our ProSRS algorithm is the best among all the algorithms. Because of the high cost of the GP-EI-MCMC algorithm, the advantage of our algorithm over GP-EI-MCMC becomes even more pronounced compared to that of the iteration-based performance measurement (Fig. 3.5).

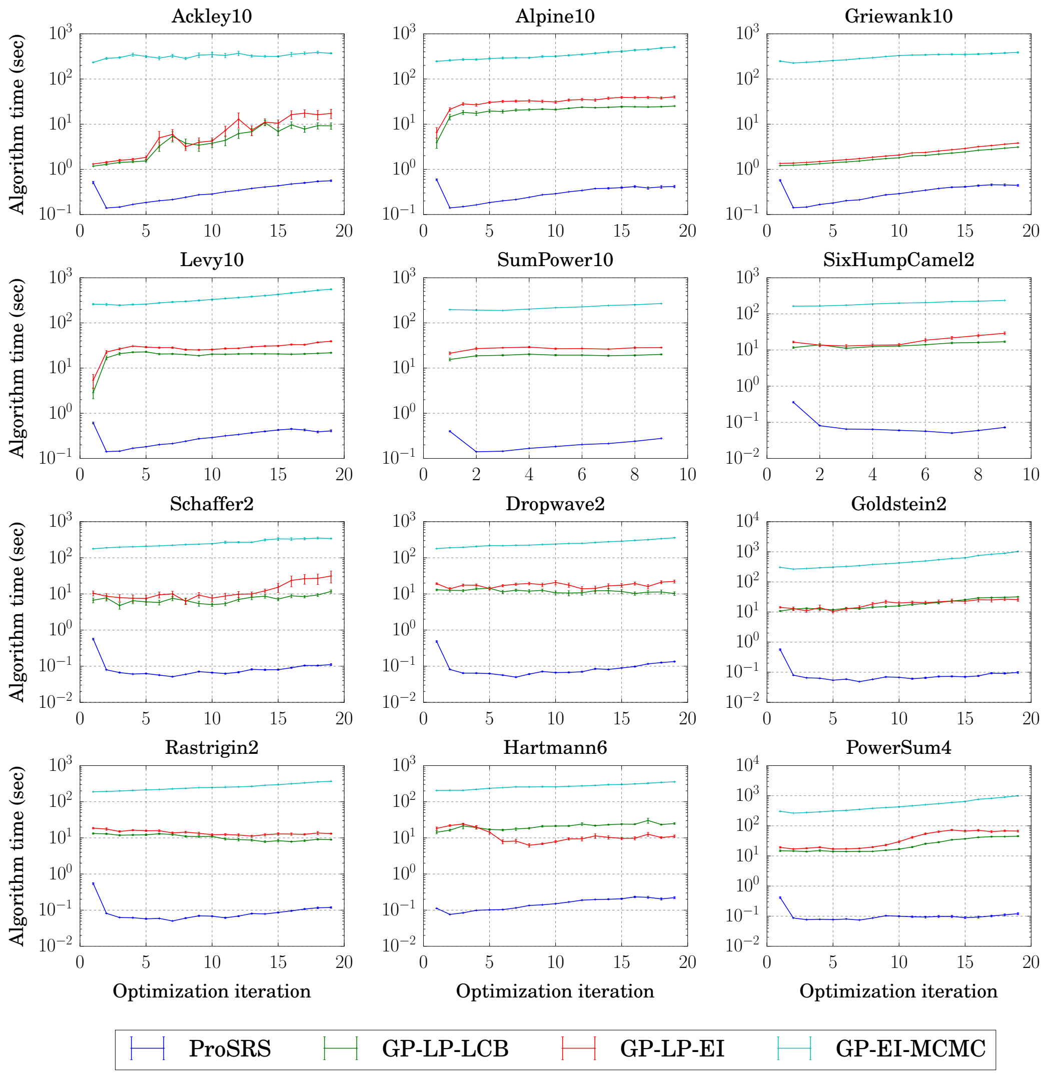

Figure 3.7: Computational costs of different algorithms for the twelve optimization benchmark problems. The plots show the mean and standard deviation of 20 independent runs. The x axis is the number of iterations in actual optimization excluding the initial DOE iteration. The y axis is the actual time that was consumed by an algorithm in each iteration, and does not include the time of parallel function evaluations.

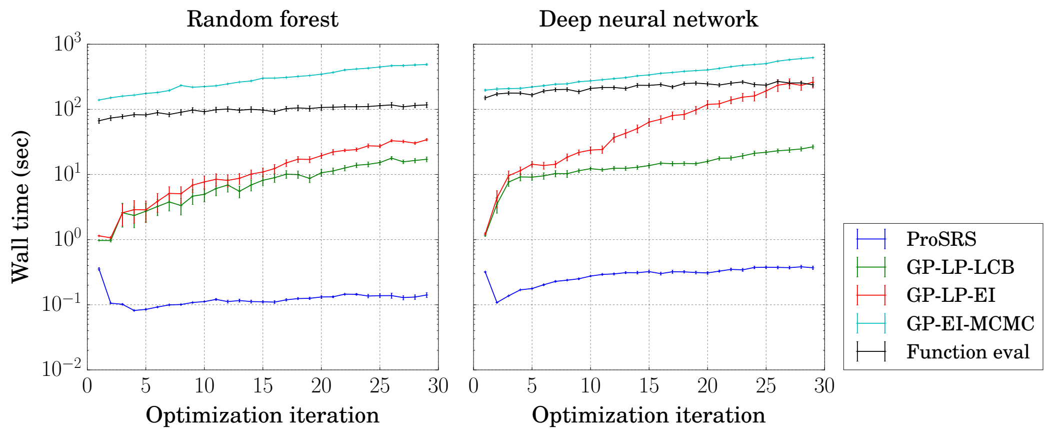

Figure 3.8: Computational costs of different algorithms for the two hyperparameter tuning problems. The plots show the mean and standard deviation of 20 independent runs. The x axis is the number of iterations in actual optimization excluding the initial DOE iteration. For different algorithms, the y axis is the actual time that was consumed by the algorithm in each iteration, and does not include the time of parallel function evaluations. The time for training and evaluating the machine learning models is shown in black.

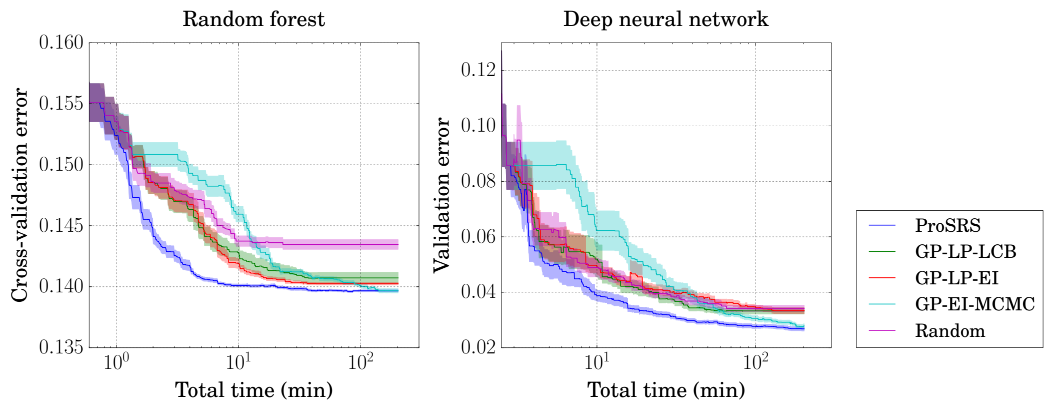

Figure 3.9: Optimization efficiency of different algorithms on the two hyperparameter-tuning problems. Total time on the horizontal axis is the actual elapsed time including both algorithm running time and time of evaluating expensive functions. The shaded areas show the standard deviation of 20 independent runs.

3.5 Application of ProSRS to a general problem

In the last section (Section 3.4), we demonstrated the effectiveness of the ProSRS algorithm by benchmarking it against several state-of-the-art Bayesian optimization algorithms on standard benchmark functions, as well as on two real hyperparameter-tuning problems. In this section, we will demonstrate, through a concrete example, the use of ProSRS for a general noisy expensive optimization problem. In this example we applied ProSRS algorithm to a model characterization problem. We will show a complete optimization workflow from defining the optimization problem, running the ProSRS algorithm, and finally selecting the best sample among evaluations.

3.5.1 A model characterization problem

We consider the problem of determining unknown parameters of a particle-resolved aerosol model, known as the PartMC model[1], to match the model output to the measurements from a particular set of laboratory chamber experiments. PartMC is a stochastic atmospheric aerosol model that simulates the evolution of aerosols at per-particle level using Monte Carlo methods. A detailed description of the model is available in Riemer et al. [14] and Zaveri et al. [34].

In this study, we simulated aerosol particles in a chamber environment with the PartMC model. We identified 12 parameters that need to be prescribed for the PartMC simulations but that are not well-constrained from the chamber experiments (see Table 3.3). We determined these unknown parameters by optimizing them over a predefined domain so that the error between the simulation outputs and the experimental measurements was minimized.

Specifically, the PartMC model was simulated with a total simulation time of 220 minutes, and the outputs were generated every two minutes in simulation. These outputs were then compared to two sets of experimental data: measurement of the overall size distribution with an SMPS instrument, and measurement of the black carbon core size distribution with an SP2 instru-

[1]PartMC code is open-source under the GNU General Public License (GPL) at `http://lagrange.mechse.illinois.edu/partmc/`.

Table 3.3: Unknown parameters in PartMC simulation

Parameter	Meaning	Domain
R_{AS}	filling inflow for AS particles	$[1, 4]$
R_{RB}	filling inflow for RB particles	$[1, 4]$
R_{dil_2}	dilution outflow during Period 2	$[0.5, 4]$
$s_{\mathrm{c,AS}}$	input scaling factor for AS particles	$[100, 400]$
$s_{\mathrm{c,RB}}$	input scaling factor for RB particles	$[100, 400]$
$s_{\mathrm{c,SMPS}}$	output scaling factor for SMPS measurements	$[100, 400]$
$s_{\mathrm{c,SP2}}$	output scaling factor for SP2 measurements	$[50, 300]$
d_{f}	fractal dimension	$[1.5, 3]$
R_0	radius of primary particles	$[3, 100]$
f	volume filling factor	$[1.35, 2]$
a	exponent in diffusive boundary layer thickness	$[0.2, 0.3]$
k_{D}	prefactor in diffusive boundary layer thickness	$[0.02, 0.1]$

ment. The error with respect to the SMPS measurements is given by

$$\epsilon_{\mathrm{SMPS}} = \sqrt{\frac{1}{T_1}\sum_{t=1}^{T_1}(\epsilon_{t,1})^2}, \tag{3.9}$$

where T_1 is the number of SMPS measurement times, and $\epsilon_{t,1}$ is the relative error of size distributions at time t, defined as

$$\epsilon_{t,1} = \sqrt{\frac{\sum_{i=1}^{N_1}\left(n_{\mathrm{PMC}}^{i,t} - n_{\mathrm{SMPS}}^{i,t}\right)^2}{\sum_{i=1}^{N_1}\left(n_{\mathrm{SMPS}}^{i,t}\right)^2}}. \tag{3.10}$$

Here $n_{\mathrm{PMC}}^{i,t}$ and $n_{\mathrm{SMPS}}^{i,t}$ represent the number concentration at size bin i and time t for the PartMC simulations and for the SMPS measurements respectively, and N_1 is the number of bins.

