Bayesian optimisation for expensive physical experiments and computer simulators with application in fluid dynamics

Mike Diessner

Thesis submitted for the degree of
Doctor of Philosophy

School of Computing
Newcastle University
Newcastle upon Tyne
United Kingdom

August 2024

For Lara.

Acknowledgements

I would like to express my sincere gratitude towards Dr. Richard D. Whalley and Prof. Kevin J. Wilson for their invaluable guidance and advice during my doctoral studies. Their support improved this thesis in many ways and greatly shaped me as a researcher.

—

I am deeply grateful to Newcastle University and the Centre for Doctoral Training in Cloud Computing for Big Data, who made this research possible. In particular, I would like to thank Jennifer Wood, Andrew Turnbull, Elaine Adams, the Experimental Fluid Dynamics Lab and my fellow PhD students for their generous help and continuous support.

—

I also want to thank Dr. Joseph O'Connor, Dr. Andrew Wynn and Prof. Sylvain Laizet from the Turbulence Simulation Group at Imperial College London for their valuable support in applying Bayesian optimisation to expensive simulators in computational fluid dynamics.

—

Zuletzt, möchte ich meinen Eltern, meiner Familie und meinen Freunden in Deutschland danken, die alles getan haben um mich von weitem zu unterstützen und immer für mich da waren.

Abstract

Expensive black-box functions such as physical experiments and computer simulators are challenging to optimise as they cannot be solved analytically, and only small numbers of function evaluations are available for optimisation. This prevents use of conventional methods that rely on gradient information or larger numbers of function evaluations, requiring a specialised optimisation strategy. Bayesian optimisation is a sample-efficient strategy that represents the objective function through a surrogate model and guides the exploration of the input space with heuristics—so-called acquisition criteria—to select promising candidate points sequentially. Expensive black-box functions are a common occurrence in fluid dynamics where the underlying systems, for example the Navier-Stokes equations, can be too complex to solve explicitly and can be viewed as a black box. In addition, the expensive nature of the associated experiments and simulations makes Bayesian optimisation a prime candidate. However, the Bayesian optimisation literature is mainly geared towards statisticians and computer scientists and is potentially challenging to scrutinise and apply for non-experts. Thus, the main motivation of this thesis is to make Bayesian optimisation more accessible and answer some fundamental questions overlooked in the literature, while also developing techniques for specific challenges encountered in but not limited to fluid dynamics. This thesis studies three topics for applying Bayesian optimisation to experiments and simulators. Firstly, it investigates key choices in Bayesian optimisation empirically, such as the choice of the acquisition criterion and the number of data points used for initialisation, and applies the findings to two computer simulators with the objective of controlling air flow to maximise the skin-friction drag reduction over a flat plate—mimicking the surface of a moving vehicle such as the wing of an aeroplane. Secondly, NUBO—an open-source Python package for optimising expensive experiments and simulators aimed at practitioners of Bayesian optimisation—is presented, and its functionalities are discussed. This transparent package allows users to tailor the optimisation loop to their specific problems and supports sequential single-point, parallel multi-point and asynchronous optimisation for bounded, constrained and mixed (discrete and continuous) input parameter spaces. Lastly, problems affected by external environmental variables that cannot be controlled are investigated, and ENVBO—a novel algorithm—is introduced. ENVBO fits a global surrogate model over all controllable and environmental variables but optimises the acquisition criterion only with regard to the controllable variables while keeping the environmental variables fixed at a current measurement. Important properties of ENVBO, such as the robustness to noisy objective functions and the number of environmental variables, are studied. ENVBO is applied to a wind farm simulator to maximise energy production by (a) finding optimal positions for four wind

turbines within a complex terrain with changing wind directions and (b) setting optimal derating factors of a row of five wind turbines subject to changing wind speeds.

Contents

List of Figures

List of Tables

List of Acronyms

AEP	Annual Energy Production
ARD	Automatic Relevance Determination
AUC	Area Under the Curve
BO	Bayesian Optimisation
BUCB	Batched Upper Confidence Bound
CFD	Computational Fluid Dynamics
CL	Constant Liar
CPU	Central Processing Unit
DACE	Design and Analysis of Computer Experiments
DIRECT	DIverging RECTangles optimisation algorithm
EGO	Efficient Global Optimisation
EI	Expected Improvement
ENVBO	Bayesian optimisation with environmental conditions
ES	Entropy Search
GB	GigaBytes
GDR	Global Drag Reduction
GP	Gaussian Process
L-BFGS-B	Limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm for Box bounds
LHS	Latin Hypercube Sampling
LogEI	Logarithmic Expected Improvement
MAP	Maximum A Posteriori
MAPE	Mean Absolute Percentage Error
MC	Monte Carlo
MEMS	Micro-Electro-Mechanical-Systems
MEP	Mean Energy Production
MES	Max-value Entropy Search
MLE	Maximum Likelihood Estimation

NM	Nelder-Mead
NUBO	Newcastle University Bayesian Optimisation
NES	Net Energy Savings
PES	Predictive Entropy Search
PI	Probability of Improvement
pip	Package installer for Python
PyPI	Python Package Index
RBF	Radial Basis Function
ROC	Receiver Operating Characteristic
SLSQP	Sequential Least Squares Programming Optimiser
STEM	Science, Technology, Engineering and Mathematics
UCB	Upper Confidence Bound

Chapter 1

Introduction

This chapter introduces the problem which is of central interest to this thesis, that of optimising expensive physical experiments and simulators, by defining key characteristics of expensive black-box functions and explaining why conventional optimisation methods all have limitations with regards to solving this problem. It then briefly outlines Bayesian optimisation and its advantages in this context. Following this, the overarching research aim and the specific objectives are described before a thesis outline, including key contributions, is given. The chapter ends by detailing all publications that resulted from this research.

1.1 Expensive physical experiments and computer simulators

Expensive physical experiments and computer simulators fall into the category of expensive-to-evaluate black-box functions [20, 53, 58]. By expensive we mean that each evaluation of input parameters generates high costs. These can be operational costs, such as energy, computing resources and manual labour, or equipment costs, as many experiments and simulators are expensive to set up initially. Moreover, typically the only way to gather information about the problem in question is to select sets of input values and observe their output. The inner workings between providing inputs and observing their results are unknown or too complex to determine exactly. Thus, problems are treated as black boxes, which even specialists with expert knowledge cannot predict precisely. Expensive black-box functions are discussed further in Section 2.1.

To illustrate this further, we introduce the following example from the field of fluid dynamics [123, 41, 145, 124]. Consider a wind tunnel with a flat plate parallel to the floor. When air flows over the flat plate, a layer of turbulence is created close to its surface, creating skin friction drag. This mimics the real-life situation where air flows over the wing

of an aircraft. The turbulence close to the wing—or generally to the aircraft's surface—is responsible for over half of its fuel consumption [22]. Thus, interest lies in reducing this turbulence and the resulting drag. Low-amplitude blowing actuators provide a solution [89, 106] as they allow the airflow to be actively controlled by imposing a specific blowing strategy, or profile, that reduces the turbulence. In an experiment, actuators create a uniform blowing region that blows air upwards, perpendicular to the flat plate. This manipulates the airflow over the flat plate from the blowing region onwards. The challenge is to find the optimal strategies that maximise the global drag reduction—i.e., the drag reduction over the whole flat plate. Investigations of this problem could range from simple one-dimensional setups where one blowing amplitude is uniformly imposed through to complicated designs with many parameters such as a travelling wave that is defined by a blowing amplitude, frequency and wavelength. While the setup of such experiments is relatively simple, data collection is complex as measuring skin-friction drag in experiments is difficult even for experts, and measuring the global drag reduction requires a large number of measurements over the whole flat plate. An alternative to performing experiments in a wind tunnel is to perform simulations of the behaviour of the flat plate instead. However, these simulations are costly and can take up to 12 hours on thousands of CPU cores, as discussed in Section 3.3. Furthermore, while there are governing equations for turbulence development, they are far too complex to predict the effects of different blowing strategies accurately. The problem is essentially a black box where the primary method to obtain an understanding of the relationship between blowing and skin friction drag is observing the output for the provided inputs.
If it were possible to find an optimal blowing strategy for this example experiment and transfer the results to aircraft, it could reduce the skin-friction drag, saving money, preventing emissions and protecting public health. Indeed, Bushnell and Hefner [22] estimate that a skin-friction drag reduction of just 3% would save £1 million in fuel costs annually per aircraft. This highlights the importance of optimising expensive black-box problems that can be extended to physical experiments and computer simulators in many different areas, including chemistry, physics, biology, engineering and computer science [58].

1.2 Bayesian optimisation

Typically, optimisation methods make strong assumptions, such as convexity and the availability of gradient information, about the objective function—i.e., the function at the centre of the optimisation problem representing the system that we seek to optimise—that will not always hold for expensive physical experiments and computer simulators. Indeed, most assumptions cannot be verified as the given problem is a black box that prohibits such verification. Other commonly used methods require large numbers of function evaluations

to be successful and assume implicitly that observations are cheap. This contradicts the nature of expensive black-box problems.
Suitable strategies in this context typically use a three-step process to select data points sequentially and explore the input space efficiently. First, the objective function is represented by a surrogate model fitted to the available data. Second, the next point is selected by maximising a heuristic (a.k.a. acquisition functions) that guides the optimisation. Third, this point is observed from the objective function, and the process is repeated with the new input-output pair. Typically, the first point used to initialise the three-step process is chosen at random but it can also be determined by expert knowledge. Bayesian optimisation is a surrogate model-based optimisation strategy and thus suitable to optimise expensive, derivative-free, black-box functions [20, 53, 58]. Section 2.2 discusses Bayesian optimisation in relation to other approaches in more depth.
Although developed in the 1960s for applications in engineering [111, 110], Bayesian optimisation has had a renaissance since the late 1990s [171, 94] mainly due to the improved computational capabilities of modern CPUs. Bayesian optimisation has been developed steadily and expanded over the following decades to address problems with noise, constraints, high-dimensional inputs, and multiple objectives, tasks and fidelities. It has been applied to problems in many different scientific fields in the area of science, technology, engineering and mathematics (STEM) [58]. Despite these efforts, more work is required to understand Bayesian optimisation better and extend it to unique applications. For example, there has been limited attention given to physical experiments in environments where not all influential variables can be controlled and the aim is to find optimal values for the controllable variables conditional on these uncontrollable variables. Furthermore, there is a gap between the state-of-the-art in Bayesian optimisation research and what is available to scientists from other disciplines looking to apply it to their experiments and simulators. Typically, software can be overly complex and challenging to comprehend in detail. Thus, simpler implementations allowing users to fully understand and verify each algorithm are required.

1.3 Research aim and objectives

This thesis' overarching aim is **to further the understanding, methodology and application of Bayesian optimisation for expensive physical experiments and computer simulators**. The following objectives have been set to achieve this aim:

1. Investigate the properties of key choices to be made in Bayesian optimisation to make general recommendations for specific problems (simulation study).

2. Transfer findings from the simulation study to expensive experiments and simulators

(application).

3. Develop software that implements methods transparently and enables scientists and practitioners to optimise expensive experiments and simulators (software development).

4. Develop an approach to optimise expensive experiments with randomly changing environmental variables that cannot be controlled (methodology development).

5. Apply Bayesian optimisation to experiments and simulators in flow control to maximise turbulent drag reduction (application).

1.4 Thesis outline and contributions

Chapter 2 provides the background material necessary to understand this thesis. It includes a mathematical formulation of expensive black-box problems that are the focus of this work. Further, Bayesian optimisation is introduced, its advantages over other methods are discussed, and a brief account of its history is given. This chapter focuses on Gaussian processes used for surrogate modelling and the acquisition functions used to guide the optimisation over the input parameter space. Methods for special cases, such as noisy, constrained and high-dimensional optimisation, are reviewed and sign-posted for further reading.

Chapter 3 investigates the influence of fundamental choices and different strategies on the performance of a Bayesian optimisation algorithm. Notably, various acquisition functions with different numbers of data points used to initialise the algorithms are investigated. Cheap test functions with different shapes and properties are used to find robust Bayesian optimisation strategies. This investigation will show that acquisition functions based on confidence bounds are superior to other methods for our test functions and that the number of initial data points is not a significant driver of performance. Furthermore, acquisition functions approximated by Monte Carlo sampling prove competitive with the exact acquisition functions and have the advantage of enabling parallel evaluations via multiple concurrent simulations. These findings will then be applied to two simulators in computational fluid dynamics (CFD), aiming to find optimal blowing strategies of active blowing actuators that maximise global drag reduction and net energy savings. While solutions with significant global drag reduction (over 22%) are found, the net energy savings are modest (below 1%), indicating the need for cheaper actuators. This work addresses research objectives 1 and 2 and has been published in Diessner et al. [41].

Chapter 4 presents the Python package NUBO (Newcastle University Bayesian Optimisation) developed to enable researchers and practitioners to use Bayesian optimisation to optimise their expensive black-box problems. NUBO focuses on providing a transparent,

user-friendly and flexible implementation of Bayesian optimisation. It supports sequential single-point, parallel multi-point and asynchronous optimisation of bounded, constrained and mixed parameter input spaces. This chapter contains code snippets that show how NUBO can be used in practice. Comparisons to other Bayesian optimisation packages show that the focus on simplicity does not come at the cost of performance. This chapter addresses research objective 3 and has been published in Diessner, Wilson, and Whalley [42].

Chapter 5 considers an extension of Bayesian optimisation where some variables are uncontrollable. Specifically, the work herein assumes that some variables are given externally by the environment and change randomly. The algorithm ENVBO is developed, extending Bayesian optimisation to leverage the correlation of the objective function realisations with small changes in the environmental variables. The aim is to find the controllable inputs that maximise the objective function conditional on the environmental variables. ENVBO is applied to a wind farm simulator and benchmarked against standard Bayesian optimisation, the Nelder-Mead algorithm [143] and the sequential least squares programming optimiser (SLSQP) [107]. The comparison shows that ENVBO finds better or at least comparable solutions to the benchmarks while requiring fewer function evaluations. Thus, ENVBO presents a sample-efficient algorithm for optimising expensive black-box problems with randomly changing variables. This chapter addresses research objective 4 and has been published in Diessner, Wilson, and Whalley [43] and Diessner et al. [40].

Lastly, the **Conclusion** summarises the contributions in this thesis and provides research directions for future works.

1.5 Related publications

All research chapters of this thesis (Chapters 3–5) have been published (or are in the process of being published) as first-author papers in journals or conference proceedings. Additionally, the following list of publications features two second-author papers where Bayesian optimisation was applied to experiments and simulators in active flow control. The work in the following articles is entirely my own if not otherwise stated below the articles.

Journal articles

[41] M. Diessner, J. O'Connor, A. Wynn, S. Laizet, Y. Guan, K. Wilson, and R. D. Whalley. "Investigating Bayesian Optimization for Expensive-to-Evaluate Black Box Functions: Application in Fluid Dynamics". In: *Frontiers in Applied Mathematics and Statistics* 8 (2022), p. 1076296

The simulations for the applications in Section 4 of this article were run by Joseph O'Connor who also provided information about their specific implementation in Xcompact3D [8]. This study is included as Chapter 3 in this thesis.

[145]J. O'Connor, M. Diessner, K. Wilson, R. D. Whalley, A. Wynn, and S. Laizet. "Optimisation and Analysis of Streamwise-Varying Wall-Normal Blowing in a Turbulent Boundary Layer". In: *Flow, Turbulence and Combustion* 110.4 (2023), pp. 993–1021

My contribution to this work is co-planning the setup of the simulations, writing Section 3.2 of this article on Bayesian optimisation (including generating the figures), and preparing the Python code for the Bayesian optimisation algorithm. This work is not included in this thesis.

[42] M. Diessner, K. Wilson, and R. D. Whalley. "NUBO: A Transparent Python Package for Bayesian Optimisation". In: *arXiv preprint arXiv:2305.06709* (2023). Accepted by Journal of Statistical Software

This journal article is included as Chapter 4 in this thesis.

[43] M. Diessner, K. J. Wilson, and R. D. Whalley. "On the Development of a Practical Bayesian Optimization Algorithm for Expensive Experiments and Simulations With Changing Environmental Conditions". In: *Data-Centic Engineering* 5 (2024), e45

This study is included as Chapter 5 in this thesis.

Conference articles

[40] M. Diessner, X. Chen, K. J. Wilson, and R. D. Whalley. "Optimising Active Flow Control Strategies for Random and Controlled Wind Speeds via Bayesian Optimisation". In: *Symposium on Turbulence and Shear Flow Phenomena (TSFP).* TSFP. 2024

The application published in this article is included in Section 5.6.

[27] X. Chen, M. Diessner, K. J. Wilson, and R. D. Whalley. "Optimizing Wall Blowing for Global Skin-Friction Drag Reduction Using a Bayesian Optimization Framework". In: *Symposium on Turbulence and Shear Flow Phenomena (TSFP).* TSFP. 2024

My contribution to this work is co-planning the setup of the simulations and preparing the Python code for the Bayesian optimisation algorithm. This conference article is not included in this thesis.

Chapter 2

Background

This chapter lays the methodological foundation for the investigation of specific aspects of Bayesian optimisation in the subsequent chapters. Section 2.1 defines expensive black-box problems whose optimisation is central to this research. Section 2.2 introduces Bayesian optimisation and explains why it is favoured in the literature over other algorithms to optimise expensive black-box functions. Section 2.3 describes Gaussian processes used as a surrogate model to represent an objective function and presents alternative surrogate models from the literature. Section 2.4 discusses how different acquisition functions guide the optimisation. Section 2.5 provides an overview of special cases of Bayesian optimisation that can potentially be important for optimising physical experiments and computer simulators, such as noisy and parallel observations.

2.1 Expensive black-box problems and notation

This thesis considers problems where the overarching aim is discovering a global maximiser $\boldsymbol{x}^*$—a d-dimensional input vector in $\mathbb{R}^d$—that maximises a continuous objective function f such that

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmax}}\, f(\boldsymbol{x}), \tag{2.1}$$

where $\boldsymbol{x}$ is a vector of inputs and $f(\cdot)$ is the objective function, defined below. This optimisation is challenging as the objective function is assumed to be a black box that is expensive-to-evaluate and derivative-free [20, 53, 58]. Thus, mathematical properties such as convexity or concavity are unknown and cannot be used within the optimisation, observing the result of an arbitrary input point generates high costs—such as in resources, time and manual labour—and gradient information about the objective function cannot be used to aid and speed-up the optimisation. The only way of gathering information about the objective function is to provide it with an input point $\boldsymbol{x}$ and observe the corresponding

output y such that

$$y = f(\boldsymbol{x}) : \mathcal{X} \mapsto \mathbb{R}, \tag{2.2}$$

where $\boldsymbol{x} \in \mathbb{R}^d$ is a d-dimensional input vector and y is a scalar output. The input space $\mathcal{X}$ is typically a hyper-rectangle restricted by a lower and an upper bound for each input dimension such that $\mathcal{X} \in [\boldsymbol{a}, \boldsymbol{b}]^d$ where $a^k \leq x^k \leq b^k$ for all $k = 1, \ldots, d$ [53, 20]. The problem described above considers a deterministic objective function. Alternatively, the objective function can also be stochastic such that

$$y = f(\boldsymbol{x}) + \epsilon : \mathcal{X} \mapsto \mathbb{R}, \tag{2.3}$$

where ϵ is a noise term, which is typically considered to be sampled from a Gaussian distribution with zero mean and variance σ^2 such that $\epsilon \sim \mathcal{N}\left(0, \sigma^2\right)$. Section 2.5.1 discusses the optimisation of noisy objective functions.
In this thesis, we denote the input vector $\boldsymbol{x}_i$ containing individual input values x_i^k for all $k = 1, \ldots, d$ such that

$$\boldsymbol{x}_i = \begin{bmatrix} x_i^1 & x_i^2 & \ldots & x_i^d \end{bmatrix}, \tag{2.4}$$

and a set of multiple data points $\boldsymbol{x}_i$ for all $i = 1, \ldots, n$ as the input matrix $\boldsymbol{X}$ such that

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_n \end{bmatrix}. \tag{2.5}$$

We further denote the outputs corresponding to multiple data points $\boldsymbol{X}$ as the output vector $\boldsymbol{y}$ such that

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} f(\boldsymbol{x_1}) \\ f(\boldsymbol{x_2}) \\ \vdots \\ f(\boldsymbol{x_n}) \end{bmatrix} = f(\boldsymbol{X}), \tag{2.6}$$

and data $\mathcal{D}$ as a set of input-output pairs such that

$$\mathcal{D} = (\boldsymbol{X}, \boldsymbol{y}). \tag{2.7}$$

Within the Bayesian optimisation loop we use indices for the data $\mathcal{D}$, the inputs $\boldsymbol{X}$ and the outputs $\boldsymbol{y}$ to indicate the number of included elements. The data used to initialise an algorithm is indexed with 0, i.e., $\mathcal{D}_0$, $\boldsymbol{X}_0$ and $\boldsymbol{y}_0$, while data of a specific optimisation step n is indexed with n, i.e., $\mathcal{D}_n$, $\boldsymbol{X}_n$ and $\boldsymbol{y}_n$. For example, data for the 20^{th} optimisation step is referred to as $\mathcal{D}_{20}$ and includes the initial data $\mathcal{D}_0$ and input-output pairs $(\boldsymbol{x}_n, y_n)$

for all $n = 1, 2, \ldots, 20$.

2.2 Bayesian optimisation

Many methods for optimising black-box functions have been previously proposed in the literature ranging from heuristical methods such as evolutionary algorithms [220], simulated annealing [105] and tabu search [65] over reinforcement learning [189] to derivative-free methods [31, 113] such as directional direct search algorithms [4], simplicial direct search algorithms [143] and trust-region methods [31]. However, these methods assume that the objective function can be evaluated cheaply and require large numbers of function evaluations to be successful. Thus, they are unsuitable for optimising expensive black-box functions, e.g., physical experiments and computer simulators that generate high costs with each observation [202].
Suitable methods for optimising expensive black-box functions typically employ a three-step process where data points are selected sequentially, and the input space is explored sample-efficiently:

1. The objective function is represented by fitting a surrogate model to the available data.
2. A heuristic based on the surrogate model is used to select the next data point to evaluate.
3. The objective function is evaluated at the selected input, and the process is repeated.

Due to the importance of the surrogate model, these methods are called surrogate-based methods [202]. Within this group, different surrogate models are used, such as polynomials [61], radial basis functions [74, 163, 164, 165] and support vector machines [33, 35]. In contrast to these typically frequentist methods, Bayesian optimisation uses Bayesian models—typically a Gaussian process—as their surrogate model of choice [20, 175, 53, 58]. Bayesian models have the advantage of including prior beliefs about the objective function, such as smoothness, and can require fewer function evaluations to achieve a good fit. They are thus more sample-efficient than frequentist models that can require much greater numbers of evaluations [131, 94, 169, 20]. Booker [15] introduced the Surrogate management framework based on kriging [128] with Gaussian processes in the area of design and analysis of computer experiments (DACE) [167], which also uses a Bayesian model and is very similar to Bayesian optimisation.

2.2.1 A brief history and applications

This section gives a brief account of the history of Bayesian optimisation, as Bayesian optimisation has a rich body of literature that would exceed this thesis' limit. Garnett

[58] provides a comprehensive history that goes back further than this section.
The origin of Bayesian optimisation can be traced back to the early 1960s, where Kushner [111, 110] introduced an optimisation algorithm on a one-dimensional noisy function using a Gaussian process as a surrogate model and sampling heuristics that later evolved into upper confidence bound (Section 2.4.2) and probability of improvement (Section 2.4.1). Research from 1971 onwards linked Bayesian optimisation to the optimisation of expensive multi-modal functions and introduced new heuristics—notably expected improvement [168] (Section 2.4.1) and knowledge gradient [133, 134] (Section 2.4.3). Limitations in computational capabilities at the time resulted in the use of cheaper models, such as Wiener processes and Ornstein-Uhlenbeck processes, for which both heuristics collapse onto each other [58]. The evolution of technologies allowed computationally expensive models such as Gaussian processes to be used in the late 1990s, which incentivised reevaluation of the methods proposed two decades earlier. Schonlau [171] and Jones, Schonlau, and Welch [94] introduced their efficient global optimisation (EGO) algorithm and studied expected improvement in a deterministic setting. Auer [5] and Srinivas et al. [184] reexamined the upper confidence bound and studied its theoretical regret, while Frazier and Powell [51] and Scott, Frazier, and Powell [173] studied the knowledge gradient for discrete and continuous input spaces, respectively. Villemonteix, Vazquez, and Walter [200] introduced an information-based strategy (Section 2.4.3) into the Bayesian optimisation framework that was later studied and further developed into entropy search [78], predictive entropy search [80] and max-value entropy search [204]. Closely related to Bayesian optimisation is the work on multi-armed bandits that investigates a similar problem but in a discrete parameter space [58, 12, 114].
Bayesian optimisation has been applied to many scientific fields, predominantly in the science, technology, engineering and mathematics (STEM) area. Applications include material science [54, 147] and drug discovery [142] in chemistry, physics [44] and biology [117]. In engineering it has been used for civil engineering [59, 70], electrical engineering [122], mechanical engineering [186] and robotics [23, 127]. Furthermore, Bayesian optimisation is often used in machine learning and artificial intelligence, e.g., to configure algorithms [87] and to tune hyperparameters [178]. Garnett [58] provides a thorough list of applications.

2.2.2 Introduction of method

Bayesian optimisation [134, 94, 20, 53, 58] is a sample-efficient global optimisation algorithm designed for the optimisation of expensive black-box functions introduced in Section 2.1. The strategy sequentially selects points, observes them by evaluating the objective function—i.e., by conducting an experiment or running a simulator—and uses the new data point to perform subsequent selections of points. During this selection process, the aim is to extract as much information as possible from each data point and keep

the overall number of function evaluations as small as possible. This ensures that the global optimum is found in a minimum number of function evaluations, keeping costs low and making optimisation feasible.

Algorithm 1 illustrates the Bayesian optimisation algorithm. The algorithm runs in an optimisation loop where each optimisation step consists of three substeps. First, a surrogate model $\mathcal{M}$ is fitted to the training data. The training data is the set of all observed data points from all previous optimisation steps. While a Gaussian process (introduced in Section 2.3) is typically chosen as the surrogate model, other models are possible. However, the model must be capable of returning a prediction and its associated uncertainty for the entire input space. Second, an acquisition criterion α—a heuristic that guides the sequential optimisation process as explained in more detail in Section 2.4—is computed. This criterion bases its computation on the prediction from the surrogate model and corresponding uncertainty and, when maximised, suggests the next candidate point to be observed from the objective function in step three.

Algorithm 1 Standard Bayesian optimisation algorithm

Require: Evaluation budget N, number of initial points n_0, surrogate model $\mathcal{M}$, acquisition function α.
Sample n_0 initial data points $\boldsymbol{X}_0$ via a space-filling design and gather observations $\boldsymbol{y}_0$.
Set $n = 0$.
while $n < N - n_0$ **do**
  Fit surrogate model $\mathcal{M}$ to training data $\mathcal{D}_n = (\boldsymbol{X}_n, \boldsymbol{y}_n)$.
  Find $\boldsymbol{x}_{n+1}$ that maximises an acquisition criterion α based on model $\mathcal{M}$, i.e., solve $\operatorname{argmax}_{\boldsymbol{x}} \alpha(\boldsymbol{x})$.
  Evaluate $\boldsymbol{x}_{n+1}$, observing y_{n+1}
  Increment n.
end while
return Input vector $\boldsymbol{x}^*$ with the highest corresponding observation y^*.

Before performing the first step of the Bayesian optimisation algorithm, a small number of observed input-output pairs $(\boldsymbol{X}_0, \boldsymbol{y}_0)$ are required to initialise the algorithm. These initial data points ensure that the surrogate model can model the objective function from the first iteration, and the acquisition criterion can guide the selection process effectively. The initial data points can be found by randomly sampling from a uniform distribution or using a space-filling design such as Latin hypercube sampling [129, 86]. Consider a two-dimensional input space. Latin hypercube sampling divides the space into n_0 columns and rows with equal size. It then randomly allocates n_0 data points so that each row and column contains exactly one single point. From the $n_0 \times n_0$ squares that the hypercube created, only n_0 are populated with a data point at which to observe the objective function. Within the squares, the points are positioned randomly. Various methods exist for finding Latin hypercube samples that fill the space effectively. For example, maximin

Latin hypercubes try to select points by maximising the minimal distance between all points. Section 4.2.3 discusses a simple implementation of Latin hypercube sampling.
A few strategies for stopping the Bayesian optimisation algorithm are used. The simplest and typically used strategy is running Bayesian optimisation until a predetermined evaluation budget is exhausted [58]. This is displayed in Algorithm 1 where Bayesian optimisation is run until the objective function is evaluated N times. As the focus lies on expensive objective functions, such as physical experiments and computer simulators, the budget is typically set to a total number of function evaluations in the low hundreds or less. Other strategies are stopping the algorithm once a specified target value of the output is achieved or terminating Bayesian optimisation if the algorithm is not able to find a candidate that improves upon the current best for a certain number of optimisation steps [1]. The final stopping rule links termination to values of the acquisition function. For example, improvement-based policies (discussed in Section 2.4.1) quantify how likely a given point is to improve upon the best candidate point to date. Thus, a natural stopping criterion is to terminate once the likelihood of improving falls below a predefined threshold [94, 171].
Algorithm 1 is illustrated in Figure 2.1 by showing an optimisation loop consisting of eight optimisation steps, or iterations, on a one-dimensional function with one local and one global maximum. Three data points are selected randomly and are evaluated by the objective function to form the initial training data for iteration one. A Gaussian process is fitted to the three observations, resulting in a prediction and uncertainty quantification for the entire input range. These are then used in the computation of the expected improvement acquisition criterion. Maximisation of the criterion guides the selection of the next candidate point. The new data point is then observed from the objective function and added to the training data for the second optimisation step, which now consists of four training points. While Bayesian optimisation first exploits its knowledge about the highest prediction given current knowledge and focuses on the local maximum in iterations one to four, it explores the remaining input space, driven by the high uncertainty, from iteration five onwards and discovers the global maximum. This exploitation-exploration trade-off is a fundamental property of the acquisition criterion. The trade-off specifies that there has to be a balance between searching in areas with high predictions (exploitation) and in areas with high uncertainty (exploration) to search the input space effectively [58, 53, 96].
Bayes' theorem allows incorporating and updating prior beliefs into the Bayesian optimisation algorithm [20]. Particularly, a prior—a distribution encoding our belief about the objective function—is assumed and updated each time new evidence is collected. The current state of knowledge about the objective function is incorporated via the surrogate model. Depending on the selection of the model and the incorporated prior beliefs, certain objective functions are excluded from the set of possible objective functions. For example,

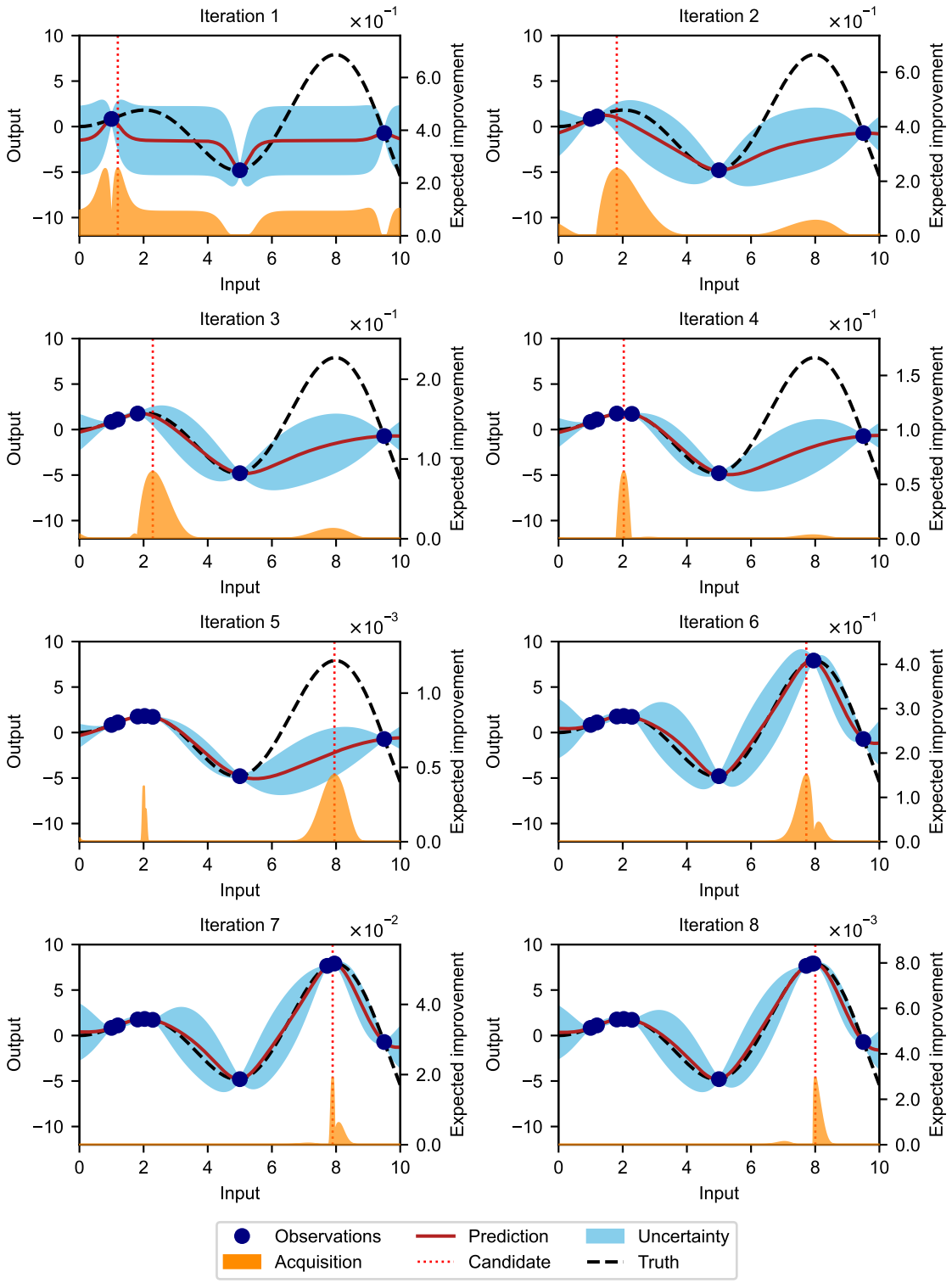

Figure 2.1: Bayesian optimisation applied to a 1-dimensional function with one local and one global maximum. Expected improvement is used as the acquisition function. The input space is bounded by $[0, 10]$.

if a model is chosen that only allows a smooth representation of the objective function, then erratic functions are not possible under that model and are thus disregarded.
While some methods aim to extend Bayesian optimisation into high-dimensional input spaces (Section 2.5.4), the standard Bayesian optimisation algorithm introduced in this section best suits problems with low to moderate dimensionality. Frazier [53] limits Bayesian optimisation to less than 20 input dimensions. Furthermore, the focus of Bayesian optimisation lies in the optimisation of continuous objective functions, as outlined in Section 2.1. However, this does not mean optimisation with discrete or categorical inputs is impossible. For example, Garrido-Merchán and Hernández-Lobato [60] and Daulton et al. [36] present possible solutions and Section 4.2.2 discusses a mixed optimisation strategy for small numbers of discrete variables.

2.3 Surrogate modelling with Gaussian processes

Surrogate modelling is an essential part of Bayesian optimisation and enables sample-efficient optimisation, which is at the heart of expensive black-box optimisation. A surrogate model is a statistical model that uses available data to represent the objective function and informs the sequential selection of candidate points. Based on the available data, the surrogate model's prediction can be understood as the best estimate of the objective function's shape.
Bayesian optimisation focuses on using Bayesian models that allow the incorporation of prior knowledge about the objective function into the model, restricting the possible set of represented functions from all possible functions to a smaller subset of functions that share properties such as smoothness [20, 53]. Another property of models used in Bayesian optimisation is that they return a prediction for the entire input space along with corresponding uncertainty quantification. This uncertainty quantifies the variance around a certain prediction. For deterministic objective functions, there is expected to be almost zero variance close to observed data points. At the same time, there will be larger variance for areas where no nearby point has been observed to date [58, 69].
Different surrogate models were proposed and studied in the literature for Bayesian optimisation. Hutter, Hoos, and Leyton-Brown [87] investigate the use of random forests [18, 34] and offer an implementation within the SMAC Python package [118]. Additionally, density ratio estimation [11, 177, 196] and neural networks [179, 183, 125] were considered. However, the limited number of available data points in expensive black-box optimisation restricts the complexity of the latter and prevents deep neural networks [58]. While these examples show that different surrogate models are viable for Bayesian optimisation, Gaussian process regression [162] has seen the widest adaption and has been researched extensively since the 1970s [20]. It is a flexible modelling strategy that is capable of rep-

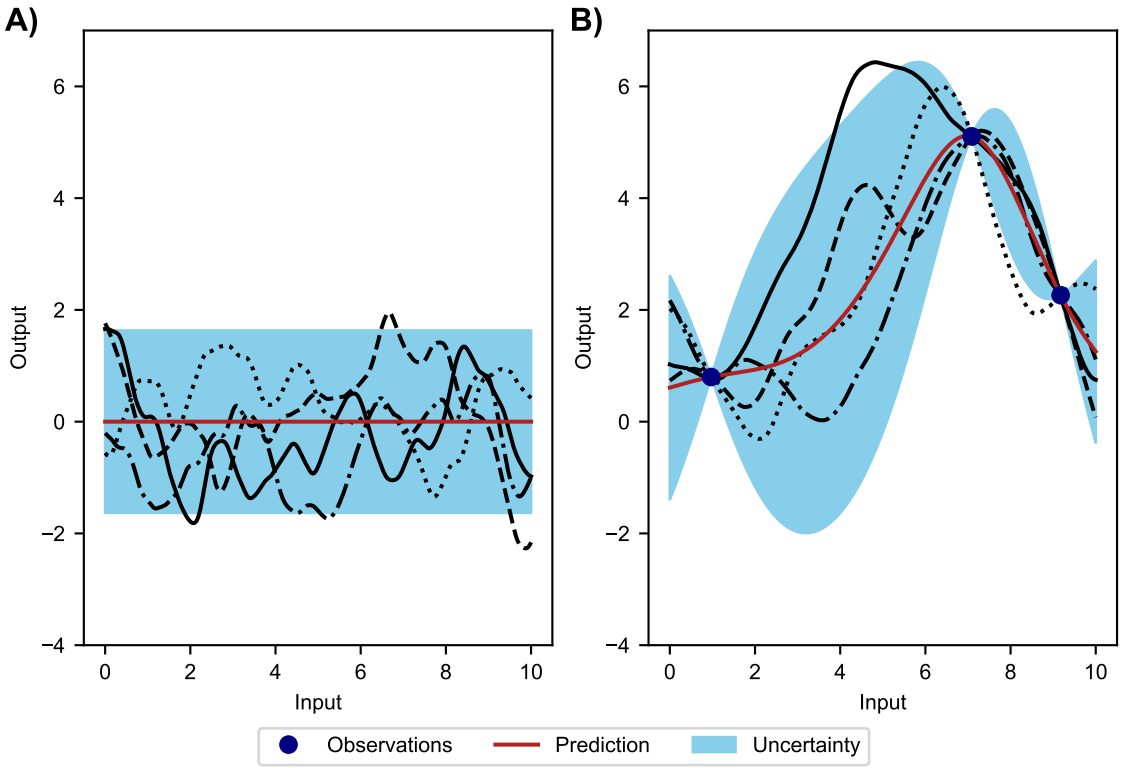

Figure 2.2: Samples from **A)** the prior distribution and **B)** the posterior predictive distribution of a Gaussian process with a zero mean function and a Matérn 5/2 covariance kernel.

resenting a wide range of possible objective functions and offers a straightforward way of incorporating prior beliefs about the objective function [20, 53, 69]. Therefore, this section and this thesis focus on using Bayesian optimisation based on Gaussian process regression. Alternative surrogate models are left for future work. Rasmussen and Williams [162] provide a comprehensive overview of Gaussian process regression and its implementation.

2.3.1 Gaussian process regression

A Gaussian process $\mathcal{GP}$ [162, 58] is a flexible non-parametric regression model that is based on the Gaussian distribution. By non-parametric we do not mean that Gaussian processes have no parameters but that the objective function has no explicit parametric form. Each point of the input space is represented by a (multivariate) Gaussian distribution of the objective function output—that is, infinitely many Gaussian distributions are arranged one after another. Thus, a Gaussian process can be considered as a distribution over all possible realisations of the objective function. Figure 2.2 shows the prior and the posterior distribution of a Gaussian process, which the remainder of this section discusses. It also gives four realisations, or samples, from the Gaussian process, illustrating the different forms the objective function can take. For each input value, the corresponding Gaussian

distribution $\mathcal{N}\left(\mu, \sigma^2\right)$ is represented by the prediction and the 95% prediction interval, calculated as $\mu \pm 1.96 \times \sigma^2$. Samples of the prior distribution are primarily random, while the three observations taken into account in the posterior predictive distribution give the realisations a clear structure. In this deterministic case, all realisations interpolate between the observations and change randomly between them according to the specific Gaussian distributions. Additionally, Figure 2.2 shows that the resulting realisations of a Gaussian process are smooth as values along the input axis are not independent of each other.

Prior distribution

Mathematically, a Gaussian process is a finite collection of random variables with a joint Gaussian distribution and is defined by specifying a prior mean function $\mu_0(\cdot)$ and a positive semidefinite covariance kernel $\Sigma_0(\cdot,\cdot)$ [162, 58]

$$\mathcal{GP}\left(\mu_0(\cdot), \Sigma_0(\cdot,\cdot)\right). \tag{2.8}$$

The mean function takes an input vector—or data point—$\boldsymbol{x}$ from the input space $\mathcal{X}$ and returns the mean as a scalar output such that $\mu_0(\boldsymbol{x}) : \mathcal{X} \mapsto \mathbb{R}$, while the covariance kernel takes two input vectors $\boldsymbol{x}$ and $\boldsymbol{x}'$ and returns the covariance as a scalar output such that $\Sigma_0(\boldsymbol{x}, \boldsymbol{x}') : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$. This results in a prior distribution where function values $f(\boldsymbol{X}_n)$ are modelled by a multivariate Gaussian distribution

$$f(\boldsymbol{X}_n) \sim \mathcal{N}\left(m(\boldsymbol{X}_n), K(\boldsymbol{X}_n, \boldsymbol{X}_n)\right) \tag{2.9}$$

with mean vector

$$m(\boldsymbol{X}_n) = \begin{bmatrix} \mu_0(\boldsymbol{x}_1) \\ \mu_0(\boldsymbol{x}_2) \\ \dots \\ \mu_0(\boldsymbol{x}_n) \end{bmatrix} \tag{2.10}$$

and covariance matrix

$$K\left(\boldsymbol{X}, \boldsymbol{X}'\right) = \begin{bmatrix} \Sigma_0(\boldsymbol{x}_1, \boldsymbol{x}'_1) & \Sigma_0(\boldsymbol{x}_1, \boldsymbol{x}'_2) & \dots & \Sigma_0(\boldsymbol{x}_1, \boldsymbol{x}'_n) \\ \Sigma_0(\boldsymbol{x}_2, \boldsymbol{x}'_1) & \Sigma_0(\boldsymbol{x}_2, \boldsymbol{x}'_2) & \dots & \Sigma_0(\boldsymbol{x}_2, \boldsymbol{x}'_n) \\ \vdots & \vdots & \ddots & \dots \\ \Sigma_0(\boldsymbol{x}_n, \boldsymbol{x}'_1) & \Sigma_0(\boldsymbol{x}_n, \boldsymbol{x}'_2) & \dots & \Sigma_0(\boldsymbol{x}_n, \boldsymbol{x}'_n) \end{bmatrix}. \tag{2.11}$$

Prior mean function

The choice of the specific prior mean function, and particularly the prior covariance kernel, allows the incorporation of prior beliefs about the objective function [20, 58]. Popular prior

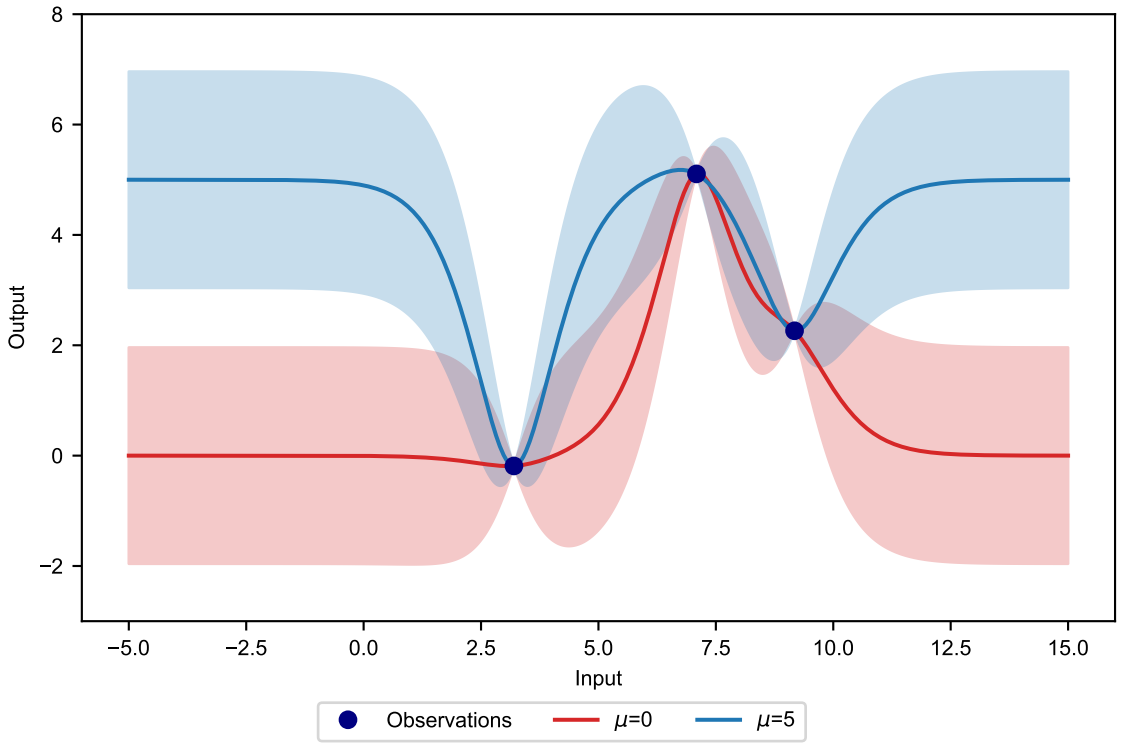

Figure 2.3: Comparison of Gaussian processes with zero mean function and constant mean function ($c = 5$). Predictions are similar when interpolating and default to the mean function when extrapolating.

mean functions are the zero mean function

$$\mu_{\text{zero}}(\boldsymbol{x}) = 0 \tag{2.12}$$

and the constant mean function

$$\mu_{\text{constant}}(\boldsymbol{x}) = c. \tag{2.13}$$

These simple prior mean functions are favoured in Bayesian optimisation, and the focus lies on the selection of prior covariance kernels because they affect the shape of the Gaussian process more significantly [58]. Indeed, subsequent sections show that the mean function only affects the model's prediction, while the covariance function affects the prediction and its uncertainty. However, the choice of mean function is essential when only a few data points are available. Figure 2.3 shows two Gaussian processes that only differ in terms of the mean function. One Gaussian process uses a zero mean function (red), and the other uses a constant mean function with $c = 5$ (blue). While the models are similar in areas between two data points, they differ significantly towards the outer bounds. In

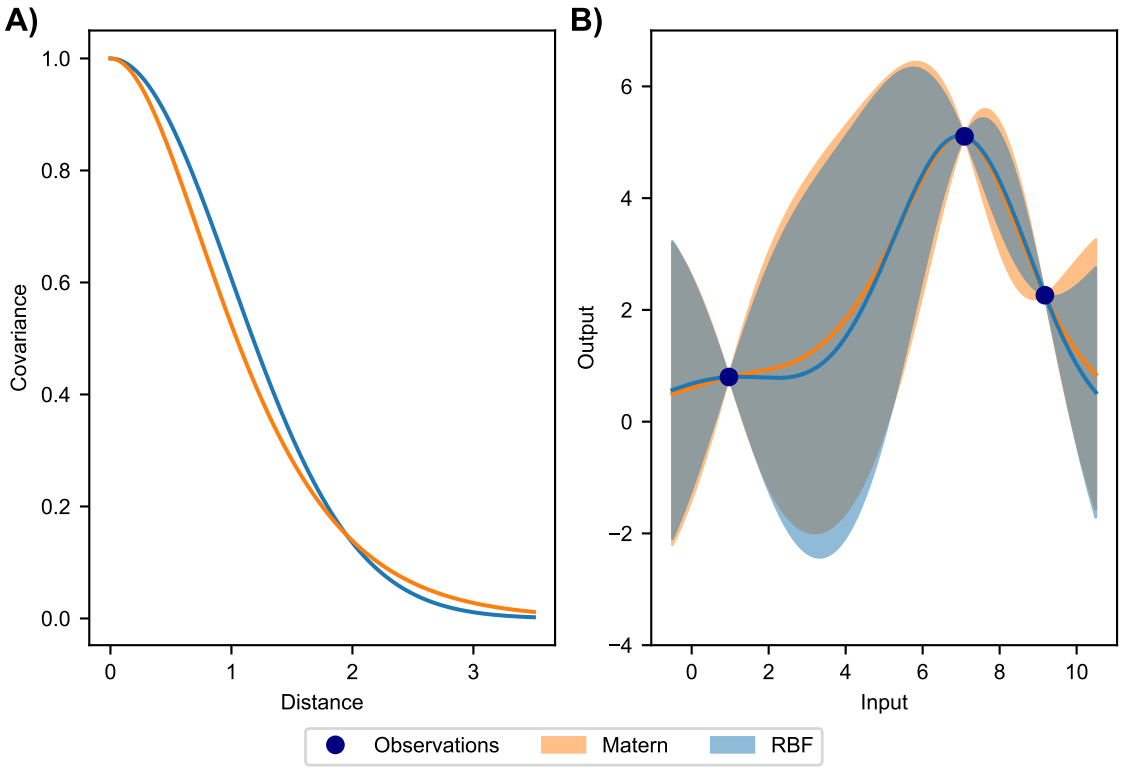

Figure 2.4: Comparison of the RBF and the Matérn 5/2 covariance kernel in Gaussian process regression. **A)** The larger the distance between two data points, the smaller the covariance. **B)** Gaussian processes with zero mean function fitted to three data points.

these areas, there are too few data points for the covariance kernel to affect the prediction, and the model defaults to the prior mean function. Thus, the choice of mean function is less crucial when interpolating than when extrapolating. Despite the popularity of simple prior mean functions, more complex functions, such as low-order polynomials [53], radial basis functions [19] and concave quadratic functions [58], are also used.

Prior covariance kernel

The prior covariance kernel specifies the correlation between any pair of data points, which controls the properties of the Gaussian process, such as its smoothness. In general, points that lie closer together will be correlated more strongly than points that are far apart [131, 20]. The radial basis function (RBF) or squared exponential kernel

$$\Sigma_{\text{RBF}}\left(\boldsymbol{x}, \boldsymbol{x}'\right) = \sigma_f^2 \exp\left(-\frac{r^2}{2l^2}\right), \tag{2.14}$$

where $r = |\boldsymbol{x} - \boldsymbol{x}'|$ is the distance between the two points, l is the characteristic length scale and σ_f^2 is the signal variance or output scale, is a popular covariance kernel used

in the literature. The length scale indicates the persistence of correlations of the output along the input axes, with larger length scales indicating a longer-lasting correlation. The output scale is a scalar that can adjust the magnitude of the variance, with larger values resulting in a larger deviation from the mean [162, 20, 69]. While the RBF kernel is popular, there is criticism that it is a naive choice [20] and unrealistically smooth [178] for Bayesian optimisation. Often, the Matérn kernel with parameter $\nu = \frac{5}{2}$

$$\Sigma_{\text{Matérn}}\left(\boldsymbol{x}, \boldsymbol{x}'\right) = \sigma_f^2 \left(1 + \frac{\sqrt{5}r}{l} + \frac{5r^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}r}{l}\right) \tag{2.15}$$

is preferred as it is twice differentiable but not as smooth as the RBF kernel [162, 178] and thus a practical choice for Bayesian optimisation. However, the difference between kernels can be relatively small, as Figure 2.4 shows. In **A)**, the correlations of the RBF and the Matérn 5/2 kernel for different distances between points are compared. A length scale $l = 1$ is assumed for both kernels. The correlation for the Matérn kernel is lower than that of the RBF kernel for points with a distance of less than 2. After that, the correlation is slightly larger. A small difference in prediction and uncertainty is noticeable after two zero-mean Gaussian processes using the RBF and the Matérn 5/2 kernel are fitted to three observations (**B)**). While these are perhaps the most frequently used covariance kernels, Rasmussen and Williams [162] and Garnett [58] discuss many other choices.

Covariance functions can be extended to include one characteristic length scale l_d for each input dimension d. In this case, outputs are correlated for longer distances along input dimensions with large length scales and, thus, less relevant for changes in the prediction as varying their values affects the prediction little. Along input dimensions with small length scales outputs are correlated for shorter distances, and even small changes in the input values can affect the prediction significantly. Gaussian processes with covariance functions that include multiple length scales are characterised by automatic relevance determination (ARD) of the input dimensions [141]. Here, the inverse of the length scales can be interpreted as the relevance of the corresponding dimensions [162]. The Gaussian process will estimate large length scales for irrelevant dimensions, automatically assigning them less importance.

The RBF and the Matérn kernel—and many of the kernels from Rasmussen and Williams [162]—are stationary kernels. Thus, the correlation fully depends on the distance r between two data points. Some real-world objective functions, however, are possibly non-stationary, requiring more observations to achieve accurate predictions. As data points are expensive to observe, using a non-stationary kernel can be beneficial [180, 162, 175].

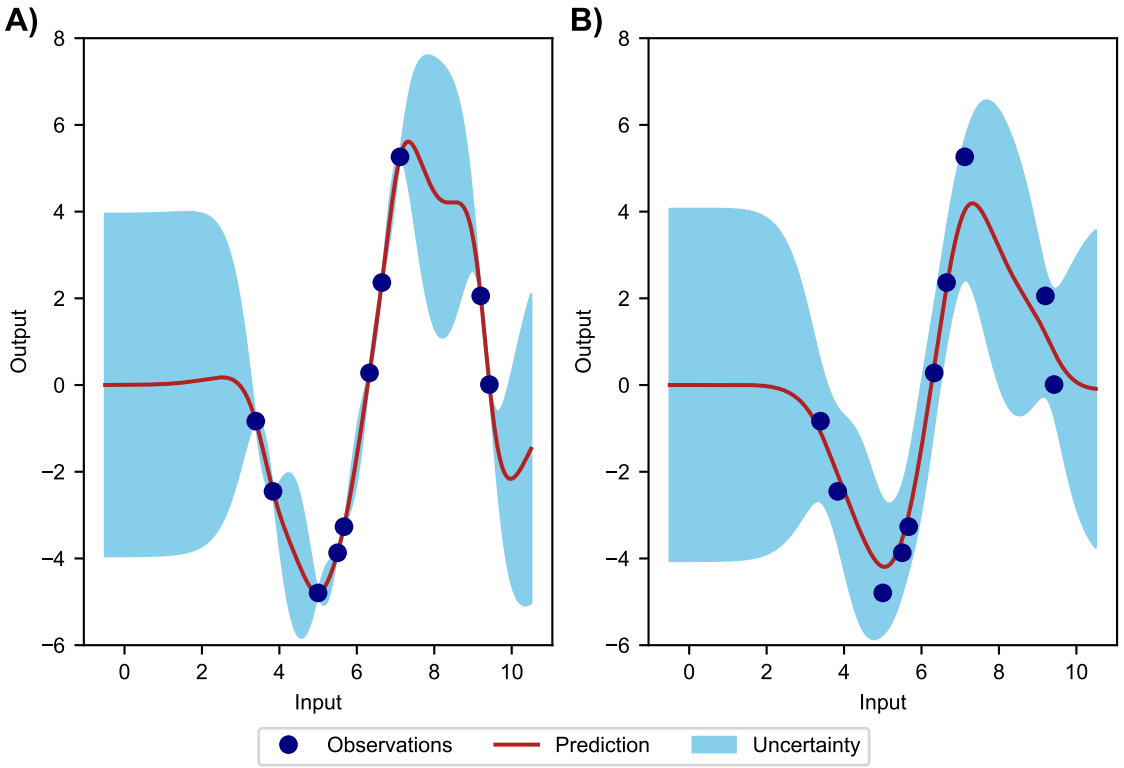

Figure 2.5: Influence of the nugget in Gaussian process regression. **A)** Without the nugget, the posterior variance of the Gaussian process is zero at all observations, and the posterior mean interpolates through all observations. **B)** With the nugget, the posterior variance of the Gaussian process is no longer zero at the observations, and the posterior mean does not necessarily interpolate through the observations.

Posterior predictive distribution

After observing some data points, the prior distribution can be updated with Bayes' rule [162] to ascertain the conditional Gaussian distribution of the surrogate model given the data $\mathcal{D}_n$ and test points $\boldsymbol{X}_*$.

$$f\left(\boldsymbol{X}_*\right) \mid \mathcal{D}_n, \boldsymbol{X}_* \sim \mathcal{N}\left(\mu_n\left(\boldsymbol{X}_*\right), \sigma_n^2\left(\boldsymbol{X}_*\right)\right). \tag{2.16}$$

This conditional distribution is called the posterior predictive distribution. It can be computed in closed form with posterior mean

$$\mu_n\left(\boldsymbol{X}_*\right) = K\left(\boldsymbol{X}_*, \boldsymbol{X}_n\right)\left[K\left(\boldsymbol{X}_n, \boldsymbol{X}_n\right) + \sigma_k^2 I\right]^{-1}\left(\boldsymbol{y} - m\left(\boldsymbol{X}_n\right)\right) + m\left(\boldsymbol{X}_*\right) \tag{2.17}$$

and posterior variance

$$\sigma_n^2\left(\boldsymbol{X}_*\right) = K\left(\boldsymbol{X}_*, \boldsymbol{X}_*\right) - K\left(\boldsymbol{X}_*, \boldsymbol{X}_n\right)\left[K\left(\boldsymbol{X}_n, \boldsymbol{X}_n\right) + \sigma_k^2 I\right]^{-1} K\left(\boldsymbol{X}_n, \boldsymbol{X}_*\right), \tag{2.18}$$

where σ_k^2 is a nugget term or noise variance reflecting, for example, the stochastic noise added to the objective function ϵ in Equation 2.3. Figure 2.5 shows that the posterior mean of the Gaussian process no longer interpolates between observations and is more flexible. Gramacy and Lee [72] make a comprehensive case for adding a nugget. In their work, nuggets improve the algorithm's computational stability and performance, such as the coverage and goodness-of-fit. They can also yield improved results in situations where statistical assumptions—such as stationarity of the process—are violated. Furthermore, Gramacy and Lee [72] reason why nuggets should be added to deterministic problems besides these technical advantages. For example, while there might not be a measurement error in a deterministic simulation, the simulation itself only approximates reality. Using models that take these biases into account can improve performance in these cases.
As discussed in the previous sections, Equations 2.17 and 2.18 show that the covariance kernel affects the posterior mean and the posterior variance, while the mean function only affects the posterior mean. In regions where data points are too far apart to be considered correlated by the covariance kernel, the prediction defaults towards the mean function [58]. This can be seen in Figure 2.3 and Figure 2.5 where the prediction converges towards the zero mean function in the interval $[0, 3]$ for the input. This stresses the potential dangers of extrapolating with Gaussian processes.

Model selection

Model selection describes the process of deciding which specific model to choose for the given data. Even when the prior mean function and prior covariance kernel are specified, a wide range of models—and thus representations of objective functions—is possible depending on the values assigned to the Gaussian process parameters [58]. The number of parameters of a Gaussian process varies according to the specified prior mean function and prior covariance kernel. Typically, they consist of at least one length scale l, the output scale σ_f^2, the nugget σ_k^2 and possibly some parameter in the mean function, such as the constant mean value c. Hence, the aim is to find parameter values that specify a model that fits well to the observed data.
In Bayesian analysis, prior distributions (probability density functions) are chosen for the parameters $\boldsymbol{\theta}$ that represent beliefs about the parameter values before observing any data. After observing data $\mathcal{D}_n$, a Gaussian process model is selected, and parameters $\boldsymbol{\theta}$ are estimated. One way to achieve this is maximising the model's posterior density function such that

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p\left(\boldsymbol{\theta} \mid \mathcal{D}_n\right) \propto p\left(\boldsymbol{\theta}\right) p\left(\boldsymbol{y}_n \mid \boldsymbol{X}_n, \boldsymbol{\theta}\right), \tag{2.19}$$

where $p\left(\boldsymbol{y}_n \mid \boldsymbol{X}_n, \boldsymbol{\theta}\right)$ is the marginal likelihood of the data (also referred to as the evidence) and $p\left(\boldsymbol{\theta}\right)$ is the prior distribution of the parameters. While arbitrary distributions can

be chosen to reflect knowledge about $\boldsymbol{\theta}$, the evidence $p(\boldsymbol{y}_n \mid \boldsymbol{X}_n, \boldsymbol{\theta})$ can conveniently be expressed as the multivariate Gaussian distribution modelling the function evaluations $\mathcal{N}(m(\boldsymbol{X}_n), K(\boldsymbol{X}_n, \boldsymbol{X}_n))$ given in Equation 2.9. Typically, the logarithm of the posterior $\log(p(\boldsymbol{\theta})) + \log(p(\boldsymbol{y}_n \mid \boldsymbol{X}_n, \boldsymbol{\theta}))$ is maximised as the values of the posterior can be close to zero and challenging to optimise. This estimation strategy is known as maximum a posteriori (MAP) estimation [58].
If no knowledge about the shape of the objective function is known a priori—and thus the model selection process cannot be aided by declaring an informative prior distribution for $p(\boldsymbol{\theta})$—, a uniform distribution can be chosen as an uninformative prior. This is convenient as the marginal likelihood can be expressed in closed form and maximised with gradient methods. In practice, uniformative priors are often assumed implicitly and the log marginal likelihood

$$\log p(\boldsymbol{y}_n \mid \boldsymbol{X}_n) = -\frac{1}{2}(\boldsymbol{y}_n - m(\boldsymbol{X}_n))^T \left[K(\boldsymbol{X}_n, \boldsymbol{X}_n) + \sigma_k^2 I\right]^{-1} (\boldsymbol{y}_n - m(\boldsymbol{X}_n)) - \frac{1}{2}\log|K(\boldsymbol{X}_n, \boldsymbol{X}_n) + \sigma_k^2 I| - \frac{n}{2}\log 2\pi \quad (2.20)$$

is maximised. Values that maximise this likelihood are then assigned to the parameters of the Gaussian process. This model selection strategy is also known as maximum likelihood estimation (MLE) [162] and is a special case of MAP estimation in this context [58].

2.4 Guiding optimisation with acquisition functions

Acquisition functions are heuristics computed to guide the optimisation over the parameter space. In the sequential optimisation process, they determine what point (or points) should be evaluated and observed next from the objective function. The exploitation-exploration trade-off, mentioned in Section 2.2, is an essential property of acquisition functions for Bayesian optimisation [96]. This property ensures a sample-efficient optimisation procedure by taking into account the prediction and the uncertainty of the posterior predictive distribution of the Gaussian process model. Thus, Bayesian optimisation explores areas with high uncertainty but also exploits areas with high predicted means [53, 58]. Maximisation of acquisition functions can present a challenging problem itself, particularly in higher dimensions. In the literature, multi-start gradient-based optimisers such as L-BFGS-B [119, 222, 69] or global optimisers such as the DIRECT optimisation algorithm [93, 92, 20] achieve good results. This section presents a selection of acquisition functions that have received the most attention in the literature. The focus lies mainly on acquisition functions used in subsequent chapters of this thesis. Garnett [58] provides an in-depth derivation of these acquisition functions.

2.4.1 Improvement-based acquisition functions

Improvement-based approaches are perhaps the best-studied acquisition functions available from the literature. They select points with the most significant potential to improve on a certain specified target. In Bayesian optimisation, this target is usually the best output y^{best} of all available data points to date. The probability of improvement (PI) was introduced in the 1960s by Kushner [111, 110]. It is defined by

$$\alpha_{PI}\left(\boldsymbol{X}_*\right) = \Phi\left(\frac{\mu_n\left(\boldsymbol{X}_*\right) - y^{best}}{\sigma_n\left(\boldsymbol{X}_*\right)}\right), \tag{2.21}$$

where $\mu_n\left(\cdot\right)$ and $\sigma_n\left(\cdot\right)$ are the mean and the variance of the Gaussian process posterior predictive distribution and $\Phi\left(\cdot\right)$ is the cumulative distribution function of the standard Gaussian distribution $\mathcal{N}\left(0, 1\right)$.

While PI has a tendency to be overly greedy [91, 175, 20], that is it focuses too much on exploitation at the cost of exploration, expected improvement (EI) strikes a better balance. EI [168, 94, 171] is defined as

$$\alpha_{EI}\left(\boldsymbol{X}_*\right) = \left(\mu_n\left(\boldsymbol{X}_*\right) - y^{best}\right)\Phi\left(z\right) + \sigma_n\left(\boldsymbol{X}_*\right)\phi\left(z\right), \tag{2.22}$$

where $z = \frac{\mu_n(\boldsymbol{X}_*) - y^{best}}{\sigma_n(\boldsymbol{X}_*)}$ and $\phi\left(\cdot\right)$ is the probability density function of the standard Gaussian distribution $\mathcal{N}\left(0, 1\right)$.

Jones, Schonlau, and Welch [94] and Jones [91] list the lack of a hyperparameter that requires tuning and the possibility of a natural stopping rule for algorithms using EI—that is stopping once the expected improvement falls below a threshold—as the strengths of EI. However, they criticise the fact that EI (as well as PI) takes the estimated uncertainty of the Gaussian process as true, which might lead to over-exploration. Bull [21] studies the convergence rate of EI and show that it can converge near-optimally.

Snoek, Larochelle, and Adams [178] investigate EI empirically and compare different flavours of EI, such as (i) using Gaussian processes for which parameters were estimated via maximum likelihood or a fully Bayesian treatment, (ii) parallel EI (see Section 2.5.3) and (iii) EI per second. The latter is a version of EI that aims to maximise the objective function and minimise the wall clock time of the optimisation simultaneously. Furthermore, they compare EI against UCB on a test function, showing better performance of EI.

Sometimes, a parameter ξ is added to PI and EI to control the trade-off between exploitation and exploration, where higher values result in more exploration. Here, ξ is subtracted from the numerator of z and multipe strategies for setting the parameter have been proposed, such as a variable approach over the iterations [110, 121] or keeping ξ constant [91].

Although expected improvement can never be non-positive mathematically, it can become zero numerically when computed due to the floating point precision of the programming language. This results in flat areas where the expected improvement is zero and cannot be optimised correctly. To prevent numerically vanishing values, log expected improvement (LogEI) was proposed by Ament et al. [2] as

$$\alpha_{\text{LogEI}}\left(\boldsymbol{X}_*\right) = \begin{cases} \log_h\left(\frac{\mu_n(\boldsymbol{X}_*)-y^{best}}{\sigma_n(\boldsymbol{X}_*)}\right) + \log\left(\sigma_n\left(\boldsymbol{X}_*\right)\right) & \text{if } \sigma_n\left(\boldsymbol{X}_*\right) > 0 \\ 0 & \text{if } \sigma_n\left(\boldsymbol{X}_*\right) = 0 \end{cases}, \tag{2.23}$$

where

$$\log_h\left(z\right) = \begin{cases} \log\left(\phi\left(z\right) + z\Phi\left(z\right)\right) & \text{if } z > -1 \\ -z^2/2 - c_1 + \texttt{log1mexp}\left(\log\left(\texttt{erfcx}\left(-z/\sqrt{2}\right)|z|\right) + c_2\right) & \text{if } -1/\sqrt{\epsilon} < z \leq -1 \\ -z^2/2 - c_1 - 2\log\left(|z|\right) & \text{if } z \leq -1/\sqrt{\epsilon} \end{cases}, \tag{2.24}$$

where $c_1 = \log\left(2\pi\right)/2$, $c_2 = \log\left(\pi/2\right)/2$, ϵ is the numerical precision, and `log1mexp` and `erfcx` are stable implementations of $\log\left(1-\exp\left(z\right)\right)$ and $\exp\left(z^2\right)\operatorname{erfc}\left(z\right)$ respectively, where erfc is the complementary error function.

2.4.2 Confidence bound-based acquisition functions

Another popular acquisition function is the upper confidence bound (UCB) [5, 184], which traces back to the work of Lai and Robbins [112]. It is also referred to as an optimistic strategy as it assumes a predefined level of the posterior variance of the Gaussian process to be true [175]. It is defined as

$$\alpha_{UCB}\left(\boldsymbol{X}_*\right) = \mu_n\left(\boldsymbol{X}_*\right) + \sqrt{\beta}\sigma_n\left(\boldsymbol{X}_*\right), \tag{2.25}$$

where β is the trade-off parameter that balances exploration and exploitation. Common strategies for determining a value for β are fixing it to a single value or varying the value as in the GP-UCB algorithm [185]. Chapter 3 studies this parameter empirically. Furthermore, Srinivas et al. [185] and De Freitas, Smola, and Zoghi [37] theoretically investigate the regret and convergence rate of upper confidence bound for the stochastic (Equation 2.3) and the deterministic problem (Equation 2.1), respectively. The latter presents a branch and bound approach that sequentially restricts the input space once a certain coverage of data points is achieved. This approach speeds up optimisation for deterministic problems significantly, however, the authors acknowledge limitations in extending it to the noisy problem.

2.4.3 Information-based acquisition functions

A large group of information-based policies consider the entropy of the posterior predictive distribution $p(\boldsymbol{x} \mid \mathcal{D}_n)$ and aim to find a candidate point $\boldsymbol{x}$ that maximises its reduction. While entropy search (ES) [200, 78] and predictive entropy search (PES) [79] are computationally expensive, max-value entropy search (MES) [204] uses information about simple to compute maximal output values $p(y_* \mid \mathcal{D}_n)$ instead of costly to compute entropies. In detail, the aim is to maximise the mutual information gain between the maximum output y_* and the next candidate point. This gain can be approximated with Monte Carlo estimation such that

$$\alpha_{MES}(\boldsymbol{X}_*) \approx \frac{1}{K} \sum_{y_* \in Y_*} \left[\frac{\gamma_{y_*}(\boldsymbol{X}_*)\, \phi(\gamma_{y_*}(\boldsymbol{X}_*))}{2\Phi(\gamma_{y_*}(\boldsymbol{X}_*))} - \log\left(\Phi(\gamma_{y_*}(\boldsymbol{X}_*))\right) \right], \tag{2.26}$$

where $\gamma_{y_*}(\boldsymbol{x}) = \frac{y_* - \mu_n(\boldsymbol{x})}{\sigma_n(\boldsymbol{x})}$ and K is the number of Monte Carlo samples. Wang and Jegelka [204] give two strategies to approximate y_*—an approximation using a Gumbel distribution and sampling from the Gaussian process posterior. The performance of MES is studied empirically in Chapter 3. Other information-based strategies include Thompson sampling [195, 176] and knowledge gradient [52, 53].

2.4.4 Portfolios

Portfolios consider multiple acquisition functions at each step and choose the best-performing one. Prominent portfolios include the GP-Hedge algorithm [83] based on the work of Auer et al. [6] and the entropy search portfolio [176] combining information-based acquisition functions. The Hedge portfolio algorithm is presented in Algorithm 2 and is investigated in Chapter 3. It requires the computation and optimisation of all acquisition functions $\alpha \in A$ in the portfolio at each Bayesian optimisation iteration. At each optimisation step, the next candidate point x_{n+1} to evaluate from the objective function is selected randomly according to probabilities $p_{n+1}(\alpha)$ calculated from the performance of the corresponding acquisition function α. The Hedge portfolio uses rewards r^α and gains g^α of the acquisition functions to compute these probabilities. Rewards are defined as the surrogate model's prediction of the candidate point, i.e., $r_n^\alpha = \mu_n(\boldsymbol{x}_n^\alpha)$, and gains are the sum of all rewards to date $g_n^\alpha = \sum_{i=0}^n r_n^\alpha$. The probability that candidate point $\boldsymbol{x}_n^\alpha$ is selected to be evaluated from the objective function is given as the ratio of the exponential gains multiplied by η and the sum of all acquisition functions' exponential gains multiplied by η. The hedge parameter η is also called the learning rate of the algorithm. For larger values, the gains will be exponentially larger and the differences in probabilities across the acquisition function will be more than proportionally larger. Thus, high-performing acquisition functions will be more than proportionally favoured. If η is smaller, the probabilities of the acquisition

functions will be more similar and they will have a more equal chance of being chosen. The probabilities associated with the acquisition functions influence the selection of which acquisition function is used at each iteration, and can be viewed as a meta-acquisition function, where the hedge parameter η is a trade-off parameter that balances exploitation (large values) and exploration (small values). Similar to Srinivas et al. [185] who propose a variable trade-off parameter for upper confidence bound, Hoffman, Brochu, De Freitas, et al. [83] propose setting $\eta = \sqrt{8 \ln k/t}$, where k is the number of acquisition functions in the portfolio and t is the number of the current iteration. This will favour exploitation in the early iterations and shift increasingly towards exploration. At the first optimisation step, all gains are zero and the selection of the candidate point $\boldsymbol{x}_1^\alpha$ is completely random. Optimising all acquisition functions at every optimisation step means that portfolios are computationally costly compared to single acquisition functions. However, the performance of different acquisition functions varies for each iteration. While one acquisition function could be optimal in one iteration, another acquisition function could be preferred in a different iteration. The Hedge algorithm aims to select the best acquisition function at each iteration to deliver a better solution than when just considering one individual acquisition function.

Algorithm 2 Hedge portfolio algorithm

Require: Evaluation budget N, number of initial points n_0, surrogate model $\mathcal{M}$, acquisition functions A, hedge parameter η.
  Sample n_0 initial training data points $\boldsymbol{X}_0$ via a space-filling design and gather observations $\boldsymbol{y}_0$.
  Set $n = 0$.
  Initialise gains $g_0^\alpha = 0$ for all α in A.
  Fit surrogate model $\mathcal{M}$ to training data $\mathcal{D}_n = (\boldsymbol{X}_n, \boldsymbol{y}_n)$.
  while $n < N - n_0$ **do**
    Find $\boldsymbol{x}_{n+1}^\alpha = \operatorname{argmax}_{\boldsymbol{x}} \alpha(\boldsymbol{x})$ for all α in A.
    Select point $\boldsymbol{x}_{n+1} = \boldsymbol{x}_{n+1}^\alpha$ with probability $p_{n+1}(\alpha) = \exp(\eta g_n^\alpha) \;/\; \sum_{i=1}^k \exp(\eta g_n^i)$.
    Evaluate $\boldsymbol{x}_{n+1}$, observing y_{n+1}.
    Fit surrogate model $\mathcal{M}$ to training data $\mathcal{D}_{n+1} = (\boldsymbol{X}_{n+1}, \boldsymbol{y}_{n+1})$.
    Get rewards $r_{n+1}^\alpha = \mu_{n+1}(\boldsymbol{x}_{n+1}^\alpha)$.
    Get gains $g_{n+1}^\alpha = g_n^\alpha + r_{n+1}^\alpha$.
    Increment n.
  end while

2.4.5 Monte Carlo acquisition functions

Some of the aforementioned analytical acquisition functions—such as expected improvement and upper confidence bound—can also be approximated by Monte Carlo sampling. This has the benefit of not requiring the explicit computations, which can become com-

putationally challenging, particularly when considering parallel multi-point approaches, as discussed in Section 2.5.3. The analytical acquisition functions can be reparameterised such that they are dependent on samples from a Gaussian distribution [212, 211]. For example, Equations 2.21, 2.22 and 2.25 become

$$\alpha_{PI}^{MC}\left(\boldsymbol{X}_*\right) = \max\left(\sigma\left(\frac{\mu_n\left(\boldsymbol{X}_*\right) + \boldsymbol{L}\boldsymbol{z} - y^{best}}{\tau}\right)\right) \tag{2.27}$$

$$\alpha_{EI}^{MC}\left(\boldsymbol{X}_*\right) = \max\left(ReLU\left(\mu_n\left(\boldsymbol{X}_*\right) + \boldsymbol{L}\boldsymbol{z} - y^{best}\right)\right) \tag{2.28}$$

$$\alpha_{UCB}^{MC}\left(\boldsymbol{X}_*\right) = \max\left(\mu_n\left(\boldsymbol{X}_*\right) + \sqrt{\frac{\beta\pi}{2}}|\boldsymbol{L}\boldsymbol{z}|\right) \tag{2.29}$$

where $\boldsymbol{z}$ are base samples from a standard multivariate Gaussian distribution such that $\boldsymbol{z} \sim \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{I}\right)$, where $\boldsymbol{0}$ is a vector of zeros and $\boldsymbol{I}$ an identity matrix of suitable size. Additionally, $\boldsymbol{L}$ is the lower triangular matrix of the Cholesky decomposition such that $\boldsymbol{L}\boldsymbol{L}^\intercal = K\left(\boldsymbol{X}_n, \boldsymbol{X}_n\right)$, $\sigma\left(\cdot\right)$ is the sigmoid nonlinearity, and $ReLU\left(\cdot\right)$ is the rectified linear unit function. These Monte Carlo acquisition functions are computed many times and then averaged to give an approximation of the analytical acquisition functions. Chapter 3 empirically studies these Monte Carlo acquisition functions.

2.5 Special cases

This section presents variants of Bayesian optimisation for specific problems such as noisy objective functions, parallel evaluations and high-dimensional optimisation. The aim is to give an overview of the different directions of research associated with Bayesian optimisation and to signpost important articles that will be considered further in the thesis beyond this background chapter.

2.5.1 Noise

Many objective functions—particularly physical experiments—are stochastic due to errors when measuring individual observations. The corresponding results or measurements will differ when observing the same data point multiple times. Section 2.1 showed how this can be represented as a maximisation problem by adding noise term ϵ sampled from a Gaussian distribution with zero mean and a small variance to the objective function $f\left(\cdot\right)$. As a first step, a nugget σ_k^2 is usually added to the Gaussian process to guarantee the model's flexibility and avoid interpolating through all observations, as discussed in Section 2.3. This nugget can either be homoskedastic and take the same single value over the full input domain or heteroskedastic and vary over the domain—for example, by modelling the noise through a second Gaussian process [66]. While adding a nugget suffices for many

Bayesian optimisation algorithms, such as ones using Thompson sampling, upper confidence bound, entropy search and knowledge gradient [53], others, such as improvement-based methods, are the focus of a considerable amount of research to enable the optimisation of stochastic objective functions [69, 155, 205, 9, 85, 88]. These are predominantly concerned with determining a good substitute or estimate of y^{best} used in Equation 2.22 as the target on which the algorithm aims to improve. With stochastic objective functions, this target is no longer deterministic and various methods have been proposed, such as the plug-in method that uses the maximum of the posterior mean of the Gaussian process as the target [155]. Gramacy [69] make a case for such a simple strategy by pointing out that the Bayesian optimisation literature seems to over-emphasise potential differences between deterministic and stochastic objective functions, while it is not greatly debated within the surrogate modelling literature.

2.5.2 Constraints

Constrained optimisation [3, 169] is another well-researched topic in Bayesian optimisation, where one or multiple constraints $g_i(\cdot)$ restrict the feasible input space $\mathcal{X}$ such that

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}\in\mathcal{X}}{\operatorname{argmax}}\, f(\boldsymbol{x}) \tag{2.30}$$

$$\text{subject to } g_i(\boldsymbol{x}) \geq 0, \qquad i = 1, \ldots, I. \tag{2.31}$$

While these problems can be solved by using an optimiser allowing input constraints such as the SLSQP algorithm [107] when constraint functions $g_i(\cdot)$ can be cheaply evaluated, they are much more challenging to manage when evaluations of $g_i(\cdot)$ are expensive. Expected improvement can be generalised to be used in the latter case by declaring the improvement to be zero if the inputs are not feasible [172, 57, 53]. In instances where the constraints are unknown, expected improvement can be extended to allow optimisation [71, 181, 62, 70]. Predictive entropy search has also been studied for unknown constraints [79].

2.5.3 Parallel and asynchronous evaluations

For problems where more than one candidate point can be evaluated simultaneously—particularly for simulators that can be run in parallel on different cores or computers—determining multiple candidate points at each optimisation step can speed up optimisation significantly. While the focus has been on single-step optimal acquisition functions in this case, it shifts to multi-step lookahead functions that account for the batch size of the simultaneous evaluations and select points accordingly for this specific problem.
Methods adapted to use the single-step acquisition functions for batched optimisation

include local penalisation [67] and sequential simulation [58], where points are selected in a loop until the batch is full. Different methods exist for upper confidence bound [38, 32, 109, 102], and expected improvement, such as the kriging believer and constant liar approaches [64], where previously selected points of a batch are added to the training data with outputs imputed as the minimum, mean or maximum of all outputs $\boldsymbol{y}$. The batch points are evaluated once the batch is full and imputed outputs are replaced with their actual observations.

While there are attempts to develop true multi-step lookahead acquisition functions for expected improvement [64, 29, 126], knowledge gradient [215], predictive entropy search [174] and Thompson sampling [81, 99], many suffer from challenging explicit computations. However, Monte Carlo versions of many single-step acquisition functions can be extended naturally to multi-point lookahead optimisation and avoid challenging mathematics [211]. Section 2.4.5 discussed Monte Carlo acquisition functions.

Due to the randomness in the Monte Carlo samples, these acquisition functions can only be optimised by stochastic optimisers, such as Adam [104]. However, there is some empirical evidence that fixing the base samples for individual Bayesian optimisation loops does not affect the performance negatively [7]. This method would allow deterministic optimisers, such as L-BFGS-B [222] and SLSQP [107], to be used but could potentially introduce bias due to sampling randomness.

Two optimisation strategies for multi-point batches are proposed in the literature [211]: The first is a joint optimisation approach, where the acquisition functions are optimised over all points of the batch simultaneously. The second option is a greedy sequential approach where one point after another is selected, holding all previous points fixed until the batch is full. Empirical evidence shows that both methods approximate the acquisition successfully. However, the greedy approach seems to have a slight edge over the joint strategy for some examples [211]. It is also faster to compute for larger batches.

Closely linked to parallel evaluations are asynchronous evaluations. Here, the problem consists of continuing optimisation, while some points have not yet been evaluated and are still pending. Conveniently, many of the methods mentioned above can also be used for asynchronous optimisation, such as sequential simulation and Monte Carlo acquisition functions [64, 178, 58].

2.5.4 High-dimensional optimisation

A major focus of Bayesian optimisation research is the extension into high-dimensional parameter space. Standard Bayesian optimisation is only reliable with a low to medium number of parameters. Frazier [53] defines the input space for problems that can be solved by Bayesian optimisation as no more than 19 dimensions. The main challenge with high-dimensional optimisation is the curse of dimensionality, which says that the parameter

space to explore grows exponentially with the number of parameters [158, 58]. There are roughly four approaches to extending Bayesian optimisation to high-dimensional space.
First, many methods assume a low effective dimensionality, which is considered a requirement for success in many areas [116]. While there are many inputs, only a small fraction actually influence the output. Determining the effective inputs is the main challenge of this approach, and many methods rely on random embeddings [207, 206, 13, 140, 115, 135, 39]. Raponi et al. [161] use principal component analysis to find a low-dimensional structure.
Second, other methods assume that the objective function can be decomposed into many simpler functions that together combine to form the overall objective function. This additive structure is leveraged—for example with additive Gaussian processes—in Kandasamy, Schneider, and Póczos [100], Gardner et al. [55], Rolland et al. [166], Mutny and Krause [138] and Hoang et al. [82].
The third approach aims to improve the Gaussian process model to handle the problem of high dimensionality. Oh, Gavves, and Welling [146] use a cylindrical covariance kernel to transform the input space and to speed up finding the optimum. Moreover, Eriksson and Jankowiak [47] and Liu et al. [120] try to locate sparse subspaces with a fully Bayesian approach that places priors on the parameters of the Gaussian process.
The final approach aims to restrict the search space by placing a trust region—essentially a hyperrectangle—around the best observation and performing Bayesian optimisation only within this trust region [48]. Depending on the results, the trust region is scaled to be smaller or larger, and multiple trust regions can be used simultaneously to ensure the global optimum is located.

2.5.5 Multiple objectives, fidelities and tasks

The maximisation problem introduced in Section 2.1 can be extended to multi-objective, multi-fidelity and multi-task optimisation.
In multi-objective optimisation, the aim is no longer to optimise one single objective function $f(\cdot)$ but rather multiple objective functions $f_i(\cdot)$ simultaneously. A global optimum for all objective functions is only possible if it happens to occur at the same inputs for all functions. The main goal is to find the best trade-off between the objective functions [58, 53]. Acquisition functions such as expected improvement [46, 157, 219, 218], predictive entropy search [79] and max-value entropy search [10, 49] have been extended to allow multi-objective optimisation. Furthermore, Paria, Kandasamy, and Póczos [150] propose the use of random scalarisations.
Multi-fidelity optimisation assumes that the objective function $f(\cdot)$ can be evaluated with different fidelities s such that $f(\boldsymbol{x}, s)$, where higher s yield lower fidelities and the original objective function from Equation 2.1 can be written as $f(\boldsymbol{x}, 0)$ [53]. It is often

assumed that higher fidelities yield more accurate results, making this a complex problem for which Bayesian optimisation has to determine the next candidate to observe and at what fidelity. While Forrester, Sóbester, and Keane [50] focus on updating the surrogate model to account for multiple fidelities, many authors focus on the acquisition functions such as expected improvement [182, 84, 155], upper confidence bound [97, 98], predictive entropy search [221, 130] and max-value entropy search [191].
Multi-task optimisation assumes that the objective function $f(\cdot)$ is dependent on different tasks or variables that are correlated [53, 58]. In an experiment, for example, the observation could be dependent and influenced by external environmental conditions that may or may not be controlled. The aim is to leverage the correlation between tasks and find one solution that performs well for all tasks [24] or multiple solutions, with each being optimal for a certain task [108]. Chapter 5 discusses this case further and introduces an optimisation strategy for changing environmental conditions. In the literature, the focus lies on applying multi-task Gaussian processes to Bayesian optimisation [190] and extending acquisition functions to account for different tasks—such as expected improvement [209, 73], entropy search [190] and knowledge gradient [216, 198].

2.6 Summary

This chapter provided essential background information on the optimisation of physical experiments and computer simulators for the subsequent chapters by introducing expensive black-box problems and presenting Bayesian optimisation as a promising solution for their optimisation. Compared to other optimisation strategies, Bayesian optimisation (i) uses Bayesian models to incorporate prior beliefs about the objective function and (ii) selects potential candidate points sequentially guided by heuristics, resulting in a sample-efficient approach. We focused mainly on surrogate modelling with Gaussian processes and the selection of candidate points with acquisition functions before giving an overview of exceptional cases of expensive black-box optimisation, such as parallel, high-dimensional and multi-fidelity problems.
Subsequent chapters build upon this background knowledge and empirically investigate the performance of different acquisition functions and other fundamental properties of Bayesian optimisation (Chapter 3), present an open-source Python package for the optimisation of expensive physical experiments and computer simulators (Chapter 4), and develop a method for optimising problems with randomly changing environmental conditions (Chapter 5).

Chapter 3

An empirical investigation of Bayesian optimisation

Summary

Bayesian optimisation provides an effective method to optimise expensive-to-evaluate black-box functions. It has been widely applied to problems in many fields, notably in computer science, e.g., machine learning, to optimise hyperparameters of neural networks, and in engineering, e.g., fluid dynamics, to optimise control strategies that maximise drag reduction. This chapter empirically studies and compares the performance and the robustness of standard Bayesian optimisation algorithms on a range of synthetic test functions to provide general guidance on the design of Bayesian optimisation algorithms for specific problems. It investigates the choice of acquisition function, the effect of different numbers of training samples, the exact and Monte Carlo-based calculation of acquisition functions, and both single-point and multi-point optimisation. The test functions considered cover a wide range of challenges and, therefore, serve as a suitable test bed to understand the performance of Bayesian optimisation to specific challenges and in general. To illustrate how these findings can be used to inform a Bayesian optimisation setup tailored to a specific problem, two simulations in the area of computational fluid dynamics are optimised, giving evidence that suitable solutions can be found in a relatively small number of evaluations of the objective function for complex, real problems. The results of our investigation can be applied to other areas, such as machine learning and physical experiments, where objective functions are expensive to evaluate and their mathematical expressions are unknown.

3.1 Introduction

Apart from the applications discussed in Chapter 2, Bayesian optimisation has recently been used successfully in the data-intensive field of computational fluid dynamics (CFD). Examples include Talnikar et al. [192] who developed a Bayesian optimisation framework for the parallel optimisation of large eddy simulations. They used this framework (i) on a one-dimensional problem, i.e., a problem with one input variable, to determine the wave speed of a traveling wave that maximises the skin-friction drag reduction in a turbulent channel flow and (ii) on a four-dimensional problem to find an efficient design for the trailing edge of a turbine blade that minimises the turbulent heat transfer and pressure loss. For the former, Bayesian optimisation was able to locate a wave speed to generate a skin-friction drag reduction of 60%, while for the latter within 35 objective function evaluations a design was found that reduced the heat transfer by 17% and the pressure loss by 21%. Mahfoze et al. [123] utilised Bayesian optimisation on two three-dimensional problems to locate optimal low-amplitude wall-normal blowing strategies to reduce the skin-friction drag of a turbulent boundary layer and maximise net power savings. The two problems use different methods to estimate the net power savings and achieve 5% savings with 8.3% global drag reduction within 18 function evaluations, and 0.7% savings with 5.6% global drag reduction within 11 function evaluations, respectively. Lastly, Nabae and Fukagata [139] maximised the skin-friction drag reduction in a turbulent channel flow by optimising the velocity amplitude and the phase speed of a travelling wave-like wall deformation. They achieved a maximum drag reduction of 60.5%.