Similarly, the error between the PartMC simulations and the SP2 measurements is given by

$$\epsilon_{\mathrm{SP2}} = \sqrt{\frac{1}{T_2}\sum_{t=1}^{T_2}(\epsilon_{t,2})^2}, \quad \text{where } \epsilon_{t,2} = \sqrt{\frac{\sum_{i=1}^{N_2}\left(n_{\mathrm{PMC}}^{i,t} - n_{\mathrm{SP2}}^{i,t}\right)^2}{\sum_{i=1}^{N_2}\left(n_{\mathrm{SP2}}^{i,t}\right)^2}}. \tag{3.11}$$

The total error ϵ is the root mean square error (RMSE) over the two types of measurements:

$$\epsilon = \sqrt{\frac{\epsilon_{\text{SMPS}}^2 + \epsilon_{\text{SP2}}^2}{2}}. \tag{3.12}$$

Let us denote the unknown parameters in Table 3.3 as a 12-dimensional vector x. Then the number concentrations $n_{\text{PMC}}^{i,t}$, $n_{\text{SMPS}}^{i,t}$ and $n_{\text{SP2}}^{i,t}$ all depend on the value of x. As a result, the total error ϵ (Eq. 3.12) is also dependent on the vector x. Hence, finding the unknown parameters can be formally cast into solving a noisy optimization problem:

$$\underset{x \in \mathcal{D}}{\operatorname{argmin}}\, E(x) = \mathbb{E}_{\omega}[\epsilon(x, \omega)], \tag{3.13}$$

where the expected error E is the optimization objective function, ϵ is the total error defined in Eq. 3.12, ω captures the randomness in the PartMC simulations and $\mathcal{D}$ is the optimization domain given in Table 3.3. Here the optimization objective E is not observed directly but can be estimated via independent random samples of the function ϵ. Moreover, the function ϵ is expensive to evaluate since one evaluation requires running a PartMC simulation with some vector x and then computing the error according to Eq. 3.9–Eq. 3.12.

3.5.2 Optimization procedure

After the optimization problem was defined, we fed it into the ProSRS algorithm. We monitored the optimization progress versus iterations, and stopped running the algorithm when it appeared to have converged. After ProSRS completed, we needed to select, from all the samples evaluated by ProSRS, the one with the lowest expected error. Since the underlying expected error function E (Eq. 3.13) is unknown and we have only one noisy evaluation of function ϵ per sample, selecting the true best sample is not trivial. Indeed, this can be regarded as a discrete expensive optimization problem, more commonly known as a ranking and selection problem [44]. There are several ranking and selection algorithms in the literature such as Nelson et al. [81] and Ni et al. [82]. Here we present a very simple algorithm (Alg. 4) that we found works well in this example. In this algorithm, we started with choosing m top samples from the evaluations as the candidates

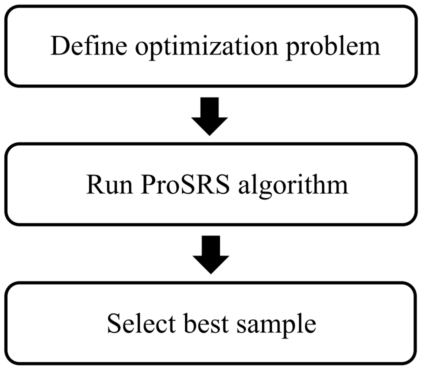

Figure 3.10: A general workflow for solving a noisy expensive optimization problem with ProSRS algorithm.

for the best sample, then performed Monte Carlo evaluations for each candidate sample, and finally selected the best sample to be the one with the lowest Monte Carlo mean estimate. The complete procedure for solving an optimization problem with ProSRS is illustrated in Fig. 3.10.

```
Algorithm 4 Select best sample from evaluations
  Input: evaluations during optimization: $\{(x_1, \epsilon_1), (x_2, \epsilon_2), \ldots, (x_n, \epsilon_n)\}$
  Parameters: number of candidates $m$ ($m \leq n$), number of repeats $R$
  Sorting $x_i$ based on value $\epsilon_i$ (ascending order) gives $(x'_1, x'_2, \ldots, x'_n)$
  Select top $m$ samples as candidates $C = (x'_1, x'_2, \ldots, x'_m)$
  Obtain mean estimate $\hat{\epsilon}_i$ for each sample in $C$ using $R$-repeat Monte Carlo
  Output: best sample $x^* = x'_j$, where $j = \operatorname{argmin}_{i=1,2,\ldots,m} \hat{\epsilon}_i$
```

3.5.3 Optimization result

For the SMPS measurements, the number of times T_1 (Eq. 3.9) is 66 and the number of size bins N_1 (Eq. 3.10) is 106. For the SP2 measurements, the number of times T_2 and the number of bins N_2 are 54 and 200 respectively (see Eq. 3.11). As a result, the optimization problem is essentially to fit a total of $T_1 N_1 + T_2 N_2 = 66 \times 106 + 54 \times 200 = 17796$ data points with 12 parameters listed in Table 3.3.

We ran the ProSRS algorithm with 800 iterations on 32-core XE compute nodes of Blue Waters[1]. We configured the algorithm to use all the cores

[1]Blue Waters: `https://bluewaters.ncsa.illinois.edu`.

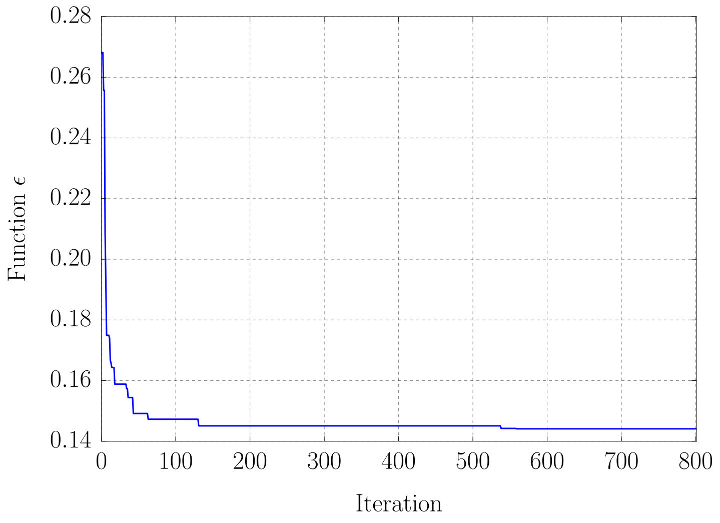

Figure 3.11: Optimization curve for the PartMC characterization problem. The y axis is the noisy function value of the ProSRS algorithm output.

of a node (i.e., ProSRS proposed 32 points for parallel evaluations at each iteration).

Figure 3.11 shows the optimization progress of the ProSRS algorithm. As we can see, ProSRS made significant progress within the initial 50 iterations before it gradually stabilized. Figure 3.12 shows the computational time of ProSRS and that of expensive function evaluations. We see that the computational cost of running ProSRS is about 1–2 orders of magnitude lower than that of evaluating the error function ϵ (Eq. 3.12). The general trend of the running time for the ProSRS algorithm stays roughly constant with the iterations.

After running the ProSRS algorithm, we ran the sample selection algorithm (Alg. 4) with number of candidates $m = 100$ and number of repeats $R = 10$. The best sample found (i.e., optimal parameters) is shown in Table 3.4. From Fig. 3.13 and 3.14, we see that the outputs of the optimized PartMC model agree well with the experimental data for both SMPS and SP2 measurements.

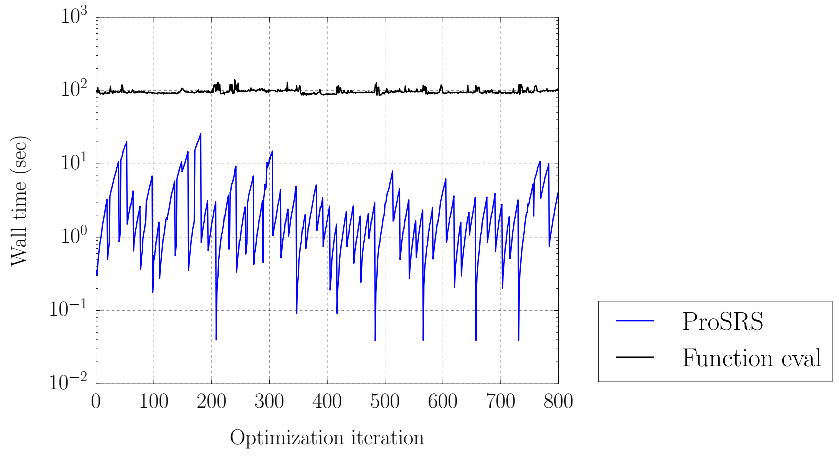

Figure 3.12: Computational costs of ProSRS algorithms (blue curve) and the evaluations of error function ϵ (black curve). The x axis is the number of iterations in the optimization, excluding the initial DOE iteration.