While these examples, and the adaptation of Bayesian optimisation in general, show that Bayesian optimisation performs well on various problems, such as computer simulators and hyperparameter tuning of machine learning models, there needs to be a more extensive study on the many different types of Bayesian optimisation algorithms in the literature. In particular, there is a research gap regarding the performance and robustness of Bayesian optimisation when applied to distinct challenges. This chapter aims to address this gap by considering a wide range of algorithms and problems with increasing levels of complexity that are, consequently, increasingly difficult to solve. The comparison and the thorough analysis of these algorithms can inform the design of Bayesian optimisation algorithms and allow them to be tailored to the unique problem. Two novel simulations in computational fluid dynamics are optimised to show how the findings can guide the setup of Bayesian optimisation, demonstrating that Bayesian optimisation can find suitable solutions to complex problems in a relatively small number of function evaluations. The results of these applications show that Bayesian optimisation can find promising solutions for expensive-to-evaluate black-box functions. In these specific cases, solutions were found that (a) maximise the drag reduction over a flat plate and (b) achieve drag reduction and net energy

savings simultaneously. For these experiments, the NUBO framework [42] is utilised. NUBO is introduced formally and discussed in Chapter 4.
The chapter is structured as follows. Section 3.2 discusses the synthetic benchmark functions and their advantages over other common benchmarking methods. In Section 3.2.1, four different sets of simulations are presented. These experiments study multiple analytical single-point acquisition functions, varying numbers of initial training points, acquisition functions utilising Monte Carlo sampling and various multi-point acquisition functions. Finally, Section 3.2.2 discusses the main findings and relates them to the problems presented in the introduction and more generally. These findings inform the algorithm used to optimise the setup of two computational fluid dynamics experiments in Section 3.3. Lastly, a brief conclusion is drawn in Section 3.4.

3.2 Synthetic test functions

While there are various methods of benchmarking the performance of optimisation algorithms, such as sampling objective functions from Gaussian processes or using hyperparameter optimisation of machine learning models as the objective function [178, 20, 204], this chapter focuses on using synthetic test functions [188]. Synthetic test functions have the main advantage that their global optima and their underlying shape are known. Thus, the difference between the results and the true optimum can be evaluated when comparing the acquisition functions outlined in Section 2.4 on these functions. For other benchmarking methods, this might not be possible as, for example, in the case of optimising the hyperparameters of a model, the true optimum, i.e., the solution optimising the model's performance, is rarely known. Hence, results can only be discussed relative to other methods, irrespective of how close any method is to the true optimum. Furthermore, knowing the shape of the test function is advantageous as it provides information on how challenging the test function is to optimise. For example, knowing that a test function is particularly smooth and has a single optimum indicates that it is less complex and thus less challenging to optimise than a test function that possesses multiple local optima, in each of which the algorithm could potentially get stuck.
This chapter assesses the Bayesian optimisation algorithms on eight different synthetic test functions spanning various challenges. Table 3.1 presents the test functions and gives details on the number of input dimensions, the shape and the number of optima to give an idea about the complexity of the individual functions. Figure 3.1 shows some of the functions (those that can be presented in three-dimensional space) and indicates the increasing complexity of the functions. Detailed mathematical definitions for all test functions can be found in Appendix A.1. The number of input dimensions for the test functions was chosen to be equal to or slightly greater than the number in the articles

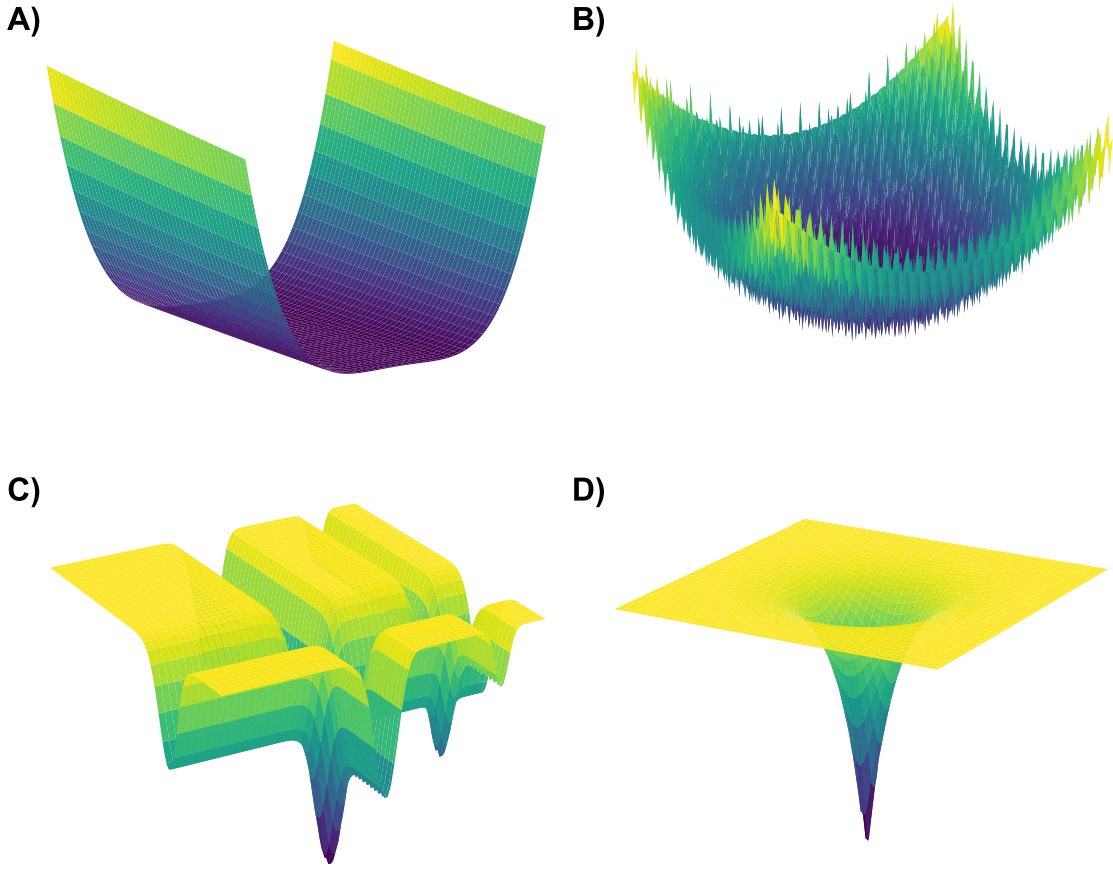

Figure 3.1: Different challenges and levels of complexity represented by the shapes of four test functions where **A)** is the Dixon-Price function, **B)** is the Griewank function, **C)** is the Michalewicz function and **D)** is the modified Ackley function.

on optimising simulations recently published in the fluid dynamics community, which is typically three to eight [123, 136, 192, 139]. The 10D Sphere and 10D Dixon-Price functions are less challenging problems with a high degree of smoothness and only one optimum. They are, therefore, considered for a higher number of input dimensions. The 8D Griewank function adds a layer of complexity by introducing oscillatory properties. The 6D Hartmann function increases the level of difficulty further through its multi-modality. It has six optima with only one global optimum. This function is also considered in two noisy variants to simulate typical measurement uncertainty encountered during experiments in fluid dynamics. For these noisy variants, the standard deviation of the added Gaussian noise is chosen so that it represents the measurement errors of state-of-the-art Micro-Electro-Mechanical-Systems (MEMS) sensors which directly measure time-resolved skin-friction drag in turbulent air flows (e.g., the flow over an aircraft); that is,

Table 3.1: Overview of the seven synthetic test functions.

Test function	Number of input dimensions D	Shape	Number of optima
Sphere	10	Bowl-shaped	1
Dixon-Price	10	Valley-shaped	1
Griewank	8	Oscillatory	1
Hartmann	6	Multi-modal	6
Noisy Hartmann	6	Noisy	6
Michalewicz	5	Steep edges	120
Ackley	6	Mostly flat	1

1.4 to 2.4% in an experimental setting [45]. Based on the 6D Hartmann function's range, the corresponding standard deviations for the Gaussian noise, taken as the limits of a 99% confidence interval, are, therefore, 0.0155 and 0.0266, respectively. The most complex test functions are the 5D Michalewicz and the 6D Ackley functions (this chapter considers a modified Ackley function with $a = 20.0$, $b = 0.5$ and $c = 0.0$). While the former has 120 optima and steep ridges, the latter is mostly flat, with a single global minimum in the centre of the space. The high gradient of the function close to the optimum, and the large flat areas represent a great level of difficulty, as illustrated in the bottom row of Figure 3.1.

3.2.1 Results

This section focuses on the results for four of the eight test functions considered in this chapter. The results for the other four test functions and more extensive tables can be found in Appendices A.3 and A.4, respectively. The 8D Griewank function, the 6D Hartmann function in variations without noise and with the high noise level and the 6D Ackley function are examined here, as they represent increasing levels of complexity and come with unique challenges as illustrated in Section 3.2 above. The following budgets were imposed on the test functions to mirror real-world applications with small evaluation budgets: 200 evaluations for the Griewank and Hartmann functions and 500 for the more complex Ackley function. If not otherwise stated, the number of initial training points is equivalent to five points per input dimension of the given test function, i.e., 40 training points for the Griewank function and 30 training points for the Hartmann and Ackley functions. For each method in the following sections, 50 different optimisation runs were run to investigate how robust and sensitive the methods are to varying initial training points. Each run was initialised with a different set of initial training points sampled from a maximin Latin hypercube design [129, 86]. However, the points were identical for all methods for a specific test function. All experiments were run on container instances on

the cloud with the same specifications (two CPU cores and 8GB of memory) to make runs comparable.
Overall, four different sets of experiments are considered. Firstly, analytical single-point acquisition functions are compared. Secondly, the effects of varying the number of initial training points are investigated. Thirdly, analytical methods are compared with Monte Carlo methods, and lastly, multi-point or batched methods are compared to the single-point results. Space-filling designs exhausting the full evaluation budgets were sampled from a maximin Latin hypercube as a performance baseline.

Analytical single-point acquisition functions

Section 2.4 describes five different groups of acquisition functions: improvement-based, confidence bound-based, information-based, portfolios and Monte Carlo. This section tests representative functions from the improvement-based, confidence bound-based and portfolio acquisition functions on the synthetic test functions outlined above. The focus lies on analytical single-point acquisition functions, which are widely used and thus present a natural starting point. Information-based and Monte Carlo acquisition functions are typically not analytical and are investigated in following sections. Overall, seven different methods are considered: PI, EI, UCB with a variable and fixed β (5.0 and 1.0) and a Hedge portfolio that combines the PI, EI and variable UCB acquisition functions. For a detailed discussion of these functions, see Section 2.4.
Figure 3.2 presents the best solutions, defined by the output value closest to the global optimum, found so far at each evaluation for each acquisition function. The outputs are normalised to the unit range, where 1.00 represents the global optimum. Most methods perform very well on the Griewank function, all reaching 1.00, and both variations of the Hartmann function with and without added noise (all reaching 1.00 and >0.97, respectively). All acquisition functions find the optimum or a solution reasonably close to the optimum within the allocated evaluation budget. However, PI and variable UCB typically require more evaluations to find a solution close to the optimum. All acquisition functions perform noticeably better than the Latin hypercube benchmark and exhibit much less variation over the 50 runs, as indicated by the 95% confidence intervals calculated at each evaluation as $\mu \pm 1.96 \times \sigma$, where μ is the average output over the 50 runs and σ is the standard deviation of the outputs. There is little difference between the Hartmann function with and without added noise; indeed, the results are almost identical. The Ackley function, on the other hand, is more challenging. While the UCB methods still perform very well (all >0.96), the performance of the portfolio and PI decrease slightly (0.91 and 0.88, respectively), and EI performs considerably worse (0.63) with more variability between runs.
Similar conclusions can be drawn from Table 3.2, which presents each method's area under

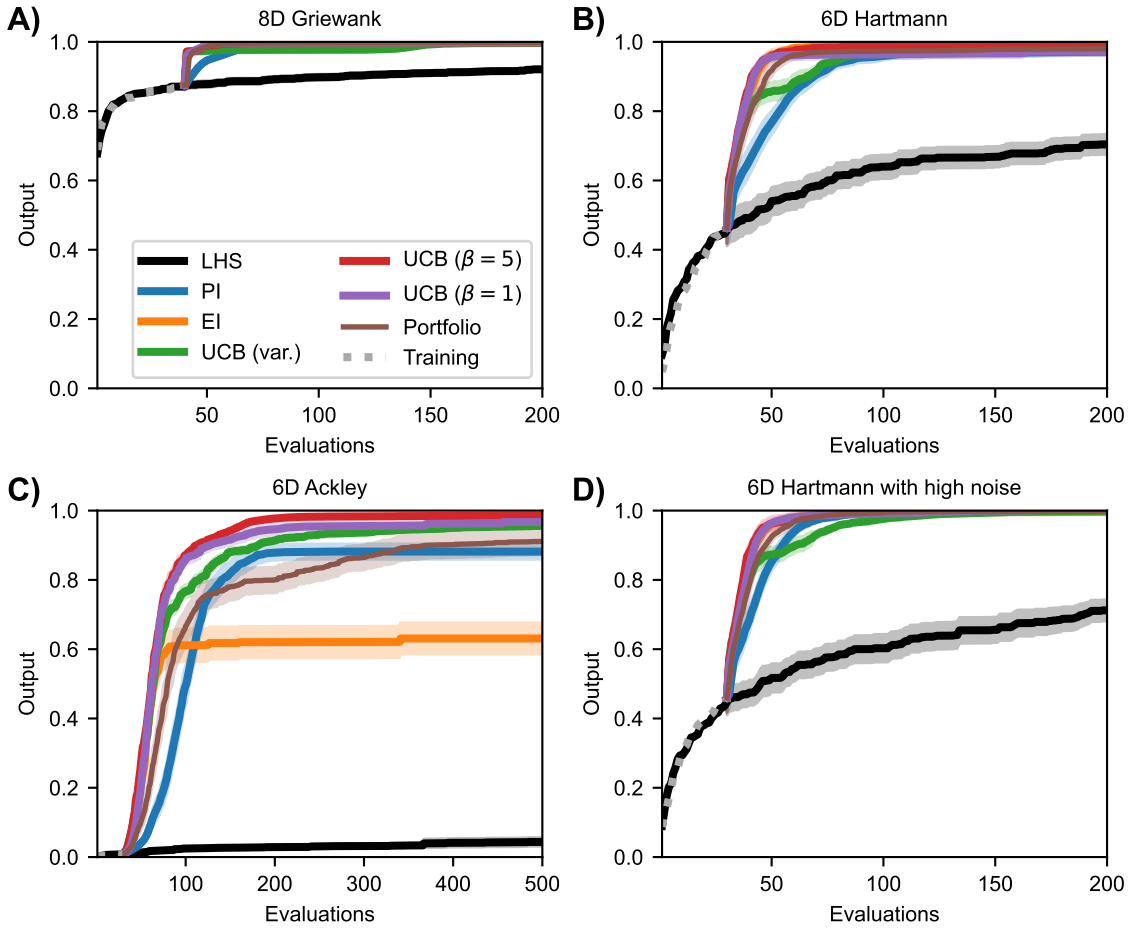

Figure 3.2: Performance plots for analytical single-point acquisition functions with five initial starting points per input dimension. Solid lines represent the mean over the 50 runs while the shaded area represents the 95% confidence intervals.

the receiver operating characteristic (ROC) curve (AUC) [75, 17]. The AUC indicates how quickly the individual methods find solutions near the optimum. A perfect score (1.0) would indicate that the algorithm finds the optimum perfectly at the first iteration. The lower the score (a) the further the algorithm is away from the optimum and (b) the more evaluations are required for the algorithm to find promising solutions. While all methods score at least 0.98 for the Griewank function with a standard error of at most 0.01, the scores worsen slightly for the Hartmann function and significantly for the Ackley function. In particular, the AUC of EI for the Ackley function is poor (0.59) and exhibits high variability between the 50 runs (standard error: 0.15). At the opposite end of the performance spectrum lie the optimistic methods, i.e., the UCB with variable and fixed β, that perform very well on all test functions with low standard errors. These results suggest that while the choice of acquisition function is less relevant for simple and moderately

Table 3.2: Averaged AUC with standard error for analytical single-point acquisition functions with five initial training points per input dimension.

Method	Griewank	Hartmann	Noisy Hartmann	Ackley
PI	0.99 (± 0.00)	0.91 (± 0.04)	0.95 (± 0.02)	0.76 (± 0.08)
EI	**1.00** (± 0.00)	**0.97** (± 0.02)	0.97 (± 0.02)	0.59 (± 0.15)
UCB (variable)	0.98 (± 0.01)	0.94 (± 0.03)	0.95 (± 0.03)	0.85 (± 0.01)
UCB (β=5)	**1.00** (± 0.00)	**0.97** (± 0.02)	**0.98** (± 0.02)	**0.90** (± 0.02)
UCB (β=1)	**1.00** (± 0.00)	0.95 (± 0.04)	0.97 (± 0.02)	0.88 (± 0.05)
Hedge	**1.00** (± 0.00)	0.95 (± 0.03)	0.97 (± 0.02)	0.77 (± 0.11)

complex objective functions such as the Griewank or Hartmann functions, it is important when solving challenging problems such as the Ackley function, especially if large flat areas characterise them. Here, the optimistic acquisition functions are advantageous and should be preferred over the Hedge portfolio and improvement-based approaches. It should be noted that this may only be true for the specific portfolio defined previously. Implementing a different collection of acquisition functions could yield different results.

Varying number of initial training points

The training points initialising the Bayesian optimisation algorithm directly affect the surrogate model (Gaussian process), representing the objective function. A larger number of training points yields a Gaussian process that will typically represent the objective function better as it incorporates more data and, thus, more information. However, the more evaluations of the total budget allocated for these initial training points, the fewer points can be evaluated as part of the Bayesian optimisation algorithm. This trade-off indicates that the number of training points (and, by extension, their selection) is a crucial choice in Bayesian optimisation and should be considered thoroughly. This section explores this trade-off by taking the same experimental setup as the previous section but varying the number of initial training points. Overall, setups with one, five and ten initial points per input dimension of the objective function are considered.
Figures 3.3 and 3.4 depict the performance plots for the case with one and ten training

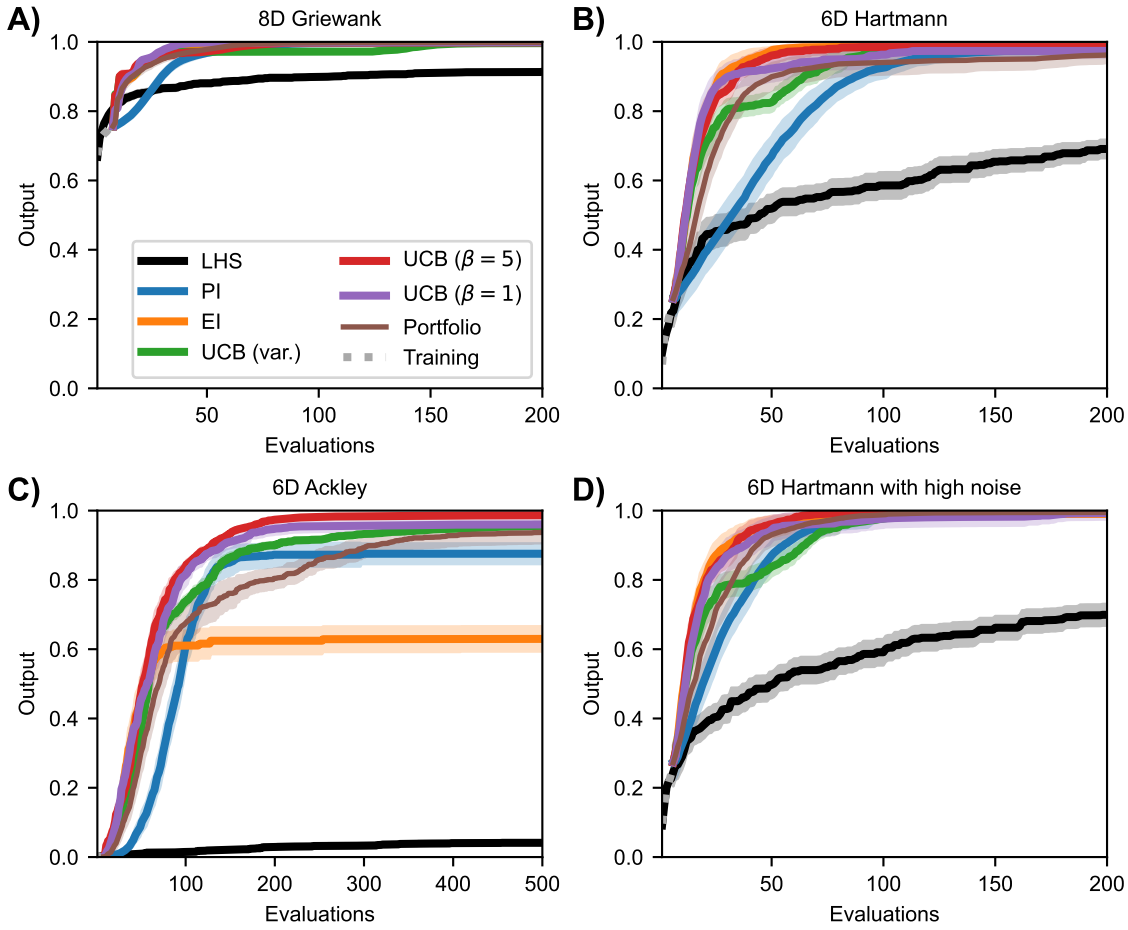

Figure 3.3: Performance plots for analytical single-point acquisition functions with one initial starting point per input dimension. Solid lines represent the mean over the 50 runs while the shaded area represents the 95% confidence intervals.

points per dimension, respectively. If the number of points is reduced to one point per dimension, the individual methods find solutions virtually identical to the results with five training points per dimension. EI still performs much worse than other methods for the Ackley function. Furthermore, most methods find their best solution in a comparable or slightly higher number of evaluations than before, as the AUC values in Table 3.3 show. This is expected, as using fewer initial training points means that the Bayesian optimisation algorithm has less information at the earlier iterations than when using more initial training points. However, the difference in mean AUC is small, and the results suggest that Bayesian optimisation quickly makes up for this lack of information. PI and the Hedge portfolio have the highest decrease in performance on average for the Hartmann function, with the mean AUC decreasing by 0.09 and 0.06, respectively. The variability between runs of the Hedge algorithm also rises, as indicated by an AUC standard error

Table 3.3: Averaged AUC with standard error for analytical single-point acquisition functions with one initial training point per input dimension.

Method	Griewank	Hartmann	Noisy Hartmann	Ackley
PI	0.97 (± 0.01)	0.82 (± 0.09)	0.90 (± 0.06)	0.73 (± 0.11)
EI	**0.99** (± 0.00)	**0.95** (± 0.03)	**0.95** (± 0.03)	0.58 (± 0.13)
UCB (variable)	0.97 (± 0.01)	0.91 (± 0.04)	0.91 (± 0.03)	0.82 (± 0.01)
UCB (β=5)	0.98 (± 0.00)	0.94 (± 0.03)	**0.95** (± 0.03)	**0.87** (± 0.02)
UCB (β=1)	**0.99** (± 0.00)	0.93 (± 0.08)	0.93 (± 0.09)	0.85 (± 0.06)
Hedge	0.98 (± 0.01)	0.89 (± 0.11)	0.92 (± 0.05)	0.76 (± 0.09)

0.08 higher. PI and the portfolio also perform worse for the Noisy Hartmann function, where the mean AUC decreased by 0.05 for both methods. Intuitively, this makes sense as the surrogate model at the start of the Bayesian optimisation algorithm includes less information than before. Thus, it takes more evaluations to find a good solution. In early iterations, the individual methods deviate from one another more than when five training points per dimension are used.

If the number of starting points is increased to ten points per dimension, there is essentially no change to the case with just five starting points per dimension for the Griewank and the two Hartmann functions. For the Ackley function, however, the performance of both the best solution found and the AUC worsen (Table 3.4) for EI and, most significantly, PI. The average best solution for the latter decreases by 0.37 to only 0.51, while EI decreases by 0.10 to 0.53. Both policies struggle with the large area of the test function that gives identical response value and is hence flat (see Section 3.2). The optimistic policies and the portfolio perform much better, and no real change is noticeable from the results discussed in Section 3.2.1.

These results suggest that choosing more training points to initialise the Bayesian optimisation algorithm cannot necessarily be equated with better performance and solutions. This is particularly true considering there was no improved performance when increasing from five to ten points per input dimension. On the other hand, reducing the number of training points did not yield noticeably worse results. Overall, a similar picture as in the previous section emerges: while all methods perform well on more straightforward

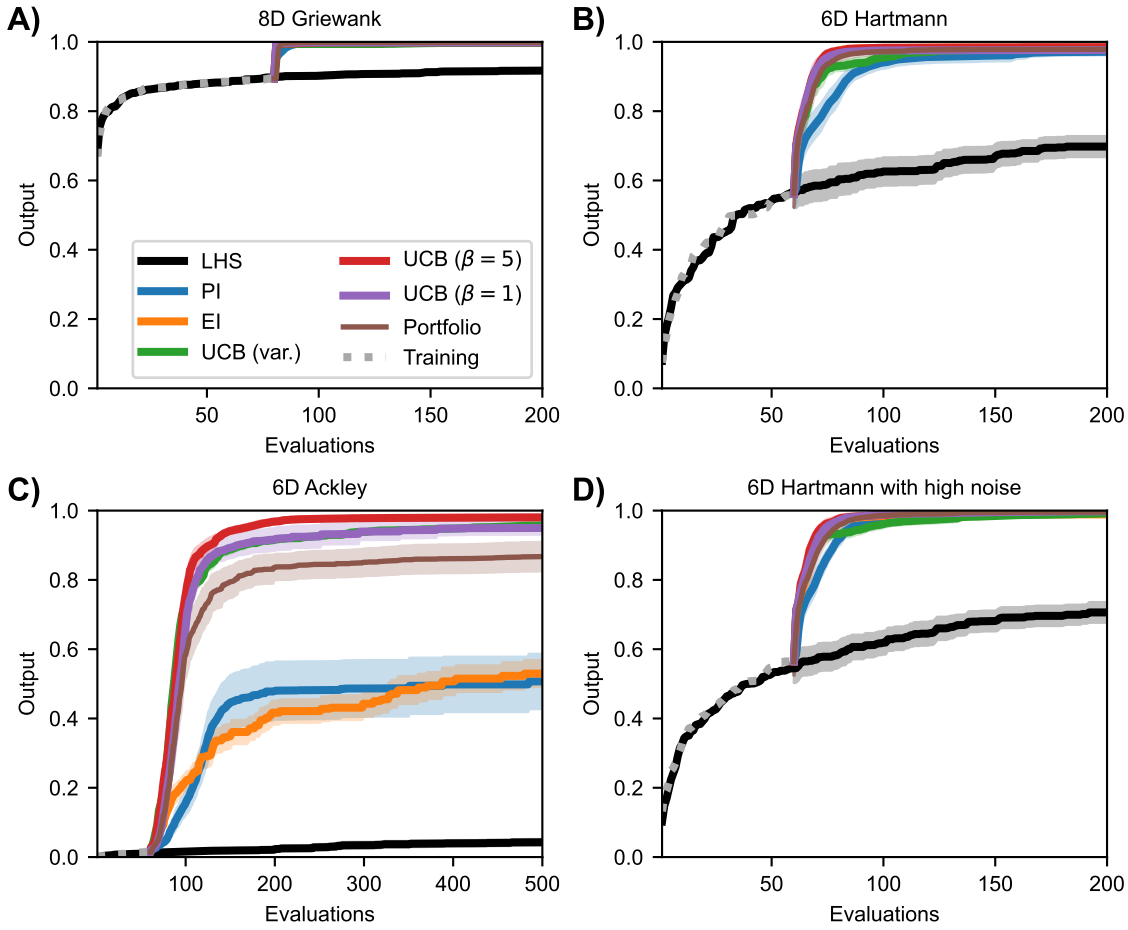

Figure 3.4: Performance plots for analytical single-point acquisition functions with ten initial starting points per input dimension. Solid lines represent the mean over the 50 runs while the shaded area represents the 95% confidence intervals.

problems, optimistic policies achieve the best results on the more challenging problems independent of the number of starting points. Five training points per dimension appeared sufficient for the test functions we considered. There was no discernible improvement when moving to ten training points and a small performance loss when reducing to one training point per dimension.

Monte Carlo single-point acquisition functions

The previous experiments considered analytical acquisition functions. This section further assesses the Monte Carlo approach outlined in Section 2.4.5. As not all acquisition functions can be rewritten to suit such an approach, the experiments are restricted to PI, EI and UCB with variable and fixed β. Max-value entropy search (MES) is also used as a new method, as introduced in Section 2.4.3.

Table 3.4: Averaged AUC with standard error for analytical single-point acquisition functions with ten initial training points per input dimension.

Method	Griewank	Hartmann	Noisy Hartmann	Ackley
PI	**1.00** (± 0.00)	0.93 (± 0.05)	0.96 (± 0.02)	0.43 (± 0.25)
EI	**1.00** (± 0.00)	**0.97** (± 0.02)	0.97 (± 0.02)	0.41 (± 0.07)
UCB (variable)	**1.00** (± 0.00)	0.95 (± 0.03)	0.96 (± 0.03)	0.87 (± 0.01)
UCB (β=5)	**1.00** (± 0.00)	**0.97** (± 0.02)	**0.98** (± 0.02)	**0.91** (± 0.03)
UCB (β=1)	**1.00** (± 0.00)	0.96 (± 0.03)	**0.98** (± 0.03)	0.86 (± 0.08)
Hedge	**1.00** (± 0.00)	0.96 (± 0.03)	0.97 (± 0.02)	0.78 (± 0.13)

Table 3.5: Averaged AUC with standard error for Monte Carlo single-point acquisition functions with five initial training points per input dimension.

Method	Griewank	Hartmann	Noisy Hartmann	Ackley
MC PI	0.99 (± 0.00)	0.92 (± 0.06)	0.95 (± 0.04)	0.17 (± 0.12)
MC EI	**1.00** (± 0.00)	**0.97** (± 0.02)	0.97 (± 0.04)	0.05 (± 0.03)
MC UCB (variable)	0.98 (± 0.01)	0.95 (± 0.02)	0.95 (± 0.04)	0.86 (± 0.01)
MC UCB (β=5)	**1.00** (± 0.00)	**0.97** (± 0.02)	**0.98** (± 0.03)	**0.91** (± 0.01)
MC UCB (β=1)	**1.00** (± 0.00)	0.96 (± 0.07)	0.97 (± 0.07)	0.89 (± 0.03)
MES	0.99 (± 0.00)	**0.97** (± 0.02)	0.97 (± 0.03)	0.49 (± 0.12)

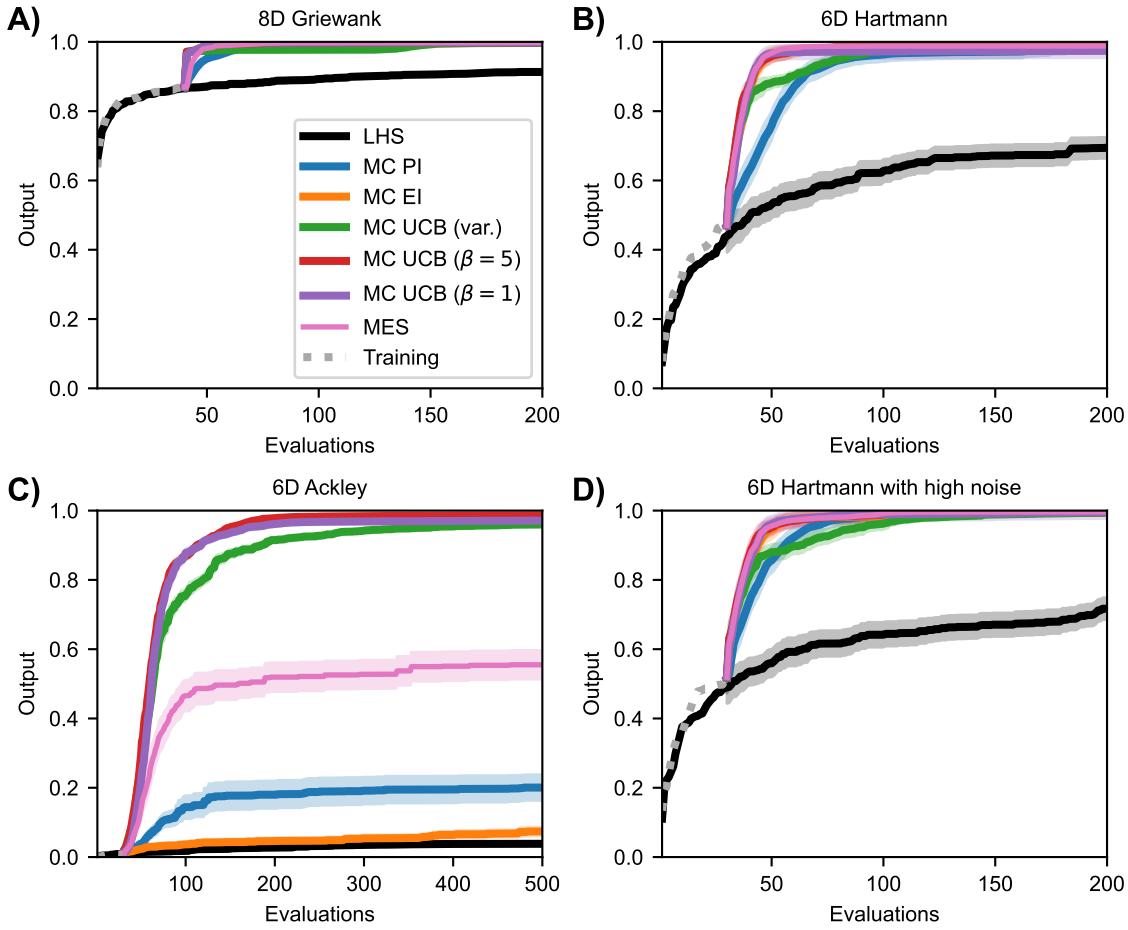

Figure 3.5: Performance plots for Monte Carlo single-point acquisition functions with five initial starting points per input dimension. Solid lines represent the mean over the 50 runs while the shaded areas represent the 95% confidence intervals.

For the Griewank and both Hartmann functions, all results are almost identical to the analytical case except for the variability between runs, which increased slightly for some methods. However, Figure 3.5 clearly shows that the performance for PI and EI on the Ackley function decreased significantly. The average best solutions decreased by 0.68 and 0.56, and the average AUC (Table 3.5) decreased by 0.59 and 0.54, respectively. Table 3.5 shows that MES performs well on the Griewank and Hartmann functions, reaching an AUC above 0.97 with low standard errors. However, MES performs much worse when it comes to the more complex Ackley function. With a mean AUC of 0.49, its performance ranks below the optimistic acquisition functions but above the improvement-based methods. Earlier sections showed that improvement-based policies (particularly EI) perform poorly on the Ackley function when using analytical acquisition functions. However, the results from this comparison show that their performance suffers even more severely when

using the Monte Carlo approach. One reason could be that the Monte Carlo variants essentially approximate the analytical acquisition functions and are less accurate as discussed in Section 2.4.5. Another well-established problem of improvement-based acquisition is that their acquisition values are numerically zero in many areas and thus challenging to maximise. This resulted in inconsistent performance in the past [2]. However, there seems to be little to no change when using optimistic policies. This suggests that, similar to previous sections, optimistic policies should be preferred when optimising complex and challenging objective functions with large flat areas using Monte Carlo acquisition functions.

Multi-point acquisition functions

The previous sections focused on single-point approaches, where each iteration of the Bayesian optimisation algorithm yields one new point sampled from the objective function before the next iteration of the optimisation loop. While this makes sense for objective functions that can be evaluated quickly or that do not allow parallel evaluations, it might slow down the optimisation process needlessly when objective functions take a long time to evaluate and allow parallel evaluations. Thus, this section implements multi-point acquisition functions that propose a batch of points at each iteration, which are evaluated simultaneously before the next iteration.

Section 2.4.5 outlined how Monte Carlo approaches using the reparameterisation trick can be extended to compute batches naturally. MES does not use reparameterisation and is thus not naturally extendable to the multi-point setting. Hence, we consider the same acquisition functions as in the previous section, but this time for a batch size of five points. Each acquisition function is optimised with two different methods—sequentially and jointly. The latter computes all batch points in a single step, while the former selects one batch point after the other, as discussed in Section 2.4.5. For example, for a batch of five points, the sequential strategy repeats the selection process five times [7]. This approach, also known as greedy optimisation, might be preferable and yield better results [211]. While analytical functions cannot be extended to the multi-point case as easily as the Monte Carlo evaluations, some frameworks allow the computation of batches. This section considers the constant liar approach with a lie equivalent to the minimal (CL min) and maximal (CL max) value so far [64] and the GP-BUCB algorithm, an extension of the UCB function [38]. For more details, see Section 2.5.3.

Figure 3.6 shows the results for a selection of multi-point acquisition functions. They essentially find identical solutions, on average, to the single-point approach for the Griewank function and Hartmann function with and without noise. Only selected methods are shown in the plot. Specifically, two variations of the UCB approach are not included as they are almost identical to the UCB with β=1 (see Appendices A.3 and A.4). While there are no

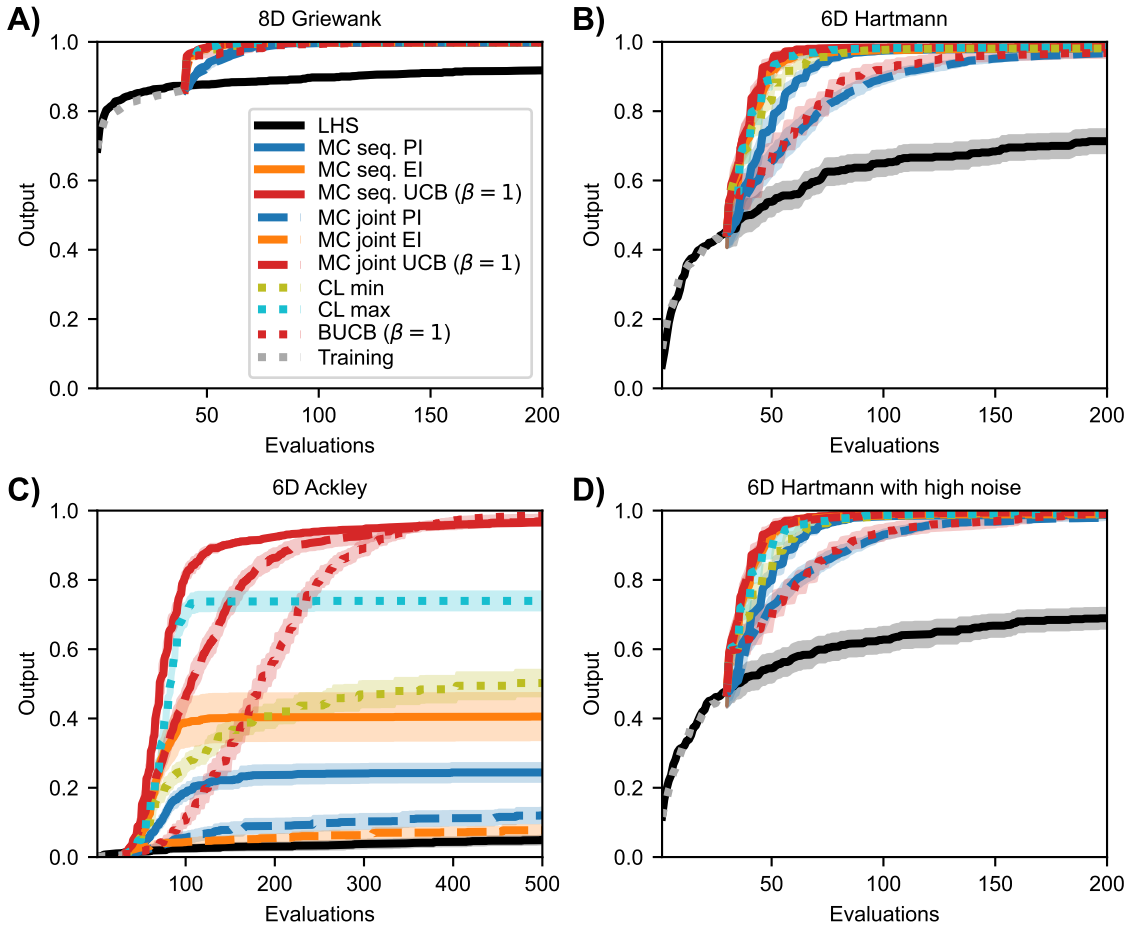

Figure 3.6: Performance plots for multi-point acquisition functions with five initial training points per input dimension. Solid lines represent the mean over the 50 runs while shaded areas represent the 95% confidence intervals. PI in blue, EI in orange, UCB in red.

changes to the AUC for the Griewank function, some differences for the Hartmann functions suggest that methods require a different number of evaluations to find the best value, as Table 3.6 shows. Although the joint Monte Carlo approach has a lower mean AUC for PI and UCB (variable and β=5), the decrease is less severe than for the GP-BUCB methods, which all decrease by 0.08 to 0.10. The other methods perform comparably to the single-point case, and the variability between the sequential and joint Monte Carlo UCB runs with $\beta = 1$ even decreases. The results of the noisy Hartmann test function are very similar to those of the Hartmann function without noise. In general, over all experiments, no discernible drop in performance can be declared as a result of adding noise up to 2.4% of the overall objective function range.

Most methods find similar best solutions to the Ackley test function as their single-point counterparts. However, there are some changes in the improvement-based policies: the

Table 3.6: Averaged AUC with standard errors for multi-point acquisition functions with five initial training points per input dimension.

Type	Method	Griewank	Hartmann	Noisy Hartmann	Ackley
Sequential Monte Carlo	PI	0.99 (± 0.00)	0.92 (± 0.03)	0.94 (± 0.02)	0.22 (± 0.08)
	EI	**1.00** (± 0.00)	0.95 (± 0.03)	**0.97** (± 0.02)	0.37 (± 0.22)
	UCB (variable)	0.98 (± 0.01)	0.94 (± 0.03)	0.94 (± 0.03)	0.84 (± 0.01)
	UCB (β=5)	0.99 (± 0.00)	0.95 (± 0.02)	0.96 (± 0.02)	**0.88** (± 0.02)
	UCB (β=1)	**1.00** (± 0.00)	**0.96** (± 0.02)	**0.97** (± 0.02)	0.86 (± 0.02)
Joint Monte Carlo	PI	0.99 (± 0.00)	0.86 (± 0.05)	0.89 (± 0.04)	0.09 (± 0.06)
	EI	**1.00** (± 0.00)	0.95 (± 0.03)	0.96 (± 0.02)	0.06 (± 0.03)
	UCB (variable)	0.97 (± 0.02)	0.89 (± 0.04)	0.90 (± 0.04)	0.81 (± 0.02)
	UCB (β=5)	0.99 (± 0.00)	0.94 (± 0.03)	0.95 (± 0.03)	0.87 (± 0.02)
	UCB (β=1)	**1.00** (± 0.00)	**0.96** (± 0.02)	**0.97** (± 0.02)	0.78 (± 0.06)
Analytical	CL min	0.99 (± 0.00)	0.94 (± 0.02)	0.95 (± 0.02)	0.40 (± 0.08)
	CL max	**1.00** (± 0.00)	0.95 (± 0.03)	0.96 (± 0.03)	0.67 (± 0.09)
	BUCB (variable)	0.98 (± 0.02)	0.85 (± 0.05)	0.85 (± 0.05)	0.65 (± 0.03)
	BUCB (β=5)	0.98 (± 0.00)	0.88 (± 0.05)	0.90 (± 0.05)	0.66 (± 0.05)
	BUCB (β=1)	0.99 (± 0.00)	0.88 (± 0.06)	0.90 (± 0.06)	0.65 (± 0.05)

sequential Monte Carlo EI and PI, and CL max, find better solutions than before (by 0.33, 0.04 and 0.11, respectively). Conversely, CL min and joint Monte Carlo PI provide inferior solutions in the batched case (0.13 and 0.08, respectively). Looking at how quickly the individual methods find good values on average, i.e., the AUC, it is clear that the analytical multi-point and joint Monte Carlo methods perform worse than the single-point implementations for the Ackley function: the AUC of all analytical multi-point methods worsened by 0.19 to 0.24, except for CL max that improved by 0.08. Similarly, all joint methods perform more poorly than in the sequential single-point optimisation where UCB with $\beta = 1$ sees the most significant drop of 0.11. EI is the exception and stays approximately the same. The sequential Monte Carlo EI provides the most significant improvement with an increase of 0.32. However, this improvement is mainly caused by the very poor performance of the single-point Monte Carlo EI acquisition function. Furthermore, this approach comes with a higher variability, as the standard error of the AUC rises by 0.19.

Regarding the best solutions found, multi-point methods present a good alternative to single-point acquisition functions. They generally find better solutions comparable to the single-point approach but for a slightly larger number of objective function evaluations. However, while it requires more evaluations, the multi-point approach would still be faster when computing batches in parallel. Multi-point acquisition functions will be most beneficial for expensive-to-evaluate objective functions that can be evaluated in parallel. With some exceptions, this benefit requires more evaluations until solutions comparable to single-point methods are found. Overall, combined with sequential optimisation, the optimistic methods appear favourable over the rest, as the red lines in Figure 3.6 clearly show.

3.2.2 Discussion

Six main conclusions can be drawn from the simulation results presented in detail above. This section discusses these findings and relates them to the applied problems that motivated this work, i.e., optimising experiments in engineering, particularly fluid dynamics, and tuning hyperparameters of neural networks and other statistical models.

The first findings concern the choice of acquisition functions related to the complexity of the problem. The results show that this choice is less critical when optimising simpler objective functions. Improvement-based, confidence bound-based, and information-based acquisition functions, as well as portfolios, performed well on a wide range of synthetic test functions with up to ten input dimensions (see Appendices A.3 and A.4 for more examples to reinforce this result). However, for more complex functions, such as the Ackley function, the optimistic methods performed significantly better than the rest and thus should be favoured. For the Bayesian optimisation algorithm, all acquisition functions considered

in this chapter can yield good results. However, the choice of acquisition function must be considered more carefully with increasing complexity of the objective function. This indicates the importance of expert knowledge when applying Bayesian optimisation to a specific problem, such as a drag reduction problem in fluid dynamics. Basing the choice of acquisition function on specific knowledge about the expected complexity of the objective function could significantly increase the Bayesian optimisation algorithm's performance. In cases where no expert knowledge or other information about the objective function is accessible, results suggest that the optimistic methods are a good starting point. This result is reinforced by the statistical analysis in Appendix A.2 that shows that methods indeed perform significantly differently from each other and that confidence bound-based acquisition functions find the best solutions with the least variation between the 50 replications.

Secondly, the results suggest that the number of initial training points is not critical in achieving good performance from the algorithm. There is little difference when increasing the number of starting points from one point per input dimension to ten points per dimension. While the performance of the acquisition functions differs initially for the former case, they still find comparable results to the cases of five or ten points per input dimension over all iterations. This means that Bayesian optimisation efficiently explores the space even when provided with only a few training points. Deciding on the number of starting points is an important decision in problems where evaluating a point is expensive. For example, when evaluating a set of hyperparameters for a complex model such as a deep neural network, the model must be retrained for each set of hyperparameters, racking up time and computing resource costs. In most cases, the problem involves dividing a predefined budget into two parts: evaluations to initialise the algorithm and evaluations for points proposed by the Bayesian optimisation algorithm. The decision of how many points to allocate for the training budget is important. Using too many could mean that the Bayesian optimisation algorithm does not have sufficient evaluations available to find a suitable solution, i.e., the budget is exhausted before a promising solution is found. The simulations in this chapter suggest that this decision might not be as complex as it initially seems, as only a few training points are necessary for the algorithm to achieve good results. Using only a small number of evaluations for training points saves more evaluations for the optimisation itself, thus increasing the chances of finding a good solution.

The third finding regards the information-based acquisition functions. PI and EI failed to find good results for the Ackley function across the different simulations. The reason for this is the large area of the parameter space where all response values are identical; thus, the response surface is flat. As discussed in Section 2.4.1, improvement-based acquisition functions propose a point most likely to improve upon the previous best point. With a flat function like the Ackley function, all initial starting points likely fall into the flat area

(especially in higher dimensions where the flat area grows exponentially with the number of input dimensions). The posterior mean of the Gaussian process will then be very flat and will, in turn, predict that the underlying objective function is flat as well. This leads to a very flat acquisition function, as most input points will have a small likelihood of improving upon the previous best points. Gramacy [69] mentions problems when optimising with a flat EI acquisition function that results in the evaluation of points that are not optimal, which is especially problematic when the shape of the objective function is not known and flat regions cannot be ruled out a priori. If such properties are possible in a particular applied problem, the results suggest that using a different acquisition function, such as an optimistic policy, achieves better results. This again highlights the importance of expert knowledge for applications to specific problems.

Fourthly, the simulations show that Monte Carlo acquisition functions yield comparable results to analytical functions. These functions use Monte Carlo sampling to compute the acquisition functions instead of analytically solving them. Utilising sampling to compute a function that can be solved analytically might not appear useful at first glance, as it is essentially just an approximation of the analytical results. However, this approach makes it straightforward to compute batches of candidate points (as outlined in Section 2.5.3), which presents the foundation for the following finding.

The fifth finding suggests that multi-point acquisition functions perform comparably to single-point approaches. As discussed previously, all methods found solutions close to the optimum across the range of considered problems, except improvement-based methods for the Ackley function. Multi-point approaches are particularly beneficial for cases where the objective function is expensive-to-evaluate and allows parallel evaluations (e.g., in high-fidelity turbulence resolving simulations [123]). For these problems, the total time required to conclude the whole experiment can be reduced as multiple points are evaluated in parallel each time. However, more evaluations might be required to achieve results comparable to the single-point approach. This method is particularly advantageous for applications involving simulations that can be evaluated in parallel, e.g., the computational fluid dynamics simulations mentioned in the introduction.

Lastly, the results showed no decrease in performance when adding noise to the objective function, in this case the 6D Hartmann function. When optimising the function with low and high noise (corresponding to the measurement error range of MEMS sensors to mirror the circumstances of an applied example in fluid dynamics), the same solutions were found in a comparable number of evaluations as for the deterministic case. This result can be attributed to the nugget that is added to the deterministic and noisy cases, as discussed in Section 2.3. These results are promising, as they show that Bayesian optimisation can handle noisy objective functions just as efficiently as deterministic functions. This enables the use of Bayesian optimisation for physical experiments where measurements

cannot be taken without error and simulations where noise can be introduced unknowingly. Consider, for example, a physical experiment in which the drag created by air blowing over a surface can be reduced by blowing air upwards orthogonally to the same surface using multiple actuators (for further details, refer to Section 3.3). As each actuator has a large, if not infinite, number of settings, finding the optimal overall blowing strategy is a complex problem that is a prime candidate for Bayesian optimisation. However, the drag cannot be measured without noise, as the measurement errors associated with the MEMS sensors introduced in Section 3.2 show. As it is impossible to collect noise-free data in such circumstances, and taking the same measurement twice would yield slightly different results, Bayesian optimisation must perform equally well on these problems.
While these findings show that using Bayesian optimisation to optimise expensive black-box functions is promising, some limitations should be noted. Firstly, the acquisition functions considered in this chapter represent only a subset of those available in the literature. The general results inferred from this selection might only extend to some acquisition functions. Secondly, the dimensionality of the test functions was selected to be no greater than ten. While Bayesian optimisation is generally considered to work best in this range, and even up to 20 input parameters [53], it would appear unlikely that these results would hold in a much higher-dimensional space [135]. While this is an ongoing area of research, multiple algorithms attempt to make Bayesian optimisation viable for higher dimensions, as discussed in Section 2.5.4. Thirdly, the conclusions drawn in this chapter are based on the synthetic test functions chosen and their underlying behaviour. When encountering objective functions with shapes and challenges different from those considered in this chapter, the findings might not hold. Lastly, the noise added to the Hartmann function could be too low to represent the full space of realistic experiments. It is possible that the measurement error, or other sources of noise, from an experiment or simulation, are too large for Bayesian optimisation to work effectively. More investigation is needed to find the maximal noise levels that Bayesian optimisation can tolerate while performing well.

3.3 Application

While most of the work in this section is my own, Joseph O'Connor from the Turbulence Simulation Group at Imperial College London set up and ran the computational fluid dynamics simulations and wrote the second paragraph describing the details of this simulation work as published in Diessner et al. [41].

In this section, we apply the findings from the investigation on synthetic test functions in Section 3.2 to the selection and design of Bayesian optimisation algorithms to specific applications, in this case, simulation models in the area of computational fluid dynamics

(CFD). Consider high-fidelity simulations involving the turbulent flow over a flat plate, as illustrated in Figure 3.7. Initially, the flow within the boundary layer is laminar. However, after a critical streamwise length, the flow transitions into a turbulent regime, characterised by increased turbulence activity and increased skin-friction drag. This setup mimics the flow encountered on many vehicles, e.g., the flow over an aircraft wing, a high-speed train or a ship's hull, as outlined in Section 1.1. These simulations aim to minimise the turbulent skin-friction drag by utilising active control via actuators, which are seen as a key upstream technology approach for the aerospace sector that allows us to either blow fluid away from the plate or suck fluid towards the plate. These actuators are located towards the beginning of the plate but after the transition region, where the flow is fully turbulent. An averaged skin-friction drag coefficient is then measured along the plate from the blowing region. Within the blowing region, a very large number of blowing setups are possible. For example, a simple setup could be a 1D problem where fluid is blown away from the plate uniformly over the entire blowing region with a constant amplitude. In this case, the only parameter to optimise would be the amplitude of the blowing. However, many more complex setups are possible (e.g., see Mahfoze et al. [123]). These numerical simulations are a prime candidate for Bayesian optimisation as they fulfil the characteristics of an expensive black-box function: the underlying mathematical expression is too complex to solve analytically, and each objective function evaluation is expensive. Indeed, one evaluation (a high-fidelity simulation with converged statistics) can take up to 12 hours on thousands of CPU cores and requires the use of specialist software since the total turbulence activity covering the flat plate must be simulated in order to correctly resolve the quantity of interest (i.e., skin-friction drag). To perform these numerical simulations, the open-source flow solver Xcompact3D [8] is utilised on the UK supercomputing facility ARCHER2. The simulations were performed by the Turbulence Simulation Group at Imperial College London. This section optimises two blowing profiles that follow this setup: a travelling wave with four degrees of freedom and a gap problem with three degrees of freedom, where a gap separates two blowing areas with individual amplitudes.

The computational setup consists of a laminar Blasius solution at the inlet, a convective condition at the outlet, a homogenous Neumann condition in the far-field, and periodic conditions in the spanwise direction. The domain dimensions are $L_x \times L_y \times L_z = 750\delta_0 \times 80\delta_0 \times 30\delta_0$, where δ_0 is the boundary layer thickness at the inlet. The Reynolds number at the inlet is $Re_{\delta_0} = 1250$, based on the boundary layer thickness (i.e., the thin layer of fluid above the plate where the flow velocity is reduced due to the presence of the plate) and free-stream velocity (u_∞., i.e., the speed of the moving vehicle) at the inlet of the simulation domain. This corresponds to a momentum Reynolds number of $Re_{\theta_0} \approx 169$ to 2025, from the inlet to the outlet for the canonical case (no control). The mesh size is

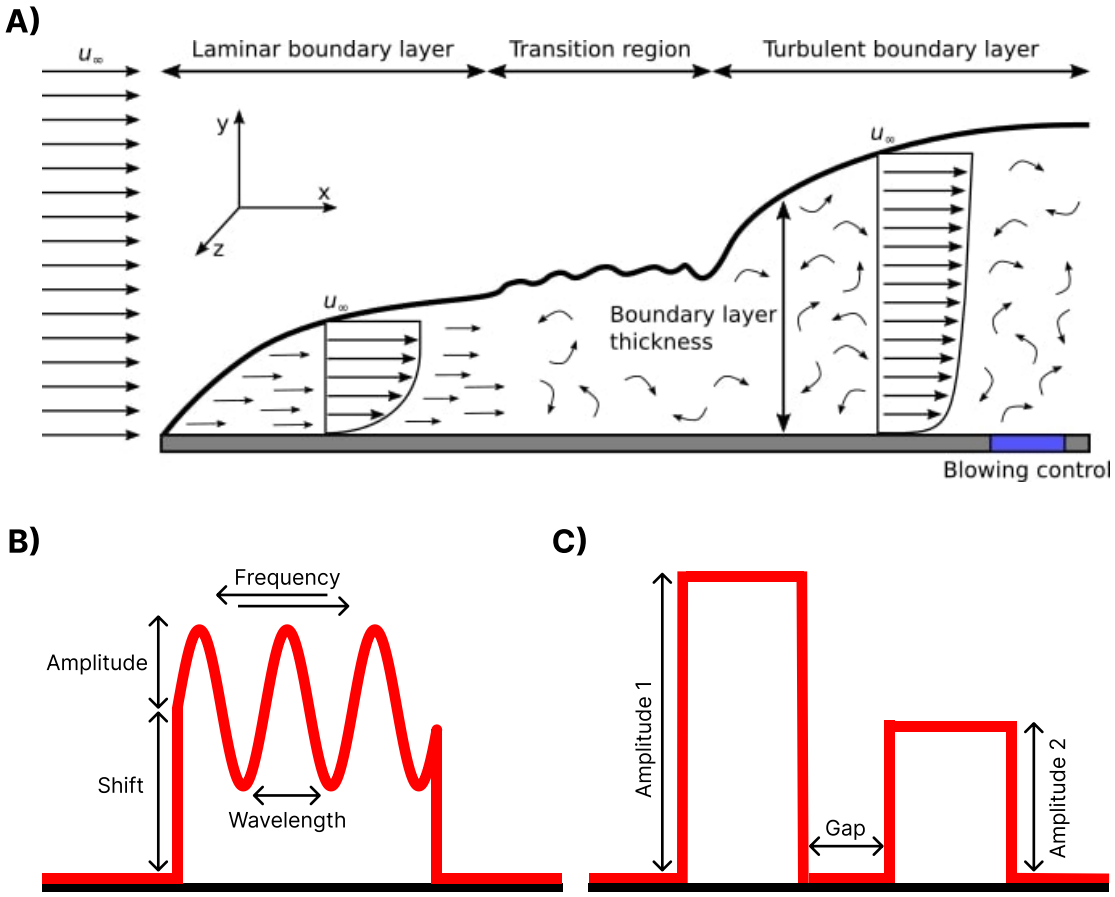

Figure 3.7: CFD simulations: **A)** illustrates the flow over a flat plate. Initially, the boundary layer is laminar. However, at a critical streamwise length from the leading edge, the flow transitions to a turbulent boundary layer, characterised by increased turbulent activity and skin-friction drag. The blue-shaded region illustrates the location of the blowing control region in the present study. **B)** shows the travelling wave blowing profile specified by an amplitude, a wavelength, a travelling frequency and a shift parameter. **C)** shows the gap blowing profile specified by two blowing areas with individual amplitudes separated by a gap.

chosen to be $n_x \times n_y \times n_z = 1537 \times 257 \times 128$, with uniform spacing in the streamwise (x-axis in Figure 3.7 **A)**) and spanwise (z-axis) directions and non-uniform spacing in the wall-normal direction (y-axis) to properly resolve the near-wall effects. This results in a mesh resolution of $\Delta x^+ = 31$, $0.54 \leq \Delta y^+ \leq 705$, and $\Delta z^+ = 15$ in viscous (inner) units, where the inner scaling is concerning the friction velocity (i.e., scaled by the wall-shear stress generated by the skin-friction drag force) at the start of the control region for the canonical case. The control region extends from $x = 68\delta_0$ to $145\delta_0$ in the streamwise direction, corresponding to a Reynolds number range of $Re_\theta \approx 479$ to 703, for the canonical case. To accelerate the transition to turbulence, a random volume forcing approach [170], located at $x = 3.5\delta_0$, is used to trip the boundary layer [41].

Table 3.7: Results for travelling wave experiment. Horizontal line separates initial training points and points proposed by Bayesian optimisation. Points with global drag reduction over 22% in bold.

Evaluation	Amplitude	Shift	Wavenumber	Frequency	GDR [in %]
1	0.50	-0.74	0.02	-0.22	-25.71
2	0.96	0.50	0.00	0.20	12.26
3	0.85	0.67	0.01	0.23	15.91
4	0.09	-0.48	0.02	-0.07	-15.63
5	0.42	-0.16	0.01	-0.02	-4.67
6	0.62	0.89	0.00	-0.05	19.69
7	0.44	-0.50	0.00	-0.22	-16.73
8	0.19	0.31	0.01	-0.19	8.33
9	0.02	-0.34	0.00	0.17	-10.94
10	0.68	-0.98	0.01	-0.12	-34.70
11	0.31	0.62	0.01	0.12	14.86
12	0.73	0.85	0.01	-0.15	19.52
13	0.34	0.16	0.02	0.08	4.28
14	0.81	0.06	0.02	0.04	1.94
15	0.90	-0.02	0.01	0.01	-1.22
16	0.24	-0.77	0.01	0.15	-26.98
17	0.84	1.00	0.01	0.04	21.43
18	**0.39**	**1.00**	**0.01**	**-0.00**	**22.07**
19	1.00	1.00	0.00	-0.25	21.77
20	1.00	1.00	0.02	0.00	21.83
21	0.48	1.00	0.02	0.13	21.90
22	**0.01**	**1.00**	**0.02**	**-0.25**	**22.06**
23	0.01	1.00	0.00	-0.25	21.88
24	**0.01**	**1.00**	**0.02**	**0.25**	**22.19**
25	**1.00**	**1.00**	**0.02**	**-0.25**	**22.03**
26	0.01	1.00	0.00	0.25	21.95
27	**0.01**	**1.00**	**0.02**	**0.01**	**22.00**
28	**0.42**	**1.00**	**0.01**	**-0.25**	**22.07**
29	**0.01**	**1.00**	**0.01**	**0.25**	**22.04**
30	**0.01**	**1.00**	**0.01**	**0.01**	**22.17**
31	0.43	0.79	0.00	-0.22	18.45
32	1.00	1.00	0.00	0.25	21.88
33	0.01	1.00	0.01	-0.25	21.93
34	0.47	1.00	0.02	-0.25	21.75
35	1.00	1.00	0.02	0.25	21.83
36	0.83	0.94	0.02	-0.02	21.02

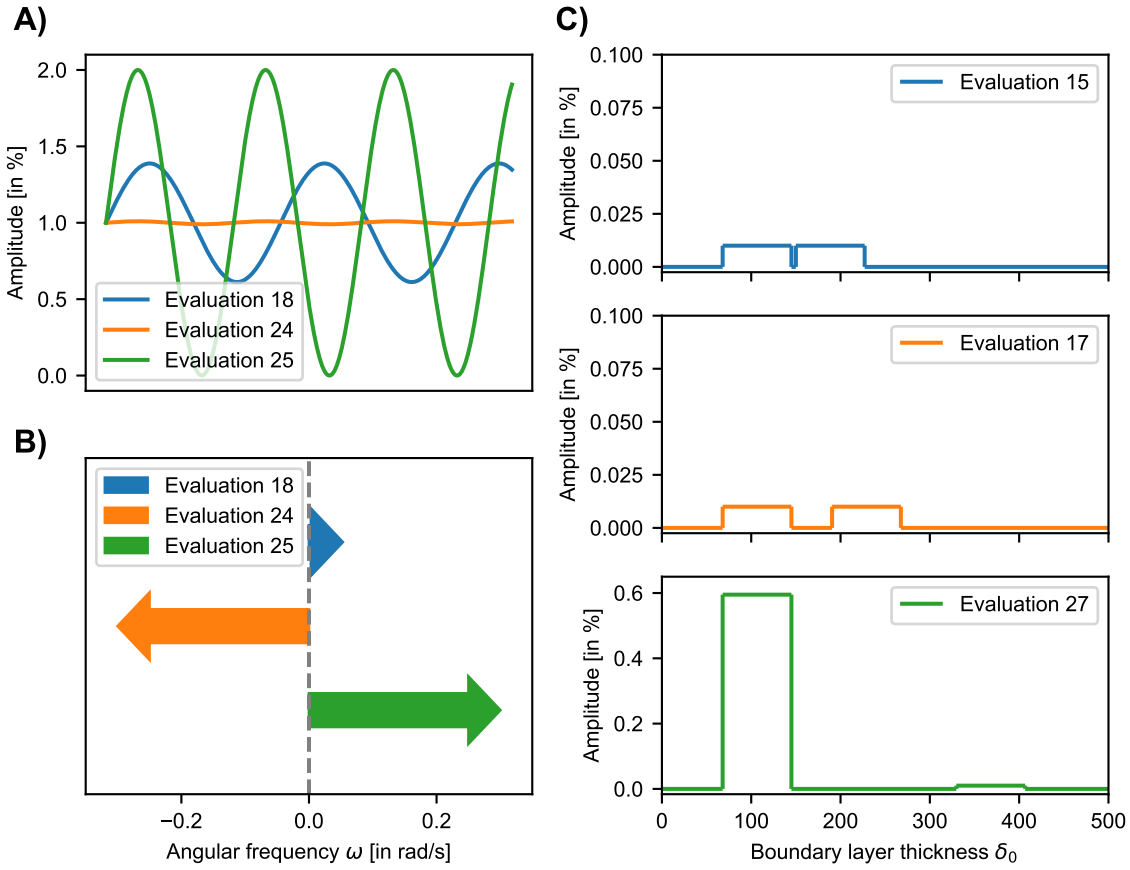

Figure 3.8: Results of CFD simulations: **A)** gives the travelling wave blowing profiles for iterations 18, 24 and 25 where the arrows in **B)** illustrate the direction and strength of travel. **C)** presents the gap blowing profiles for iterations 15, 17 and 27.

Figure 3.7 depicts the two blowing profiles in question and defines the parameters for the optimisation. The travelling wave with four degrees of freedom, given in Subfigure 3.7 **B)**, is a wave defined by an amplitude in the range 0.01 to 1.00% of the overall free-stream velocity and a wavelength between 0.00 and 0.02 (inner scaling). The angular frequency, restricted to values between -0.25 and 0.25 (inner scaling), allows the wave to travel up and downstream. Lastly, a shift parameter displaces the wave vertically up and down. This parameter is restricted to values between -1.00 and 1.00% of the free-stream velocity. The blowing turns into suction for cases where the blowing profile is negative. The gap configuration with three degrees of freedom, illustrated in Subfigure 3.7 **C)**, includes two blowing regions with individual amplitudes in the range 0.01 to 1.00% of the overall free-stream velocity and a gap restricted to between 5 δ_0 and 355 δ_0. While the aim for both problems is the maximisation of the global drag reduction (GDR), defined as the globally averaged skin-friction drag reduction with respect to the canonical case, the gap problem with three degrees of freedom also considers the energy consumption

Table 3.8: Results for gap experiment. Horizontal line separates initial training points and points proposed by Bayesian optimisation. Points with net energy savings in bold.

Evaluation	Amplitude 1	Amplitude 2	Gap	GDR [in %]	NES [in %]
1	0.97	0.55	110.35	32.86	-4.31
2	0.28	0.12	191.95	10.19	-1.31
3	0.87	0.89	225.60	34.75	-7.41
4	0.40	0.41	289.40	17.75	-3.36
5	0.13	0.69	241.42	16.14	-4.34
6	0.71	0.25	133.84	22.47	-1.66
7	0.66	0.80	352.52	26.74	-7.97
8	0.53	0.64	179.61	25.28	-3.18
9	0.83	0.03	318.82	19.83	-0.97
10	0.19	0.96	51.96	25.34	-3.53
11	0.04	0.30	92.16	8.72	-0.96
12	0.49	0.45	31.67	22.86	-0.87
13	0.27	0.01	5.00	7.72	-0.28
14	1.00	0.01	5.00	22.04	-2.84
15	**0.01**	**0.01**	**5.00**	**0.91**	**0.25**
16	0.15	0.38	5.00	13.91	-0.77
17	**0.01**	**0.01**	**45.54**	**0.98**	**0.32**
18	**0.01**	**0.01**	**355.00**	**0.72**	**0.06**
19	0.01	0.10	5.00	3.01	-0.44
20	0.47	0.01	355.00	11.84	-0.39
21	**0.01**	**0.01**	**131.32**	**0.77**	**0.11**
22	0.01	0.01	237.00	0.62	-0.04
23	0.01	0.63	5.00	15.33	-0.15
24	**0.01**	**0.01**	**77.50**	**0.80**	**0.13**
25	**0.01**	**0.01**	**5.00**	**0.91**	**0.25**
26	0.28	0.63	5.00	21.72	-1.24
27	0.60	0.01	184.71	14.83	-0.01
28	0.38	0.67	5.00	24.86	-1.31

of the actuators and tries to find profiles that reduce the drag while also achieving net energy savings (NES). NES is achieved when the energy used by the blowing device is smaller than the energy saved by the drag reduction (much of the energy expenditure in aerodynamics/hydrodynamics applications is used to overcome the skin-friction drag). In this work, NES is calculated following the approach of Mahfoze et al. [123], where the input power for the blowing device is estimated from real-world experimental data and a

relationship between input power and blowing velocity is derived (see Mahfoze et al. [123] for more details).

The results of the synthetic test functions from Section 3.2.1 inform the Bayesian optimisation algorithm used for both problems. For the surrogate model, a Gaussian process with a zero mean function and a Matérn kernel with $\nu = 5/2$ was defined, and its hyperparameters were estimated from the training data using maximum likelihood estimation, as discussed in Section 2.3. While CFD simulations are expensive, they allow for parallel evaluations. This is done by concurrently running multiple simulations and combining the results once all simulations are completed. Therefore, these setups are well suited for the multi-point approach presented in Section 3.2 that, as our investigations showed, has no clear disadvantage compared to the single-point approach. Based on previous work, for example Mahfoze et al. [123], the possibility that the underlying objective function is characterised by large flat areas similar to the 6D Ackley function cannot be ruled out. Thus, an acquisition function should be implemented that allows the selection of batches and performs well even when encountering flat areas. The sequential Monte Carlo upper confidence bound acquisition function with the trade-off hyperparameter β=1 yielded very good results for the Ackley function and all other test functions and is thus chosen with a batch size of four to optimise the CFD problems in this section. Section 3.2.1 showed that Bayesian optimisation can find reasonable solutions even with a relatively small number of initial training data points. Hence, for both problems, four points per input dimension were randomly selected from a maximin Latin hypercube [129, 86].

Table 3.7 presents the results of the 16 training points plus 20 points (or five batches of four points) selected using Bayesian optimisation by maximising the GDR for the travelling wave problem. While the highest GDR of the initial training data was 19.69%, Bayesian optimisation improves upon this value with each evaluated batch, finding multiple strategies that achieve a GDR above 22%, with the best strategy from batch 2, giving a GDR of 22.19%. Three blowing profiles, including the overall best solution found, are depicted in Figure 3.8 **A)** and **B)**. Overall, the shift parameter seems to be the main driver behind the drag reduction, as almost all strategies selected by Bayesian optimisation implement the upper limit of this parameter, independent of the other parameter values.

While blowing at a high amplitude yields increased skin-friction drag reduction, it also consumes more energy. The second experiment addresses this point and accounts for the energy consumption, following Mahfoze et al. [123], when optimising the blowing profile. Table 3.8 provides the results for 12 initial training points and four batches, proposed via Bayesian optimisation, by maximising the NES. The initial strategies selected by the Latin Hypercube did not find a solution that achieved both GDR and NES. Bayesian optimisation found multiple strategies that achieved both, in which the NES and the GDR were relatively small (0.11 to 0.32% and 0.77 to 0.98%, respectively). However, the algorithm

also found one strategy with NES of -0.01% and a GDR of 14.83%. While this strategy did not achieve NES, it did not increase overall energy use, and a small increase in the efficiency of the actuators could yield NES with a considerable GDR. Compared to the high-intensity blowing for the travelling wave, the amplitudes are clustered towards the lower end of the parameter space in this experiment. This results from the objective of optimising NES, which penalises high-velocity blowing due to its increased power requirements. Figure 3.8 **C)** illustrates this by providing the blowing profiles of three solutions: the two solutions with the most significant net energy savings and the solution with a high GDR, as previously described.

3.4 Conclusion

In this chapter, Bayesian optimisation algorithms, implemented with different types of acquisition functions, were benchmarked regarding their performance and their robustness on synthetic test functions inspired by applications in engineering and machine learning. Synthetic test functions have the advantage that their shape and their global optima are known. This allows the algorithms to be evaluated on (a) how close their best solutions are to the global optimum and (b) how well they perform on specific challenges, such as oscillating functions or functions with steep edges. This evaluation can indicate the advantages and shortcomings of the individual acquisition functions and inform researchers of the best approach for their specific problem.

This chapter conducted four sets of comparisons. First, analytical single-point acquisition functions were compared to each other. Second, the effect of varying the number of initial training data points was investigated. Third, the analytical approach was contrasted with acquisition functions based on Monte Carlo sampling. Fourth, the single-point approach was compared to the multi-point or batched approach.

Six main conclusions could be drawn from these experiments: (i) While all acquisition functions performed well on simple test functions, optimistic policies, such as the upper confidence bound, dealt best with challenging problems. (ii) Varying the number of initial training data points did not significantly affect the performance of the individual methods. (iii) Improvement-based acquisition functions struggled with flat test functions. (iv) Monte Carlo and multi-point acquisition functions present a good alternative to the widely used analytical single-point methods. (v) The multi-point approach is particularly advantageous when the objective function takes a long time to evaluate and allows parallel evaluations. (vi) Bayesian optimisation performs equally well on noisy objective functions up to the moderate levels of noise investigated.

Finally, two experiments in computational fluid dynamics were taken as illustrative examples of how the findings of this chapter can be used to guide the design of a Bayesian

optimisation algorithm and tailor it to unique problems. In detail, a multi-point approach was employed that used the Monte Carlo upper confidence bound acquisition function, allowing multiple points to be evaluated in parallel with concurrent simulations. For both experiments, Bayesian optimisation improved upon the training points immediately and found solutions that result in global drag reduction for the travelling wave and global drag reduction as well as net energy savings for the experiment in which a gap separated two blowing areas. However, the effects of the second experiment remained relatively small. These optimisation studies are promising in designing a robust and efficient control strategy to reduce drag around moving vehicles.

Chapter 4

NUBO: A transparent software package for Bayesian optimisation

Summary

NUBO, short for Newcastle University Bayesian Optimisation, is a Bayesian optimisation framework for optimising expensive-to-evaluate black-box functions, such as physical experiments and computer simulators. Bayesian optimisation is a cost-efficient optimisation strategy that uses surrogate modelling via Gaussian processes to represent an objective function and acquisition functions to guide the selection of candidate points to approximate the global optimum of the objective function. NUBO focuses on transparency and user experience to make Bayesian optimisation accessible to researchers from all disciplines. Clean and understandable code, precise references, and thorough documentation ensure transparency, while a modular and flexible design, easy-to-write syntax, and careful selection of Bayesian optimisation algorithms ensure a good user experience. NUBO allows users to tailor Bayesian optimisation to their specific problem by writing a custom optimisation loop using the provided building blocks. It supports sequential single-point, parallel multi-point, and asynchronous optimisation of bounded, constrained, and mixed (discrete and continuous) parameter input spaces. Only algorithms and methods that are extensively tested and validated to perform well—in the literature and in Section 3—are included in NUBO. This ensures that the package remains compact and does not overwhelm the user with an unnecessarily large number of options. The package is written in Python but does not require expert knowledge of Python to optimise simulators and experiments. NUBO is distributed as open-source software under the BSD 3-Clause licence.

4.1 Introduction

Bayesian optimisation can be used to optimise expensive black-box functions, such as physical experiments and computer simulators, in many different fields, as discussed in Chapters 1 and 2 of this thesis. The setup of such problems is typically unique and requires a software implementation that is easily adjusted and tailored to the specific problem. However, most available software implementations either lack the necessary functionality to optimise non-standard problems or are overly complex and challenging to use—particularly for scientists who are not confident in coding. With NUBO, we have developed an open-source implementation of Bayesian optimisation aimed at researchers with expertise in disciplines other than statistics and computer science. To ensure that our target audience can understand and use Bayesian optimisation to its full potential, NUBO focuses particularly on (a) transparency through clean and understandable code, precise references, and thorough documentation and (b) user experience through a modular and flexible design, easy syntax, and a careful selection of implemented algorithms.
The remainder of this chapter is structured as follows. In Section 4.2 we discuss the implementation of Bayesian optimisation in NUBO. Section 4.3 illustrates how NUBO can be used to optimise expensive black-box functions through a non-trivial case study. Section 4.4 compares NUBO and its performance to other implementations of Bayesian optimisation. Finally, we conclude and give an outlook on future work in Section 4.5.

4.2 NUBO

NUBO is a Bayesian optimisation package written in Python that focuses on transparency and user experience to make Bayesian optimisation accessible to researchers from a wide range of disciplines whose expertise is not necessarily statistics or computer science. With this overall goal in mind, NUBO ensures transparency by implementing clean and comprehensible code, precise references and thorough documentation within Diessner, Wilson, and Whalley [42] and on our website at `www.nubopy.com`. The latter includes instructions for installing NUBO, a primer on Bayesian optimisation in general, all source code, off-the-shelf algorithms, examples of boilerplate code, guidance on creating custom optimisation loops, and documentation for all functions and classes included in NUBO. The former is our reference paper on which this chapter is based. We avoid implementing overly complex and convoluted functions and objects that require the retracing of individual elements through multiple files to be fully understood. We prioritise user experience defined by a modular and flexible design that can be intuitively tailored to unique problems, easy-to-read and -write syntax, and a careful selection of Bayesian optimisation algorithms. The latter is essential as we try not to overwhelm the user with a larger number of op-

tions, but instead focus on what is essential to optimise computer simulators and physical experiments successfully.

To create a powerful package with good longevity, starting with a strong foundation is important. NUBO is built upon the Torch[1] ecosystem [151] that provides a strong scientific computation framework for working with tensors, a selection of powerful optimisation algorithms, such as `torch.Adam` [104], automatic differentiation capabilities to compute gradients of acquisition functions via `torch.autograd`, and GPU acceleration. Furthermore, GPyTorch[2] [56], the package we use to implement Gaussian processes for our surrogate modelling, is also based on Torch and combines seamlessly with NUBO. We borrow the L-BFGS-B [222] and SLSQP [107] optimisation algorithms from SciPy[3] [201] for the deterministic optimisation of the acquisition functions and use NumPy[4] [76] to make data suitable for these optimisers.

NUBO and all its required dependencies can be installed from the Python Package Index (PyPI) [159] with the package installer pip [194] via the terminal:

```
pip install nubopy
```

4.2.1 Gaussian processes

NUBO uses the GPyTorch [56] package to implement Gaussian processes for surrogate modelling. While GPyTorch allows the definition of many different Gaussian processes through its various mean functions, covariance kernels, and methods for hyper-parameter estimation, we provide a predefined Gaussian process in the `nubo.models` module that follows the work of Snoek, Larochelle, and Adams [178]. The `GaussianProcess` is specified by a constant mean function, and the Matérn 5/2 ARD kernel that, due to its flexibility, is well suited for practical optimisation as it can represent a wide variety of real-world objective functions [178, 141]. The code below implements a Gaussian process and estimates its hyper-parameters from some training inputs `x_train` and training outputs `y_train` by maximising the log-marginal likelihood in Equation 2.20 with the `fit_gp` function. The hyper-parameters include the constant in the mean function, the output scale and length scales in the covariance kernel, and the noise in the Gaussian likelihood. The training inputs and training outputs are specified as a `torch.Tensor` of size $n \times d$ and length n, respectively, where n is the number of points, and d is the number of input dimensions.

[1] See `https://pytorch.org/` for documentation and `https://pypi.org/project/torch/` for package information on PyPI.

[2] See `https://gpytorch.ai/` for documentation and `https://pypi.org/project/gpytorch/` for package information on PyPI.

[3] See `https://scipy.org/` for documentation and `https://pypi.org/project/scipy/` for package information on PyPI.

[4] See `https://numpy.org/` for documentation and `https://pypi.org/project/numpy/` for package information on PyPI.

Calling the function `fit_gp` results in a trained Gaussian process that can subsequently be used for Bayesian optimisation.

```
>>> from nubo.models import GaussianProcess, fit_gp
>>> from gpytorch.likelihoods import GaussianLikelihood
>>>
>>>
>>> likelihood = GaussianLikelihood()
>>> gp = GaussianProcess(x_train, y_train, likelihood=likelihood)
>>> fit_gp(x_train, y_train, gp=gp, likelihood=likelihood)
```

While Gaussian processes can estimate noise, for example observational noise occurring when taking measurements, we might prefer to specify the noise explicitly if it is known. We can exchange the `GaussianLikelihood` for the `FixedNoiseGaussianLikelihood` and specify the noise for each training point. The `FixedNoiseGaussianLikelihood` allows us to decide if any additional noise should be estimated by setting the `learn_additional_noise` attribute to `True` or `False`. The snippet below fixes the variance of the observational noise of each training point at 0.025 and estimates any additional noise.

```
>>> from nubo.models import GaussianProcess, fit_gp
>>> from gpytorch.likelihoods import FixedNoiseGaussianLikelihood
>>>
>>>
>>> noise = torch.ones(x_train.size(0)) * 0.025
>>> likelihood = FixedNoiseGaussianLikelihood(noise=noise,
...                                           learn_additional_noise=True)
>>> gp = GaussianProcess(x_train, y_train, likelihood=likelihood)
>>> fit_gp(x_train, y_train, gp=gp, likelihood=likelihood)
```

4.2.2 Bayesian optimisation

Before describing the individual optimisation options in detail, we want to illustrate NUBO's user experience—its easy-to-read and -write syntax, flexibility, and modularity—for a simple Bayesian optimisation step that can be further divided into four substeps. First, we define the input space. Here, we want to optimise a six-dimensional objective function that is bounded by the hyper-rectangle $[0, 1]^6$ specified as `bounds`, a 2×6 `torch.Tensor`, where the first row provides the lower bounds and the second row the upper bounds for all six input dimensions. Second, we load the training inputs `x_train` and the training outputs `y_train`. This training data can be manually selected or generated using a space-filling design, such as Latin hypercube sampling introduced in Section 4.2.3. Third, we define and train the Gaussian process implemented in NUBO as discussed in

Section 4.2.1, or set up a custom Gaussian process with GPyTorch. Fourth, we specify an acquisition function that takes the fitted Gaussian process as an argument and chooses an optimisation method. In this case, we use the upper confidence bound introduced in Equation 2.25 and optimise it with the L-BFGS-B algorithm [222] using the `single` function.

```
>>> import torch
>>> from nubo.acquisition import UpperConfidenceBound
>>> from nubo.models import GaussianProcess, fit_gp
>>> from nubo.optimisation import single
>>> from gpytorch.likelihoods import GaussianLikelihood
>>>
>>>
>>> bounds = torch.tensor([[0., 0., 0., 0., 0., 0.],
...                        [1., 1., 1., 1., 1., 1.]])
>>>
>>> x_train = # load inputs as torch.Tensor
>>> y_train = # load outputs as torch.Tensor
>>>
>>> likelihood = GaussianLikelihood()
>>> gp = GaussianProcess(x_train, y_train, likelihood=likelihood)
>>> fit_gp(x_train, y_train, gp=gp, likelihood=likelihood)
>>>
>>> acq = UpperConfidenceBound(gp=gp, beta=4)
>>> x_new, _ = single(func=acq, method="L-BFGS-B", bounds=bounds)
```

NUBO is very flexible and allows users to swap out individual elements for other options. For example, we can substitute the `UpperConfidenceBound` acquisition function or the `single` optimisation strategy without changing any of the other lines of code. This makes it easy and fast to tailor Bayesian optimisation to specific problems.
The remainder of this section introduces NUBO's optimisation strategies. Figure 4.1 shows a flowchart that helps users decide on the correct acquisition function and optimiser for their specific problem.

Sequential single-point optimisation

In NUBO, we differentiate between two optimisation strategies: single-point and multi-point optimisation. When using the single-point strategy via the `single` function, NUBO uses the analytical acquisition functions discussed in Section 2.4 to find the next point to be evaluated by the objective function. The corresponding observation must be added to

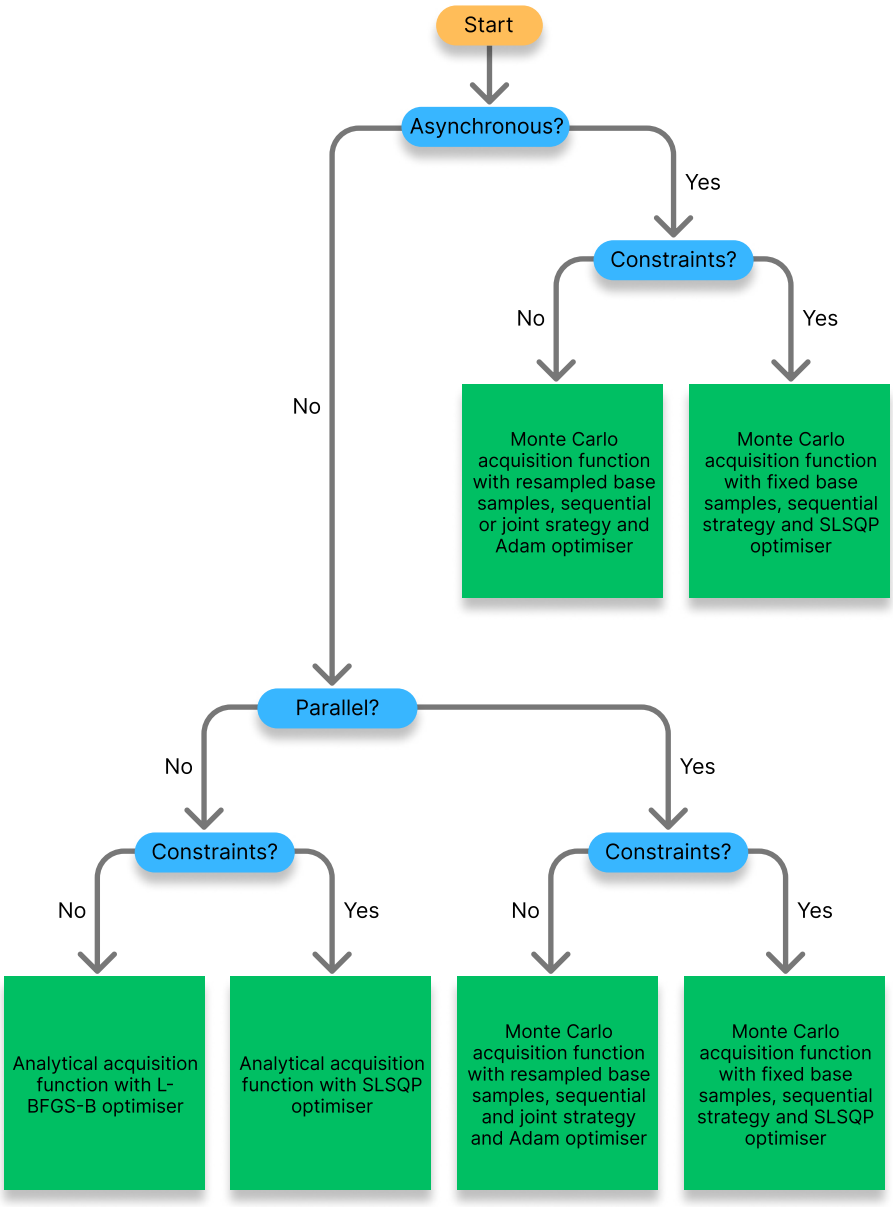

Figure 4.1: NUBO flowchart. Overview of the recommended algorithms for specific problems. Start in yellow, decisions in blue, and recommended algorithm in green.

the dataset before the next iteration of the optimisation loop can begin.