Table 3.4: Optimization results

Parameter	Optimal value	Parameter	Optimal value
R_{AS}	1.509	$s_{\mathrm{c,SP2}}$	217.43
R_{RB}	1.889	d_{f}	2.146
$R_{\mathrm{dil_2}}$	1.077	R_0	37.339
$s_{\mathrm{c,AS}}$	305.19	f	1.411
$s_{\mathrm{c,RB}}$	255.43	a	0.230
$s_{\mathrm{c,SMPS}}$	253.18	k_{D}	0.086

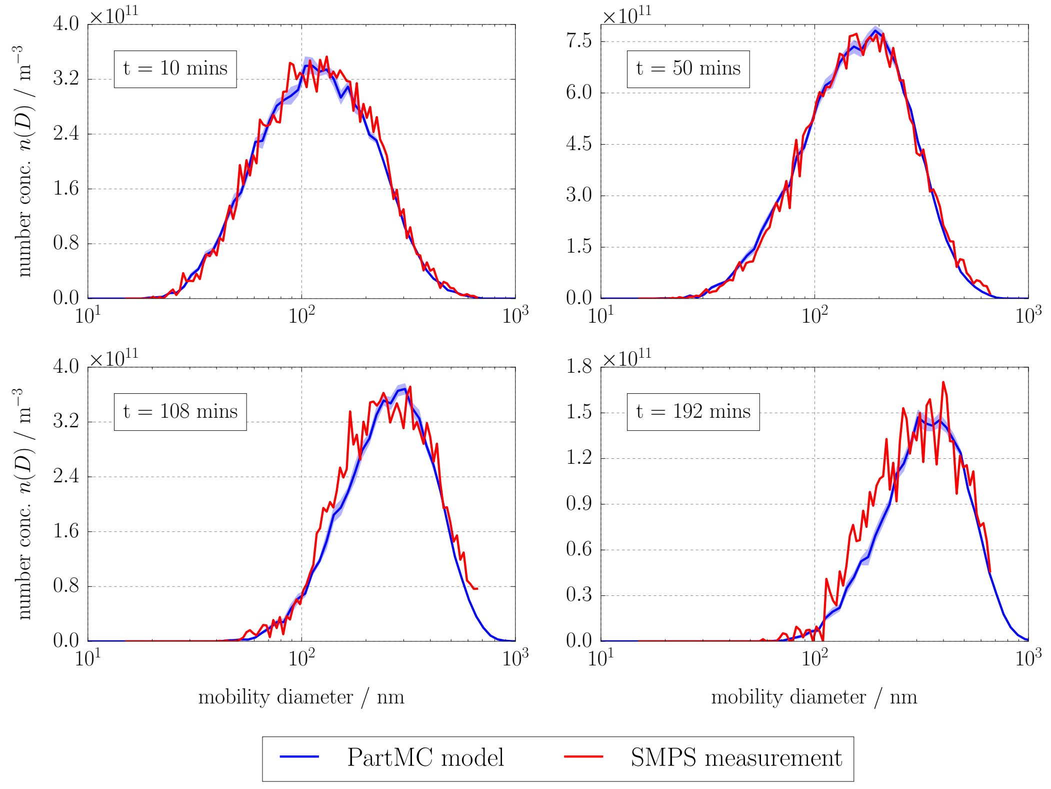

Figure 3.13: The outputs of the optimized PartMC model (with the optimal parameters in Table 3.4) versus SMPS measurements. The shaded area shows the standard deviation of 10 independent runs.

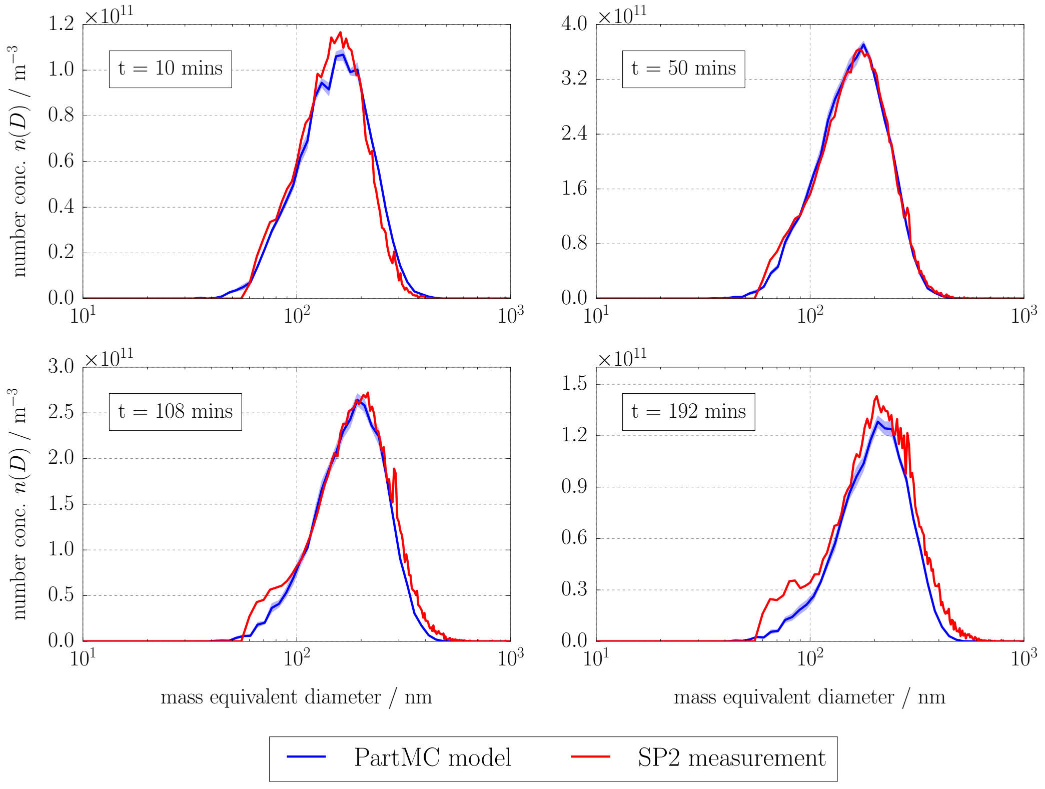

Figure 3.14: The outputs of the optimized PartMC model (with the optimal parameters in Table 3.4) versus SP2 measurements. The shaded area shows the standard deviation of 10 independent runs.

3.6 Conclusions

In this chapter we introduced a novel parallel surrogate optimization algorithm, namely the ProSRS algorithm, for noisy expensive optimization problems. We developed a "zoom strategy" for efficiency improvement, a weighted radial basis regression procedure, and a new SRS method combining the two types of candidate points in the original SRS work. We proved an analytical result for our algorithm (Theorem 3.1): if ProSRS is run for sufficiently long, with probability converging to one there will be at least one sample among all the evaluations that will be arbitrarily close to the global minimizer of the objective function. Numerical experiments show that our algorithm outperforms three current parallel Bayesian optimization algorithms on both optimization benchmark problems and two real hyperparameter-tuning problems. Our algorithm not only shows better optimization performance per iteration but is also orders of magnitude cheaper to run. We also demonstrated the application of ProSRS to the problem of characterizing a complex aerosol model against experimental measurements. This application serves as an example to illustrate a workflow of solving a general optimization problem with our algorithm.

CHAPTER 4

CONCLUSIONS

In this dissertation we demonstrated the use of machine learning to accelerate compute-intensive tasks in two distinct fields: simulation and optimization.

In Chapter 2 we developed an adaptive control variates algorithm to expedite the simulations of mean-field particle systems. Within this algorithm, we treated the body of particles as training data and used machine learning to automatically construct highly-correlated control variates from the data. It was through this learning mechanism that we were able to achieve significant variance reduction in spite of highly complex particle dynamics. We proved two important properties of our algorithm: unbiasedness of the estimator and asymptotic greater efficiency than naive Monte Carlo. We validated our theoretical claims through a simple 1D example, and demonstrated the effectiveness of our algorithm in simulating a complex aerosol particle model.

In Chapter 3 we introduced a parallel surrogate optimization algorithm, namely the ProSRS algorithm, for optimizing noisy expensive black-box functions. The algorithm uses weighted radial basis regression to learn the underlying response surface from the evaluation data. In doing so, the trend of the objective function is well exploited. Hence, our algorithm is able to achieve good optimization progress using relatively few function evaluations. In addition to the radial basis regression, we developed a tree-based technique, known as the "zoom strategy", to help reduce the computational cost while not compromising the optimization convergence. We proved an asymptotic convergence result for our algorithm. Moreover, we benchmarked our algorithm against state-of-the-art Bayesian optimization algorithms, finding that our algorithm was not only generally superior in optimization performance but was also orders of magnitude cheaper to run. We further demonstrated a workflow of solving a general optimization problem with ProSRS by applying it to the problem of characterizing the parameters needed to fit an expensive aerosol model to experimental data.

APPENDIX A

ASSUMPTION VALIDATION FOR NUMERICAL STUDY 1

Using induction, we have the following formula for the particle state $x_{t,i}$ and the mean field ϕ_t with $k \neq 1$:

$$x_{t,i} = \frac{1}{N} \sum_{j=1}^{N} \left[k^t x_{0,j} + \left(\frac{k - k^t}{1-k} \right) e^{bx_{0,j}} \right] + e^{bx_{0,i}}, \quad t \geq 1, \tag{A.1}$$

$$\phi_t = \frac{1}{N} \sum_{j=1}^{N} \left[k^t x_{0,j} + \left(\frac{1 - k^t}{1-k} \right) e^{bx_{0,j}} \right], \quad t \in \mathbb{N}. \tag{A.2}$$

Now we validate Assumptions 2.5–2.9 separately as follows:

1. **Check Assumption 2.5**

This is trivially satisfied (see the remarks of Theorem 2.3 on the domain Θ).

2. **Check Assumption 2.6**

Since we train cubic polynomial regression models, the basis function η in this assumption is simply

$$\eta(x_{0,1}) = [1, x_{0,1}, x_{0,1}^2, x_{0,1}^3]^T, \tag{A.3}$$

and the subspace ν is given by

$$\nu = \{a_0 + a_1 x_{0,1} + a_2 x_{0,1}^2 + a_3 x_{0,1}^3 \mid a_0, a_1, a_2, a_3 \in \mathbb{R}\}. \tag{A.4}$$

Clearly, $1 \in \nu$. Because $x_{0,1}$ is a standard Gaussian random variable, the random variable $a_0 + a_1 x_{0,1} + a_2 x_{0,1}^2 + a_3 x_{0,1}^3$ has finite second moment for any fixed $a_0, a_1, a_2, a_3 \in \mathbb{R}$. Hence, $\nu \subseteq \mathcal{L}^2$.

The correlation matrix is

$$\mathbb{E}\left[\eta(x_{0,1})\,(\eta(x_{0,1}))^T\right] = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 3 \\ 1 & 0 & 3 & 0 \\ 0 & 3 & 0 & 15 \end{bmatrix}, \tag{A.5}$$

which is non-singular.

3. **Check Assumption 2.7**

From Eq. A.2, it is easy to see that the mean field ϕ_t converges to its expectation, denoted as c_t, in mean square as $N \to \infty$ (recall that $x_{0,j}$ are i.i.d.). As a result, for any $t \geq 1$,

$$x_{t,1} = \Psi_t(\phi_0, \phi_1, \ldots, \phi_{t-1}, x_{0,1}) = k\phi_{t-1} + e^{bx_{0,1}} \xrightarrow[N\to\infty]{m.s.} kc_{t-1} + e^{bx_{0,1}}. \tag{A.6}$$

4. **Check Assumption 2.8**

Define a random vector $\xi \coloneqq [x_{0,1}, x_{0,1}^2, x_{0,1}^3]^T$. Then the projection of $x_{t,1}$ onto the subspace ν in Eq. A.4 is given by

$$\Pi_\nu(x_{t,1}) = \mathbb{E}[x_{t,1}] + \mathrm{Cov}(x_{t,1}, \xi)\mathrm{Cov}(\xi, \xi)^{-1}(\xi - \mathbb{E}[\xi]) = (\beta_t')^T\,[1, \xi^T]^T, \tag{A.7}$$

from which we solve for β_t' for each $t = 1, 2, \ldots, T$.

Using the projection expression in Eq. A.7 and after some calculation, the covariance in the assumption is found to have the following form:

$$\mathrm{Cov}\left(x_{t,2}, \Pi_\nu(x_{t,1})\right) = \frac{A_t}{N} + \frac{B_t}{N^2}, \tag{A.8}$$

where the constants A_t and B_t are given by

$$\begin{aligned} A_t &= D_t\mathrm{Cov}(\xi, \xi)^{-1}\mathrm{Cov}\left(\xi, e^{bx_{0,1}}\right), \\ B_t &= D_t\mathrm{Cov}(\xi, \xi)^{-1}(D_t)^T, \end{aligned}$$

with

$$D_t = \mathrm{Cov}\left(k^t x_{0,1} + \frac{k - k^t}{1-k}e^{bx_{0,1}}, \xi\right).$$

With the parameters in the model, it can be checked that $A_t > 0$ for all time $t = 1, 2, \ldots, T$. Hence, $\liminf_{N\to\infty} \mathrm{Cov}\big(g(x_{t,2}), (\beta_t')^T \eta(x_{0,1})\big) N = A_t > 0$.

5. **Check Assumption 2.9**

Computing the variance of projection gives

$$\begin{aligned}\mathrm{Var}\Big(\Pi_\nu\big(\Psi_t(c_0, c_1, \ldots, c_{t-1}, x_{0,1})\big)\Big) &= \mathrm{Var}\Big(\Pi_\nu\big(kc_{t-1} + e^{bx_{0,1}}\big)\Big) \\ &= \mathrm{Cov}(e^{bx_{0,1}}, \xi)\mathrm{Cov}(\xi, \xi)^{-1}\left(\mathrm{Cov}(e^{bx_{0,1}}, \xi)\right)^T. \quad \text{(A.9)}\end{aligned}$$

It is easy to see that the vector $\mathrm{Cov}(e^{bx_{0,1}}, \xi)$ is not a zero vector. Hence, the variance in Eq. A.9 is positive by noting that the covariance matrix $\mathrm{Cov}(\xi, \xi)^{-1}$ is positive definite.

APPENDIX B

MEAN FIELD CONVERGENCE FOR AEROSOL PARTICLE MODEL

We show that the concentration of gas species converges to some constant in the limit of a large number of particles for a simplified case of the aerosol model (Theorem B.1).

Assumption B.1 *Gas emission, gas dilution and gas chemistry are neglected. There is only one gas species. No doubling or halving for number of particles. The volume of air box does not change with time. No dilution or emission for aerosol particles. There is only one initial source. There is one chemical species for each aerosol particle.*

Assumption B.2 *The dynamics of the gas-particle partitioning is given by the following equations [83]:*

$$x_{t+1,i} = x_{t,i} + G(R_{t,i})\Delta t \left(\frac{\phi_{t-1} - \phi_t}{ln(\phi_{t-1}) - ln(\phi_t)} \right), \tag{B.1}$$

$$\phi_t = \phi_{-1} exp \left(-\Delta t \sum_{\tau=0}^{t} \sum_{j=1}^{N} k_{\tau,j} \right), \quad i = 1, 2, \ldots, N, \quad t \in \mathbb{N} \tag{B.2}$$

$$k_{t,i} = \frac{G(R_{t,i})}{V_{box}}, \quad R_{t,i} = \sqrt[3]{\frac{3}{4\pi} x_{t,i}}, \quad G(x) = \frac{x(a_2x^2 + a_1x)}{b_2x^2 + b_1x + b_0}, \tag{B.3}$$

where $x_{t,i}$ is the volume of aerosol particle i at time t, Δt is the time step size, $R_{t,i}$ is the radius of the particle, $k_{t,i}$ is the first order mass transfer coefficient, and N is total number of particles. The variable ϕ_t denotes the gas concentration at time t, and ϕ_{-1} is a positive constant for the initialized gas concentration. Coefficients a_1, a_2, b_0, b_1 and b_2 in the function G are some positive constants.

Theorem B.1 *For the aerosol model defined in Alg. 5 and under Assumptions B.1–B.2, the gas concentration ϕ_t converges to a constant in probability as the mean number of particles $M_{init} \to \infty$ (M_{init} is defined in Eq. 2.66) for each $t = 0, 1, \ldots, T$.*

Proof. To show this, we will first prove several Lemmas (Lemma B.1–B.4).

Lemma B.1 *Let $\{N_m\}_{m\in\mathbb{N}}$ be a sequence of random variables taking values in $\mathbb{N}$ such that $\frac{N_m}{m} \to b > 0$ in probability as $m \to \infty$. Let $\{x_n\}_{n\in\mathbb{N}}$ be another sequence of random variables taking values in Ω. Let $S_m(y) = \sum_{i=1}^{N_m} \phi(x_i, y)$ with the convention that $S_m(y) = 0$ if $N_m = 0$, where $\phi(x, y)$ is a real-valued function continuous at $y = a$ uniformly in $x \in \Omega$ (i.e., $\forall\epsilon > 0$, $\exists\delta > 0$ such that $\|y - a\| < \delta$ implies $\sup_{x\in\Omega} |\phi(x, y) - \phi(x, a)| < \epsilon$). Suppose a sequence of random vectors $z_m \xrightarrow{p.} a$ and the sequence $\frac{S_m(a)}{m} \xrightarrow{p.} c$. Then*[1] $\frac{S_m(z_m)}{m} \xrightarrow{p.} c$.

Proof. It suffices to show $\frac{S_m(z_m)-S_m(a)}{m} \xrightarrow{p.} 0$. That is, to show for any $\epsilon > 0$,

$$\mathrm{P}\left(\left|\frac{S_m(z_m) - S_m(a)}{m}\right| < \epsilon\right) \to 1. \tag{B.4}$$

Since ϕ is continuous at $y = a$ uniformly in $x \in \Omega$, there exists $\delta > 0$ such that $\|z_m - a\| < \delta$ implies $|\phi(x, z_m) - \phi(x, a)| < \epsilon/(2b)$ for all $x \in \Omega$. By the triangle inequality, we have

$$\begin{aligned}\left|\frac{S_m(z_m) - S_m(a)}{m}\right| &= \frac{\left|\sum_{i=1}^{N_m} \big(\phi(x_i, z_m) - \phi(x_i, a)\big)\right|}{m} \\ &\le \frac{\sum_{i=1}^{N_m} |\phi(x_i, z_m) - \phi(x_i, a)|}{m} \le \frac{N_m}{2mb}\epsilon.\end{aligned}$$

Hence, $\|z_m - a\| < \delta$ and $\frac{N_m}{m} < 2b$ would imply $\left|\frac{S_m(z_m)-S_m(a)}{m}\right| < \epsilon$. So to prove Eq. B.4, we only need to show

$$\mathrm{P}\left(\|z_m - a\| < \delta, \quad \frac{N_m}{m} < 2b\right) \to 1. \tag{B.5}$$

Since $\frac{N_m}{m} \xrightarrow{p.} b > 0$ and $z_m \xrightarrow{p.} a$, $\mathrm{P}\left(\frac{N_m}{m} < 2b\right) \to 1$ and $\mathrm{P}\left(\|z_m - a\| < \delta\right) \to 1$ so that Eq. B.5 immediately follows. □

Lemma B.2 *Let $N \sim Pois(M)$, where $M > 0$ is a Poisson distribution parameter. Conditioned on $N > 0$, let $X_1, X_2, \ldots, X_N$ be identically distributed and pairwise uncorrelated with finite mean μ and finite variance σ^2 (μ and σ^2 are constants that do not depend on N). Let $S_N = \sum_{i=1}^{N} X_i$ with the*

[1] We assume that x_n, N_m and z_m are defined on the same probability space.

convention $S_0 = 0$. Then

$$\frac{S_N}{M} \xrightarrow[M\to\infty]{p.} \mu.$$

Proof. By the tower property,

$$\mathbb{E}\Big[\frac{S_N}{M}\Big] = \mathbb{E}\Big[\mathbb{E}\Big[\frac{S_N}{M} \mid N\Big]\Big] = \frac{\mathbb{E}[N]}{M}\mu = \mu.$$

By the law of total variance,

$$\begin{aligned}\operatorname{Var}\Big(\frac{S_N}{M}\Big) &= \operatorname{Var}\Big(\mathbb{E}\big[\frac{S_N}{M} \mid N\big]\Big) + \mathbb{E}\Big[\operatorname{Var}\big(\frac{S_N}{M} \mid N\big)\Big] \\ &= \operatorname{Var}\Big(\frac{N}{M}\mu\Big) + \mathbb{E}\Big[\frac{N}{M^2}\sigma^2\Big] \\ &= \frac{\mu^2+\sigma^2}{M}.\end{aligned}$$

Since $\mu^2 + \sigma^2$ is finite, by Markov's inequality we have for all $\epsilon > 0$,

$$\mathrm{P}\left(\Big|\frac{S_N}{M} - \mu\Big| > \epsilon\right) \le \operatorname{Var}\Big(\frac{S_N}{M}\Big)\epsilon^{-2} = \frac{\mu^2+\sigma^2}{M\epsilon^2} \xrightarrow{M\to\infty} 0.$$

Hence, $\frac{S_N}{M} \xrightarrow{p.} \mu$ as desired. □

Lemma B.3 *Suppose g is a real-valued function uniformly continuous on $\mathcal{S} \subseteq \mathbb{R}$. Let $\phi(x, y)$ be an $\mathcal{S}$-valued function continuous at $y = a$ uniformly in $x \in \Omega$. Then the composite function $g\big(\phi(x, y)\big)$ is continuous at $y = a$ uniformly in $x \in \Omega$.*

Proof. Since g is uniformly continuous on $\mathcal{S}$, for any $\epsilon > 0$, there exists $\delta > 0$ such that for all x_1, $x_2 \in \mathcal{S}$ with $|x_1 - x_2| < \delta$, $|g(x_1) - g(x_2)| < \epsilon$. Since ϕ is continuous at $y = a$ uniformly in $x \in \Omega$, by definition there exists $\xi > 0$ such that for all y with $\|y - a\| < \xi$ and all $x \in \Omega$, we have $|\phi(x, y) - \phi(x, a)| < \delta$, which implies $|g\big(\phi(x, y)\big) - g\big(\phi(x, a)\big)| < \epsilon$. Thus, $g\big(\phi(x, y)\big)$ is continuous at $y = a$ uniformly in $x \in \Omega$. □

Lemma B.4 *Suppose the partial derivative $\partial\phi(x, y)/\partial y$ exists on $\mathcal{D} = \{(x, y) \mid y = a, x \in \Omega\}$ and is bounded on $\mathcal{D}$. Then $\phi(x, y)$ is continuous at $y = a$ uniformly in $x \in \Omega$.*

Proof. Since $\partial_y\phi$ exists on $\mathcal{D}$, $\lim\limits_{y\to a} \frac{\phi(x,y)-\phi(x,a)}{y-a} = \partial_y\phi(x, a)$ for all $x \in \Omega$. Since $\partial_y\phi(x, a)$ is bounded on $x \in \Omega$, there exists $\delta > 0$ and $C > 0$ such that for any y with $|y-a|<\delta$ and any $x\in\Omega$, $|\phi(x,y)-\phi(x,a)|\le C|y-a|$. As a result, for any $\epsilon>0$, if we take $\xi=\min(\delta,\frac{\epsilon}{C})$, then for any y with $|y-a|<\xi$ and any $x\in\Omega$,

$$|\phi(x,y)-\phi(x,a)|\le C|y-a|<C\xi\le C\frac{\epsilon}{C}=\epsilon.$$

Hence, $\phi(x,y)$ is continuous at $y=a$ uniformly in $x\in\Omega$. □

Now we prove the theorem (Theorem B.1) as follows.