The code below shows how the analytical expected improvement (EI) and the analytical upper confidence bound (UCB) can be specified within NUBO. The former takes the best training output to date as the argument `y_best`, while the latter accepts the trade-off hyper-parameter β as the `beta` argument. For bounded optimisation problems with analytical acquisition functions, the optimisation method of the `single` function should be set to `method="L-BFGS-B"` and the arguments `num_starts` (default 10) and `num_samples`

(default 100) can be set to enable multi-start optimisation, where the selected optimisation algorithm is run multiple times. Each start is initialised at the best points from a large number of points sampled from a Latin hypercube. This reduces the risk of the optimiser getting stuck in a local optimum. Section 4.2.3 introduces Latin hypercube sampling in more detail. The `single` function returns the best start and its acquisition value. Sequential single-point optimisation can be paired with constrained and mixed optimisation, as detailed later in this section.

```
>>> from nubo.acquisition import ExpectedImprovement, UpperConfidenceBound
>>> from nubo.optimisation import single
>>>
>>>
>>> acq = ExpectedImprovement(gp=gp, y_best=torch.max(y_train))
>>> acq = UpperConfidenceBound(gp=gp, beta=4)
>>> x_new, _ = single(func=acq, method="L-BFGS-B", bounds=bounds,
...                   num_starts=5, num_samples=50)
```

Parallel multi-point optimisation

The second optimisation strategy available in NUBO is multi-point optimisation. This strategy uses the Monte Carlo acquisition functions outlined in Section 2.4.5 to find multiple points, also called batches, in each iteration of the Bayesian optimisation loop. This strategy is particularly beneficial for objective functions supporting parallel evaluations, as points can be queried simultaneously, speeding up optimisation.
NUBO uses the Monte Carlo versions of expected improvement `MCExpectedImprovement` and upper confidence bound `MCUpperConfidenceBound` in conjunction with either the `multi_joint` or `multi_sequential` function to compute batches. The two different options for the multi-point optimisation strategy are discussed in Section 2.4.5. In short, in addition to the arguments of the analytical acquisition functions, both Monte Carlo acquisition functions accept the number of Monte Carlo samples to be used to approximate the acquisition function as the `samples` argument (default 512). For the optimisation functions, the number of points to be computed can be passed to the `batch_size` argument, while the `method` should be set to `"Adam"` to enable stochastic optimisation via the Adam algorithm [104]. The Adam algorithm can be fine-tuned by setting the learning rate `lr` (default 0.1) and the number of optimisation steps `steps` (default 100). Parallel multi-point optimisation can be paired with asynchronous, constrained, and mixed optimisation, as detailed later in this section.

```
>>> from nubo.acquisition import MCExpectedImprovement,
...                              MCUpperConfidenceBound
```

```
>>> from nubo.optimisation import multi_joint, multi_sequential
>>>
>>>
>>> acq = MCExpectedImprovement(gp=gp, y_best=torch.max(y_train),
...                             samples=256)
>>> acq = MCUpperConfidenceBound(gp=gp, beta=4, samples=256)
>>> x_new, _ = multi_joint(func=acq, method="Adam", lr=0.1,
...                        steps=100, batch_size=4, bounds=bounds)
>>> x_new, _ = multi_sequential(func=acq, method="Adam", lr=0.1,
...                             steps=100, batch_size=4, bounds=bounds)
```

To enable the use of deterministic optimisers, such as L-BFGS-B [222] and SLSQP [107], the base samples from a standard normal distribution $\mathcal{N}(0, 1)$ used to compute the Monte Carlo samples as described in Section 2.4.5 can be fixed by setting `fix_base_samples=True` (default `False`).

```
>>> from nubo.acquisition import MCUpperConfidenceBound
>>> from nubo.optimisation import multi_joint, multi_sequential
>>>
>>>
>>> acq = MCUpperConfidenceBound(gp=gp, beta=4, fix_base_samples=True)
>>> x_new, _ = multi_joint(func=acq, method="L-BFGS-B",
...                        batch_size=4, bounds=bounds)
>>>
>>> acq = MCUpperConfidenceBound(gp=gp, beta=4, fix_base_samples=True)
>>> x_new, _ = multi_sequential(func=acq, method="L-BFGS-B",
...                             batch_size=4, bounds=bounds)
```

Asynchronous optimisation

NUBO supports asynchronous optimisation, that is, the continuation of the optimisation loop, while some points are being evaluated by the objective function. In this case, the Monte Carlo acquisition functions `MCExpectedImprovement` or `MCUpperConfidenceBound` are used. Similar to the sequential multi-point approach, pending points are added to the test points, and Monte Carlo samples are taken from their joint multivariate Gaussian distribution [178]. The code snippet below assumes that the two points `x_pend` are currently being evaluated. To continue the optimisation, these points can be fed into the acquisition function by setting `x_pending=x_pend`, and NUBO will take them into account for the subsequent iteration.

```
>>> import torch
>>> from nubo.acquisition import MCUpperConfidenceBound
>>> from nubo.optimisation import multi_joint, multi_sequential
>>>
>>>
>>> x_pend = torch.tensor([[0.2, 0.9, 0.8, 0.4, 0.4, 0.1],
...                        [0.1, 0.3, 0.7, 0.2, 0.1, 0.2]])
>>> acq = MCUpperConfidenceBound(gp=gp, beta=4, x_pending=x_pend)
>>> x_new, _ = multi_joint(func=acq, method="Adam",
...                        batch_size=4, bounds=bounds)
>>> x_new, _ = multi_sequential(func=acq, method="Adam",
...                             batch_size=4, bounds=bounds)
```

While Monte Carlo acquisition functions are approximations of the analytical functions, they are used for computing multiple points, where analytical functions are generally intractable. The Monte Carlo approach can also be used for single-point asynchronous optimisation by setting `batch_size=1`.

Constrained optimisation

The simplest case of the maximisation problem in Equation 2.1 can be extended by including one or more input constraints

$$\begin{aligned} & \boldsymbol{x}^* = \underset{\boldsymbol{x}\in\mathcal{X}}{\operatorname{argmax}}\, f(\boldsymbol{x}), & & \\ \text{subject to} \quad & g_i(\boldsymbol{x}) = 0 & \forall i = 1,\ldots,I \quad & \text{[Equality constraint]} \\ & h_j(\boldsymbol{x}) \geq 0 & \forall j = 1,\ldots,J \quad & \text{[Inequality constraint].} \end{aligned} \tag{4.1}$$

In these instances, NUBO allows constrained Bayesian optimisation by using the SLSQP algorithm to optimise the acquisition function. Implementing this method requires the additional step of specifying the constraints `cons` as a dictionary—an ordered collection of key-value pairs within curly brackets—for one constraint or a list of dictionaries for multiple constraints. Each constraint requires two entries. The first is `"type"` and can either be set to `"ineq"` for inequality constraints or `"eq"` for equality constraints. The second is `"fun"`, which takes a function representing the constraint. The optimiser selects points for which the constraint functions are greater than or equal to zero for inequality constraints and exactly zero for equality constraints. The code snippet below specifies two constraints: the first is an inequality constraint that requires the sum of the first two input dimensions to be smaller than or equal to 0.5. The second is an equality constraint that requires dimensions four, five, and six to sum to 1.2442.

```
>>> import torch
>>>
>>>
>>> bounds = torch.tensor([[0., 0., 0., 0., 0., 0.],
...                        [1., 1., 1., 1., 1., 1.]])
>>> cons = [{"type": "ineq", "fun": lambda x: 0.5 - x[0] - x[1]},
...         {"type": "eq", "fun": lambda x: 1.2442 - x[3] - x[4] - x[5]}]
```

After setting up the input space using the bounds and constraints, the Bayesian optimisation loop is similar to before. We need to set the `method` argument of the optimisation function to `"SLSQP"` and provide the function with the constraints `cons`.

```
>>> from nubo.acquisition import UpperConfidenceBound
>>> from nubo.optimisation import single
>>>
>>>
>>> acq = UpperConfidenceBound(gp=gp, beta=4)
>>> x_new, _ = single(func=acq, method="SLSQP",
...                   bounds=bounds, constraints=cons)
```

Constrained Bayesian optimisation can be used with analytical and Monte Carlo acquisition functions, as well as single-point, multi-point, asynchronous, and mixed optimisation.

Mixed optimisation

Bayesian optimisation predominantly focuses on problems with continuous input parameters since Gaussian processes model all input dimensions as continuous variables. However, NUBO supports optimising mixed input parameter spaces via a workaround. To do this, NUBO first computes all possible combinations of the discrete parameters. Then, it maximises the acquisition function for all continuous parameters while holding one combination of the discrete parameters fixed. Once the acquisition function is maximised for each possible combination of discrete parameters, the best overall solution is returned. This can be very time-consuming for many discrete dimensions or discrete values. In the case of a discrete variable with a very large number of possible values, an alternative is to treat it as a continuous variable and round proposed values to the nearest discrete value. To implement mixed optimisation in NUBO, bounds are specified as before, but the discrete dimensions are additionally defined in a dictionary where the keys are the dimensions (starting from zero), and the values are a list of all possible values for the discrete inputs. The code below specifies dimensions one and five as `disc`.

```
>>> import torch
>>>
```

```
>>>
>>> bounds = torch.tensor([[0., 0., 0., 0., 0., 0.],
...                        [1., 1., 1., 1., 1., 1.]])
>>> disc = {0: [0.2, 0.4, 0.6, 0.8],
...         4: [0.3, 0.6, 0.9]}
```

After setting up the input space specified by the bounds and discrete values, the Bayesian optimisation loop is similar to before. We only need to provide the function with the dictionary specifying the discrete dimensions `discrete=disc`.

```
>>> from nubo.acquisition import UpperConfidenceBound
>>> from nubo.optimisation import single
>>>
>>>
>>> acq = UpperConfidenceBound(gp=gp, beta=4)
>>> x_new, _ = single(func=acq, method="L-BFGS-B",
...                   bounds=bounds, discrete=disc)
```

Mixed Bayesian optimisation can be used in conjunction with analytical and Monte Carlo acquisition functions as well as single-point, multi-point, asynchronous, and constrained optimisation.

4.2.3 Test functions and utilities

NUBO provides a selection of test functions and utilities—functions automising frequently repeating tasks—to make implementing and testing Bayesian optimisation algorithms more convenient. The ten test functions were selected from the virtual library of Surjanovic and Bingham [188] and represent a variety of challenges, such as bowl-shaped, plate-shaped, valley-shaped, uni-modal, and multi-modal functions. The functions can be imported from the `nubo.test_functions` module and instantiated by providing the number of dimensions (except for the Hartmann function that comes in 3D and 6D versions), the standard deviation of any noise that should be added, and whether the function should be minimised or maximised. These functions are equipped with the following attributes: the number of dimensions `dims`, the bounds `bounds`, and the inputs and outputs of the global optimum `optimum`.

```
>>> from nubo.test_functions import Ackley, Hartmann6D
>>>
>>>
>>> func = Ackley(dims=5, noise_std=0.1, minimise=False)
>>> func = Hartmann6D(minimise=False)
```

```
>>> dims = func.dims
>>> bounds = func.bounds
```

The `gen_inputs` function from the `nubo.utils` module allows efficient generation of input data that covers the input space by sampling a larger number of random Latin hypercube designs [129, 86] and returning the design with the largest minimal distance between all points. Figure 4.2 compares Latin hypercube sampling to random sampling for two input dimensions. While many random points are near each other, points from the Latin hypercube design effectively cover the whole input space by placing only one point in each row and column. The exact position of the point within the selected square is random. Latin hypercubes were discussed in more detail in Section 2.2.2. The code snippet below generates five points for each input dimension of the Hartmann function initiated above and uses `func` to compute the corresponding outputs.

```
>>> from nubo.utils import gen_inputs
>>>
>>>
>>> x_train = gen_inputs(num_points=dims * 5,
...                      num_dims=dims,
...                      bounds=bounds)
>>> y_train = func(x_train)
```

Finally, we discuss three convenience functions that can be used for data transformation. `normalise` and `unnormalise` can be used to scale input data to the unit cube $[0, 1]^d$ and back to its original domain by providing the bounds of the input space. Furthermore, the outputs can be centred at zero with a standard deviation of one with the `standardise` function.

```
>>> from nubo.utils import standardise, normalise, unnormalise
>>>
>>>
>>> x_norm = normalise(x_train, bounds=bounds)
>>> x_train = unnormalise(x_norm, bounds=bounds)
>>> y_stand = standardise(y_train)
```

4.3 Case study

We present the general workflow for optimising an expensive-to-evaluate black-box function with NUBO by providing a detailed case study in which a test function with six input dimensions is optimised. This case study demonstrates how the user can specify

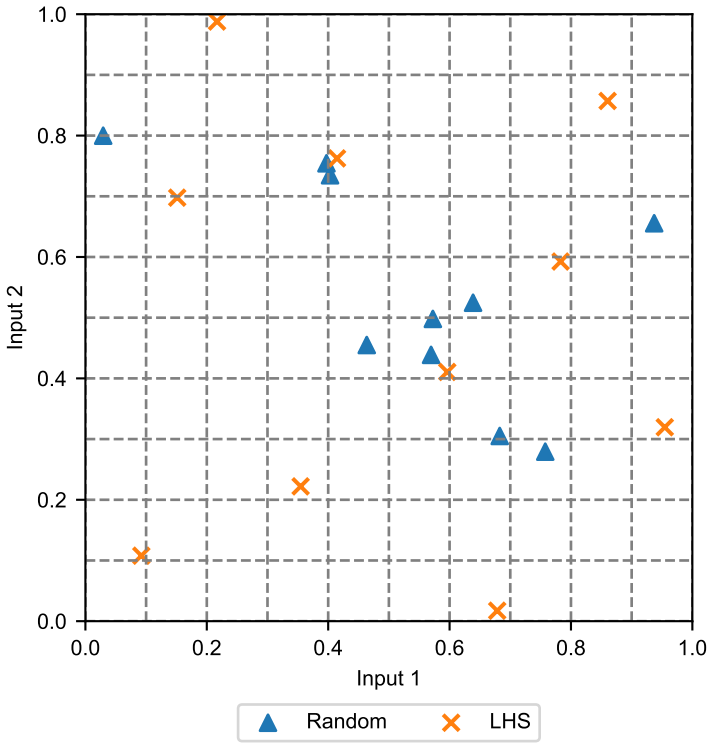

Figure 4.2: Latin hypercube sampling compared to random sampling.

the parameter input space, generate initial training data, and define and run the Bayesian optimisation loop.

To ensure reproducibility of the case study, we set the seed for the pseudo-number generator within Torch to 123. We set some format options for the `print` function such that values are rounded to the fourth decimal place and are not formatted in scientific notation to increase readability.

```
>>> import torch
>>>
>>>
>>> torch.manual_seed(123)
>>> torch.set_printoptions(precision=4, sci_mode=False)
```

A typical objective function optimised with Bayesian optimisation is expensive to evaluate and thus not feasible in a case study that aims to illustrate how NUBO can be applied. Hence, we will use one of the synthetic test functions provided by NUBO as a surrogate expensive-to-evaluate black-box function. We use the six-dimensional Hartmann function that possesses multiple local and one global minimum. Its input space is bounded by the hyper-rectangle $[0, 1]^6$. Observational noise, such as measurement error, is represented

by adding a small amount of random Gaussian noise to the function output by setting `noise_std=0.1`. `minimise` is set to `False` to transform the minimisation problem into a maximisation problem as required for Bayesian optimisation with NUBO.

```
>>> from nubo.test_functions import Hartmann6D
>>>
>>>
>>> black_box = Hartmann6D(noise_std=0.1, minimise=False)
```

With our objective function specified, we can focus on defining the input space. Our objective function has six inputs, all bounded by $[0, 1]$. As introduced in Section 4.2.2, the bounds are defined as a $2 \times d$ `torch.tensor`, where the first row specifies the lower bounds and the second row specifies the upper bounds. This case study also highlights the mixed parameter optimisation capabilities of NUBO (see Section 4.2.2) by assuming that the first input is a discrete parameter restricted to 0.2, 0.4, 0.6, and 0.8. We can implement this by specifying a dictionary, where the key is the input dimension and the value is a list of all possible values the input can take, that is `{0: [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]}`. Note that indexing starts at zero in Python.

```
>>> dims = 6
>>> bounds = torch.tensor([[0., 0., 0., 0., 0., 0.],
...                        [1., 1., 1., 1., 1., 1.]])
>>> discrete = {0: [0.0, 0.1, 0.2, 0.3, 0.4, 0.5,
...                 0.6, 0.7, 0.8, 0.9, 1.0]}
```

The Bayesian optimisation loop requires initial training data. This is important to train the Gaussian process that emulates the objective function. This case study uses the `gen_inputs` function introduced in Section 4.2.3 to generate 30 initial data points from a Latin hypercube design. We round the first input dimension to fit the discrete values specified above as Latin hypercube designs return continuous values. These points are evaluated by the objective function to produce our training data pairs consisting of input parameters `x_train` and observations `y_train`.

```
>>> from nubo.utils import gen_inputs
>>>
>>>
>>> x_train = gen_inputs(num_points=dims * 5,
...                      num_dims=dims,
...                      bounds=bounds)
>>> x_train[:, 0] = torch.round(x_train[:, 0], decimals=1)
>>> y_train = black_box(x_train)
```

Next, we specify the Bayesian optimisation algorithm we plan to use in our optimisation loop. We define the `bo` function, that takes our training pairs `(x_train, y_train)` and returns the next candidate point `x_new` which is evaluated by the objective function, in four steps. First, we set up our surrogate model as the Gaussian process provided by NUBO with a Gaussian likelihood as discussed in Section 4.2.1. Second, we train the Gaussian process `gp` with our training data by maximising the likelihood with the Adam algorithm [104] via the `fit_gp` function. Here, we set a custom learning rate `lr` and the number of optimisation steps `steps`. Third, we define an acquisition function that will guide our optimisation. Assuming that our objective function allows parallel function evaluations, we compute multi-point batches at each iteration and choose a Monte Carlo acquisition function, in this case `MCUpperConfidenceBound`. The acquisition function `acq` is instantiated by providing it with the fitted Gaussian process `gp`, a value for the trade-off hyper-parameter `beta`, and the number of Monte Carlo samples used to approximate the acquisition function. For further details on Monte Carlo acquisition functions, refer to Section 2.4.5. The exact choice of these parameters depends on the objective function in question, however, general guidance can be found in Chapter 3 and Diessner et al. [41], where we investigate fundamental properties of Bayesian optimisation, including parameter choices. Fourth, we maximise the acquisition function `acq` with the `multi_sequential` function that uses the sequential strategy for computing multiple candidate points. We compute four candidate points at each iteration by setting `batch_size=4` and providing the previously specified bounds and discrete values. The Adam optimiser is used as Monte Carlo acquisition functions require a stochastic optimiser due to their inherent randomness introduced by drawing the Monte Carlo samples. The optimiser is initialised at two different initial points chosen as the two points with the highest acquisition value out of 100 potential points sampled from a Latin hypercube design. We chose two initialisations to keep the computational overhead within the replication script low. In practice, a higher number of initialisations might be beneficial.

```
>>> from nubo.acquisition import MCUpperConfidenceBound
>>> from nubo.models import GaussianProcess, fit_gp
>>> from nubo.optimisation import multi_sequentia
>>> from gpytorch.likelihoods import GaussianLikelihood
>>>
>>>
>>> def bo(x_train, y_train):
>>>
>>>     likelihood = GaussianLikelihood()
>>>     gp = GaussianProcess(x_train, y_train, likelihood=likelihood)
>>>
```

```
>>>     fit_gp(x_train, y_train, gp=gp, likelihood=likelihood,
...            lr=0.1, steps=200)
>>>
>>>     acq = MCUpperConfidenceBound(gp=gp, beta=4, samples=128)
>>>
>>>     x_new, _ = multi_sequential(func=acq,
...                                 method="Adam",
...                                 batch_size=4,
...                                 bounds=bounds,
...                                 discrete=discrete,
...                                 lr=0.1,
...                                 steps=200,
...                                 num_starts=2,
...                                 num_samples=100)
>>>
>>>     return x_new
```

Finally, we specify the entire optimisation loop, that is, a simple `for`-loop that computes the next batch of candidate points using the defined Bayesian optimisation algorithm `bo`, evaluates the candidate points using the objective function `black_box`, and adds the new data pairs `(x_new, y_new)` to the training data. We let the optimisation loop run for ten iterations (of batches of four points) and print all evaluations, where the first six columns are the inputs and the final column is the output from the objective function. The first 30 rows give the initial training data generated by the Latin hypercube design, while the last 40 rows were chosen by the Bayesian optimisation algorithm. The results show that NUBO improves upon the initial space-filling design and produces points consistent with the bounds and discrete values that specify the parameter input space.

```
>>> iters = 10
>>>
>>> for iter in range(iters):
>>>
>>>     x_new = bo(x_train, y_train)
>>>
>>>     y_new = black_box(x_new)
>>>
>>>     x_train = torch.vstack((x_train, x_new))
>>>     y_train = torch.hstack((y_train, y_new))
>>>
>>> print(torch.hstack([x_train, y_train.reshape(-1, 1)]))
```

```
tensor([[0.2000, 0.6523, 0.1574, 0.7822, 0.3039, 0.8603,  0.1251],
        [0.5000, 0.9127, 0.8746, 0.4787, 0.6523, 0.1249,  2.2907],
        [0.4000, 0.5638, 0.0459, 0.6200, 0.7056, 0.2929,  0.6744],
        [0.2000, 0.3003, 0.2290, 0.8110, 0.9529, 0.2384,  0.0442],
        [0.1000, 0.7809, 0.5374, 0.1381, 0.5655, 0.5679,  0.6123],
        [0.7000, 0.3454, 0.9352, 0.0283, 0.7969, 0.7874,  0.0732],
        [0.9000, 0.0395, 0.4250, 0.2010, 0.8243, 0.9836, -0.0281],
        [0.1000, 0.2542, 0.8055, 0.0806, 0.0381, 0.1833,  0.0791],
        [0.2000, 0.8412, 0.2388, 0.0388, 0.8542, 0.4247,  0.1126],
        [0.8000, 0.7370, 0.3922, 0.5172, 0.3952, 0.6280,  0.2454],
        [0.6000, 0.1313, 0.5940, 0.6993, 0.2744, 0.0337, -0.0489],
        [0.9000, 0.8016, 0.2760, 0.9357, 0.4104, 0.7371,  0.0214],
        [0.6000, 0.6316, 0.9115, 0.3722, 0.0273, 0.0787,  0.5736],
        [0.7000, 0.8821, 0.1944, 0.1973, 0.6821, 0.2330,  0.0460],
        [0.4000, 0.6779, 0.8546, 0.8716, 0.7558, 0.6497,  0.0783],
        [1.0000, 0.5682, 0.1289, 0.4361, 0.1452, 0.4574,  0.0586],
        [0.3000, 0.2194, 0.6718, 0.2631, 0.4686, 0.3043,  0.6239],
        [1.0000, 0.3801, 0.7743, 0.7396, 0.8837, 0.8693, -0.0627],
        [0.4000, 0.9939, 0.6117, 0.2750, 0.9850, 0.8189,  0.0045],
        [0.6000, 0.4286, 0.9720, 0.3607, 0.5942, 0.4977,  0.2505],
        [0.9000, 0.1827, 0.6516, 0.1159, 0.9271, 0.3751,  0.0581],
        [0.5000, 0.7077, 0.4601, 0.9771, 0.2619, 0.5356, -0.0289],
        [0.0000, 0.4767, 0.3555, 0.6558, 0.1989, 0.0162,  0.1569],
        [0.3000, 0.4371, 0.7413, 0.9112, 0.0723, 0.9252,  0.9560],
        [0.5000, 0.5276, 0.0843, 0.7266, 0.2118, 0.5302,  0.2011],
        [0.7000, 0.1468, 0.3188, 0.3087, 0.3526, 0.3344,  0.5923],
        [0.0000, 0.9432, 0.5099, 0.5586, 0.6024, 0.6905, -0.0087],
        [0.1000, 0.0802, 0.0017, 0.4008, 0.5019, 0.1411,  0.2529],
        [0.8000, 0.0221, 0.7221, 0.8440, 0.1240, 0.9540,  0.2324],
        [0.5000, 0.2958, 0.4872, 0.5731, 0.4418, 0.7037,  0.7130],
        [0.2000, 0.9085, 0.8757, 0.4770, 0.6390, 0.1237,  1.1281],
        [0.7000, 0.0000, 0.3683, 0.0712, 0.0515, 0.4062,  0.1484],
        [0.3000, 0.7281, 0.9619, 0.0431, 0.0000, 0.1199,  0.0894],
        [0.8000, 1.0000, 0.8457, 1.0000, 1.0000, 0.0000, -0.0373],
        [0.8000, 0.9280, 0.8906, 0.4856, 0.6641, 0.1195,  0.1296],
        [0.8000, 1.0000, 1.0000, 0.5266, 0.6308, 0.0358,  0.3261],
        [1.0000, 0.5787, 0.4309, 0.1291, 0.4129, 0.5590,  0.2360],
        [0.3000, 0.0000, 1.0000, 0.5009, 0.6804, 0.1289,  0.0289],
```

```
        [0.4000, 0.9628, 0.9097, 0.4806, 0.4525, 0.0000,  2.6153],
        [0.5000, 0.0000, 1.0000, 1.0000, 1.0000, 1.0000, -0.0507],
        [1.0000, 1.0000, 0.0000, 1.0000, 0.0000, 1.0000, -0.1304],
        [0.3000, 0.0000, 0.0000, 0.0000, 0.0000, 1.0000,  0.1087],
        [0.4000, 1.0000, 1.0000, 0.5046, 1.0000, 0.0000,  2.5231],
        [0.4000, 1.0000, 1.0000, 0.4928, 0.0000, 1.0000,  0.0980],
        [0.4000, 1.0000, 1.0000, 0.4467, 1.0000, 1.0000, -0.0251],
        [0.5000, 1.0000, 0.0000, 0.5280, 0.0000, 0.0000,  2.2291],
        [0.4000, 0.9658, 0.0000, 0.4804, 0.0000, 0.0898,  2.5967],
        [0.4000, 0.8883, 1.0000, 0.5189, 0.0000, 0.0000,  3.1767],
        [0.4000, 1.0000, 0.0000, 0.3745, 1.0000, 0.0000,  1.6312],
        [0.7000, 0.0000, 1.0000, 0.5100, 1.0000, 1.0000, -0.0249],
        [0.4000, 0.8786, 0.0161, 0.5639, 0.0581, 0.0000,  3.1551],
        [0.5000, 0.8284, 1.0000, 0.6011, 0.0000, 0.0000,  2.5988],
        [0.4000, 0.9136, 1.0000, 0.5669, 0.0000, 0.0802,  3.2133],
        [0.6000, 1.0000, 0.0000, 0.7980, 1.0000, 1.0000, -0.0130],
        [0.4000, 0.8908, 1.0000, 0.5868, 0.0000, 0.0000,  3.0625],
        [0.4000, 0.8291, 1.0000, 0.5621, 0.0000, 0.0862,  2.8820],
        [0.0000, 1.0000, 0.0000, 0.0000, 0.0000, 0.0000,  0.0447],
        [0.3000, 0.8511, 1.0000, 0.7110, 0.0000, 0.0000,  2.1879],
        [0.4000, 0.8976, 0.4139, 0.5711, 1.0000, 0.0425,  2.9899],
        [0.5000, 0.9286, 0.0000, 0.6761, 0.0000, 0.1140,  2.2584],
        [0.4000, 0.9195, 0.0000, 0.6349, 0.0000, 0.1203,  2.5906],
        [0.1000, 0.2126, 1.0000, 0.5955, 0.0000, 1.0000,  0.4850],
        [0.4000, 0.8943, 1.0000, 0.5565, 0.0000, 0.0097,  2.9492],
        [0.2000, 0.4618, 0.0000, 0.1665, 1.0000, 1.0000,  0.1278],
        [0.2000, 1.0000, 1.0000, 0.9451, 0.0000, 0.0000,  0.4609],
        [0.5000, 1.0000, 1.0000, 0.5839, 0.0000, 0.3425,  0.7659],
        [0.4000, 0.8899, 1.0000, 0.5622, 0.0000, 0.0421,  3.1330],
        [0.4000, 0.8026, 0.0000, 0.5356, 0.0000, 0.0000,  2.7828],
        [0.2000, 0.0556, 1.0000, 1.0000, 0.0000, 0.6695,  0.1134],
        [0.4000, 0.7124, 1.0000, 0.6842, 0.0000, 0.0000,  2.3028]],
        dtype=torch.float64)
```

NUBO efficiently explores the parameter space by switching between exploring areas with high uncertainty and high predicted values. This means the algorithm does not monotonically converge to a single solution as conventional optimisation algorithms would. Thus, the approximate solution to an objective function is the best value found during the optimisation. In this case study, the approximate solution, i.e., the solution with the highest

output (the final column in the Python output above), was found at iteration 53, and the inputs and outputs are printed below.

```
>>> best_iter = int(torch.argmax(y_train))
>>>
>>> print("Approximate solution")
>>> print("--------------------")
>>> print(f"Evaluation: {best_iter + 1}")
>>> print(f"Inputs: {x_train[best_iter]}")
>>> print(f"Output: {y_train[best_iter]:.4f}")

Approximate solution
--------------------
Evaluation: 53
Inputs: tensor([0.4000, 0.9136, 1.0000, 0.5669, 0.0000, 0.0802],
            dtype=torch.float64)
Output: 3.2133
```

We compare the results provided by NUBO with those from random sampling and using a space-filling design, in this case, Latin hypercube sampling (LHS). The code below generates results for the total budget of 70 evaluations for both sampling methods and plots the results, with the number of evaluations on the x-axis and the cumulative best output for each method on the y-axis. Figure 4.3 shows that NUBO (green line) provides a better solution than either alternative approach and is very close to the true maximum of 3.32237. NUBO succeeds in accurately approximating the true optimum.

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np >>>
>>>
>>> torch.manual_seed(123)
>>> random = black_box(torch.rand((70, dims)))
>>> lhs = black_box(gen_inputs(num_points=70, num_dims=dims, bounds=bounds))
>>>
>>> plt.plot(range(1, 71), np.maximum.accumulate(random), label="Random")
>>> plt.plot(range(1, 71), np.maximum.accumulate(lhs), label="LHS")
>>> plt.plot(range(1, 71), np.maximum.accumulate(y_train), label="NUBO")
>>> plt.hlines(3.32237, 0, 71, colors="red", linestyles="dashed",
...            label="Maximum")
>>> plt.title("Comparison against random designs")
>>> plt.xlabel("Evaluations") >>> plt.ylabel("Output")
```

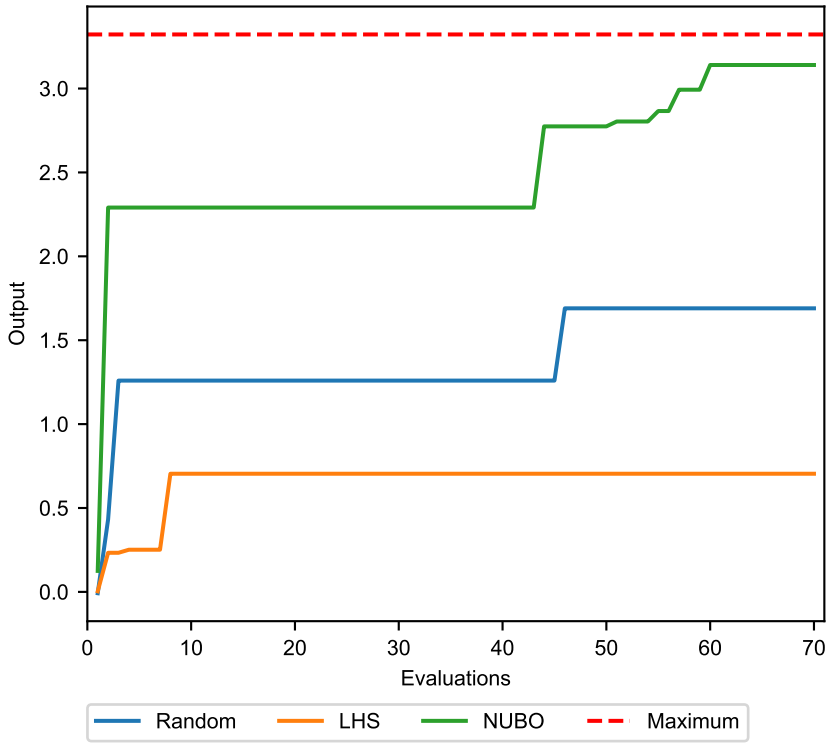

Figure 4.3: Results of the Bayesian optimisation algorithm implemented with NUBO, as defined in this case study, compared to random sampling and Latin hypercube sampling.