Without loss of generality, we assume $V_{\text{box}}=M_{\text{init}}$ so that the proportionality constant in Eq. 2.66 is one. For notational brevity, we omit the subscript of M_{init} so that $V_{\text{box}}=M_{\text{init}}=M$. Since there is one initial source by Assumption B.1, we have $N\sim\text{Pois}(M)$, which is clear from Eq. 2.64. Further define

$$S_t:=\begin{cases}0, & N=0,\\ \sum\limits_{i=1}^{N}G(R_{t,i}), & N>0,\end{cases} \tag{B.6}$$

$$F_t:=\left(\frac{3\phi_{-1}}{4\pi}\right)\exp\left(-\sum_{\tau=0}^{t-1}\frac{S_\tau\Delta t}{M}\right)\left[1-\exp\left(-\frac{S_t\Delta t}{M}\right)\right]\left(\frac{S_t}{M}\right)^{-1}, \tag{B.7}$$

with the convention $\sum\limits_{\tau=0}^{t-1}(\bullet)=0$ if $t=0$. With some algebra, it can be shown that the gas-partitioning dynamics (Eq. B.1, Eq. B.2 and Eq. B.3) is equivalent to

$$\phi_t=\phi_{t-1}\exp\big(-\Delta t\frac{S_t}{M}\big), \tag{B.8}$$

$$R_{t+1,i}=\sqrt[3]{(R_{t,i})^3+F_tG(R_{t,i})},\quad i=1,2,\ldots,N. \tag{B.9}$$

Since the particle volume $x_{0,i}$ ($=V_{0,i}$) is lognormally-distributed (see Eq. 2.65), the particle radius $R_{0,i}$ is also lognormal. As a result, $G(R_{0,i})$ has a finite second moment. Note that $G(R_{0,i})$ ($i=1,2,\ldots,N$) are i.i.d. given positive N and they are always positive. Consequently, by Lemma B.2,

$$\frac{S_0}{M}\xrightarrow[M\to\infty]{p.}\mathbb{E}\big[G(R_{0,1})\big]>0. \tag{B.10}$$

Recursively applying Eq. B.8, we have

$$\phi_t = \phi_{-1}\exp\big(-\Delta t\frac{S_0}{M}\big)\exp\big(-\Delta t\frac{S_1}{M}\big)\dots\exp\big(-\Delta t\frac{S_t}{M}\big).$$

Since ϕ_{-1} is a constant, to prove ϕ_t converges to a constant in probability, we only need to show $\frac{S_t}{M}$ converges to a constant in probability for all $t = 0, 1, \dots, T$ thanks to the continuous mapping theorem. With Eq. B.10, it suffices to show the following statement: *if* $\frac{S_0}{M}, \frac{S_1}{M}, \dots, \frac{S_t}{M}$ *all converge to some positive constants in probability, then* $\frac{S_{t+1}}{M}$ *also converges to a positive constant in probability.*

From Eq. B.9, it is clear $R_{t+1,i}$ is a function of $R_{0,i}$, F_0, $F_1, \dots, F_t$. As a result, we can write

$$R_{t+1,i} = H_t(R_{0,i}, F_0, F_1, \dots, F_t) \tag{B.11}$$

for some H_t. Define $z_t \coloneqq (F_0, F_1, \dots, F_t)$. Then Eq. B.6 becomes

$$S_{t+1}(z_t) = \begin{cases} 0, & N = 0, \\ \sum\limits_{i=1}^{N} G\big(H_t(R_{0,i}, z_t)\big), & N > 0. \end{cases}$$

By Lemma B.1, in order to show $\frac{S_{t+1}(z_t)}{M}$ converges to a positive constant in probability, it suffices to verify the following conditions:

1. $\frac{N}{M} \xrightarrow{p.} 1$ as $M \to \infty$.
2. $z_t \xrightarrow{p.} a_t$ as $M \to \infty$, where a_t is a constant vector $\in \mathbb{R}^{t+1}$.
3. $\frac{S_{t+1}(a_t)}{M} \xrightarrow{p.} C_t$ as $M \to \infty$, where C_t is a positive constant.
4. $G\big(H_t(x, z_t)\big)$ is continuous at $z_t = a_t$ uniformly in $x > 0$.

Condition 1 is a trivial corollary of Lemma B.2 by setting all X to 1.

Since $\frac{S_0}{M}, \frac{S_1}{M}, \dots, \frac{S_t}{M}$ all converge to some positive constants in probability, by the continuous mapping theorem, $F_0, F_1, \dots, F_t$ all converge to some (positive) constants, say $\omega_0, \omega_1 \dots \omega_t$, in probability ($F_t$ is defined in Eq. B.7). Elementwise convergence in probability implies convergence of the random vector in probability. Now letting $a_t = (\omega_0, \omega_1, \dots, \omega_t)$ gives Condition 2.

Using simple induction, it is easy to check that $G\big(H_t(R_{0,i}, a_t)\big)$ has finite second moment. Because $G\big(H_t(R_{0,i}, a_t)\big)$ $(i = 1, 2, \ldots, N)$ are i.i.d. conditioned on positive N, we have that $\frac{S_{t+1}(a_t)}{M} \xrightarrow{p.} \mathbb{E}\big[G\big(H_t(R_{0,1}, a_t)\big)\big] > 0$ by Lemma B.2. This gives Condition 3.

Function G in Eq. B.3 is continuous on $[0, +\infty)$ and has the property that $\lim_{x\to\infty}\big(G(x) - (ax + b)\big) = 0$ for some $a, b \in \mathbb{R}$. Since $ax + b$ is uniformly continuous on $[0, +\infty)$, G is also uniformly continuous on $[0, +\infty)$. Hence, by Lemma B.3, to show Condition 4 we only need to show $H_t(x, z_t)$ is continuous at $z_t = a_t$ uniformly in $x > 0$. To show this, we will use induction.

First note that by Eq. B.9 and the definition of H_t in Eq. B.11, $H_0(x, z_0) = \sqrt[3]{x^3 + z_0 G(x)}$. Consequently, the partial derivative is $d(x) := \frac{\partial H_0}{\partial z_0}\Big|_{z_0 = a_0} = \frac{1}{3}\big(x^3 + a_0 G(x)\big)^{-\frac{2}{3}} G(x)$. Since $G(x) \sim \mathcal{O}(x)$ as $x \to \infty$, by observation we have $\lim_{x\to 0^+} d(x) = \lim_{x\to\infty} d(x) = 0$. Note that d is continuous on $(0, +\infty)$. Hence, d must be bounded on $(0, +\infty)$. By Lemma B.4, $H_0(x, z_0)$ is continuous at $z_0 = a_0$ uniformly in $x > 0$.

To complete the induction, now we only need to show the following statement: *If $H_t(x, z_t)$ is continuous at $z_t = a_t$ uniformly in $x > 0$, then $H_{t+1}(x, z_{t+1})$ is also continuous at $z_{t+1} = a_{t+1}$ uniformly in $x > 0$.* Now fix an arbitrary $\epsilon > 0$.

By Eq. B.9, Eq. B.11 and using the triangle inequality, we have

$$\begin{aligned}
&|H_{t+1}(x, z_{t+1}) - H_{t+1}(x, a_{t+1})| \\
&= \left|\sqrt[3]{\big(H_t(x, z_t)\big)^3 + F_{t+1} G\big(H_t(x, z_t)\big)} - \sqrt[3]{\big(H_t(x, a_t)\big)^3 + \omega_{t+1} G\big(H_t(x, a_t)\big)}\right| \\
&\leq \left|\sqrt[3]{\big(H_t(x, z_t)\big)^3 + F_{t+1} G\big(H_t(x, z_t)\big)} - \sqrt[3]{\big(H_t(x, z_t)\big)^3 + \omega_{t+1} G\big(H_t(x, z_t)\big)}\right| \\
&\quad + \left|\sqrt[3]{\big(H_t(x, z_t)\big)^3 + \omega_{t+1} G\big(H_t(x, z_t)\big)} - \sqrt[3]{\big(H_t(x, a_t)\big)^3 + \omega_{t+1} G\big(H_t(x, a_t)\big)}\right|.
\end{aligned} \tag{B.12}$$

Applying the same technique that is for establishing the first step of induction (i.e., the step of showing uniform continuity of $H_0(x, z_0)$), we have that there exists $\delta_1 > 0$ such that for any z_{t+1} with $\|z_{t+1} - a_{t+1}\| < \delta_1$ (which implies

$|F_{t+1} - \omega_{t+1}| < \delta_1$) and for any $x > 0$,

$$\left| \sqrt[3]{\big(H_t(x, z_t)\big)^3 + F_{t+1} G\big(H_t(x, z_t)\big)} - \sqrt[3]{\big(H_t(x, z_t)\big)^3 + \omega_{t+1} G\big(H_t(x, z_t)\big)} \right| < \frac{\epsilon}{2}. \tag{B.13}$$

Note that $\sqrt[3]{x^3 + \omega_{t+1} G(x)}$ is uniformly continuous on $(0, +\infty)$ (this can be seen using exactly the same arguments that we have established for the uniform continuity of function G). Also note that by our induction condition, $H_t(x, z_t)$ is continuous at $z_t = a_t$ uniformly in $x > 0$. As a result, by using Lemma B.3, we have that $\sqrt[3]{\big(H_t(x, z_t)\big)^3 + \omega_{t+1} G\big(H_t(x, z_t)\big)}$ is continuous at $z_t = a_t$ uniformly in $x > 0$. Hence, there exists $\delta_2 > 0$ such that for any z_{t+1} with $\|z_{t+1} - a_{t+1}\| < \delta_2$ (which implies $\|z_t - a_t\| < \delta_2$) and for any $x > 0$,

$$\left| \sqrt[3]{\big(H_t(x, z_t)\big)^3 + \omega_{t+1} G\big(H_t(x, z_t)\big)} - \sqrt[3]{\big(H_t(x, a_t)\big)^3 + \omega_{t+1} G\big(H_t(x, a_t)\big)} \right| < \frac{\epsilon}{2}. \tag{B.14}$$

Now take $\delta = \min(\delta_1, \delta_2)$. Substituting Eqs. B.13 and B.14 into Eq. B.12, we have for any z_{t+1} with $\|z_{t+1} - a_{t+1}\| < \delta$ and for any $x > 0$,

$$|H_{t+1}(x, z_{t+1}) - H_{t+1}(x, a_{t+1})| < \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon.$$

This completes the induction of Condition 4 and so also the proof of the theorem. $\square$

APPENDIX C

ALGORITHMS IN NUMERICAL STUDY 2

Algorithm 5 shows the particle-resolved aerosol model in the second numerical example. A more detailed description of the model is available in [14, 34].