```
>>> plt.legend(loc='lower center', ncol=4, bbox_to_anchor=(0.5, -0.275))
>>> plt.xlim(0, 71) >>> plt.tight_layout()
```

4.4 Comparison to other packages

Various Python packages for Bayesian optimisation exist, as listed in Table 4.1. Most of them only support sequential single-point optimisation, i.e., every point the algorithm proposes must be evaluated by the objective function before moving on to the next iteration. However, parallelism can be exploited in many cases to speed up the optimisation process. For example, consider a simulator that can be run in parallel. Evaluating all points in parallel would save time as it would only take as long as evaluating a single point sequentially. pyGPGO [90], bayes_opt[5] [144], Spearmint [77], and SMAC3 [118] do not allow parallel multi-point optimisation. Furthermore, Spearmint is not modular, resulting in relatively inflexible implementations and giving the user little control when tailoring Bayesian optimisation to unique research problems. The closest available package to NUBO is BoTorch [7] as it also supports parallel and asynchronous optimisation

[5]The package is also known under the name bayesian-optimization.

Type	Modular	Sequential	Parallel	Asynchronous	Lines of code	Version
NUBO	Yes	Yes	Yes	Yes	1,322	1.0.3
BoTorch	Yes	Yes	Yes	Yes	38,419	0.8.4
bayes_opt	Yes	Yes	No	No	1,241	1.4.3
SMAC3	Yes	Yes	No	No	11,217	2.0.0
pyGPGO	Yes	Yes	No	No	2,029	0.5.0
GPyOpt	Yes	Yes	Yes	No	4,605	1.2.6
Spearmint	No	Yes	No	No	3,662	0.1

Table 4.1: Overview of available Bayesian optimisation packages in Python. We compare whether individual packages have a modular design and support sequential single-point, parallel multi-point, and asynchronous optimisation. We also list the number of lines of code of the core package (without comments, examples, tests, etc.) and the version number.

through Monte Carlo approximations of the acquisition functions. However, compared to the lightweight implementation of NUBO, BoTorch uses a very large code base that makes code comprehension difficult, as it often requires retracing various functions and objects through many files. This can be quantified by the huge codebase represented in Table 4.1 as the total number of lines of code[6]: NUBO implements Bayesian optimisation in only 1,322 lines of code over 20 files, while BoTorch uses 38,419 lines of code—roughly 29 times more than NUBO—and spreads them between 160 files. It also provides a large number of functions and methods that enforce decisions non-expert users do not have the knowledge and experience to make. NUBO lightens this burden on the user by limiting itself to the most fundamental and well-established methods. Table 4.1 also includes GPyOpt [193]; however, it is no longer maintained and has recently been archived.

The number of code lines refers to the underlying code bases of the packages and not the lines of code a user must write to apply Bayesian optimisation. When talking about a transparent implementation, the former is a better proxy as it reflects the complexity of the whole package, that is, all functions and algorithms in the package. If a package has many thousands of lines—such as BoTorch— it is intuitive that it is more complex and thus more challenging to comprehend fully than a package with only a few hundred lines of code—such as NUBO. The number of code lines it takes to apply Bayesian optimisation is less informative as it can easily be distorted. Consider, for example, a very complex algorithm with many lines of code. It would be possible to wrap this algorithm into one function that can be called with one line of code. While this reduces the lines of code, it does not change the algorithm's complexity. Thus, the comparison in this chapter focuses on the number of lines of the underlying code bases to give an idea of the size and complexity of the packages.

Although it is difficult to provide an exhaustive comparison of the relative efficiency of each

[6]The total number of lines of code does not include comments, blank lines and files that are irrelevant to the actual algorithms, such as examples, tests and test functions.

of the packages, we have undertaken a limited comparison[7] of the following form. We compare NUBO to four of the packages mentioned above—BoTorch, bayes_opt, SMAC3 and pyGPGO—representing a reasonably wide range of complexity. All methods use Gaussian processes (introduced in Section 2.3) as the surrogate model and upper confidence bound (introduced in Section 2.4.2) as the acquisition function. Please see the replication materials published alongside Diessner, Wilson, and Whalley [42] for further details on the algorithms and the benchmarking. Two synthetic test functions from Surjanovic and Bingham [188] were selected to benchmark the five packages. The first row of plots in Figure 4.4 compares the performance of sequential single-point optimisation on **A)** the two-dimensional Levy function

$$f(\boldsymbol{x}) = \sin^2(\pi w_1) + (w_1 - 1)^2 \left[1 + 10\sin^2(\pi w_1 + 1)\right] + (w_2 - 1)^2 \left[1 + \sin^2(2\pi w_2)\right],$$

where $w_i = 1 + \frac{x_i - 1}{4}$, for $i = 1, 2$, and **B)** the six-dimensional Hartmann function (already introduced in Section 4.3 and further discussed in Section 5.4.2). The second row of plots compares the performance of parallel multi-point optimisation with batches of four points on **C)** the two-dimensional Levy function and **D)** the six-dimensional Hartmann function. As a reminder, of the compared packages, only NUBO and BoTorch offer funcitonality for multi-point optimisation (**C)** and **D)**). All functions are negated to transform them from their initial minimisation problem into a maximisation problem in line with the convention of Bayesian optimisation. The plots provide the best observation (i.e., maximum value) at the current evaluation, averaging over ten replication runs. The results show that all packages converge towards the global optimum of 0.00 for the Levy function and 3.32 for the Hartmann function. While NUBO requires more evaluations to find the global optimum for the Hartmann function (**B)** and **D)**) than more complex packages such as BoTorch, it gets closest to the true optimum in all cases after all evaluations and shows low variance in these results (Table 4.2). These results show that the simplicity of NUBO's implementation does not come at a cost in performance. NUBO can outperform packages with a similar level of complexity, such as pyGPGO and bayes_opt, and compares well against more complex packages, such as BoTorch and SMAC3. This is not to say that NUBO is the superior package for any problem, but rather that NUBO performs competitively while focusing on a transparent and simple design. This makes NUBO a good candidate for optimising expensive black-box functions in the sciences—such as physical experiments and computer simulators—where transparency is vital.
However, the time NUBO requires to complete one iteration, with a maximum of 2.20s for **D)**, is, on average, higher than for the other packages (Table 4.3). While this might be important for some areas of optimisation, it will typically be negligible when optimising

[7] All comparisons were run on an Apple Mac mini with an M2 chip and 16 GB memory.

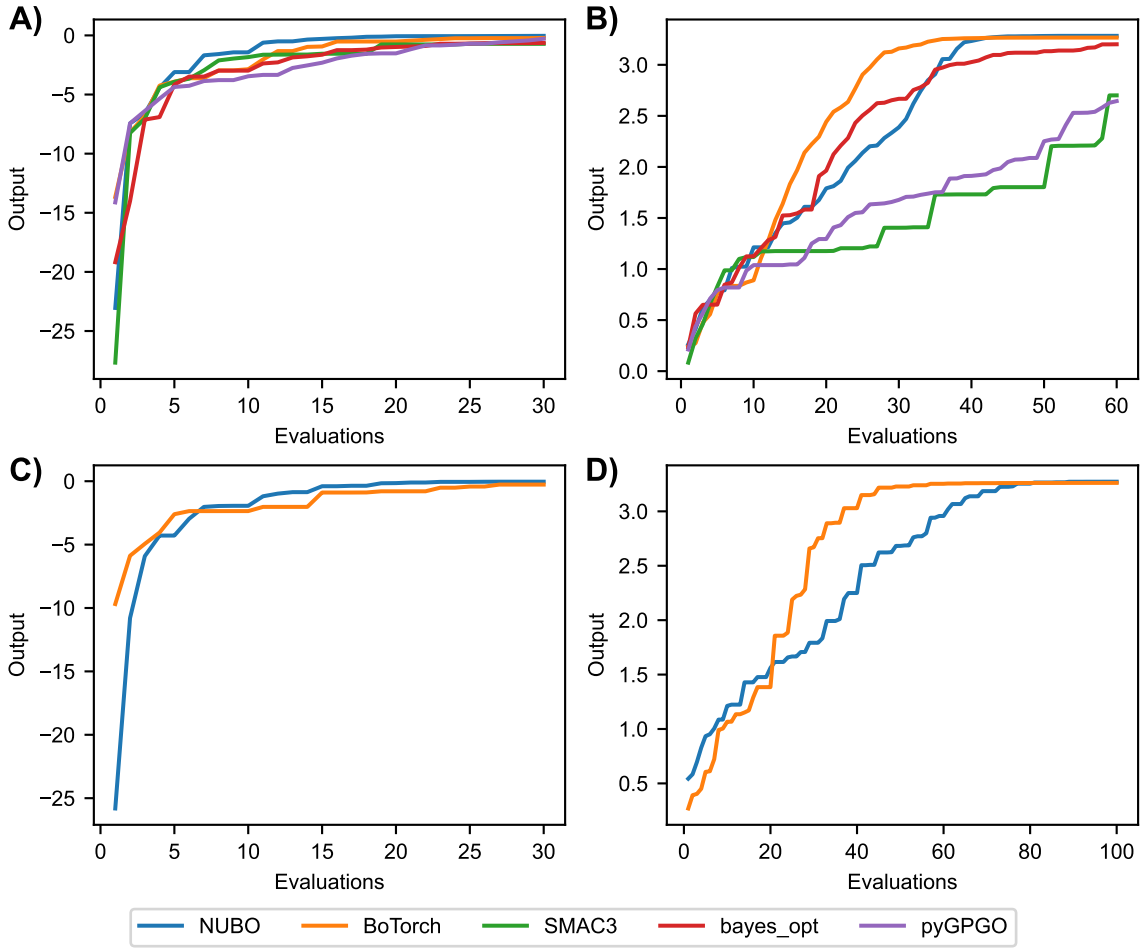

Figure 4.4: Comparison of different Python packages for Bayesian optimisation. **A)** Sequential single-point optimisation on the 2D Levy function; **B)** Sequential single-point optimisation on the 6D Hartmann function; **C)** Parallel multi-point optimisation with a batch size of four on the 2D Levy function; **D)** Parallel multi-point optimisation with a batch size of four on the 6D Hartmann function.

	Sequential		Parallel	
	2D Levy	6D Hartmann	2D Levy	6D Hartmann
NUBO	-0.04 (0.06)	3.28 (0.06)	-0.04 (0.04)	3.27 (0.06)
BoTorch	-0.21 (0.20)	3.27 (0.07)	-0.27 (0.21)	3.26 (0.06)
SMAC3	-0.71 (0.58)	2.70 (0.38)	-	-
bayes_opt	-0.64 (0.74)	3.20 (0.13)	-	-
pyGPGO	-0.28 (0.31)	2.64 (1.05)	-	-

Table 4.2: Comparison of different Python packages for Bayesian optimisation. The best observations averaged across the ten runs with corresponding standard errors are given for each package.

expensive black-box functions, as these functions are much more resource-intensive to evaluate. Thus, the small number of additional seconds that NUBO requires per iteration

	Sequential		Parallel	
	2D Levy	6D Hartmann	2D Levy	6D Hartmann
NUBO	0.60s	1.88s	0.07s	2.20s
BoTorch	0.09s	0.22s	0.00s	0.19s
SMAC3	0.08s	0.25s	-	-
bayes_opt	0.14s	0.24s	-	-
pyGPGO	0.23s	0.65s	-	-

Table 4.3: Comparison of different Python packages for Bayesian optimisation. The elapsed time per iteration averaged across the ten runs is given for each package.

is insignificant compared to the resources required to conduct an experiment or run a simulation.

Besides implementations in Python, there are some implementations in other programming languages. For example, rBayesianOptimization [217] and ParBayesianOptimization [213] implement basic Bayesian optimisation algorithms for hyper-parameter tuning similar to bayes_opt and pyGPGO in R. ParBayesianOptimization provides additional support for parallel optimisation and follows Snoek, Larochelle, and Adams [178]. Future work could compare NUBO to these packages, although they are beyond the scope of this thesis.

4.5 Conclusion

This chapter introduced NUBO, a Python package for Bayesian optimisation for expensive-to-evaluate black-box functions, such as computer simulators and physical experiments. NUBO's main objective is to make Bayesian optimisation accessible to researchers from all disciplines by providing a transparent and user-friendly implementation.

NUBO includes five sub-modules that implement Gaussian processes, acquisition functions, optimisers, test functions, and utilities. These modules provide all necessary functionalitiy for sequential single-point, parallel multi-point, and asynchronous optimisation of expensive-to-evaluate black-box functions for bounded, constrained, and mixed (discrete and continuous) input parameter spaces. We have introduced and explained these functionalities with individual code snippets and illustrated NUBO's general workflow using a detailed case study that takes a hypothetical six-dimensional expensive-to-evaluate black-box function and approximates its global optimum with a parallel multi-point Bayesian optimisation algorithm.

A brief comparison with other Python packages for Bayesian optimisation showed that NUBO performs competitively while providing a transparent and simple implementation. This makes NUBO a good candidate for optimising expensive black-box functions when transparency is vital.

In the future, we plan to extend NUBO to include optimisation strategies for multi-fidelity,

multi-objective, and high-dimensional problems.
We used NUBO in the following journal articles and conference papers:

[41] M. Diessner, J. O'Connor, A. Wynn, S. Laizet, Y. Guan, K. Wilson, and R. D. Whalley. "Investigating Bayesian Optimization for Expensive-to-Evaluate Black Box Functions: Application in Fluid Dynamics". In: *Frontiers in Applied Mathematics and Statistics* 8 (2022), p. 1076296

[145] J. O'Connor, M. Diessner, K. Wilson, R. D. Whalley, A. Wynn, and S. Laizet. "Optimisation and Analysis of Streamwise-Varying Wall-Normal Blowing in a Turbulent Boundary Layer". In: *Flow, Turbulence and Combustion* 110.4 (2023), pp. 993–1021

[42] M. Diessner, K. Wilson, and R. D. Whalley. "NUBO: A Transparent Python Package for Bayesian Optimisation". In: *arXiv preprint arXiv:2305.06709* (2023). Accepted by Journal of Statistical Software

[43] M. Diessner, K. J. Wilson, and R. D. Whalley. "On the Development of a Practical Bayesian Optimization Algorithm for Expensive Experiments and Simulations With Changing Environmental Conditions". In: *Data-Centic Engineering* 5 (2024), e45

[40] M. Diessner, X. Chen, K. J. Wilson, and R. D. Whalley. "Optimising Active Flow Control Strategies for Random and Controlled Wind Speeds via Bayesian Optimisation". In: *Symposium on Turbulence and Shear Flow Phenomena (TSFP)*. TSFP. 2024

[27] X. Chen, M. Diessner, K. J. Wilson, and R. D. Whalley. "Optimizing Wall Blowing for Global Skin-Friction Drag Reduction Using a Bayesian Optimization Framework". In: *Symposium on Turbulence and Shear Flow Phenomena (TSFP)*. TSFP. 2024

Chapter 5

Optimisation under randomly changing environmental conditions

Summary

Experiments in engineering and other scientific disciplines are typically conducted in controlled environments where parameters can be set to any desired value. The assumption that is often made is that the same applies in a real-world setting, which can be incorrect as many real-world systems are influenced by uncontrollable environmental conditions such as temperature, humidity and wind speed. When optimising such experiments, the focus should be finding optimal values conditionally on these uncontrollable variables. This chapter extends Bayesian optimisation to the optimisation of systems in changing environments that include controllable and uncontrollable parameters. The extension fits a global surrogate model over all controllable and environmental variables but optimises only the controllable parameters conditional on measurements of the uncontrollable variables. The method is validated on two synthetic test functions, and the effects of the noise level, the number of environmental parameters, the parameter fluctuation, the variability of the uncontrollable parameters, and the effective domain size are investigated. ENVBO, an extension of Bayesian optimisation implemented in NUBO, is applied to two problems considering a wind farm simulator. ENVBO finds solutions for the entire domain of the environmental variable that outperform, or are competetive with, results from optimisation algorithms that focus on fixed environmental values. ENVBO uses a fraction of the evaluation budgets of the benchmarks and can predict solutions for the entire range of the environmental variable. This makes the proposed approach very sample-efficient and cost-effective. An off-the-shelf open-source version of ENVBO is available via the NUBO Python package.

5.1 Introduction

In its original form, Bayesian optimisation is a global optimisation algorithm that aims to find a global optimum of a function in a minimum number of function evaluations, also called observations. For standard Bayesian optimisation to be effective, all parameters must be controllable, and all environmental factors influencing the output must remain constant. However, this assumption is only true in completely isolated and controlled environments. Considering more realistic scenarios where some variables cannot be controlled, this assumption will not hold in general. Environments in the real world are generally more complex, and environmental conditions, such as humidity, temperature and wind speed, are typically governed by uncontrollable external factors.

Many applications of Bayesian optimisation—implicitly or explicitly—assume a simplistic world where all environmental conditions are fixed. In active flow control, for example, where the goal is to control blowing actuators to maximise the reduction of the skin-friction drag over a flat plate, the ambient wind speed is assumed to be fixed [123, 41, 145, 124]. However, the optimal parameters found from these simulations and experiments give optima for specific wind speeds and cannot necessarily be generalised to other wind speeds. This approach to experimentation requires repeating the experiment for each wind speed to find multiple optima. Because wind speeds are assumed to be fixed, it is impossible to share observations and, thus, information between experiments. While observations from different wind speeds will likely not result in the same drag reduction, they will be correlated and contain some information that can be transferred to problems with similar wind speeds. Sharing information between different environmental conditions could decrease the number of observations required and make Bayesian optimisation more sample-efficient and cost-effective—both essential properties and important objectives of Bayesian optimisation.

This chapter presents a practical strategy for optimising expensive black-box functions such as physical experiments and computer simulations with influential environmental conditions that are governed externally and cannot be controlled during the optimisation. The strategy extends Bayesian optimisation by (a) fitting a global surrogate model over all controllable and uncontrollable variables, (b) solving the acquisition function conditionally on measurements taken for the uncontrollable variables, and (c) restricting the initial training data that typically consists of many observations generated via a space-filling design to only one observation. It is shown that ENVBO, the proposed algorithm, generalises to situations with noisy observations, multiple uncontrollable variables, and uncontrollable variables with different levels of fluctuation and variability.

To illustrate the value of this approach, two problems within a wind farm simulator are considered. The first problem aims to maximise annual power generation by finding op-

timal positions for four wind turbines. The wind direction affects the power generation significantly and is assumed to vary randomly in the simulations. Thus, the wind direction represents an influential environmental condition. The second problem aims to maximise energy production by setting the derating strategy for a row of five wind turbines with changing wind speeds. Results for the first problem show that ENVBO outperforms two other optimisation algorithms (standard Bayesian optimisation and the SLSQP algorithm [107]) used as benchmarks in all but one case. For the second problem, ENVBO performs comparably to standard Bayesian optimisation and outperforms the Nelder-Mead algorithm [143]. ENVBO has the additional benefits of using fewer function evaluations than the benchmarks and can propose wind turbine positions for any possible wind direction within the range investigated. Similar results from the benchmarks could only be achieved by repeating simulation campaigns many times for different wind speeds—an expensive, if not infeasible, task given that wind speed is a continuous variable. Thus, the algorithm is sample-efficient and cost-effective and effectively addresses the main problem of expensive black-box function optimisation. An off-the-shelf open-source version of ENVBO is available via the NUBO Python package [42] at `www.nubopy.com`.

This chapter is structured as follows. Section 5.2 gives an overview of related literature and highlights differences to the work in this chapter. Section 5.3 extends Bayesian optimisation introduced in Section 2.2 to allow optimisation with changing environmental conditions. Section 5.4 validates the approach on two synthetic test functions—the two-dimensional Levy function and the six-dimensional Hartmann function—and introduces a way to simulate randomly changing environmental conditions via random walks. Section 5.5 investigates five properties of the proposed method: noise, number of uncontrollable variables, parameter fluctuation, parameter variability and effective domain size (i.e., the actual searched space for the environmental conditions). Section 5.6 considers nine-dimensional and six-dimensional wind farm simulators with one uncontrollable variable and eight and five controllable variables, respectively. Section 5.7 discusses the results of the empirical investigation and the application to the wind farm simulator and highlights limitations and implications. Lastly, Section 5.8 summarises this chapter and draws conclusions.

5.2 Related work

This chapter focuses on a particular scenario in Bayesian optimisation [53] that is referred to as random environmental conditions [24], multi-task optimisation [190] and contextual optimisation [108]. Methods differ by the type of input parameters (discrete or continuous), the number of allowed environmental conditions or contexts (finite or infinite), and the overall goal of the optimisation (one optimal solution for all contexts or one optimal

solution for each context). When the number of contexts is infinite, finding one optimal solution for each context corresponds to finding a function that returns optimal inputs based on the context. This is the main objective of this chapter. Gaussian process-based methods aiming to optimise problems with different contexts—that is, with controllable and environmental variables—considered previously can be classified primarily into two types based on their optimisation goal.

The first aims to find one solution that yields the best result for all tasks/contexts. It is assumed that environmental variables take values according to a probability distribution. For example, if the environment is defined by its temperature, the current temperature will follow a distribution. Optimisation aims to find the optimiser that maximises the objective function, taking into account the likelihood of the different environments. Different approaches assume the distribution of the environmental variables to be discrete [209, 190], continuous [73, 216] or both [198]. These methods assume that parameters and contexts can be selected during optimisation, and many use a sequential approach, where parameters for the next candidate are selected before the context is chosen. Only Toscano-Palmerin and Frazier [198] present an approach that chooses parameters and contexts jointly and can optimise problems where contexts are uncontrollable and given randomly. Chang et al. [25] and Chang et al. [24] use this type of approach to design a femoral component for hip replacements conditional on joint force orientation and cancellous bone properties. The aim is to find one optimal design for a wide demographic with varying characteristics.

The second type aims to find one solution for each context. Thus, interest lies not in finding one global optimum but multiple optima, one for each combination of environmental conditions. Pearce and Branke [153], Char et al. [26] and Chung et al. [30] consider multiple discrete tasks, while Ginsbourger et al. [63] and Pearce and Branke [152] consider continuous environmental variables or both. These methods are closely related to the objective of this chapter. However, they have one important distinction. While the environmental values are given externally in real-world applications, it is assumed that they can be set to any desired values in the experiments and simulations above. This deviates from our problem formulation, where we explicitly regard problems with uncontrollable environmental variables—in experiments, simulations and the real world.

The research in this chapter is closely related to that of Krause and Ong [108], who modified upper confidence bound [184] to be suited to optimisation with externally given environmental conditions and derived theoretical bounds for its contextual regret. In contrast to Krause and Ong [108], this chapter considers improvement-based acquisition functions, i.e., expected improvement [94] and log expected improvement [2], gives a detailed description of the practical implementation of Bayesian optimisation with environmental conditions and provides all code at `https://github.com/mikediessner/`

`environmental-conditions-BO`. In addition, the approach in this chapter makes fewer assumptions than Krause and Ong [108], who focus on a linear and additive covariance structure for the environmental variables. In Bayesian optimisation, the optimisation problems are by definition black-box problems. This means that no knowledge about the influence of the environmental variables on the output is available. Hence, it might be challenging to determine if the environmental variable possesses a linear, additive or completely different structure. ENVBO avoids this issue by making as few assumptions as possible and estimating the structure from the data using a flexible Matérn kernel. Furthermore, using expected improvement rather than upper confidence bound as the acquisition function has the advantage of not requiring the selection of a hyperparameter. While there are some strategies for determining the tuning parameter in UCB (see Section 2.4.2), it remains a challenge to find the optimal value in practice and there currently exists no guidance in the context of environmental conditions. Expected improvement can thus be more easily applied to different problems. This yields a widely applicable strategy that generalises well to real-world problems.

The area of multi-task optimisation is also related to multi-objective optimisation. In multi-objective optimisation, the aim is no longer to optimise one single objective function but rather multiple objective functions simultaneously. A global optimum for all objective functions is only possible if it happens to be at the same values for all inputs for all functions. Thus, the main goal is to find the best trade-off between the objective functions [58, 53]. This is related to the first problem above, which aims to find a trade-off between all contexts. However, objective functions in multi-objective optimisation are generally assumed not to be correlated, while correlations are leveraged for contexts. Acquisition functions such as expected improvement [46, 157, 219, 218], predictive entropy search [79] and max-value entropy search [10, 49] have been extended to allow multi-objective optimisation, as discussed in Section 2.5.5.

5.3 Changing environmental conditions

The standard Bayesian optimisation algorithm, given in Algorithm 1, assumes that all parameters influencing the output can be controlled. However, in many cases, when optimising physical experiments, variables will be present that influence the output but cannot be controlled. This chapter refers to these uncontrollable variables as environmental variables as they are given by the environment and are uncontrollable. Examples of such uncontrollable variables are temperature, humidity and wind speed. This section presents an extension to Algorithm 1 that allows the inclusion of uncontrollable environmental variables in the optimisation process. The extension can be broken down into three parts as highlighted in Algorithm 3.

The main modification to Algorithm 1 concerns the surrogate modelling. The standard Bayesian optimisation algorithm fits a surrogate model over all controllable variables. Environmental variables are not included and are assumed to be fixed over the full optimisation process or are irrelevant to the output. Algorithm 3 does not make this assumption and includes all controllable parameters $\boldsymbol{x}_C$ and environmental variables $\boldsymbol{x}_E$ in its surrogate model. The inputs $\boldsymbol{X}_n$ of the training data $\mathcal{D}_n = (\boldsymbol{X}_n, \boldsymbol{y}_n)$ are extended from $\boldsymbol{X}_n = (\boldsymbol{X}_{n,C})$ to $\boldsymbol{X}_n = (\boldsymbol{X}_{n,E}, \boldsymbol{X}_{n,C})$, where $\boldsymbol{X}_{n,C}$ and $\boldsymbol{X}_{n,E}$ are the controllable and the environmental variables in the training data, respectively. Furthermore, the controllable and environmental variables of individual candidate points are written as $\boldsymbol{x}_{n,C}$ and $\boldsymbol{x}_{n,E}$, respectively.

The second extension regards the computation of the next candidate point, specifically, the maximisation of the acquisition function. While in Algorithm 1 all parameters are assumed to be controllable and the acquisition function can be maximised over all parameters $\max_{\boldsymbol{x}_C} \alpha(\boldsymbol{x}_C)$, Algorithm 3 must differentiate between the controllable parameters $\boldsymbol{x}_C$ and environmental variables $\boldsymbol{x}_E$. The uncontrollable variables are given by the environment and can only be measured, not manipulated. Hence, the maximisation of the acquisition function is broken down into two steps. First, the environmental variables are measured. This gives values for the uncontrollable inputs for the next candidate point $\boldsymbol{x}_{n+1,E}$. Second, the acquisition function is maximised conditional on these values for the environmental inputs $\max_{\boldsymbol{x}_C|\boldsymbol{x}_E} \alpha(\boldsymbol{x}_C)$ resulting in the controllable inputs for the next candidate point $\boldsymbol{x}_{n+1,C}$. Conditional maximisation essentially means that the environmental variables $\boldsymbol{x}_E$ are treated as fixed for the maximisation of the acquisition function for one iteration. This assumes that the environmental variables do not change significantly from the time of measuring until the evaluation of the new candidate point $\boldsymbol{x}_{n+1}$. This assumption should be realistic for most experiments, as one iteration of the Bayesian optimisation loop—i.e., measuring the environmental variables, fitting a Gaussian process and optimising the acquisition function conditional on the measurements—takes only a few seconds (see Section 4.4 for runtimes of different Bayesian optimisation packages). However, issues could arise when working with environmental variables that change rapidly. Thus, Section 5.5 investigates the influence of different fluctuation rates. The new candidate point $\boldsymbol{x}_{n+1}$ is then defined as a combination of the measurements for the environmental variables $\boldsymbol{x}_{n+1,E}$ and the results of the maximisation of the acquisition function $\boldsymbol{x}_{n+1,C}$.

The last adjustment to Algorithm 1 focuses on generating the training data. Usually, training data is produced using a space-filling design, such as a Latin hypercube [129, 86]. Under the assumption that all parameters can be controlled, the experiment can be conducted for each training point to observe its output. However, the modified Bayesian optimisation algorithm includes uncontrollable variables in its computation. Thus, it is impossible to evaluate any arbitrary combination of inputs as it is limited by the cur-

rent measurement of the environmental variables. To resolve this issue, Algorithm 3 uses one training point $\boldsymbol{x}_0$ instead of multiple points generated from a space-filling design. The initial training data is restricted to a single point, while the second data point onwards is computed via Bayesian optimisation. The first data point is generated by taking measurements for the environmental variables $\boldsymbol{x}_{0,E}$ and randomly selecting values for the controllable parameter $\boldsymbol{x}_{0,C}$. These inputs are then evaluated, resulting in a complete training input-output pair $\mathcal{D}_0 = (\boldsymbol{x}_0, y_0)$, followed by the first Bayesian optimisation loop.

Algorithm 3 Modified Bayesian optimisation algorithm with environmental conditions

Require: Evaluation budget N, surrogate model $\mathcal{M}$, acquisition function α.
Sample an initial training data point $\boldsymbol{x}_0 = (\boldsymbol{x}_{0,E}, \boldsymbol{x}_{0,C})$ where environmental parameters $\boldsymbol{x}_{0,E}$ are measured and controllable parameters $\boldsymbol{x}_{0,C}$ are randomly sampled and obtain observation y_0.
Set $n = 0$.
while $n \leq N - n_0$ **do**
Fit surrogate model $\mathcal{M}$ to training data $\mathcal{D}_n = (\boldsymbol{X}_n, \boldsymbol{y}_n)$, where $\boldsymbol{X}_n = (\boldsymbol{X}_{n,E}, \boldsymbol{X}_{n,C})$.
Measure environmental variables $\boldsymbol{x}_{n+1,E}$.
Find values for the controllable parameters $\boldsymbol{x}_{n+1,C}$ that maximise the acquisition function α conditionally on the measurements $\boldsymbol{x}_{n+1,E}$ such that $\boldsymbol{x}_{n+1} = (\boldsymbol{x}_{n+1,E}, \boldsymbol{x}_{n+1,C})$, i.e., solve $\operatorname{argmax}_{\boldsymbol{x}_C | \boldsymbol{x}_E} \alpha(\boldsymbol{x}_C)$.
Evaluate $\boldsymbol{x}_{n+1}$ by observing y_{n+1}
Increment n.
end while
return Point $\boldsymbol{x}^*$ with highest observation y^*.

In contrast to Krause and Ong [108], the proposed approach does not assume different covariance structures for controllable and environmental variables. When working with experiments and simulators, the underlying objective function is generally unknown or too complex to compute directly [53]. Even with expert knowledge, there might be insufficient information about these black boxes to confidently assume a linear or additive structure for the environmental variable. Hence, providing the surrogate model with enough flexibility to estimate the covariance structure is essential. This can be achieved using the Matérn kernel for controllable and environmental variables—or the radial basis function kernel for objective functions expected to be very smooth.

Figure 5.1 illustrates the conditional optimisation on a two-dimensional problem with one uncontrollable variable x_1 and one controllable parameter x_2. Plot **A)** shows the true output of the objective function, where yellow areas indicate high function values and blue areas indicate low function values. The goal is to find the optimal value for the controllable parameter (y-axis) that maximises the output for any uncontrollable variable (x-axis) value. Plot **B)** shows the predictive mean of a Gaussian process fitted to 20 training data points (black crosses). Following Algorithm 3, a measurement (red dashed line) of

the uncontrollable variable results in $x_1 = -0.5$. The next iteration of the optimisation loop is performed conditional on this measurement. Plot **C)** shows the predictive mean and corresponding uncertainty of the Gaussian process for $x_1 = -0.5$. The conditional optimisation takes a slice from the full surrogate model. It reduces the two-dimensional optimisation problem to a one-dimensional problem for each optimisation step, where only the controllable parameters are considered. However, the information gained from the observed data is shared between each iteration. Notice that no training points lie on the measurement line, but the model uses the available training points to inform its prediction. If Algorithm 1 were used, the uncontrollable input would be assumed to be fixed for the entire optimisation loop, and the optimisation process would need repeating for each value of x_1. Plot **D)** extends plot **C)** by adding the acquisition function. The optimal value of the controllable input x_2 is found by maximising the acquisition function, and the new candidate point is a combination of the controllable input value at this maximum and the measurement taken for the uncontrollable variable x_1. The candidate point is observed and added to the training data to be used in the next iteration of the optimisation loop.

5.4 Simulations

This section uses two previously discussed synthetic test functions[1]—the Levy function and the Hartmann function—and applies the Bayesian optimisation algorithm with environmental conditions presented in Section 5.3. Simulations for both problems are run for 100 evaluations and are repeated 30 times[2] to validate the robustness of Algorithm 3. This decreases the risk that results are influenced by the method's inherent randomness, e.g., the randomly sampled training points that initialise the algorithm. For both test functions we assume one uncontrollable variable whose value is provided by a random walk at each iteration. In the simulations, each step of the random walk adds a sample from a uniform distribution $\mathcal{U}$ to the previous value of the uncontrollable variable, such that

$$\boldsymbol{x}_{n,E} = \boldsymbol{x}_{n-1,E} + \mathcal{U}_{[-\boldsymbol{a},\boldsymbol{a}]}, \tag{5.1}$$

where $\boldsymbol{a}$ is a vector of small predefined constants that provide the minimal and maximal change of the environmental variables from one iteration to the next. This uniform assumption represents the natural fluctuation of the uncontrollable variables encountered in a real-world application, e.g., changes in temperature, humidity and wind speed. It further allows the investigation of uncontrollable variables with different fluctuation levels by increasing or decreasing the constants in $\boldsymbol{a}$ as discussed in Section 5.5. Assuming another

[1] See `https://www.sfu.ca/~ssurjano/optimization.html` for further details on the synthetic test functions.

[2] Runs, replications, and repeats are used interchangeably in this chapter.

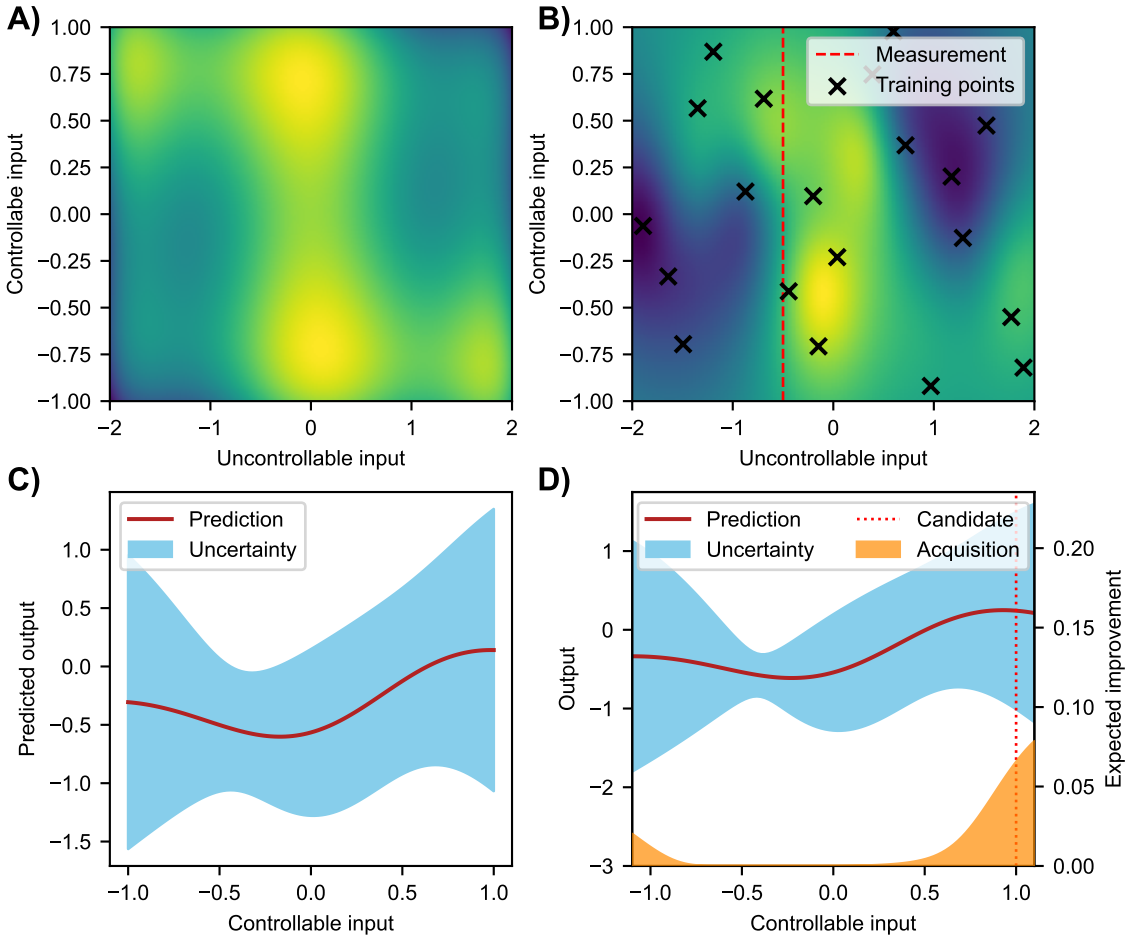

Figure 5.1: Maximisation of a two-dimensional problem with one environmental variable x_1 and one controllable variable x_2. Yellow areas indicate high outputs, and dark blue areas indicate low outputs. **A)** True objective function. **B)** Prediction of a Gaussian process with a measurement taken for the following conditional optimisation step. **C)** Gaussian process predictive posterior for optimisation conditional on the measurement. **D)** Bayesian optimisation step conditional on the measurement.

distribution for the constants in $\boldsymbol{a}$, such as a Normal distribution, is an alternative to this approach.