Algorithm 5 Particle-resolved aerosol model

```
1: $t \leftarrow 0$, box $\leftarrow \emptyset$ and set $T$
2: Add $N_\mathrm{i}$ particles for each initial source (Eq. 2.64, Eq. 2.65)
3: Initialize gas species concentrations and box volume $V_\mathrm{box}$ (Eq. 2.66)
4: while $t \leq T$ do
5:     Update gas concentrations for gas emission and dilution
6:     Add $N_{\mathrm{e},t}$ particles for each emission source (Eq. 2.67, Eq. 2.68)
7:     Randomly remove $N_{\mathrm{r},t}$ particles for dilution-out process (Eq. 2.71)
8:     Add $N_{\mathrm{d},t}$ particles for each dilution source (Eq. 2.69, Eq. 2.70)
9:     Scale $V_\mathrm{box}$ for temperature and humidity adjustment
10:    Gas-particle partitioning and gas chemistry
11:    Double or halve number of particles and $V_\mathrm{box}$ for efficient computation

12:    if $t \in$ {output times} then
13:        Output total mass concentration and scattering coefficient (Eq. 2.72)
14:    end if
15:    $t \leftarrow t + 1$
16: end while
```

Algorithm 6 is used for partitioning the training data in the second numerical example. The algorithm groups the training data so that the samples in each partition have contiguous creation times and different partitions have roughly the same sizes. The parameter N_p is used to control the partition sizes. To ensure sufficient training samples for each MARS model, we set $N_\mathrm{p} = 5000$ in our numerical experiments.

Algorithm 6 Partition training samples by particle creation times

```
1: Set $N_\mathrm{p}$ to control number of samples in a partition
2: if the total number of samples $< N_\mathrm{p}$ then
3:     The whole data is the only partition
4: else
5:     Form the first partition by grouping samples with creation time $= 0, 1, \ldots$ until the size of the group $\geq N_\mathrm{p}$
6:     Keep forming the remaining partitions until the data is exhausted
7:     if the size of the last partition $< N_\mathrm{p}$ then
8:         Join the last two partitions to form a new partition
9:     end if
10: end if
```

APPENDIX D

COMPUTATION OF CONTROL VARIATES MEAN FOR NUMERICAL STUDY 2

It can be shown that the step of doubling or halving (Line 11 of Alg. 5) does not change the mean of control variates. As a result, without loss of generality we can assume there is no doubling or halving for the aerosol simulation. Consequently, $(V_{\text{box}})_t$ is now a deterministic function of time t, and the mean of control variates is given by

$$E_t = \frac{1}{(V_{\text{box}})_t} \mathbb{E}\Big[\sum_{i=1}^{N_t} h(x_{\tau_i,i}, \tau_i; \beta_t) \mid \beta_t \Big], \tag{D.1}$$

where the function h is a piecewise MARS model (see Section 2.7.2). Partitioning the summation in Eq. D.1 based on the creation time and the particle source, we have

$$\begin{aligned} E_t = \frac{1}{(V_{\text{box}})_t} \Bigg\{ & \sum_{\text{initial sources}} \mathbb{E}\Big[\sum_i h(x_{0,i}, 0; \beta_t) \mid \beta_t \Big] \\ & + \sum_{\tau=0}^{t} \Bigg(\sum_{\text{emission sources}} \mathbb{E}\Big[\sum_i h(x_{\tau,i}, \tau; \beta_t) \mid \beta_t \Big] \\ & + \sum_{\text{dilution sources}} \mathbb{E}\Big[\sum_i h(x_{\tau,i}, \tau; \beta_t) \mid \beta_t \Big] \Bigg) \Bigg\}. \end{aligned}$$

Now it suffices to show the computation of the sum of the conditional expectations for an arbitrary creation time and an arbitrary source. Without loss of generality, consider the particles with creation time τ from the first emission source, and denote the number of these particles at time t to be $N_{\text{e},t}$ with $t \geq \tau$. Since particles with the same creation time and the same emission source are identically distributed conditioned on the learned model, we have

$$\mathbb{E}\Big[\sum_{i=1}^{N_{\mathrm{e},t}} h(x_{\tau,i},\tau;\beta_t) \mid \beta_t\Big] = \mathbb{E}\big[N_{\mathrm{e},t} \mid \beta_t\big]\mathbb{E}\big[h(x_{\tau,1},\tau;\beta_t) \mid \beta_t\big]. \tag{D.2}$$

Denote the mean number of particles of the first emission source at time τ to be $M_{\mathrm{e},\tau}$, which is equal to the product of $\lambda_{\mathrm{emit}}(\tau)$ and V_{box} (see Eq. 2.67). At each time $t \geq \tau$, there is a probability of $p_{\mathrm{dil},t}$ for each particle to be removed. As a result, we have $\mathbb{E}\big[N_{\mathrm{e},t}\big] = M_{\mathrm{e},\tau}\Pi_{j=\tau}^{t}(1-p_{\mathrm{dil},j})$. Since the learning phase and the evaluation phase are independent, $\mathbb{E}\big[N_{\mathrm{e},t} \mid \beta_t\big] = \mathbb{E}\big[N_{\mathrm{e},t}\big]$. This completes the computation of the first expectation on the right hand side of Eq. D.2. For the second expectation, note that $x_{\tau,1}$ is the product of a constant vector and a lognormal random variable (see Eq. 2.68). Consequently, $\mathbb{E}\big[h(x_{\tau,1},\tau;\beta_t) \mid \beta_t\big]$ is essentially the expectation of some function of a lognormal random variable. In this study, we use numerical quadrature to compute this expectation.

APPENDIX E

OPTIMIZATION BENCHMARK FUNCTIONS

In this section we give the expression for each optimization benchmark function in Table 3.2.

1. Ackley function

$$f(x) = -a\exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right)$$
$$+ a + \exp(1), \quad x \in \mathbb{R}^d,$$

   where d is the dimension, x_i is the i^{th} component of vector x, $a = 20$, $b = 0.2$ and $c = 2\pi$.

2. Alpine function

$$f(x) = \sum_{i=1}^{d} |x_i \sin(x_i) + 0.1x_i|, \quad x \in \mathbb{R}^d,$$

   where d is the dimension and x_i is the i^{th} component of vector x.

3. Griewank function

$$f(x) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \quad x \in \mathbb{R}^d,$$

   where d is the dimension and x_i is the i^{th} component of vector x.

4. Levy function

$$f(x) = \sin^2(\pi w_1) + \sum_{i=1}^{d-1}(w_i - 1)^2[1 + 10\sin^2(\pi w_i + 1)]$$
$$+ (w_d - 1)^2[1 + \sin^2(2\pi w_d)],$$

where w_i $(i = 1, 2, \ldots, d)$ is given by

$$w_i = 1 + \frac{x_i - 1}{4}.$$

Here d is the dimension and x_i is the i^{th} component of vector x.

5. SumPower function

$$f(x) = \sum_{i=1}^{d} |x_i|^{i+1}, \quad x \in \mathbb{R}^d,$$

where d is the dimension and x_i is the i^{th} component of vector x.

6. SixHumpCamel function

$$f(x_1, x_2) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right) x_1^2 + x_1x_2 + \left(-4 + 4x_2^2\right) x_2^2, \quad x_1, x_2 \in \mathbb{R}.$$

7. Schaffer function

$$f(x_1, x_2) = 0.5 + \frac{\sin^2(x_1^2 - x_2^2) - 0.5}{[1 + 0.001(x_1^2 + x_2^2)]^2}, \quad x_1, x_2 \in \mathbb{R}.$$

8. Dropwave function

$$f(x_1, x_2) = -\frac{1 + \cos\left(12\sqrt{x_1^2 + x_2^2}\right)}{0.5(x_1^2 + x_2^2) + 2}, \quad x_1, x_2 \in \mathbb{R}.$$

9. Goldstein-Price function

$$\begin{aligned} f(x_1, x_2) = {} & [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 \\ & + 3x_2^2)][30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 \\ & - 36x_1x_2 + 27x_2^2)], \quad x_1, x_2 \in \mathbb{R}. \end{aligned}$$

10. Rastrigin function

$$f(x) = 10d + \sum_{i=1}^{d} [x_i^2 - 10\cos(2\pi x_i)], \quad x \in \mathbb{R}^d,$$

where d is the dimension and x_i is the i^{th} component of vector x.

11. Hartmann6 function

$$f(x) = -\sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{6} A_{ij}(x_j - P_{ij})^2\right), \quad x \in \mathbb{R}^6,$$

where x_j is the j^{th} component of vector x, α_i is the i^{th} component of coefficient vector α, A_{ij} is the i^{th} row and j^{th} column of coefficient matrix A and P_{ij} is the i^{th} row and j^{th} column of coefficient matrix P. The coefficients α, A and P are given by

$$\alpha = [1, 1.2, 3, 3.2]^T,$$

$$A = \begin{bmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix},$$

$$P = 10^{-4} \begin{bmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{bmatrix}.$$

12. PowerSum4 function

$$f(x) = \sum_{i=1}^{4} \left[\left(\sum_{i=1}^{4} x_j^i\right) - b_i\right]^2, \quad x, b \in \mathbb{R}^4,$$

where vector $b = [8, 18, 44, 114]^T$ and x_j is the j^{th} component of vector x.