Subsequent sections analyse the performance of Algorithm 3 by comparing predictions from the Gaussian process models via the mean function $\mu_n(\boldsymbol{x})$ to the actual optimal values of the objective function. In both cases, results are obtained by maximising the predictive mean of the Gaussian process and the true objective function conditional on identical test values $\boldsymbol{x}'_E$ for the uncontrollable variable. To ensure that test values cover the domain of the environmental variables evenly for a fair comparison, test values are sampled from a maximin Latin hypercube design [129, 86] within the observed domain of the uncontrollable variable $[\min(\boldsymbol{x}_{N,E}), \max(\boldsymbol{x}_{N,E})]$ considered by the algorithm. A

step-by-step algorithm showing how performance is assessed can be found in Algorithm 4. We also call this observed domain the effective domain. This ensures a fair comparison by avoiding predictions outside the effective domain, requiring extrapolation. Extrapolation with Gaussian processes generally means that predictions default to the prior mean function (see Section 2.3 for further details). As ENVBO uses a constant mean function, the resulting prediction when extrapolating would tend to the constant mean estimated for the full parameter space. Any other trends and information learned from the data would not be taken into account, and the resulting prediction would very likely not reflect the truth. In cases where extrapolation cannot be avoided, making the prior mean function as informative as possible—for example, by going beyond zero and constant mean functions with polynomial and trigonometric mean functions—can improve results significantly, as trends learned from the data would not naively be discarded [156].
To assess the performance of the algorithm, the mean absolute percentage error between the maximum predictions $\boldsymbol{m}^*$ and the true optimal values of the objective function $\boldsymbol{f}^*$ are computed conditional on all k test values of the environmental variable. In this case, the mean absolute percentage error is defined as

$$\mathrm{MAPE}\left(\boldsymbol{m}^*, \boldsymbol{f}^*\right) = \frac{1}{k} \sum_{i=1}^{k} \left| \frac{m_i^* - f_i^*}{f_i^*} \right|.$$

The acquisition criterion conditional on values of the uncontrollable variables $\boldsymbol{x}_{n,E}$ is maximised with the SLSQP algorithm [107] using multiple starts. For this strategy, 100 points are sampled from a maximin Latin hypercube design [129, 86] and evaluated by the acquisition criterion. The best 20 points are then used to initialise the SLSQP algorithm, and only the best result is used as the solution for the optimisation problem. The multiple starts aim to reduce the risk of converging towards a local optimum instead of the desired global optimum of the acquisition function.

5.4.1 The two-dimensional Levy function

The Levy function

$$f\left(\boldsymbol{x}\right) = \sin^2\left(\pi w_1\right) + \left(w_1 - 1\right)^2 \left[1 + 10\sin^2\left(\pi w_1 + 1\right)\right] + \left(w_2 - 1\right)^2 \left[1 + \sin^2\left(2\pi w_2\right)\right],$$

where $w_i = 1 + \frac{x_i - 1}{4}$, for $i = 1, 2$, is a two-dimensional function with two input parameters x_1 and x_2. Although often set up as a minimisation problem to find the global minimum $f\left(\boldsymbol{x}^*\right) = 0$ at $\boldsymbol{x}^* = (1, 1)$ for the input space $[-10, 10]^2$, we choose the bounds of the parameters as $[-7.5, 7.5]$ and $[-10, 10]$ respectively to create a maximisation problem that is better suited for testing Algorithm 3. The controllable parameter is restricted from its usual range of $[-10, 10]$ to $[-7.5, 7.5]$ to avoid the steep ridges of the Levy function at

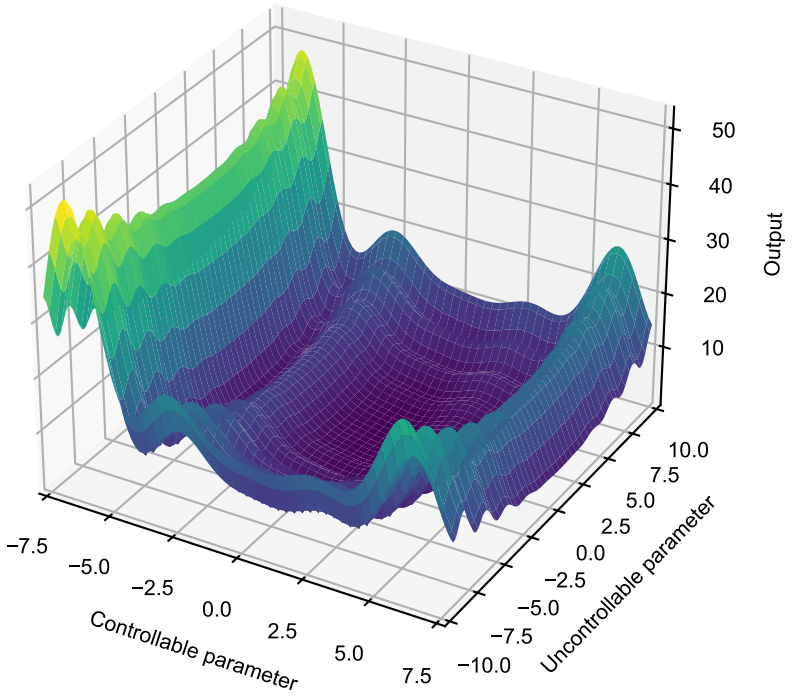

Figure 5.2: Two-dimensional negated Levy function with one controllable parameter x_1 bounded by $[-7.5, 7.5]$ and one uncontrollable variable x_2 bounded by $[-10, 10]$.

$x_1 = -10$ and $x_1 = 10$ that push the optima towards the outer bounds of the parameter space. The function given in Figure 5.2 shows a clear ridge at values of $x_2 = -6$ for the controllable parameter x_2 for all values of the uncontrollable variable x_1. The simulations use $a = 1.5$ as the uniform distribution constant of the random walk for the uncontrollable variable x_2. For this relatively simple two-dimensional function, the constant a was chosen to change by no more than 7.5% of the uncontrollable parameter's range in either direction per iteration. This assumes that the environmental conditions do not change too much between iterations. The effect of setting a to different values is investigated and discussed in Section 5.5.

Plot **A)** in Figure 5.3 shows the performance of Algorithm 3 as the mean absolute percentage error between the maximum of the Gaussian process prediction and the maximum of the true objective function conditional on 25 test values of the uncontrollable variable. Three alternatives for the acquisition function—expected improvement (EI), log expected improvement (LogEI) and upper confidence bound (UCB) as introduced in Section 2.4—are compared to a benchmark where values for the controllable variable were selected randomly. Three different values (4, 8, 16) for the trade-off parameter β of upper confidence bound are considered in Figure 5.4. While results for all parameter values are

Algorithm 4 Benchmarking the performance of ENVBO

Get the effective domain (minimal and maximal values) of the environmental variables.

if $n_E = 1$ **then**
Sample 25 test values for the environmental variables from a Latin hypercube design within their effective domain.
else if $n_E = 2$ **then**
Sample 50 test values for the environmental variables from a Latin hypercube design within their effective domain.
else if $n_E = 3$ **then**
Sample 75 test values for the environmental variables from a Latin hypercube design within their effective domain.
end if

for every ten evaluations **do**
Fit a Gaussian process to the data observed to the current evaluation.
Find the maximum posterior predictive mean conditional on each test value for the environmental variables.
end for

Find the optimal values of the objective function conditional on each test value for the environmental variables.
Assess performance by computing the mean absolute percentage error between the maximum posterior predictive mean and the optimal values of the objective function.

return Mean absolute percentage error for every ten evaluations.

comparable, upper confidence bound with trade-off parameter $\beta = 8$ performs slightly better in terms of mean performance and variance between runs. Thus, comparisons with expected improvement and log expected improvement use $\beta = 8$. The solid lines indicate the mean performance, while the shaded areas indicate the 95% confidence interval over the 30 replications. Each replication's mean absolute percentage error is computed for every ten evaluations for the same 25 test points $\boldsymbol{X}'_E$ of the environmental variable. The test values are sampled from a Latin hypercube bounded by the minimal and maximal value of the uncontrollable variable after 100 function evaluations. For further details on how performance is assessed see Algorithm 4. The mean absolute percentage error starts just below 0.8 for all alternatives. While the improvement-based algorithms (EI and LogEI) performed better than the random benchmark, the algorithm using upper confidence bound performs worse. After 100 function evaluations, expected improvement and log expected improvement have a mean absolute percentage error of 0.08 and 0.06 respectively and improve upon the random benchmark (0.17). Upper confidence bound only achieves a mean absolute percentage error of 0.24. Moreover, the optimistic strategy

shows large confidence intervals, indicating that the method is not robust. Altering the trade-off parameter β did not improve this result, as illustrated in plot **A)** in Figure 5.4. Plot **C)** presents the difference between the mean absolute percentage error of the random benchmark and the three alternative acquisition functions after 100 evaluations. This difference is computed for each of the 30 replications, and distributions of the differences are plotted for the three different acquisition functions. Negative values indicate replications where the benchmark resulted in superior solutions, while positive values indicate that the given version of Algorithm 3 performed better than the benchmark. Although no alternative is better than the benchmark for every replication, there is a clear difference between the improvement-based and the optimistic acquisition functions. Indeed, the mean of upper confidence bound (displayed by the horizontal line towards the centre of each violin plot) is worse than zero. This means that the random benchmark outperforms upper confidence bound on average. Additionally, the plot mirrors the lack of robustness discovered earlier by the large spread in differences. While upper confidence bound performs better for some replications than the random benchmark, it performs much worse for others. A Mann-Whitney U test was performed to determine which alternatives perform differently from the random benchmark using a 1% significance level. The test returned a p-value of <0.001 for the improvement-based algorithms and 0.371 for the upper confidence bound. This provides evidence against the supposition that improvement-based methods perform equally well compared to the random benchmark, reinforcing the results from the visual analysis that expected improvement and log expected improvement perform significantly better than the random benchmark. This cannot be said for the optimistic method—the test cannot reject the null hypothesis, indicating that upper confidence bound does not perform significantly differently from a random approach.

5.4.2 The six-dimensional Hartmann function

The negated Hartmann function

$$f(\boldsymbol{x}) = \sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{6} \boldsymbol{A}_{ij} \left(x_j - \boldsymbol{P}_{ij} \right)^2 \right),$$

where

$$\alpha = (1.0, 1.2, 3.0, 3.2)^T,$$

$$\boldsymbol{A} = \begin{pmatrix} 10.00 & 3.00 & 17.00 & 3.50 & 1.70 & 8.00 \\ 0.05 & 10.00 & 17.00 & 0.10 & 8.00 & 14.00 \\ 3.00 & 3.50 & 1.70 & 10.00 & 17.00 & 8.00 \\ 17.00 & 8.00 & 0.05 & 10.00 & 0.10 & 14.00 \end{pmatrix}, \text{ and}$$

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

is a six-dimensional function with six input parameters x_1, x_2, x_3, x_4, x_5 and x_6 that are defined on the hypercube $(0, 1)^6$. It has six local maxima and one global maximum with $f(\boldsymbol{x}^*) = 3.32$ at $\boldsymbol{x}^* = (0.20, 0.15, 0.48, 0.28, 0.31, 0.66)$. The simulations use $a = 0.05$ as the uniform distribution constant of the random walk for the environmental variable x_6. As this six-dimensional function is more complex than the Levy function, the constant a was chosen to change by a maximum of 5% of the range of the uncontrollable parameter in either direction, i.e., the environmental conditions do not change too much between iterations. Section 5.5 sets a to 10%, 25%, 50% and 100% and investigates the effect.

Plot **B)** in Figure 5.3 shows the performance of Algorithm 3 over the 30 repeats for the same three acquisition function as for the Levy function and compares it to the random benchmark. The mean absolute percentage error starts between 0.85 and 0.90 for all four algorithms. After 100 function evaluations, the mean absolute percentage error between the prediction from the Gaussian process and the true optimal value, using Algorithm 3 with expected improvement and upper confidence bound taking $\beta = 8$, are 0.07 and 0.06 respectively—much better than the random benchmark with 0.24. However, the algorithm using log expected improvement, with a mean absolute percentage error of 0.18 after 100 evaluations, performs only slightly better than the benchmark and significantly worse than the other versions of Algorithm 3. The wide 95% confidence intervals for log expected improvement indicate that Algorithm 3 with log expected improvement is not robust in this case, making it less reliable than expected improvement and upper confidence bound. The violin plots of the difference between the mean absolute percentage error of the benchmark and the three variations of Algorithm 3 for all 30 replications, given in plot **D)** in Figure 5.3 indicate that expected improvement performs the best with almost all replications better than the random benchmark. Log expected improvement, on the other hand, performs comparably to the benchmark on average but has a large spread, with some replication performing much worse. Upper confidence bound performs similarly to expected

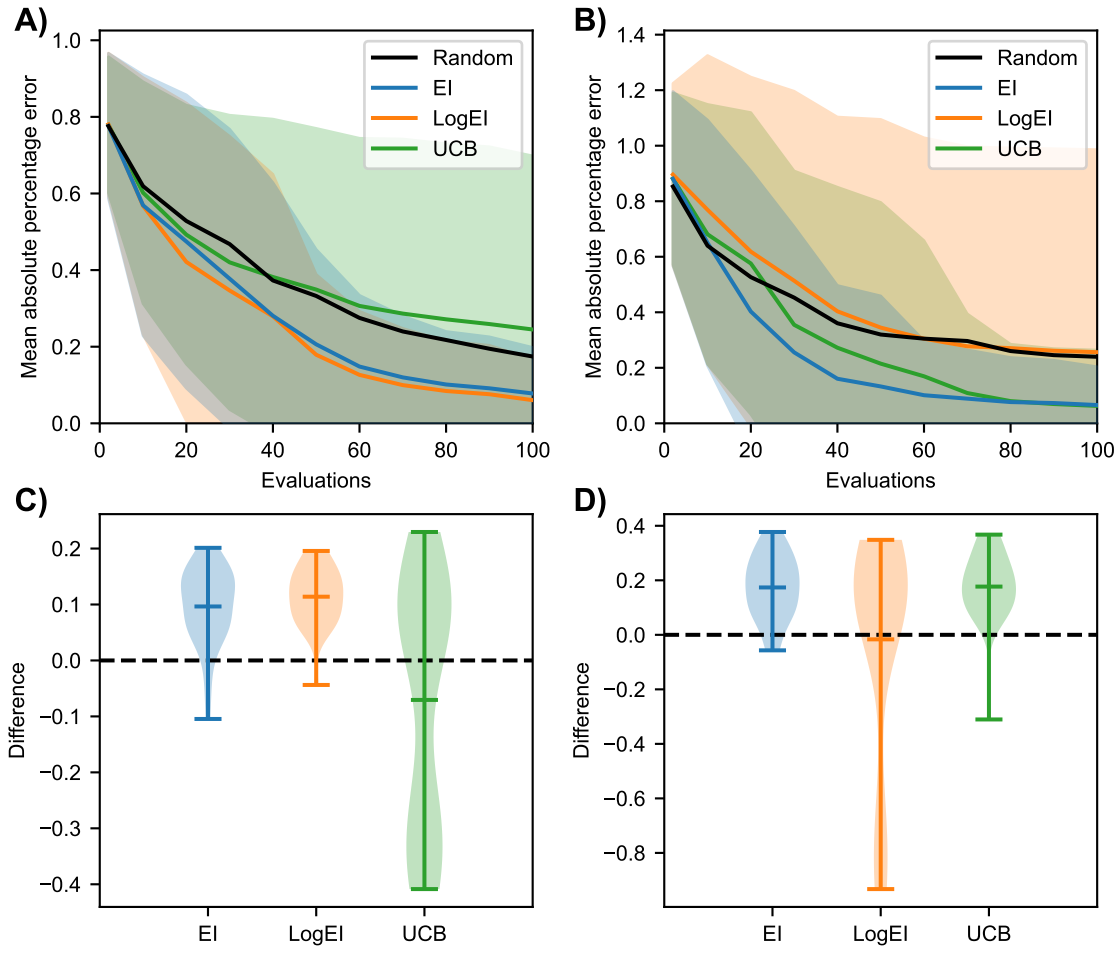

Figure 5.3: Upper row: Means (lines) and 95% confidence intervals (shaded areas) of the mean absolute percentage error between Gaussian process prediction and truth over 30 replications. Lower row: Violin plots of differences between algorithms and random benchmark after 100 function evaluations for each of the 30 replications. Two-dimensional Levy function with one uncontrollable parameter on the left and six-dimensional Hartmann function with one uncontrollable parameter on the right.

improvement but has an outlier that performs much worse than the random benchmark. Overall, expected improvement is the most robust method and is not prone to outliers. Despite these differences between algorithms, results after 100 evaluations for all three algorithms are significantly different from the random benchmark at a 1% significance level: the p-values from Mann-Whitney U tests are <0.001 for the expected improvement and the upper confidence bound version of Algorithm 3 and 0.007 log expected improvement. While this shows that the results of all methods are significantly different from the benchmark, only expected improvement and upper confidence bound perform better than the benchmark, indicated by the better average mean absolute percentage error.

Figure 5.4 shows results for different values of the trade-off parameter β. While the

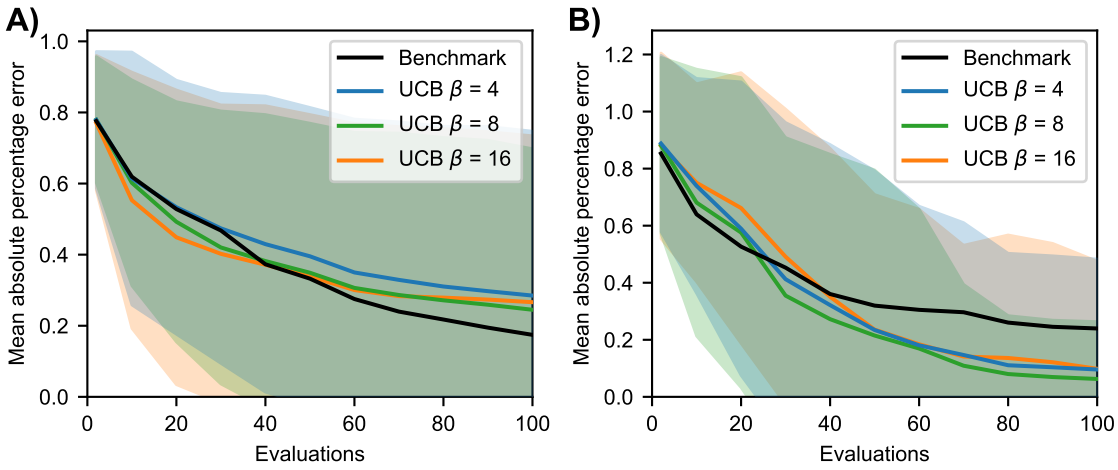

Figure 5.4: Comparison of different trade-off parameters β for the upper confidence bound acquisition function. Means (lines) and 95% confidence intervals (shaded areas) of the mean absolute percentage error between Gaussian process prediction and truth over 30 replications. **A)** shows the two-dimensional Levy function with one uncontrollable parameter and **B)** shows the six-dimensional Hartmann function with one uncontrollable parameter.

average performance is very similar after 100 evaluations, there is a difference in the 95% confidence intervals for the Hartmann function. $\beta = 8$ performs better than $\beta = 4$ and $\beta = 16$, indicating no clear trend with increasing or decreasing β.

5.5 Empirical analysis of properties

Based on the investigations of Section 5.4, we define ENVBO as a version of Algorithm 3 that uses the expected improvement acquisition function. This section explores the effect of changes to five aspects of the underlying objective function or environmental conditions on ENVBO using the negated Hartmann function. The effect of adding different levels of random Gaussian noise to the function's output, the influence of more than one uncontrollable variable, the effect of the fluctuation level (i.e. the step size a of the random walk), the impact of the variability in the uncontrollable variable, and the relationship between the algorithm performance and the effective domain size of the uncontrollable variables are investigated. An off-the-shelf version of ENVBO is available via the open-source Python package NUBO [42].

To make the comparison fair, the 30 replications of maximising the Levy and the Hartmann function with the different acquisition function used the same 30 initial starting points and random walks. For the random walks, this is achieved by sampling changes in percentages and scaling them by the maximal step size a, rather than sampling absolute values directly. Most physical experiments in engineering can only be conducted by introducing some noise, such as measurement uncertainty, that cannot be eliminated entirely. This section repli-

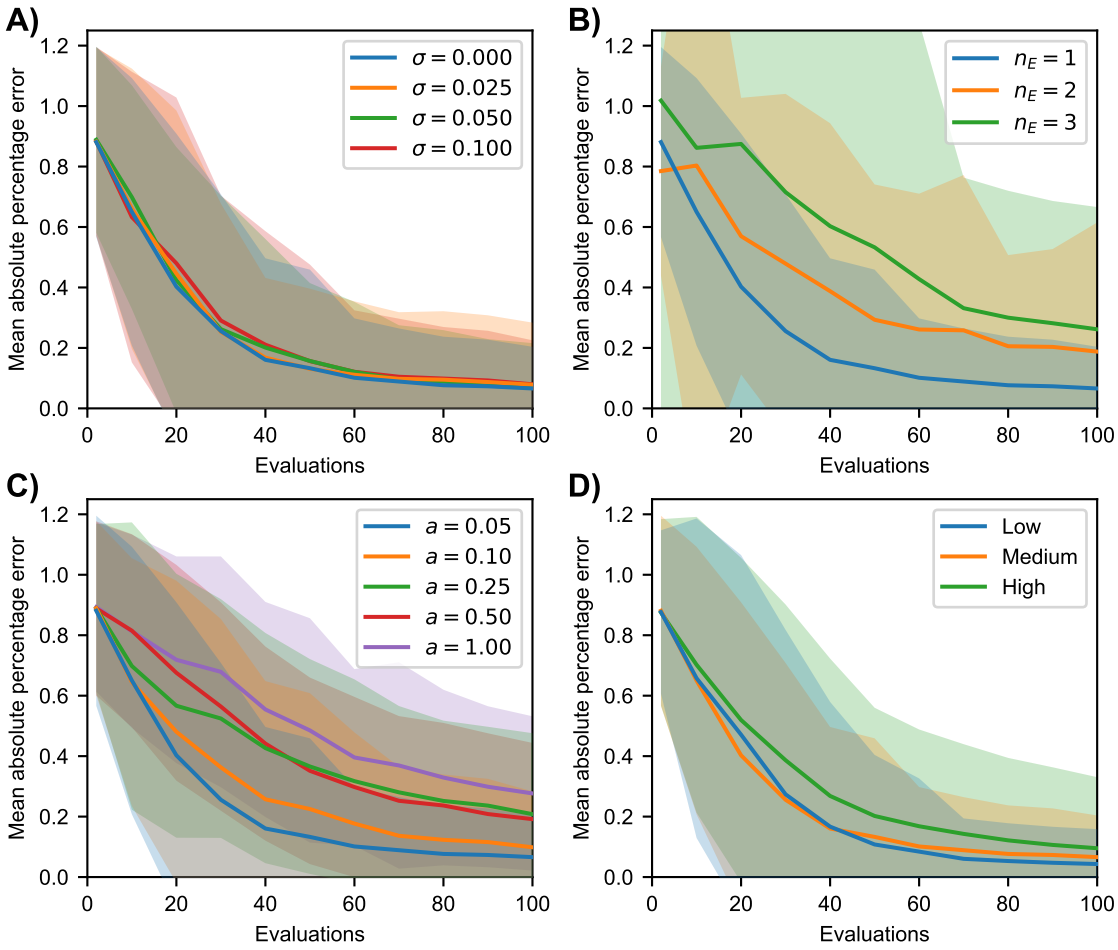

Figure 5.5: Means (lines) and 95% confidence intervals (shaded areas) of the mean absolute percentage error between the predictive mean of the Gaussian process and the truth over 30 replications for the six-dimensional Hartmann function. **A)** Comparison of randomly added noise levels, $\mathcal{N}\left(0, \sigma^2\right)$. **B)** Comparison of different numbers of uncontrollable parameters n_E. **C)** Comparison of five different step sizes a for the random walk $\mathcal{U}_{[-a,a]}$ added to the previous uncontrollable value. **D)** Comparison of uncontrollable variables with different parameter variability.

cates this situation by adding randomly generated noise ϵ to the Hartmann function. The noisy function can be defined as $g\left(\boldsymbol{x}\right) = f\left(\boldsymbol{x}\right) + \epsilon$, where $f\left(\boldsymbol{x}\right)$ is the deterministic negated Hartmann function from Equation (5.4.2). The noise is sampled from a Normal distribution centred around zero with a small standard deviation σ, such that $\epsilon \sim \mathcal{N}\left(0, \sigma^2\right)$. The simulations explore noise levels with $\sigma = 0.00$, $\sigma = 0.025$, $\sigma = 0.050$, and $\sigma = 0.100$. Considering the range of the Hartmann function, this corresponds to standard deviations of 0.75%, 1.50% and 3.00% of the full output range, respectively. This means that for any of these three cases, 68.3% of the added noise values will fall between $\pm 1\sigma$, 95.5% fall between $\pm 2\sigma$ and 99.7% fall between $\pm 3\sigma$. For $\sigma = 0.100$, this translates into noise values that decrease or increase the output by up to 3.0% of the output range 68.3% of the time,

by 6.0% of the output range 95.5% of the time, and by 9.0% of the output range 99.7% of the time. Plot **A)** in Figure 5.5 shows the performance of ENVBO for each of these four noise levels. The results indicate no clear difference in the average performance or the 95% confidence intervals between the four cases. Overall, the proposed method is not sensitive to adding modest noise levels.

For some experiments, there might be more than one influential uncontrollable variable. Plot **B)** of Figure 5.5 provides results with one, two and three uncontrollable variables. In each case, the overall dimensionality of the problem stays the same at $n = 6$. For $n_E = 1$, input six of the Hartmann function from Equation (5.4.2) is assumed uncontrollable, while input one is added to the uncontrollable variables for $n_E = 2$, and inputs one and four are added for $n_E = 3$. The inputs used as environmental variable were chosen at random. The number of test points used to evaluate the final Gaussian process model is increased from 25 to 50 for $n_E = 2$ and 75 for $n_E = 3$. The results show that the mean absolute percentage error increases with increasing numbers of environmental variables. Particularly, the 95% confidence intervals widen significantly. This result is expected as the input space of the environmental variables grows exponentially with n_E and requires exponentially more training points to cover the input space equally well as for lower n_E. Thus, more evaluations are required to achieve similar results.

Uncontrollable variables will fluctuate to different extents from one evaluation to the next, for example, when measured in physical experiments. Higher fluctuations cause big jumps in the uncontrollable variable values, while uncontrollable variables will be more stable for lower fluctuations. Plot **C)** of Figure 5.5 shows results for five different fluctuation levels implemented by varying parameter a of the uniform distribution in Equation (5.1)—the higher a, the higher the fluctuation of the uncontrollable variable. The results show that Algorithm 3 performs better for lower a, and the performance of the final Gaussian process models decreases with increasing fluctuation.

Parameter variability is closely linked to fluctuation and indicates how quickly the parameter value changes when moving along the axis. Uncontrollable variables with a low variability will only change slightly, while variables with a high variability will change considerably. Value changes of two variables could differ considerably for the same fluctuation level when they have different levels of parameter variability. As a proxy for the parameter variability, the length scales of a well-fitting Gaussian process over the full domain are considered. The length scale of a parameter quantifies the distance over which outputs are correlated when moving along the parameter's axis [69]. Consider, for example, a parameter with a large length scale. When this parameter value is changed by a certain amount, a relatively small change is expected in the output—provided everything else stays the same. The change in the output is expected to be larger for a parameter with a small length scale. For the analysis, a Gaussian process is fitted to 2000 data points

sampled from a Latin hypercube [129, 86] to achieve a well-fitting model. The resulting length scales for all six parameters in order are 1.30, 1.90, 4.81, 1.48, 1.51, and 1.47. The first input has the lowest length scale, suggesting that outputs are only correlated for a short distance when moving along its axis. This means that the parameter variability of the first input is high. In contrast, the third input has the highest length scale, indicating a low parameter variability. Moving along its axis, less change in the output is expected for the third input than for the first input. Plot **D)** of Figure 5.5 compares the first (high), third (low) and sixth input (medium) when chosen as the uncontrollable variable. Differences in the results for the low and medium parameter variability cases are minimal. At the same time, there is some difference compared to the results for the parameter with a high variability: the confidence interval is noticeably wider, and the average of the mean absolute percentage error is slightly worse, especially for evaluations 30 to 80.
Lastly, the relationship between the size of the effective domain, that is, the actual searched input space of the environmental variables, and the performance of the Gaussian process is investigated. Figure 5.6 uses the same data as Figure 5.5 but plots the effective parameter domain against the mean absolute percentage error, which provides a proxy for the performance of the Gaussian process. Each point reflects one individual replication of the 30 performed replications. The trend lines in each plot show a positive relationship between the effective domain and the mean absolute percentage error. Small effective domains generally correspond to small mean absolute percentage errors, while large effective domains generally correspond to large mean absolute percentage errors. Only plot **B)** that provides the effective domain for different numbers of uncontrollable parameters against the mean absolute percentage error shows almost no relationship.

5.6 Application to a wind farm simulator

The power generation of wind farms is highly dependent on the wind speed, the wind direction and the placement of the individual wind turbines within a particular site [68, 160]. Typically, interest lies in finding a strategy, such as the optimal wind turbine positions, that maximises the energy production conditional on either constant or variable wind speeds and directions [137, 28, 149]. Optimisation algorithms such as TOPFARM[3] can be used when function evaluations are cheap, while methods mentioned previously [209, 190, 198] present a cost-effective alternative when function evaluations are expensive. However, these methods cannot find multiple solutions for different environmental conditions within one optimisation run, which is the objective of this section. Specifically, this application aims to find (a) the optimal wind turbine positions conditional on

[3]See `https://topfarm.pages.windenergy.dtu.dk/TopFarm2/index.html` for further details on TOPFARM.

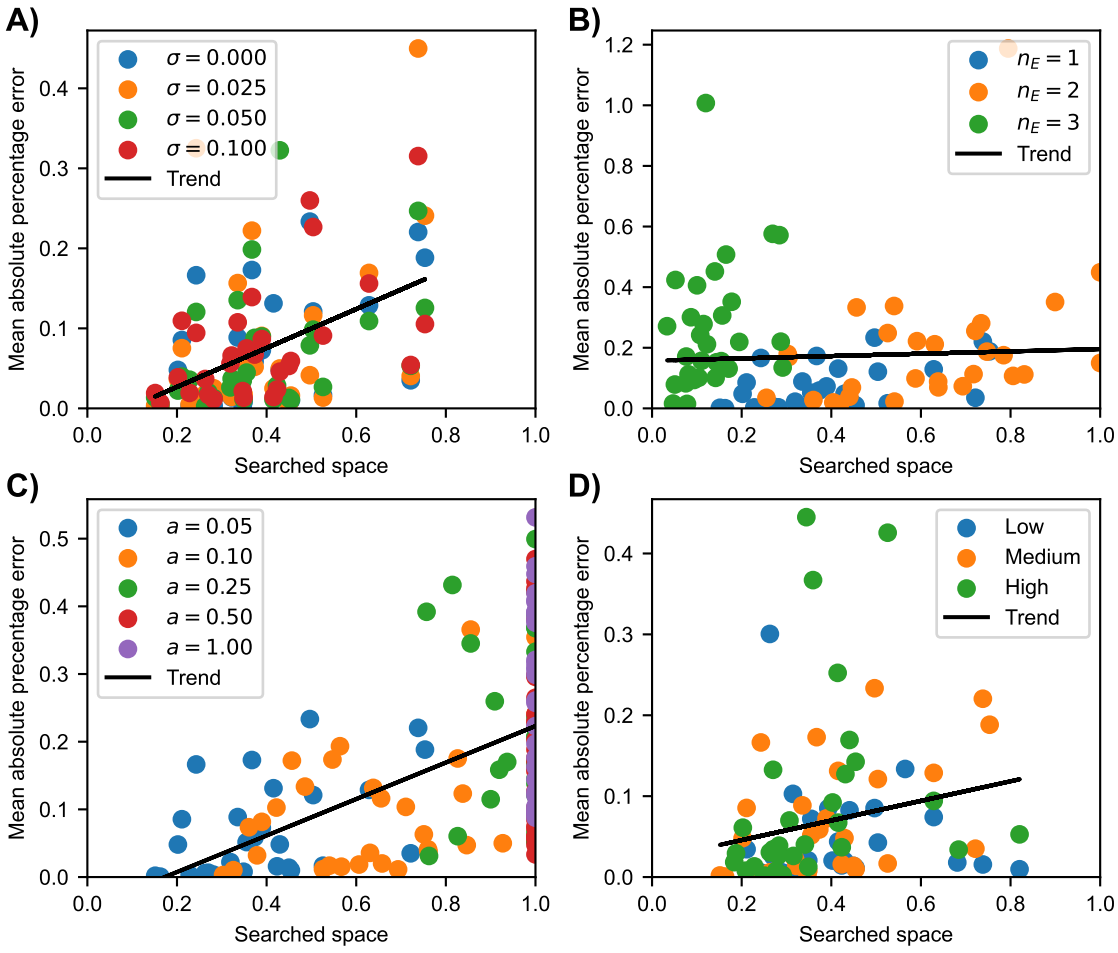

Figure 5.6: Relationships between the actual effective domain of the uncontrollable variables and the mean absolute percentage error of an individual run for the six-dimensional Hartmann function. **A)** Comparison of randomly added noise levels, $\mathcal{N}(0, \sigma^2)$. **B)** Comparison of different numbers of uncontrollable parameters n_E. **C)** Comparison of five different step sizes a for the random walk $\mathcal{U}_{[-a,a]}$ added to the previous uncontrollable value. **D)** Comparison of uncontrollable variables with different parameter variability.

randomly changing wind directions and (b) the optimal derating strategy—i.e., running individual wind turbines below their maximum capacity for a given wind speed—for a row of five wind turbines conditional on randomly changing wind speeds, resulting in one solution for each possible wind direction and speed, respectively. While the wind speed and direction are assumed to be fixed for investigations (a) and (b), respectively, they could also be randomly changing. The result would be solutions for all combinations of wind direction and wind speed.

The top row of plots of Figure 5.7 shows the effect of the wind direction on the local wind speed for a complex underlying terrain while the global wind speed is fixed at 6 m/s. The locations of high local wind speeds—necessary for high energy production—

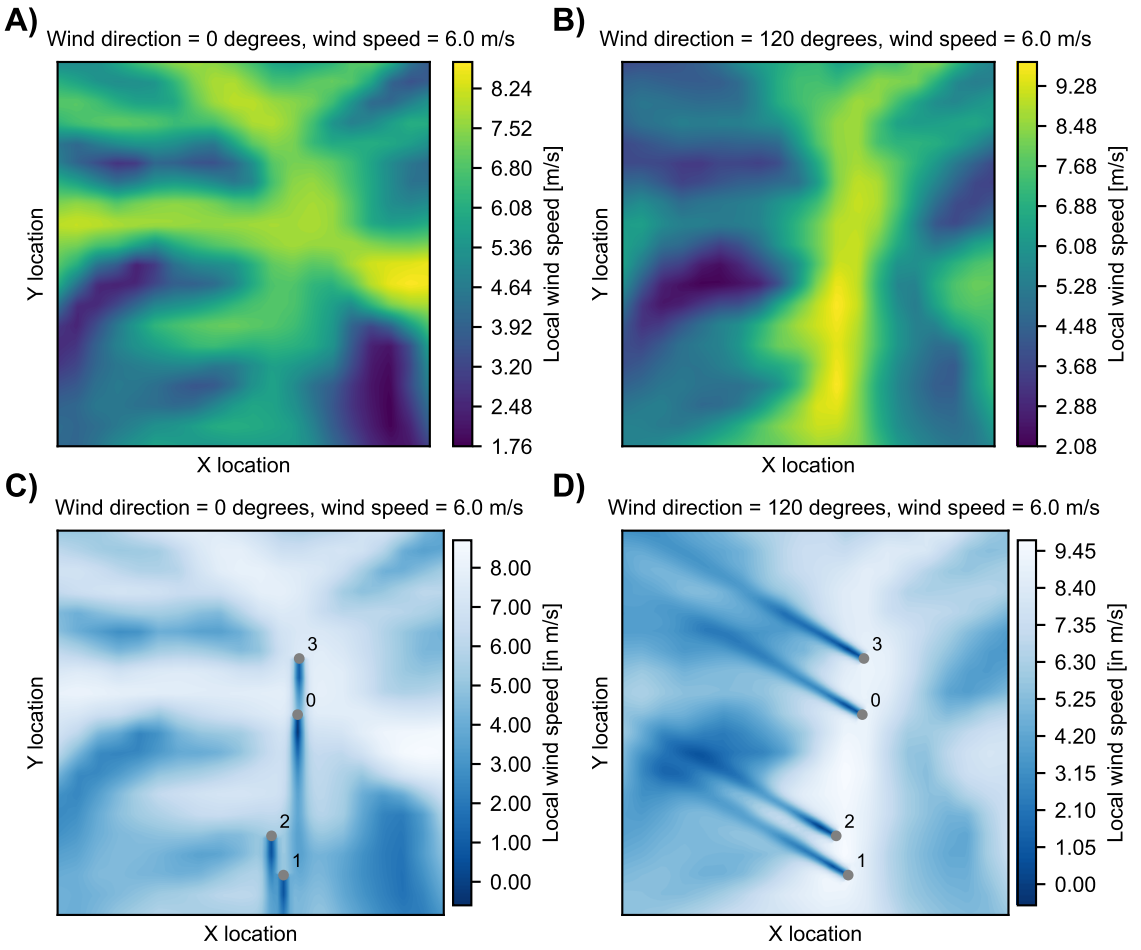

Figure 5.7: Wind farm simulator. Upper row: Local wind speed over the complex terrain for a wind direction of 0 and 120 degrees. Lower row: Wake of four wind turbines for a wind direction of 0 and 120 degrees. Wind speed is fixed at 6 m/s.

shift significantly between a wind direction of 0 and 120 degrees. While there is a band of high local wind speeds down the centre of the X location for the latter, it rotates roughly 90 degrees for the former. The plots show that ideal wind turbine positions will likely differ for the two wind directions. Another critical factor for maximising energy production is the wake of the wind turbines (lower row of Figure 5.7). While the wakes of the wind turbines located upstream do not affect wind turbines located downstream for a wind direction of 120 degrees, the wake of wind turbine 3 for a wind direction of 0 degrees heavily affects wind turbine 1. This shows that although a wind farm topology can be ideal for one wind speed, it can be suboptimal for another.

5.6.1 Wind turbine placement

In this section, a synthetic site is considered with complex terrain as shown in Figure 5.7, on which four wind turbines are to be placed to maximise the annual energy production (AEP). The wind direction is an environmental variable that varies according to a random walk as defined in Equation (5.1) where 90 and 135 degrees are the lower and upper bounds, respectively, and wind direction can change by ± 5 degrees from one iteration to the next. The global wind speed is fixed at 6 m/s. This results in a nine-dimensional problem with eight controllable parameters—one X and one Y location for each of the four wind turbines—and the wind direction as the only environmental variable. Additionally, the positioning of the wind turbines is constrained such that wind turbines have to be at least 160 metres apart to prevent the blades from colliding. Simulations are performed with PyWake [154] and use Vestas V80 wind turbines that can produce 2 MW of energy and have a blade diameter of 80 metres. Simulating the power generation of wind farms is a complex task which requires an expensive simulator that accounts for many different parameters. However, the main objective of this example is to illustrate the performance of ENVBO and enable other researchers to replicate and validate this work. Thus, a relatively simple simulator was chosen to make the results easily reproducible, so that ENVBO can be used with confidence on more expensive physical experiments and computer simulators.

ENVBO is run for a function evaluation budget of 200 and benchmarked against regular Bayesian optimisation (Algorithm 1) and the SLSQP optimisation algorithm [107]. Regular Bayesian optimisation, referred to as BO in the following paragraphs, uses expected improvement as its acquisition function to make comparisons fair. While ENVBO can return one solution for each wind direction after one optimisation run, BO and the SLSQP algorithm must be run for each possible wind direction. To compare the algorithms, four wind directions are chosen—90, 105, 120 and 135 degrees—and BO is restricted to 50 function evaluations each to reach the same function evaluation budget as ENVBO. The function evaluations of the SLSQP algorithm cannot be restricted, and the algorithm is therefore run until convergence. BO and SLSQP are run for fixed wind directions, so they do not use the random walk, in contrast to ENVBO. This makes converging towards a solution easier, as the algorithms can control all influential variables. ENVBO and BO were implemented via the open-source package NUBO [42], and an off-the-shelf version of ENVBO is available at `www.nubopy.com`. The SLSQP algorithm was implemented via the SciPy package [201]. All code for optimising the wind farm simulator is available at `https://github.com/mikediessner/environmental-conditions-BO`.

Figure 5.8 shows the results for all three algorithms and all four wind directions. The dashed circles around the wind turbine positions indicated with crosses represent the placement constraint—no wind turbine of one colour can be placed within the dashed circle of another wind turbine with the same colour. For a wind direction of 90 degrees,

ENVBO performs best with an annual power production of 4.20 GWh, followed by BO and SLSQP with 3.51 and 2.42 GWh, respectively. This is a 20% improvement over BO and a 74% improvement over SLSQP. SLSQP performs much worse than ENVBO and places at least one turbine in a suboptimal area with low local wind speeds, possibly due to converging towards a local maximum. While BO places three turbines in areas with high local wind speeds, it places one within a suboptimal area. For a wind direction of 105 degrees, ENVBO with 4.70 GWh performs substantially better than BO and SLSQP, with them achieving 0.91 and 1.86 GWh less power generation, respectively—a 24% and 65% improvement. A similar result is achieved for a wind direction of 120 degrees. ENVBO outperforms BO and SLSQP by 0.81 and 2.4 GWh or 19% and 88%, respectively. The results of the three strategies are closest for a wind direction of 135 degrees. This is the only instance where a benchmark beats ENVBO—in this case BO with an AEP of 2.88 GWh. ENVBO achieves 0.29 GWh (10%) less AEP but still outperforms SLSQP by 0.13 GWh (5%). However, the differences are much smaller than for any of the first three wind directions. Considering Figure 5.8, ENVBO is the only strategy that consistently places all four wind turbines in the band of high local wind speeds located down the centre of the X location. At least one turbine is placed off to one side of this band for all other methods.

While the results in Figure 5.8 show that ENVBO outperforms BO and SLSQP in almost all instances, they do not consider the different numbers of function evaluations required by each algorithm. Figure 5.9 plots the annual energy production and the number of function evaluations against the wind direction. The lower plot shows that SLSQP uses the most function evaluations by far, with 323, 294, 324 and 301 evaluations for the four wind directions—a total budget of 1,152 function evaluations. In contrast, ENVBO and BO both use 200 function evaluations in total. BO divides this budget equally over the four wind directions, allocating 50 evaluations per wind direction, while they are distributed via a random walk for ENVBO. Each of ENVBO's bins in the lower plot contains two to 24 function evaluations. Compared to SLSQP, both Bayesian optimisation algorithms are much more sample-efficient and use less than 20% of its evaluation budget. ENVBO uses less than half the evaluation for the four wind directions compared to BO, but it outperforms BO for three of the four wind directions, as shown in the upper plot. This shows the advantage of a global surrogate model that is fitted to all controllable and environmental variables. The Gaussian process uses all available information and can model the effect of the environmental conditions.

Although the improved performance of ENVBO compared to BO and SLSQP is already valuable, the main advantage lies in ENVBO's capability to produce a solution for each possible wind direction. This is illustrated by the 51 solutions given for ENVBO in the upper plot of Figure 5.9. SLSQP and BO can only give solutions for the specific wind

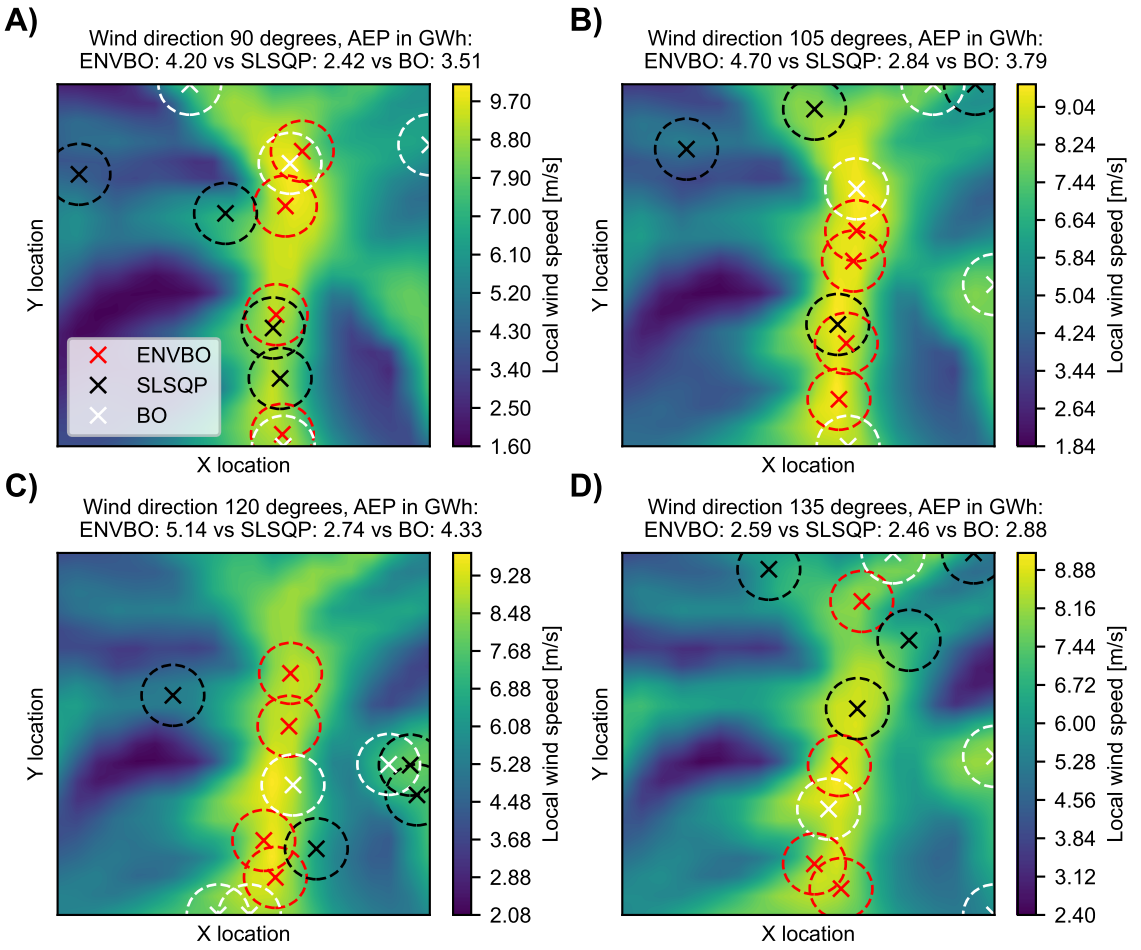

Figure 5.8: Annual energy production and placement of four wind turbines with spacing constraints—for each colour the wind turbine positions indicated by the crosses cannot fall into the dashed circles of another wind turbine. ENVBO (Algorithm 3) is benchmarked against SLSQP and BO (Algorithm 1).

direction for which they were run. To achieve similar results, SLSQP and BO need to be rerun for each wind direction, which would multiply the required function evaluations many times. ENVBO uses the available function evaluation budget much more effectively and is a more sample-efficient and cost-effective approach.

The annual energy production of a wind turbine for constant wind speeds is highly dependent on the wind direction as the wind direction affects the optimal placement within the terrain and relative to other wind turbines due to their wakes (Figure 5.7). Thus, different optimal wind turbine positions are expected for different wind directions, and optimal positions for one wind direction cannot be directly transferred to other wind directions. This is reflected in the results in Figure 5.8 where four wind turbines are placed in different positions for four wind directions. Adjusting for different wind directions is

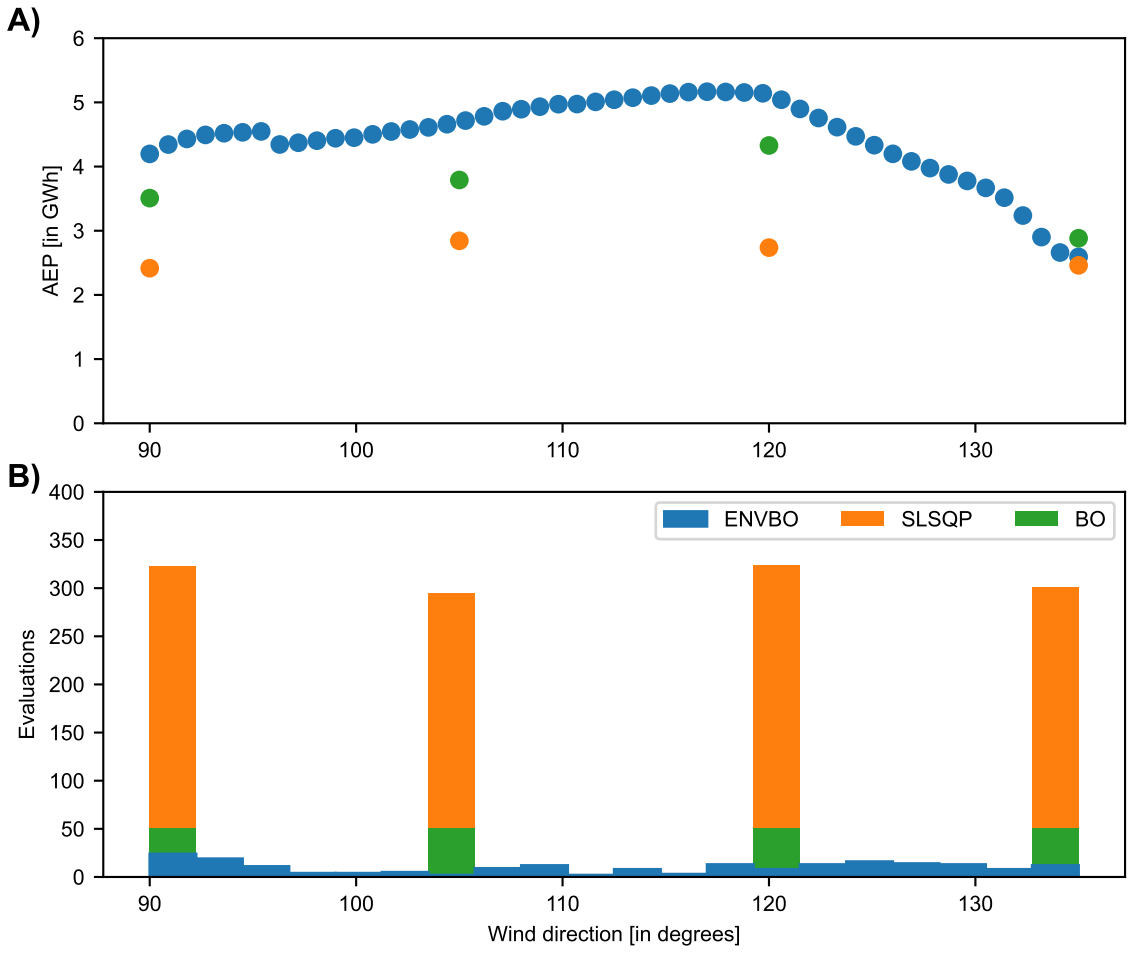

Figure 5.9: Results of ENVBO against two benchmarks—the SLSQP algorithm and standard Bayesian optimisation (Algorithm 1). **A)** Annual energy production in GWh for different wind directions. Only ENVBO can predict solutions over the full wind direction range. **B)** Number of evaluations per wind direction.

not a trivial problem and an advanced optimisation strategy, such as ENVBO, is required as shown in Figure 5.9. Overall, ENVBO finds solutions over the whole range of the environmental variables, while in most cases performing better than algorithms that focus on one fixed environmental variable at a time. This is even true for ranges of the environmental variable that are only explored briefly, as the algorithm learns the effect of the environmental variable from adjacent areas. Furthermore, ENVBO uses only a small fraction of the evaluation budget of the two benchmarks, making it particularly beneficial for optimising computer simulators and physical experiments that can be very expensive to run in engineering.

5.6.2 Wind turbine derating

This section illustrates the performance of ENVBO by considering a 5-dimensional wind turbine simulator implemented in PyWake [154]. The use case assumes a row of five wind turbines with the wind blowing directly towards the turbines so that previous wind turbines' wakes affect subsequent wind turbines' energy production. The objective is to maximise the mean energy production (MEP) over the five wind turbines by derating one or multiple wind turbines. The derating reduces the wake of the turbine and increases the potential energy generation of wind turbines affected by the wake downstream. Derating can also extend the lifetime of the wind turbine components [14, 95, 199]. Figure 5.10 shows three different strategies and their resulting mean energy production for a fixed wind speed of 18 m/s. Strategy **A)** runs all five wind turbines at full capacity without derating and produces 32.08 MW. Strategy **B)** only runs the first wind turbine at full capacity and turns the four other wind turbines off. The wake of wind turbine 0 would affect the energy production potential of wind turbines 1 and 2, as the lower local wind speeds show. This strategy produces 16.63 MW. Strategy **C)** derates wind turbines 1 and 3 by 60% and runs the other three wind turbines at full capacity. This produces 35.28 MW of energy and is superior to Strategy **A)**, showing that naively running all wind turbines at full capacity is not necessarily the best approach.

This application uses an imagined wind turbine with a height and diameter of 100 metres for which the effect of derating is computed using 1-dimensional momentum theory as outlined in the documentation of PyWake [154]. The maximum energy generation of a wind turbine is reached at 20 m/s. The five derating levels are controllable variables bounded between 0–100% while the wind direction is fixed at 270 degrees, and the wind speed is an environmental variable that changes according to a random walk [148]. The random walk uses the wind speed of the previous run and adds to it a small value sampled from a uniform distribution $\mathcal{U}[-5, 5]$. The maximum change from one iteration to the next is ± 5 m/s, and the wind speed is bounded between 6 and 50 m/s.