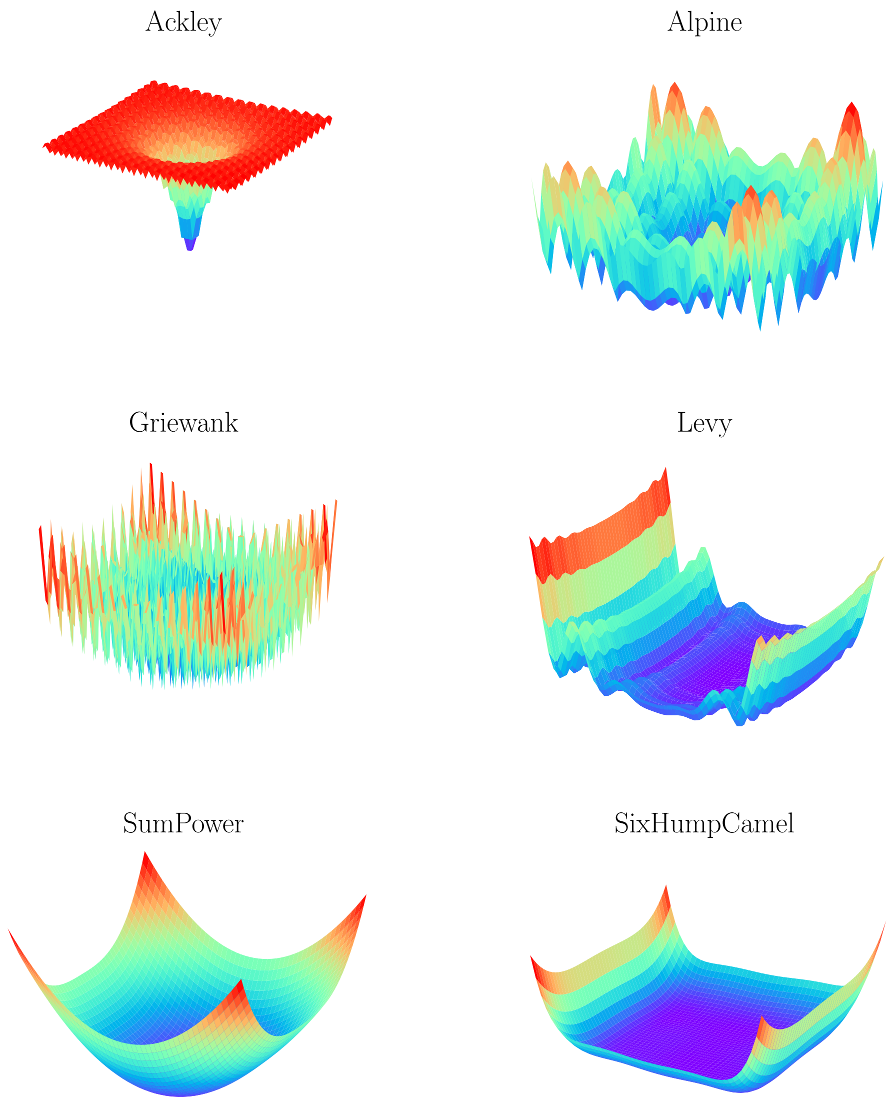

Figure E.1: Surface plots for benchmark functions Ackley, Alpine, Griewank, Levy, SumPower, and SixHumpCamel.

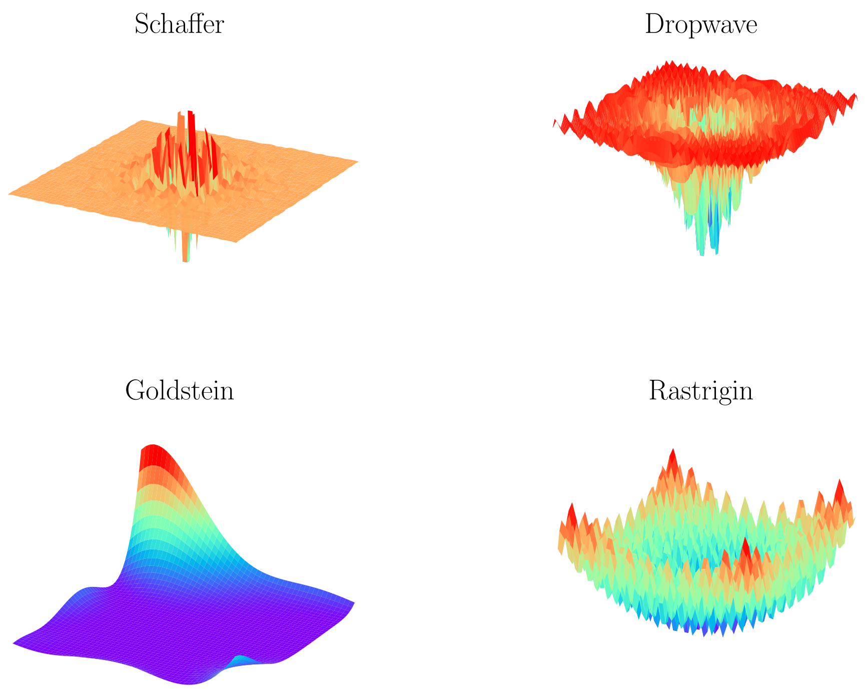

Figure E.2: Surface plots for benchmark functions Schaffer, Dropwave, Goldstein-Price and Rastrigin.

REFERENCES

[1] J. Garcke and R. Iza-Teran. Machine learning approaches for repositories of numerical simulation results. In *10th European LS-DYNA Conference*, volume 2015, 2015.

[2] F. Viana and R. Haftka. Surrogate-based optimization with parallel simulations using the probability of improvement. In *13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, page 9392, 2010.

[3] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[5] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550:354, oct 2017.

[6] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *ACM Sigmod Record*, volume 29, pages 427–438. ACM, 2000.

[7] T. Gao and J. R. Kitchin. Modeling palladium surfaces with density functional theory, neural networks and molecular dynamics. *Catalysis Today*, 2018.

[8] M. Hughes, J. K. Kodros, J. R. Pierce, M. West, and N. Riemer. Machine learning to predict the global distribution of aerosol mixing state metrics. *Atmosphere*, 9(1):15, 2018.

[9] S. Rasp, M. S. Pritchard, and P. Gentine. Deep learning to represent subgrid processes in climate models. *Proceedings of the National Academy of Sciences*, 115(39):9684–9689, 2018. doi: 10.1073/pnas.1810286115.

[10] L. Ladický, S. Jeong, B. Solenthaler, M. Pollefeys, and M. Gross. Data-driven fluid simulations using regression forests. *ACM Trans. Graph.*, 34 (6):199:1–199:9, October 2015. ISSN 0730-0301. doi: 10.1145/2816795.2818129.

[11] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin. Accelerating Eulerian fluid simulation with convolutional networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 3424–3433. PMLR, 06–11 Aug 2017.

[12] Y. Chen, J. S. Hesthaven, Y. Maday, and J. Rodríguez. Certified reduced basis methods and output bounds for the harmonic Maxwell's equations. *SIAM Journal on Scientific Computing*, 32(2):970–996, 2010.

[13] D. E. Breen, D. H. House, and M. J. Wozny. A particle-based model for simulating the draping behavior of woven cloth. *Textile Research Journal*, 64(11):663–685, 1994. doi: 10.1177/004051759406401106.

[14] N. Riemer, M. West, R. A. Zaveri, and R. C. Easter. Simulating the evolution of soot mixing state with a particle-resolved aerosol model. *Journal of Geophysical Research: Atmospheres*, 114(D9):D09202, 2009. doi: 10.1029/2008JD011073.

[15] D. Morale, V. Capasso, and K. Oelschlager. An interacting particle system modelling aggregation behavior: from individuals to populations. *Journal of Mathematical Biology*, 50(1):49–66, 2005. doi: 10.1007/s00285-004-0279-1.

[16] B. Giera, L. A. Zepeda-Ruiz, A. J. Pascall, J. D. Kuntz, C. M. Spadaccini, and T. H. Weisgraber. Mesoscale particle-based model of electrophoresis. *Journal of The Electrochemical Society*, 162(11):D3030–D3035, 2015. doi: 10.1149/2.0161511jes.

[17] D. O. Potyondy and P. A. Cundall. A bonded-particle model for rock. *International Journal of Rock Mechanics and Mining Sciences*, 41(8): 1329–1364, 2004. doi: 10.1016/j.ijrmms.2004.09.011.

[18] Z. P. Bažant, M. R. Tabbara, M. T. Kazemi, and G. Pijaudier-Cabot. Random particle model for fracture of aggregate or fiber composites. *Journal of Engineering Mechanics*, 116(8):1686–1705, 1990. doi: 10.1061/(ASCE)0733-9399(1990)116:8(1686).

[19] R.E.L. DeVille, N. Riemer, and M. West. Weighted Flow Algorithms (WFA) for stochastic particle coagulation. *Journal of Computational Physics*, 230(23):8427–8451, 2011. doi: 10.1016/j.jcp.2011.07.027.

[20] D. P. Kroese, T. Taimre, and Z. I. Botev. Handbook of Monte Carlo Methods. pages 347–380. John Wiley & Sons, Inc., Hoboken, NJ, 2011. ISBN 9781118014967. doi: 10.1002/9781118014967.

[21] P. Glasserman. Monte Carlo Methods in Financial Engineering. volume 53 of *Stochastic Modelling and Applied Probability*, pages 185–279. Springer New York, New York, NY, 2003. ISBN 978-1-4419-1822-2. doi: 10.1007/978-0-387-21617-1.

[22] R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction*. MIT press, 2018.

[23] R. Ranganath, S. Gerrish, and D. Blei. Black Box Variational Inference. In Samuel Kaski and Jukka Corander, editors, *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 814–822, Reykjavik, Iceland, 22–25 Apr 2014. PMLR.

[24] S. M. T. Ehrlichman and S. G. Henderson. Adaptive control variates for pricing multi-dimensional American options. *Journal of Computational Finance*, 11(1):65–91, 2007. doi: 10.21314/JCF.2007.167.

[25] S. G. Henderson and B. Simon. Adaptive simulation using perfect control variates. *Journal of Applied Probability*, 41(3):859–876, 2004. doi: 10.1239/jap/1091543430.

[26] S. Kim and S. G. Henderson. Adaptive control variates for finite-horizon simulation. *Mathematics of Operations Research*, 32(3):508–527, 2007. doi: 10.1287/moor.1070.0251.

[27] B. Jourdain. Adaptive variance reduction techniques in finance. In H. Albrecher, W. J. Runggaldier, and W. Schachermayer, editors, *Advanced Financial Modelling*, number 8 in Radon Series on Computational and Applied Mathematics, pages 205–222. Walter de Gruyter, 2009. ISBN 9783110213140. doi: 10.1515/9783110213140.

[28] P. A. Maginnis, M. West, and G. E. Dullerud. Application of variance reduction techniques for tau-leaping systems to particle filters. In *Proceedings of the 51st IEEE Conference on Decision and Control (CDC 2012)*, pages 6683–6689. IEEE, 2012.

[29] P. A. Maginnis, M. West, and G. E. Dullerud. Exact simulation of continuous time markov jump processes with anticorrelated variance reduced monte carlo estimation. In *Proceedings of the 53rd IEEE Conference on Decision and Control (CDC 2014)*, pages 3401–3407. IEEE, 2014.

[30] P. A. Maginnis, M. West, and G. E. Dullerud. Variance-reduced simulation of lattice discrete-time markov chains with applications in reaction networks. *Journal of Computational Physics*, 322:400–414, 2016.

[31] M. Benaim and J.-Y. Le Boudec. A class of mean field interaction models for computer and communication systems. *Performance Evaluation*, 65 (11):823–838, 2008. doi: 10.1016/j.peva.2008.03.005.

[32] R. Carmona, F. Delarue, and A. Lachapelle. Control of McKean-Vlasov dynamics versus mean field games. *Mathematics and Financial Economics*, 7(2):131–166, 2013. doi: 10.1007/s11579-012-0089-y.

[33] A. Chattopadhyay, S. Das Sarma, and A. J. Millis. Transition temperature of ferromagnetic semiconductors: a dynamical mean field study. *Phys. Rev. Lett.*, 87:227202, Nov 2001. doi: 10.1103/PhysRevLett.87.227202.

[34] R. A. Zaveri, J. C. Barnard, R. C. Easter, N. Riemer, and M. West. Particle-resolved simulation of aerosol size, composition, mixing state, and the associated optical and cloud condensation nuclei activation properties in an evolving urban plume. *Journal of Geophysical Research: Atmospheres*, 115(D17), 2010. doi: 10.1029/2009JD013616.

[35] Y. Cao, M. Y. Hussaini, T. Zang, and A. Zatezalo. A variance reduction method based on sensitivity derivatives. *Applied Numerical Mathematics*, 56(6):800–813, 2006. doi: 10.1016/j.apnum.2005.06.010.

[36] D. J. Aldous. Deterministic and stochastic models for coalescence (aggregation and coagulation): a review of the mean-field theory for probabilists. *Bernoulli*, 5(1):3–48, 1999. doi: 10.2307/3318611.

[37] R. I. Jennrich. Asymptotic properties of non-linear least squares estimators. *The Annals of Mathematical Statistics*, 40(2):633–643, 1969. doi: 10.1214/aoms/1177697731.

[38] T. Amemiya. *Advanced Econometrics*. Harvard University Press, Cambridge, MA, 1985. ISBN 9780674005600.

[39] J. Ching, N. Riemer, and M. West. Black carbon mixing state impacts on cloud microphysical properties: effects of aerosol plume and environmental conditions. *Journal of Geophysical Research: Atmospheres*, 121 (10):5990–6013, 2016. doi: 10.1002/2016JD024851.

[40] L. Fierce, T. C. Bond, S. E. Bauer, F. Mena, and N. Riemer. Black carbon absorption at the global scale is affected by particle-scale diversity in composition. *Nature Communications*, 7, 2016. doi: 10.1038/ncomms12361.

[41] J. H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–67, 1991. doi: 10.1214/aos/1176347963.

[42] T. Hastie, R. Tibshirani, and J. H. Friedman. The Elements of Statistical Learning. Springer Series in Statistics. Springer New York Inc., New York, NY, 2009. ISBN 978-0-387-84857-0. doi: 10.1007/b94608.

[43] S. Milborrow. Derived from mda:mars by T. Hastie and R. Tibshirani. *earth: Multivariate Adaptive Regression Splines*, 2011. URL `http://CRAN.R-project.org/package=earth`. R package.

[44] S. Amaran, N. V. Sahinidis, B. Sharda, and S. J. Bury. Simulation optimization: a review of algorithms and applications. *Annals of Operations Research*, 240(1):351–380, May 2016.

[45] P. Rakshit, A. Konar, and S. Das. Noisy evolutionary optimization algorithms-a comprehensive survey. *Swarm and Evolutionary Computation*, 2016.

[46] P. Köchel and U. Nieländer. Simulation-based optimisation of multi-echelon inventory systems. *International Journal of Production Economics*, 93:505–513, 2005.

[47] P. Prakash, G. Deng, M. C. Converse, J. G. Webster, D. M. Mahvi, and M. C. Ferris. Design optimization of a robust sleeve antenna for hepatic microwave ablation. *Physics in Medicine and Biology*, 53(4):1057, 2008.

[48] J. Xie, P. I. Frazier, S. Sankaran, A. Marsden, and S. Elmohamed. Optimization of computationally expensive simulations with Gaussian processes and parameter uncertainty: Application to cardiovascular surgery. In *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*, pages 406–413. IEEE, 2012.

[49] P. A. Romero, A. Krause, and F. H. Arnold. Navigating the protein fitness landscape with Gaussian processes. *Proceedings of the National Academy of Sciences*, 110(3):E193–E201, 2013.

[50] X. M. Chen, L. Zhang, X. He, C. Xiong, and Z. Li. Surrogate-based optimization of expensive-to-evaluate objective for optimal highway toll charges in transportation network. *Computer-Aided Civil and Infrastructure Engineering*, 29(5):359–381, 2014.

[51] C. Osorio and M. Bierlaire. A simulation-based optimization approach to perform urban traffic control. In *TRISTAN VII, Triennial Symposium on Transportation Analysis*, number EPFL-TALK-152410, 2010.

[52] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012. URL `https://github.com/JasperSnoek/spearmint`.

[53] J. Wu and P. Frazier. The parallel knowledge gradient method for batch Bayesian optimization. In *Advances in Neural Information Processing Systems*, pages 3126–3134, 2016.

[54] Y. Wang, J.-Y. Audibert, and R. Munos. Algorithms for infinitely many-armed bandits. In *Advances in Neural Information Processing Systems*, pages 1729–1736, 2009.

[55] S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvári. X-armed bandits. *Journal of Machine Learning Research*, 12(May):1655–1695, 2011.

[56] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

[57] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3): 462–466, 1952.

[58] J. C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, 1992.

[59] J. C. Spall. *Introduction to stochastic search and optimization: estimation, simulation, and control*, volume 65. John Wiley & Sons, 2005.

[60] S. Taghiyeh and J. Xu. A new particle swarm optimization algorithm for noisy optimization problems. *Swarm Intelligence*, 10(3):161–192, 2016.

[61] R. R. Barton and J. S. Ivey Jr. Nelder-mead simplex modifications for simulation optimization. *Management Science*, 42(7):954–973, 1996.

[62] R. C. Ball, T. M. A. Fink, and N. E. Bowler. Stochastic annealing. *Physical Review Letters*, 91(3):030201, 2003.

[63] F. Neri and A. Caponio. A differential evolution for optimisation in noisy environment. *International Journal of Bio-Inspired Computation*, 2(3-4):152–168, 2010.

[64] E. J. Anderson and M. C. Ferris. A direct search algorithm for optimization with noisy function evaluations. *SIAM Journal on Optimization*, 11(3):837–857, 2001.

[65] A. R. Conn, K. Scheinberg, and Ph. L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical Programming*, 79(1):397–414, 1997.

[66] R. G. Regis and C. A. Shoemaker. A stochastic radial basis function method for the global optimization of expensive functions. *INFORMS Journal on Computing*, 19(4):497–509, 2007.

[67] R. T. Haftka, D. Villanueva, and A. Chaudhuri. Parallel surrogate-assisted global optimization with expensive functions–a survey. *Structural and Multidisciplinary Optimization*, 54(1):3–13, 2016.

[68] E. Contal, D. Buffoni, A. Robicquet, and N. Vayatis. Parallel Gaussian process optimization with upper confidence bound and pure exploration. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 225–240. Springer, 2013.

[69] T. Desautels, A. Krause, and J. W. Burdick. Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization. *The Journal of Machine Learning Research*, 15(1):3873–3923, 2014.

[70] A. Shah and Z. Ghahramani. Parallel predictive entropy search for batch global optimization of expensive objective functions. In *Advances in Neural Information Processing Systems*, pages 3330–3338, 2015.

[71] J. Azimi, A. Fern, and X. Z. Fern. Batch Bayesian optimization via simulation matching. In *Advances in Neural Information Processing Systems*, pages 109–117, 2010.

[72] J. González, Z. Dai, P. Hennig, and N. Lawrence. Batch Bayesian optimization via local penalization. In *Artificial Intelligence and Statistics*, pages 648–657, 2016. URL `https://github.com/SheffieldML/GPyOpt`.

[73] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*, volume 1. MIT press Cambridge, 2006.

[74] J. Quinonero-Candela and C. E. Rasmussen. Analysis of some methods for reduced rank Gaussian process regression. In *Switching and Learning in Feedback Systems*, pages 98–127. Springer, 2005.

[75] R. G. Regis and C. A. Shoemaker. Parallel stochastic global optimization using radial basis functions. *INFORMS Journal on Computing*, 21(3):411–426, 2009.

[76] R. G. Regis. Trust regions in Kriging-based optimization with expected improvement. *Engineering Optimization*, 48(6):1037–1059, 2016.

[77] D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017. URL `https://archive.ics.uci.edu/ml`.

[78] Y. LeCun, C. Cortes, and C. J. C. Burges. The MNIST database of handwritten digits. URL `http://yann.lecun.com/exdb/mnist/`.

[79] J. Wu, M. Poloczek, A. G. Wilson, and P. Frazier. Bayesian optimization with gradients. In *Advances in Neural Information Processing Systems*, pages 5267–5278, 2017.

[80] Tensorflow tutorial. URL `https://www.tensorflow.org/versions/r1.1/get_started/mnist/beginners`.

[81] B. L. Nelson, J. Swann, D. Goldsman, and W. Song. Simple procedures for selecting the best simulated system when the number of alternatives is large. *Operations Research*, 49(6):950–963, 2001.

[82] E. C. Ni, S. R. Hunter, and S. G. Henderson. Ranking and selection in a high performance computing environment. In *Simulation Conference (WSC), 2013 Winter*, pages 833–845. IEEE, 2013.

[83] R. A. Zaveri, R. C. Easter, J. D. Fast, and L. K. Peters. Model for simulating aerosol interactions and chemistry (MOSAIC). *Journal of Geophysical Research: Atmospheres*, 113(D13), 2008. doi: 10.1029/2007JD008782.