ENVBO is benchmarked against two optimisation algorithms—the well-established Nelder-Mead algorithm [143] and standard Bayesian optimisation with expected improvement, as outlined in this chapter. While ENVBO can predict optimal derating combinations for the entire range of wind speeds, Nelder-Mead and standard Bayesian optimisation can not and must be run for each wind speed, keeping it fixed for the entire optimisation run. Five different wind speeds—6, 17, 28, 39 and 50 m/s—are taken as samples to compare ENVBO to the benchmarks. That is, ENVBO is run over the entire wind speed range between 6–50 m/s, but Nelder-Mead and standard Bayesian optimisation can run separately for these five fixed wind speeds. The results of this comparison are given in Table 5.1. ENVBO outperforms Nelder-Mead by 17.8%, 2.2%, 21.9%, 60.0% and 21.8% for each wind speed, respectively. Compared to standard Bayesian optimisation, ENVBO performs better for

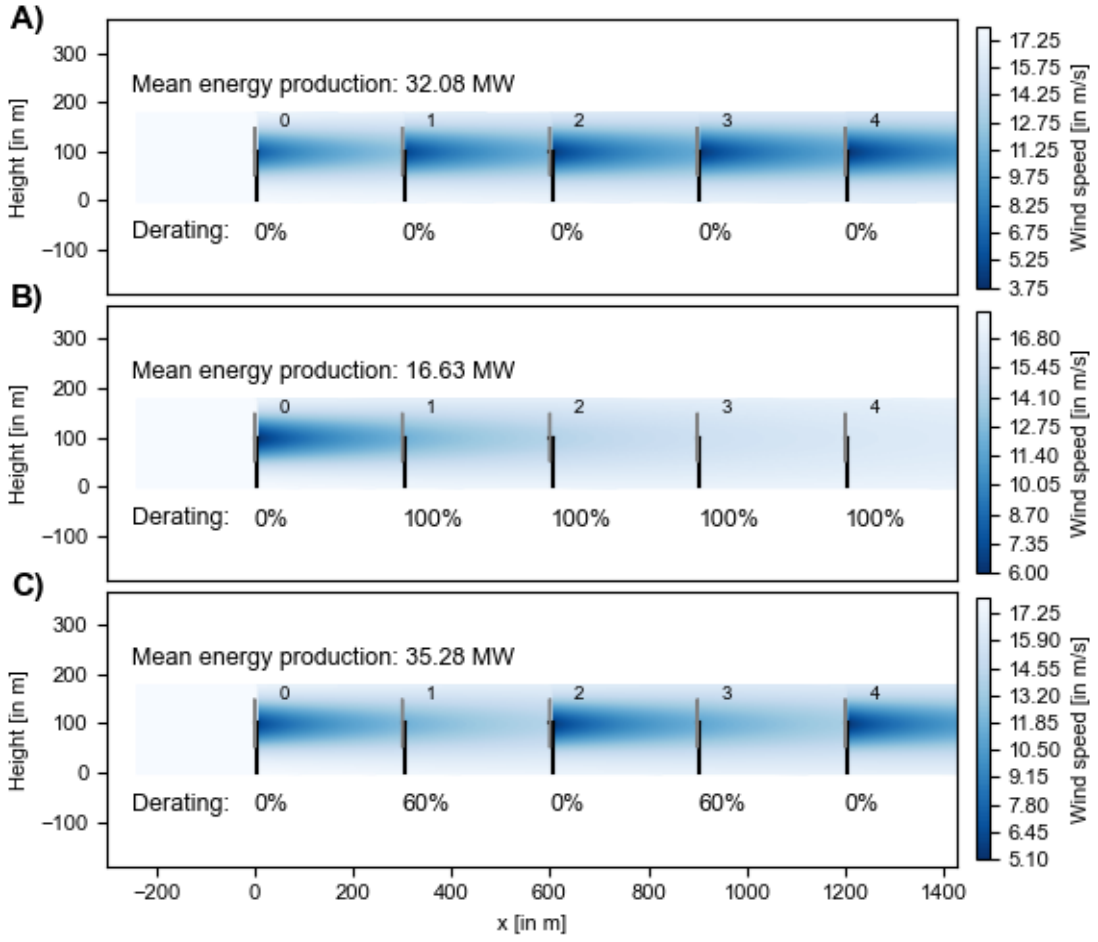

Figure 5.10: Mean energy production in MW for three different derating strategies with a wind speed of 18 m/s. **A)** All five wind turbines are run at maximum capacity (no derating). **B)** Wind turbine 0 is run at full capacity, and wind turbines 1–4 are turned off completely. **C)** Wind turbines 0, 2 and 4 are run at maximum capacity and wind turbines 1 and 3 are reduced to 60% capacity.

wind speeds of 28 and 39 m/s and slightly worse but comparably for wind speeds of 6, 17 and 50 m/s. ENVBO also finds the highest overall mean energy production at 105.99 MW for a wind speed of 39 m/s.

Figure 5.11 illustrates these results through two plots. Plot **A)** gives the mean energy production in MW for ENVBO, Nelder-Mead (NM) and standard Bayesian optimisation (BO). This shows that ENVBO and standard Bayesian optimisation come to very similar results, but Nelder-Mead is clearly worse—particularly for wind speeds 28, 39 and 50 m/s. It also presents the difference in the types of solution between ENVBO and the benchmarks. ENVBO can produce solutions for any wind speed—here depicted by a line—while the benchmarks only give results for the five specific wind speeds they are trained on. Plot **B)** connects this performance with the number of function evaluations

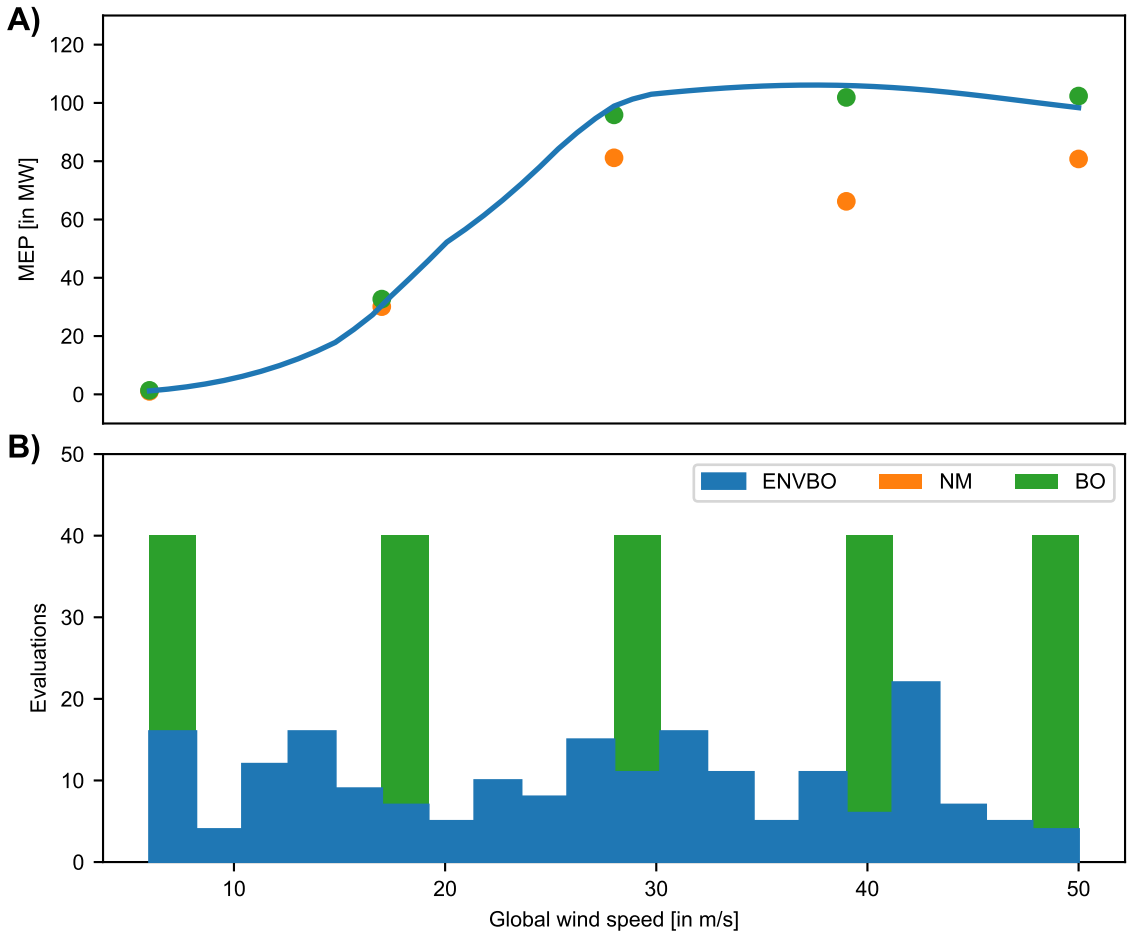

Figure 5.11: Results of ENVBO against two benchmarks—the Nelder-Mead algorithm and standard Bayesian optimisation (Algorithm 1). **A)** Mean energy production in MW for different wind speeds. Only ENVBO can predict solutions over the full wind speed range. **B)** Number of evaluations per wind speed.

necessary to get the results. ENVBO was limited to a budget of 200 evaluations, and the individual runs of Nelder-Mead and standard Bayesian optimisation were restricted to 40 function evaluations to achieve the same evaluation budget as ENVBO. Plot **B)** shows that the benchmarks have more evaluations available for the five fixed wind speeds. ENVBO's largest number of evaluations falls between wind speeds 41.2–43.4 m/s with 22 evaluations. For the other bins of the histogram, the number of function evaluations is as low as 4 for wind speeds of 8.2–10.4 m/s. Despite having fewer data points available per (bin of) fixed wind speed, ENVBO still finds solutions that outperform or are at least comparable to the benchmarks. For specific wind speeds, ENVBO leverages the correlation in the objective function values over the environmental variable and draws from the information of the surrounding wind speeds, requiring fewer data points to give a solu-

Table 5.1: Mean energy production of wind turbines optimised with ENVBO, the Nealder-Mead algorithm and standard Bayesian optimisation for the five considered wind speeds.

Wind speed [in m/s]	ENVBO [in MW]	Nelder-Mead [in MW]	BO [in MW]
6	1.19	1.01	1.37
17	30.75	30.10	32.69
28	98.91	81.17	95.90
39	105.99	66.23	101.89
50	98.37	80.78	102.36

tion. This also explains why ENVBO performs more poorly for a wind speed of 50 m/s than standard Bayesian optimisation in this example. ENVBO has very few data points available for that wind speed, and, as 50 m/s is the upper bound of the wind speed, it can only draw from information from lower wind speeds. While ENVBO performs comparably to the best benchmark overall, it should be a point of caution that ENVBO might be less accurate towards the boundaries of the environmental variables, and extrapolation should be particularly avoided. Overall, ENVBO presents as a sample-efficient and cost-effective optimisation strategy for experiments and simulations with changing environmental variables that performs well compared to well-established benchmarks.

5.7 Discussion

This section discusses the empirical results of Section 5.5 and the results of the application to the wind farm simulator in Section 5.6, interpreting the results, considering limitations and implications, and highlighting the use for the scientific community.

The effective optimisation of noisy objective functions in engineering applications has been considered from the earliest studies in Bayesian optimisation by Kushner [110] and later by Jones, Schonlau, and Welch [94] through their efficient global optimisation (EGO) algorithm, emphasising the importance of optimising noisy objective functions in engineering. Indeed, most experiments cannot be conducted without noise, and many simulators are stochastic. Gramacy and Lee [72] argue that even deterministic simulators can be seen as noisy as they are an approximation of the real world. Noise can be understood as the discrepancy between computer code and reality. This makes investigating the robustness of a proposed approach to noise essential. The empirical investigation in this chapter showed that no difference is observable between the average performance of ENVBO for deterministic and stochastic objective functions. The results show that ENVBO can optimise noisy objective functions, making it applicable to many experiments and simulators prevalent in engineering. However, the noise added to the six-dimensional Hartmann func-

tion remained mainly below 9% of the output variable range (see Section 5.5) and was assumed to be homoskedastic. Thus, one should be cautious when applying ENVBO to experiments with very high or heteroskedastic noise. A solution for the latter case could be to use a heteroskedastic Gaussian process for the surrogate modelling, which uses a second Gaussian process to model the noise [66].

The results regarding the number of uncontrollable parameters in Section 5.5 show that the performance decreases with an increasing number of uncontrollable parameters. This decrease in performance is reflected in higher mean absolute percentage errors and higher variability between replications. For example, the mean absolute percentage error after 100 evaluations is 12.2% and 19.6% higher for $n_d = 2$ and $n_d = 3$, respectively, than for $n_d = 1$. These results are not surprising since the uncontrollable input space grows exponentially with the number of environmental variables, making it more challenging to achieve even coverage over the uncontrollable space due to the randomness of the environmental variables. Furthermore, the problem of extrapolation with Gaussian processes increases with the growing number of dimensions of the uncontrollable input space. The test points used to evaluate the final prediction model given by ENVBO are generated with a Latin hypercube that uses the minimal and maximal values of all environmental variables as upper and lower bounds. With increasing numbers of the environmental variables n_E, it is very likely that while measurements for individual environmental variables fall within these bounds, the combination of these measurements for different evnironmental variables falls in areas not explored by the optimisation algorithm. The final Gaussian process model will extrapolate to predict outputs at these test points. As Section 2.3 discusses, extrapolation with Gaussian processes generally means that predictions default to the prior mean function. Gaussian processes with a more complex prior mean function can be explored [156] to counteract this issue. Moreover, the empirical investigation in this chapter assumes a single fluctuation level a for each environmental input. However, in a real-world application, different combinations of fluctuation levels are more realistic, which might require more function evaluations. In any case, the resulting prediction model must be carefully scrutinised to ensure a good fit and minimal extrapolation.

This work assumes that the environmental variables cannot be controlled and are randomly selected according to the uniform random walk in Equation 5.1. Thus, it is vital to scrutinise the performance of ENVBO based on this assumption and to consider different levels of fluctuation and parameter variability. Environmental variables are likely to fluctuate to different degrees. For example, temperature is likely to change more slowly than wind speed. Section 5.5 studies this by considering different levels of fluctuation and parameter variability, which are closely connected and might be challenging to distinguish in practice. Fluctuation, in our case, is defined as the size of a, the constant of the uniform random walk. Parameter variability is defined as the effect on the output for a given

change of the environmental variable. For higher fluctuations, there is more potential for larger effective domains of the uncontrollable variable, i.e., the range for which the environmental variables are explored. This can be explained by considering the parameter a. For $a = 1.00$, any domain value can be randomly selected at each optimisation step, and the whole domain will likely be searched. For $a = 0.05$, only values that differ by 0.05 from the previous evaluation can be randomly selected, making it less likely that the whole domain will be searched before the evaluation budget is exhausted. Suppose a Gaussian process is fitted to domains with different sizes, but the number of evaluations remains the same. In that case, predictions will typically be better for smaller domains. For example, it is plausible that a Gaussian process fitted to ten data points within the domain $[0, 0.1]$ will reflect the truth in this domain better than a Gaussian process fitted to ten data points within the much larger domain $[0, 1]$ assuming that everything else stays comparable. Similarly, a parameter with a small length scale and for which small changes affect the output significantly will be more challenging to model than a parameter with a long length scale. This is intuitive as a longer correlation length makes understanding the effect of changing the parameter easier. The results in Figure 5.5 show that the average performance and the variability in performance worsen with increasing fluctuation and parameter variability levels. Thus, the prediction reliability depends on these levels; higher levels require more function evaluations to perform similarly for environmental variables with less fluctuation. While these results seem intuitive, some limitations and caveats should be considered. First, the assumption that environmental variables change according to a uniform distribution will not generally be true. The uniform assumption can thus only be viewed as a proxy for the underlying distributions associated with arbitrary environmental variables. Second, as the environmental variables are selected randomly, the distribution of observed points could be uneven such that some regions will contain more observations than others. The final model will potentially be less accurate in regions with fewer observations. Hence, it makes sense to consider not only the prediction but also the associated uncertainty. If the uncertainty is considerable, we might not trust the prediction to the same extent. Lastly, the randomness in the environmental variables means there could be cases where the environmental variable is almost constant in the observed sample, and we cannot explore most of its range. One must be cautious not to extrapolate in these instances. A non-stationary environmental variable is an example of a case where this could be an issue.

The results of Figure 5.6 suggest an inverse relationship between the performance of ENVBO and the size of the effective domain—that is, the explored space of the environmental variables—where ENVBO performs better for smaller effective domains. Intuitively, this result makes sense as a Gaussian process should have a better fit for a smaller space than a larger space when the number of training points and the function in question stay the

same. The results depicted in plot **C)** of Figure 5.6 reinforce the reasoning that the larger the fluctuation parameter of the uniform distribution a, the more potential there is for a larger parameter space to be explored. Thus, it is plausible that ENVBO performs better with smaller values of a as the results of plot **C)** of Figure 5.5 suggest. Generally, the size of the environmental variable space grows exponentially with the number of environmental variables. Exponentially more evaluations are required to achieve the same coverage for $n_E = 3$ as for $n_E = 1$. However, in this section, the evaluation budget is fixed to 100 evaluations regardless of the number of environmental variables. Considering this, an inverse effect of the number of environmental variables and the effective domain would be expected. However, searched spaces are generally larger for $n_E = 1$ than for $n_E = 3$ but smaller than for $n_E = 2$. A possible reason for this is that the first environmental variable uses a fluctuation parameter $a = 0.05$ while the second and third environmental variables use $a = 0.1$. A change from $a = 0.05$ to $a = 0.1$ can significantly affect the domain size, as shown in plot **C)** of Figure 5.6, and might be responsible for the unusual order of effective domain sizes. In this investigation, results suggest that the size of the effective domain can inform the number of function evaluations required, and larger effective domains indicate that more function evaluations are necessary.

While these results are from the six-dimensional Hartmann function and might not be generalisable to all objective functions, the application of ENVBO to a simple wind farm simulator presented in Section 5.6 highlights the approach in an illustrative example. ENVBO outperforms the SLSQP algorithm in all cases and the standard Bayesian optimisation algorithm in three out of four cases for a nine-dimensional problem. For a second six-dimensional case study, ENVBO performs comparably to standard Bayesian optimisation and outperforms the Nelder-Mead algorithm [143]. However, the main advantages of ENVBO are that it uses fewer function evaluations per wind speed and direction than SLSQP, Nelder-Mead and standard Bayesian optimisation and makes predictions for the full range of wind speeds and directions, compared to the benchmarks that only make predictions for four wind speeds and five wind directions, respectively. These results suggest that ENVBO successfully leverages the correlation in objective function values over the environmental variable, resulting in a superior and more sample-efficient algorithm. The reason standard Bayesian optimisation can perform slightly better than ENVBO for a single wind speed and a single wind direction is likely that these values are at the upper limit of the environmental variable range, and ENVBO can only leverage information for lower wind speeds. Figures 5.9 and 5.11 show that very few function evaluations were observed for this upper limit with ENVBO, which was insufficient to yield a good model fit. As discussed previously, a more complex prior mean function could improve this result. Although this wind farm simulator is a simple representation of a complex real-world problem, it has characteristics of many engineering problems.

Together with the potential to optimise noisy objective functions, multiple environmental variables and different levels of parameter fluctuation and variability, ENVBO is a valuable tool for the broader scientific community. It does not require information about the correlation structure that is leveraged by other methods [108] and is available as an off-the-shelf algorithm as part of the Python package NUBO [42] that can conveniently be run on cloud services such as Google Colaboratory without requiring a Python installation on a local machine.

5.8 Conclusion

When optimising physical experiments, it is often the case that only some variables can be fully controlled, or interest lies in finding not one global optimum but one optimum for each value of a particular variable—essentially a function that maps environmental variable values to optimal values for the controllable parameters. To date, Bayesian optimisation has been used predominantly to find global optima. This chapter extends the Bayesian optimisation algorithm to situations with changing environmental conditions and makes the following contributions:

1. Development of ENVBO, an algorithm for optimising expensive black-box functions with randomly changing environmental conditions, that does not require any knowledge about the objective function or the environmental conditions.

2. Analysis of ENVBO and its sensitivity to noisy objective functions, the number of environmental variables and different levels of fluctuation and parameter variability.

3. Implementation of ENVBO as part of the Python package NUBO.

4. Illustration of ENVBO on two simple wind farm simulators and comparison to standard Bayesian optimisation, the SLSQP algorithm, and the Nelder-Meads algorithm.

Compared to related research in this area and particularly the closest related work of Krause and Ong [108], this chapter investigates and gives firm proposals and justification for the implementation of the algorithm and offers an easy-to-use implementation in Python. Furthermore, while Krause and Ong [108] show how known (linear or additive) structures can be exploited via composite kernels, ENVBO can also be used where no information on the behaviour of the objective function and the environmental variables is available. Lastly, ENVBO uses expected improvement as its acquisition function to guide the selection of candidate points. Compared to upper confidence bound as used in Krause and Ong [108], expected improvement performed far better in our simulations. It also has the advantage of not requiring the specification of a trade-off parameter to balance

exploitation and exploration that has the potential to negatively influence the effectiveness of the algorithm.

The proposed method fits a global surrogate model over all controllable and environmental variables and uses measurements of the environmental variables to optimise the acquisition function with regard to the controllable parameters. This conditional optimisation enables finding a model with a posterior predictive mean that provides close to optimal values of the controllable parameters for any values of the uncontrollable variables. The standard Bayesian optimisation algorithm assumes that the uncontrollable variables are fixed or disregards them entirely. Thus, ENVBO uses all available information about the objective function in one optimisation run and is more sample-efficient.

For the proposed optimisation strategy presented in Algorithm 3, three different acquisition functions—expected improvement, log expected improvement and upper confidence bound with different trade-off parameters β—were considered, and their performance compared on two synthetic test functions—the two-dimensional Levy function and the six-dimensional Hartmann function. Expected improvement performed best across the two test functions and was thus chosen as the acquisition function for the ENVBO algorithm for the case study. Furthermore, this chapter empirically investigated the properties of ENVBO, showing that the algorithm is effective in the scenarios of added noise and uncontrollable variables with high and low variability. When solving problems with more than one environmental variable, it is desirable that the final Gaussian process model is not used for extrapolation, as this can result in biased solutions. Additionally, it was found that uncontrollable variables with large fluctuations require more function evaluations than uncontrollable variables that fluctuate less. Higher fluctuation generally results in a larger effective parameter domain than lower fluctuation. When modelling, it is intuitive that larger areas require more observations—and thus information—than smaller areas to achieve identical results, assuming that all other properties are comparable.

ENVBO—an implementation of the proposed algorithm within the Python package NUBO [42]—was applied to two problems concerned with a wind farm simulator. The first problem aimed to place four wind turbines within an area with complex underlying terrain to find positions that maximise the annual power generation for different wind directions. The results were compared to two benchmarks—regular Bayesian optimisation via NUBO and the SLSQP algorithm via the SciPy package [201]—showing up to 88% better performance in all but one case across the whole range of possible wind directions, while keeping function evaluations, and thus costs, low. The second problem aimed to maximise the mean energy production by setting the derating strategy for a row of five wind turbines with changing wind speeds. ENVBO outperformed the Nelder-Mead algorithm by up to 60% and gave comparable performance to the standard Bayesian optimisation algorithm. However, ENVBO uses fewer function evaluations per value of the environmental vari-

able and produces solutions for its full range. Thus, ENVBO is a sample-efficient and cost-effective approach for optimising expensive experiments and simulators with uncontrollable environmental conditions.

In the future, we plan to apply ENVBO to more complex simulators and experiments and investigate how more complex mean functions can be used in the prior distribution of the Gaussian process to better represent areas of the input space where only a small number of observed points are available—and also to give a better representation for areas towards the edges of the ranges of the environmental variables. Section 5.7 discussed another possible extension of ENVBO to allow optimisation with heteroskedastic noise through implementing heteroskedastic Gaussian processes [66]. Furthermore, an interesting research direction is finding a method to evaluate if ENVBO should observe a point from the objective function for a given environmental variable value. There could be situations where enough data points have already been observed for an environmental condition, and we should refrain from observing another data point as it would only add very limited additional information. It might be more sample-efficient to wait for an environmental condition that has not been explored. Finally, we implicitly assume that the best observation of all previous training data points is a good target for the expected improvement function in Section 2.4.1. However, improving upon the best-observed output for any given environmental condition might not be possible. Consider the case where the single global optimum has been found, but we are still looking for local optima for different environmental conditions. No improvement would be possible in this case, and the acquisition function would suggest non-optimal candidates. A possible solution is to use the maximum posterior prediction for the given environmental condition as the target. This is similar to what Picheny et al. [155] and Gramacy [69] propose in the presence of noise.

Chapter 6

Conclusion

This thesis investigated Bayesian optimisation for expensive physical experiments and computer simulators with the overarching aim of furthering understanding, methodology and adoption.

6.1 Summary of contributions

Chapter 1 highlighted the importance of optimising these expensive black boxes and showed the potential benefits by introducing the example of reducing the turbulent skin friction drag close to an aircraft's surface, which would result in significant monetary, environmental and public health benefits. The chapter illustrated how optimising an expensive experiment mimicking the airflow over a flat plate could help find optimal drag reduction strategies. As conventional optimisation methods are not suitable for expensive derivative-free black-box optimisation, the introduction introduced methods based on surrogate modelling and, particularly, Bayesian optimisation. Chapter 2 formalised the problems and methods highlighted in the introduction mathematically and gave an overview of the related Bayesian optimisation literature. The following three chapters focused on the understanding (Chapter 3), adoption (Chapter 4) and methodology (Chapter 5) of Bayesian optimisation by addressing the objectives given in the introduction.
Chapter 3 conducted an empirical simulation study investigating key characteristics of Bayesian optimisation for expensive experiments and simulators on eight different test functions. Cheap test functions facilitated a thorough investigation and provided robust results, allowing 50 replications for each Bayesian optimisation algorithm. While results showed only a slight difference when varying the number of initial training data points—1, 5 and 10 data points per input dimension were considered—optimistic or confidence-bound acquisition functions performed best across all test functions. Furthermore, approximating the exact analytical acquisition function with Monte Carlo sampling showed comparable

results to the analytical case for single-point optimisation. For multi-point optimisation, Monte Carlo sampling enabled a natural way of computing batches of candidate points that performed better than heuristical strategies of the analytical multi-point acquisition functions.

Chapter 4 addressed the need for easily understandable and user-friendly software, allowing a broader adoption of Bayesian optimisation by non-experts. This is crucial because expensive black-box functions are prominent in many scientific fields, such as biology, chemistry, engineering and physics. However, researchers typically lack the knowledge of strategies for successful optimisation. Available packages were either overly complex or did not include all the building blocks necessary for optimisation in the sciences. Thus, NUBO was developed to close the gap between experts and users by providing a simple implementation of Bayesian optimisation with source code that can be easily verified with the provided references. Benchmarking showed that this simplicity does not come at the cost of performance, as NUBO achieved competitive, and even superior, performance compared to other popular Python packages. NUBO is published as an open-source software package and available from the Python Package Index (PyPI).

Chapter 5 developed a strategy for a challenge unique to physical experiments. In the investigated scenarios, environmental conditions can influence the output of an experiment while being uncontrollable. The chapter introduces ENVBO, a novel algorithm that allows the optimisation of problems with controllable and environmental variables. ENVBO's properties are investigated by optimising cheap test functions and benchmarking on a wind farm simulator against standard Bayesian optimisation, Nelder-Mead [143] and SLSQP [107]. For the simulator, the aim was to maximise the energy production for different wind speeds and wind directions. ENVBO was shown to be at least comparable to standard Bayesian optimisation, Nelder-Mead and SLSQP for selected environmental conditions while using fewer function evaluations per condition, i.e., per wind speed and wind direction. Furthermore, ENVBO can predict solutions for the entire range of the environmental variables.

In addition to these contributions, Bayesian optimisation and ENVBO were applied to computer simulators and physical experiments in active flow control to maximise the turbulent skin friction drag reduction. Chapter 3 found a significant drag reduction of over 22% for a four-dimensional travelling wave problem and very modest net energy savings of below 1% for a three-dimensional problem where a gap separates two blowing regions. Both problems suggest that uniform-blowing strategies were favoured. Furthermore, the modest net energy savings indicated a need for actuators with cheaper power consumption. Further experiments and simulations were conducted, and their results were published in O'Connor et al. [145] and Chen et al. [27].

6.2 Limitations and future work

The limited time available to research and write this thesis means that some research avenues were not (sufficiently) explored, and present a good starting point for future work.

Firstly, this work considered simple Gaussian processes with zero and constant mean functions. Results from Chapter 5 showed that these prior mean functions have the disadvantage of performing poorly when extrapolating as they default to their prior mean. More sophisticated prior mean functions, such as low-order polynomials [53], radial basis function [19], concave quadratic function [58] and trigonometric functions, could improve predictions.

Secondly, NUBO does not yet include all algorithms and methods available in the literature that are effective for optimising expensive physical experiments and computer simulators. More functionalities should be added in the future. At the same time, these additions should be carefully selected to ensure NUBO does not lose its compact and simple nature. Besides the more complex prior mean functions, methods to enable multi-objective, multi-fidelity and high-dimensional optimisation are potentially beneficial.

Thirdly, ENVBO was only applied to a wind farm simulator, where the environmental variables were artificially created using a random walk. It would be beneficial to apply it to expensive experiments and simulators that include actual environmental variables to validate the method further. This could give further insights into the ENVBO's performance as well as its shortcomings and strengths.

Fourthly, the growing adoption of physics-informed machine learning [101], including physics-informed Gaussian processes, presents a promising research direction. In the context of Bayesian optimisation, physics-informed models assume that while the objective function is a black box, some laws of physics apply that can be integrated into the surrogate model. In the particular case of Gaussian processes, prior knowledge of these laws can be implemented within the covariance kernel. Physics-informed models have the benefit of making modelling more robust and could improve optimisation. Indeed, initial investigations into physics-informed Bayesian optimisation report promising results [103].

Finally, high-dimensional optimisation with many input parameters is a highly researched and challenging area of Bayesian optimisation. In the literature, dimension reduction techniques such as linear embeddings proved to achieve good results (see Section 2.5.4) and should be investigated in the future.

Appendix A

Appendix of empirical investigation

A.1 Mathematical definition of test functions

6D Ackley

Function: $f(\boldsymbol{x}) = -a \exp\left(-b\sqrt{\frac{1}{6}\sum_{i=1}^{6} x_i^2}\right) - \exp\left(\frac{1}{6}\sum_{i=1}^{6}\cos(cx_i)\right) + a + \exp(1)$
with $a = 20.0$, $b = 0.5$ and $c = 0.0$
Bounds: $x_i \in [-32.768, 32.768] \quad \forall\ i = 1, 2, ..., 6$
Global Minimum: $f(\boldsymbol{x}^*) = 0 \quad$ at $\quad \boldsymbol{x}^* = (0, 0, 0, 0, 0, 0)$

10D Dixon-Price

Function: $f(\boldsymbol{x}) = (x_1 - 1)^2 + \sum_{i=2}^{10} i\left(2x_i^2 - x_{i-1}\right)^2$
Bounds: $x_i \in [-10, 10] \quad \forall\ i = 1, 2, ..., 10$
Global Minimum: $f(\boldsymbol{x}^*) = 0 \quad$ at $\quad x_i^* = 2^{-\frac{2^i - 2}{2^i}} \quad \forall\ i = 1, ..., 10$

8D Griewank

Function: $f(\boldsymbol{x}) = \sum_{i=1}^{8} \frac{x_i^2}{4000} - \prod_{i=1}^{8}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
Bounds: $x_i \in [-600, 600] \quad \forall i = 1, 2, ..., 8$
Global Minimum: $f(\boldsymbol{x}^*) = 0 \quad$ at $\quad \boldsymbol{x}^* = (0, 0, 0, 0, 0, 0, 0, 0)$

6D Hartmann

Function: $f(\boldsymbol{x}) = -\sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{6} \boldsymbol{A}_{ij}\left(x_j - \boldsymbol{P}_{ij}\right)^2\right)$
where $\alpha = (1.0, 1.2, 3.0, 3.2)^T$

$$\text{and } \boldsymbol{A} = \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}$$

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

Bounds: $x_i \in (0,1) \quad \forall i = 1,2,...,6$

Global Minimum: $f(\boldsymbol{x}^*) = -3.32237$

at $\boldsymbol{x}^* = (0.20169, 0.150011, 0.476874, 0.275332, 0.311652, 0.6573)$

5D Michalewicz

Function: $f(\boldsymbol{x}) = -\sum_{i=1}^{5} \sin(x_i) \sin^{20}\left(\frac{ix_i^2}{\pi}\right)$

Bounds: $x_i \in [0,\pi] \quad \forall i = 1,2,...,5$

Global Minimum: $f(\boldsymbol{x}^*) = -4.687658$

10D Sphere

Function: $f(\boldsymbol{x}) = \sum_{i=1}^{10} x_i^2$

Bounds: $x_i \in [-5.12, 5.12] \quad \forall i = 1,2,...,10$

Global Minimum: $f(\boldsymbol{x}^*) = 0 \quad \text{at} \quad \boldsymbol{x}^* = (0,0,0,0,0,0,0,0,0,0)$

A.2 Supplementary statistical analysis

In this appendix, we provide a formal statistical comparison of the algorithms of the analytical single-point acquisition functions and the multi-point acquisition functions, both with five initial starting points per input dimension—we see these as the two most essential comparisons in Chapter 3. The other investigations in Section 3.2.1 either represent a similar setup (analytical single-point comparison with different numbers of initial training points) or are used to facilitate the comparison of the multi-point Monte Carlo acquisition functions.

To investigate if there are statistically significant differences between the methods, the non-parametric Friedman test is performed on the best output value found for every replication for each algorithm. This conducts a hypothesis test to see if there is a difference in performance between the algorithms. First, we focus on the analytical single-point algorithms of Section 3.2.1. The tests reported in Table A.1 provide p-values, which are all well below the 0.1% significance level, indicating that we have very strong evidence

of a difference in the performance of the different methods. We can investigate these differences using the box plots in Figure A.1, which show where the differences between the algorithms lie. We see substantial differences in the Ackley function. The results are similar to the multi-point approaches of Section 3.2.1. We again see significant differences between the optimal output found by the different methods (at the 0.1% level) using the Friedman tests reported in Table A.1, and the box plots in Figure A.2 reinforce these results.

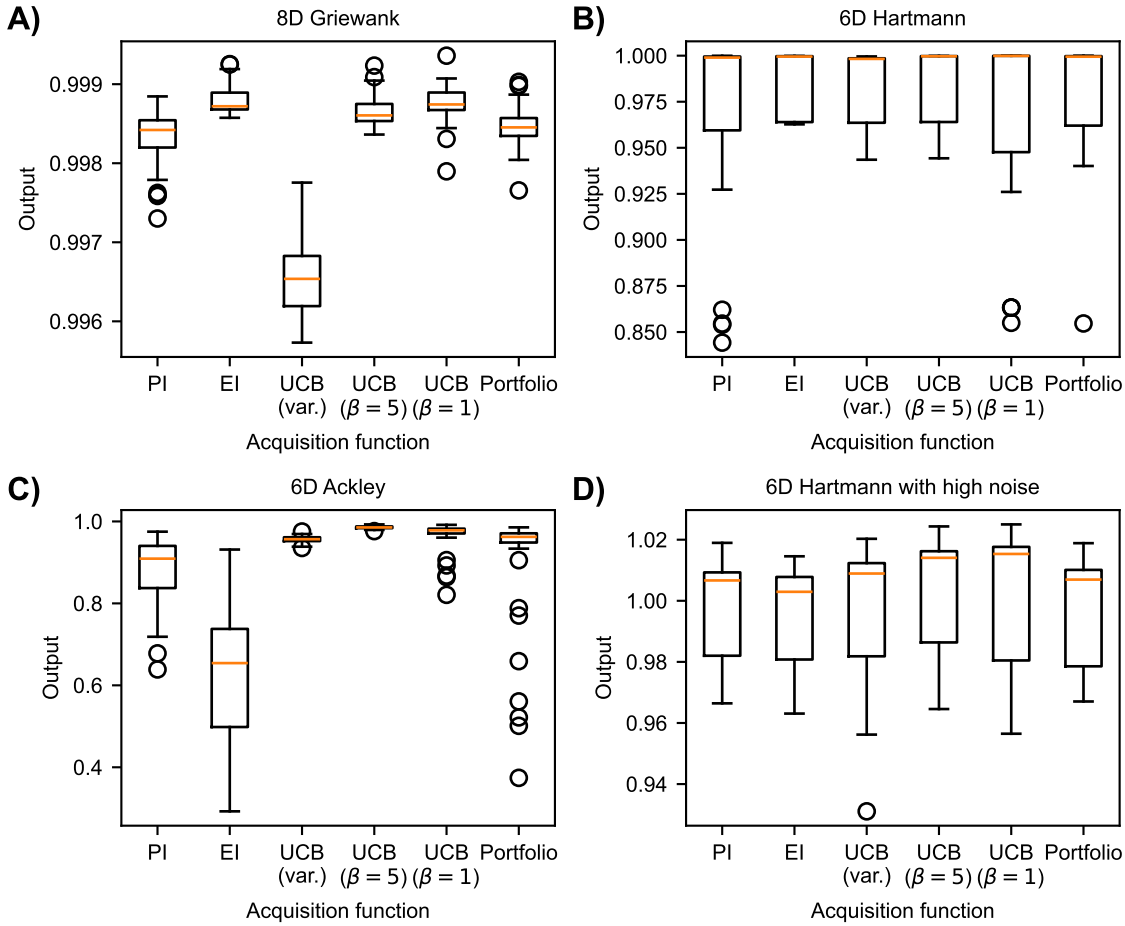

Figure A.1: Box plots of best values found for analytical single-point acquisition functions with five initial starting points per input dimension. The orange line indicates the median, the box extends from the lower to the upper quartile range and the whiskers indicate the range of the best values without the outliers shown as circles.

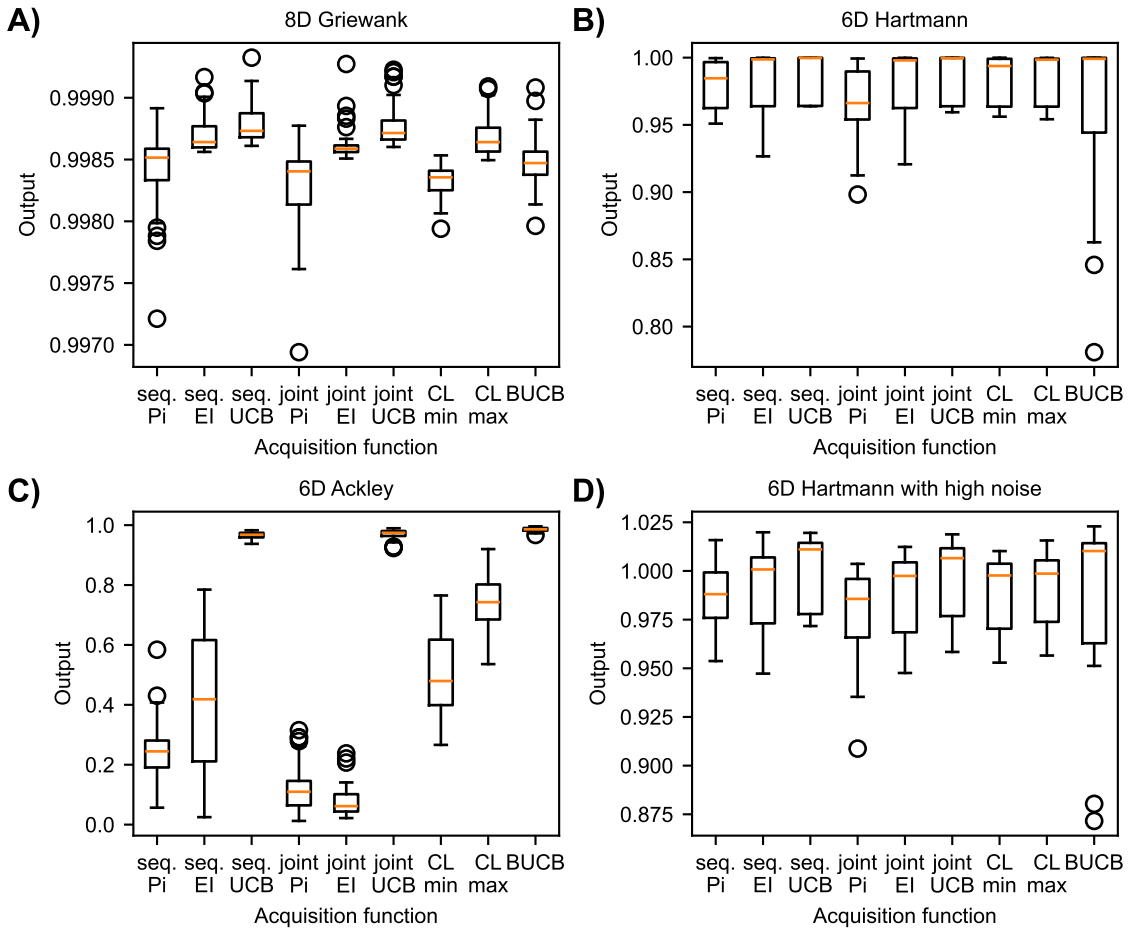

Figure A.2: Box plots of best values found for multi-point acquisition functions with five initial starting points per input dimension. The orange line indicates the median, the box extends from the lower to the upper quartile range and the whiskers indicate the range of the best values without the outliers shown as circles.

Table A.1: Friedman test for analytical single-point and multi-point acquisition function with five initial starting points per input dimension.

Test function	Single-point		Multi-point	
	Test statistic	p-value	Test statistic	p-value
Griewank	169.10	1.14e-34	254.97	1.53e-50
Hartmann	81.86	3.42e-16	159.59	1.94e-30
Noisy Hartmann	78.80	1.50e-15	137.30	8.64e-26
Ackley	199.10	4.43e-41	362.75	1.72e-73

A.3 Supplementary figures

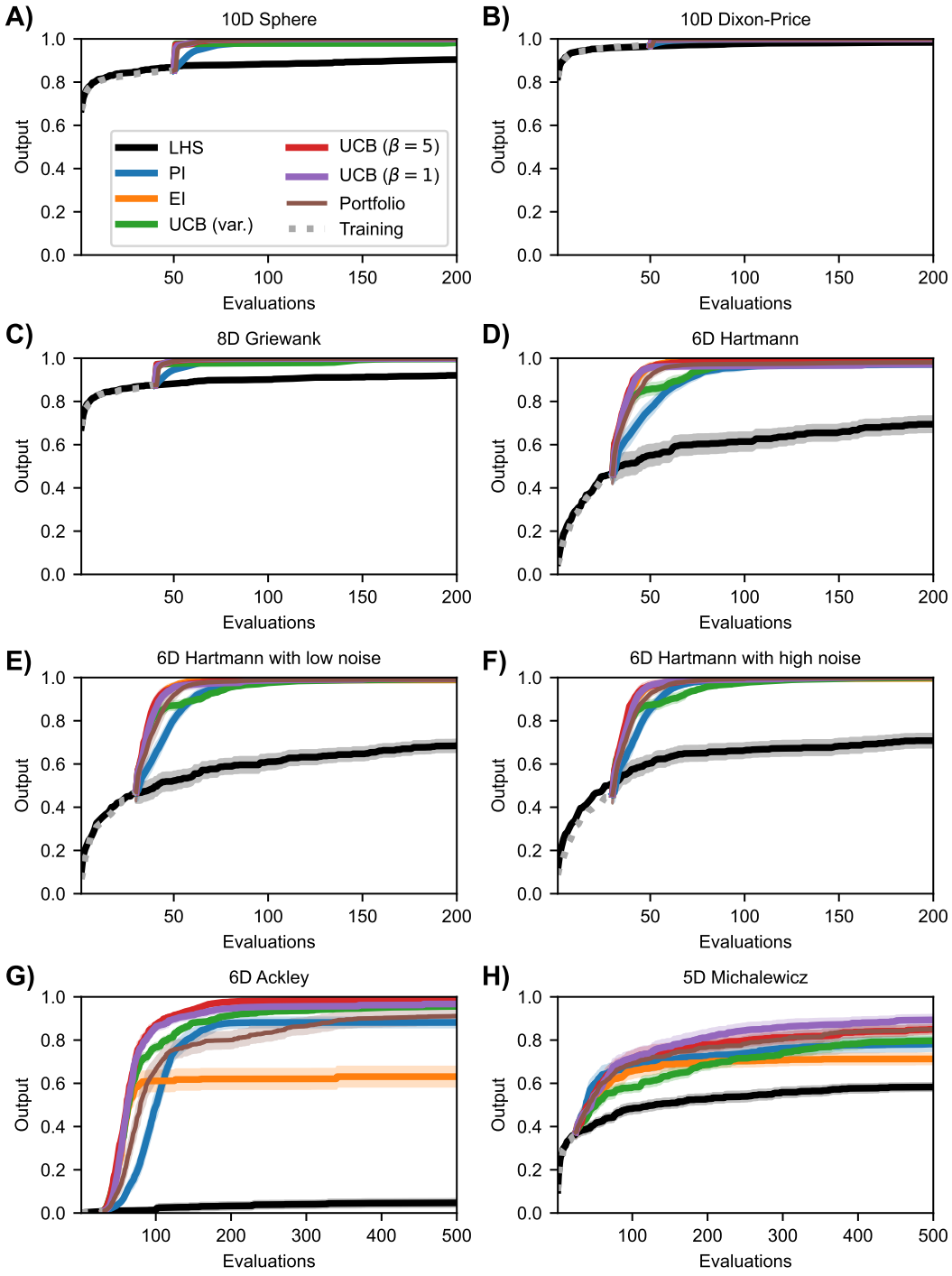

Figure A.3: Performance plots for analytical single-point acquisition functions with five initial starting points per input dimension. Solid lines represent the mean over the 50 runs while the shaded area represents the 95% confidence intervals.

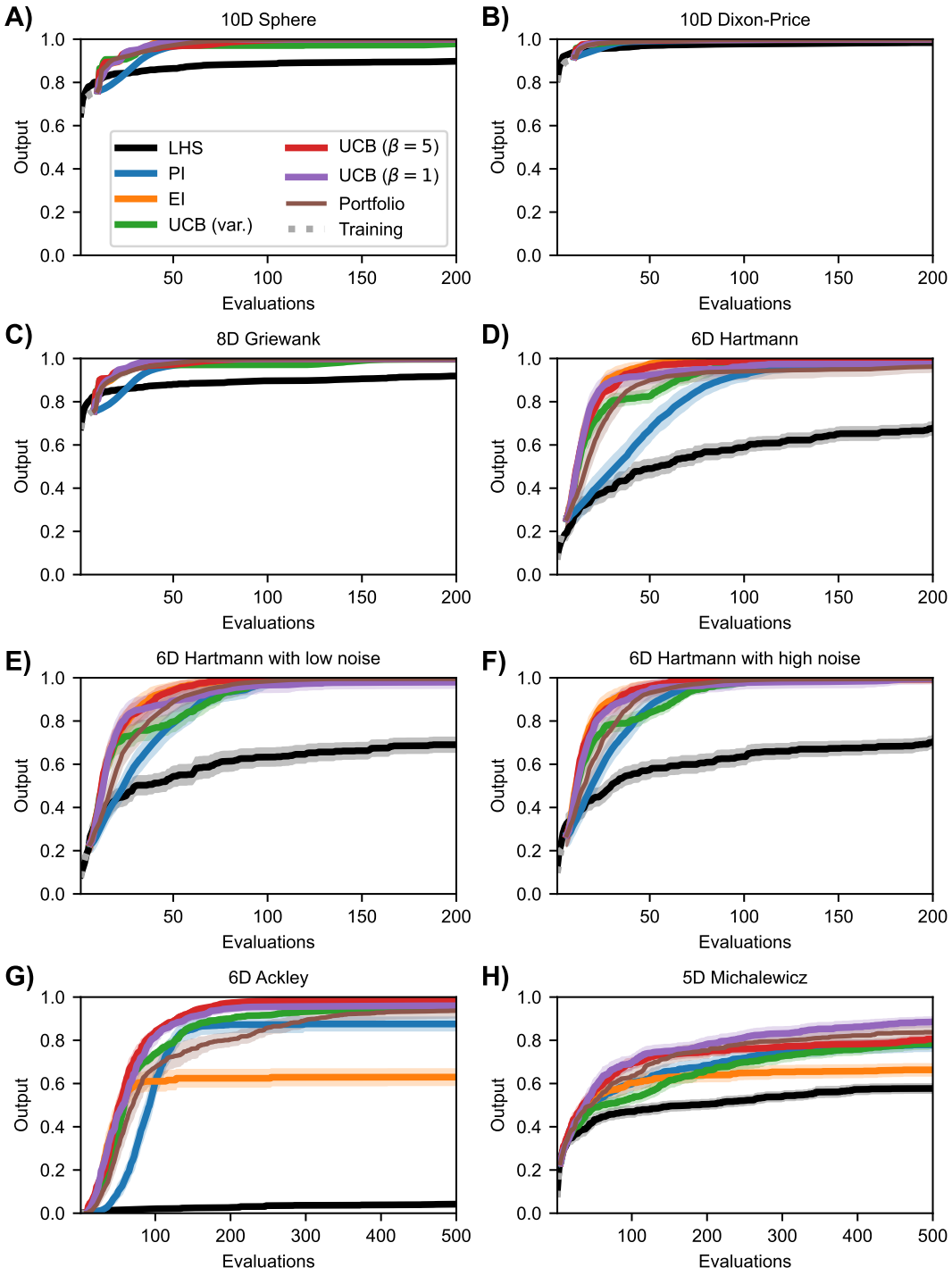

Figure A.4: Performance plots for analytical single-point acquisition functions with one initial starting point per input dimension. Solid lines represent the mean over the 50 runs while the shaded area represents the 95% confidence intervals.

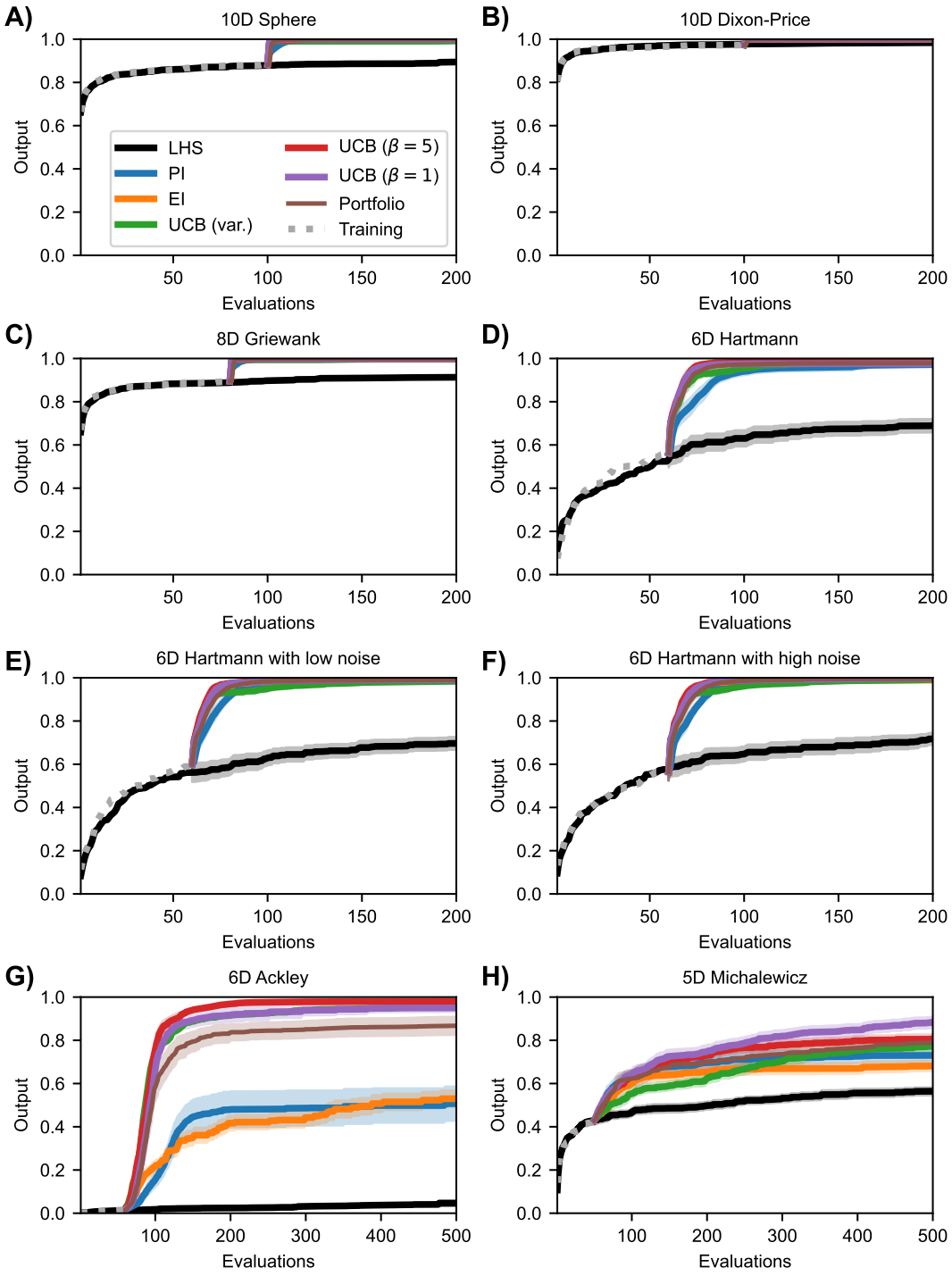

Figure A.5: Performance plots for analytical single-point acquisition functions with ten initial starting points per input dimension. Solid lines represent the mean over the 50 runs while the shaded area represents the 95% confidence intervals.

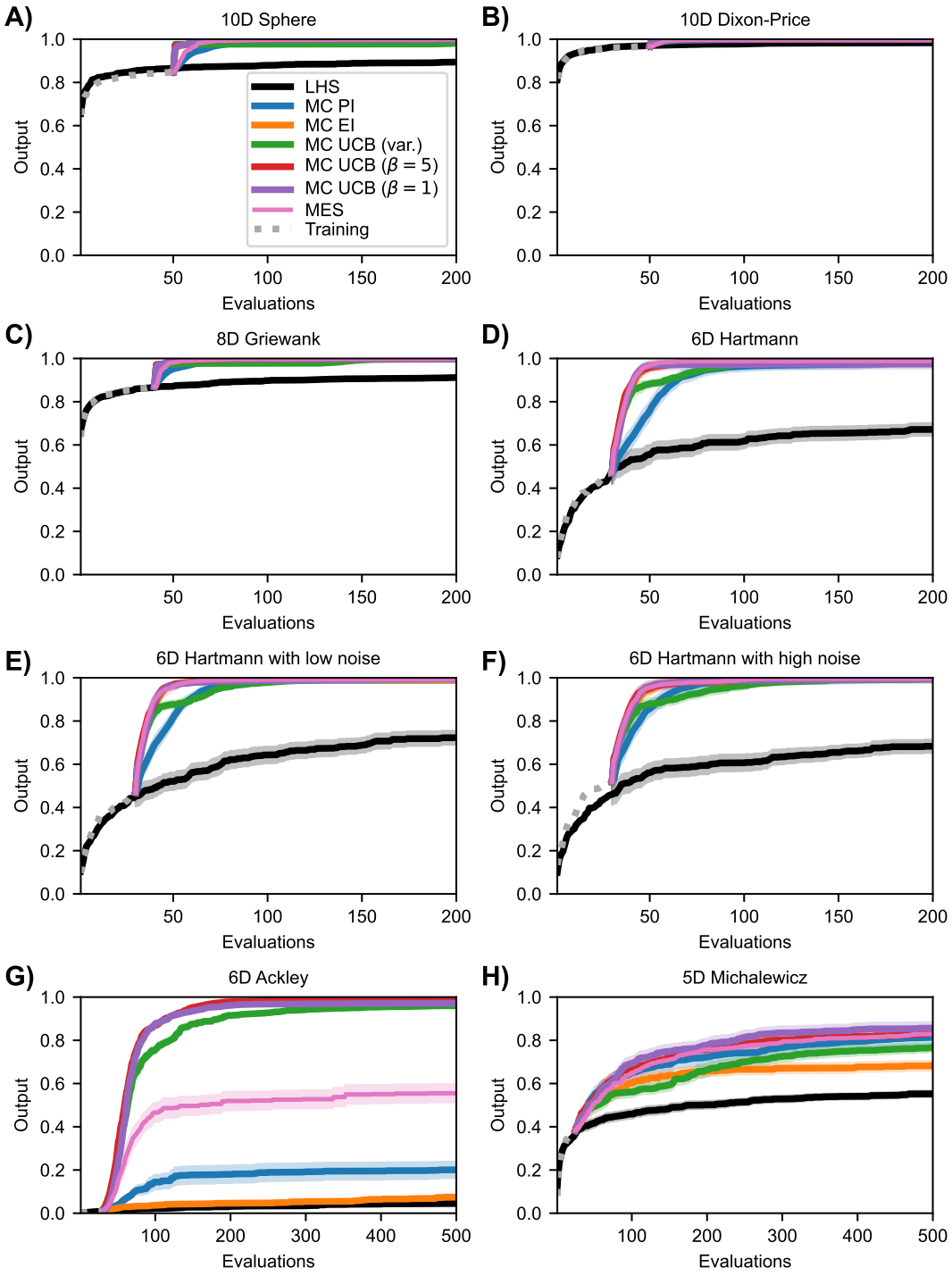

Figure A.6: Performance plots for Monte Carlo single-point acquisition functions with five initial starting points per input dimension. Solid lines represent the mean over the 50 runs while the shaded areas represent the 95% confidence intervals.

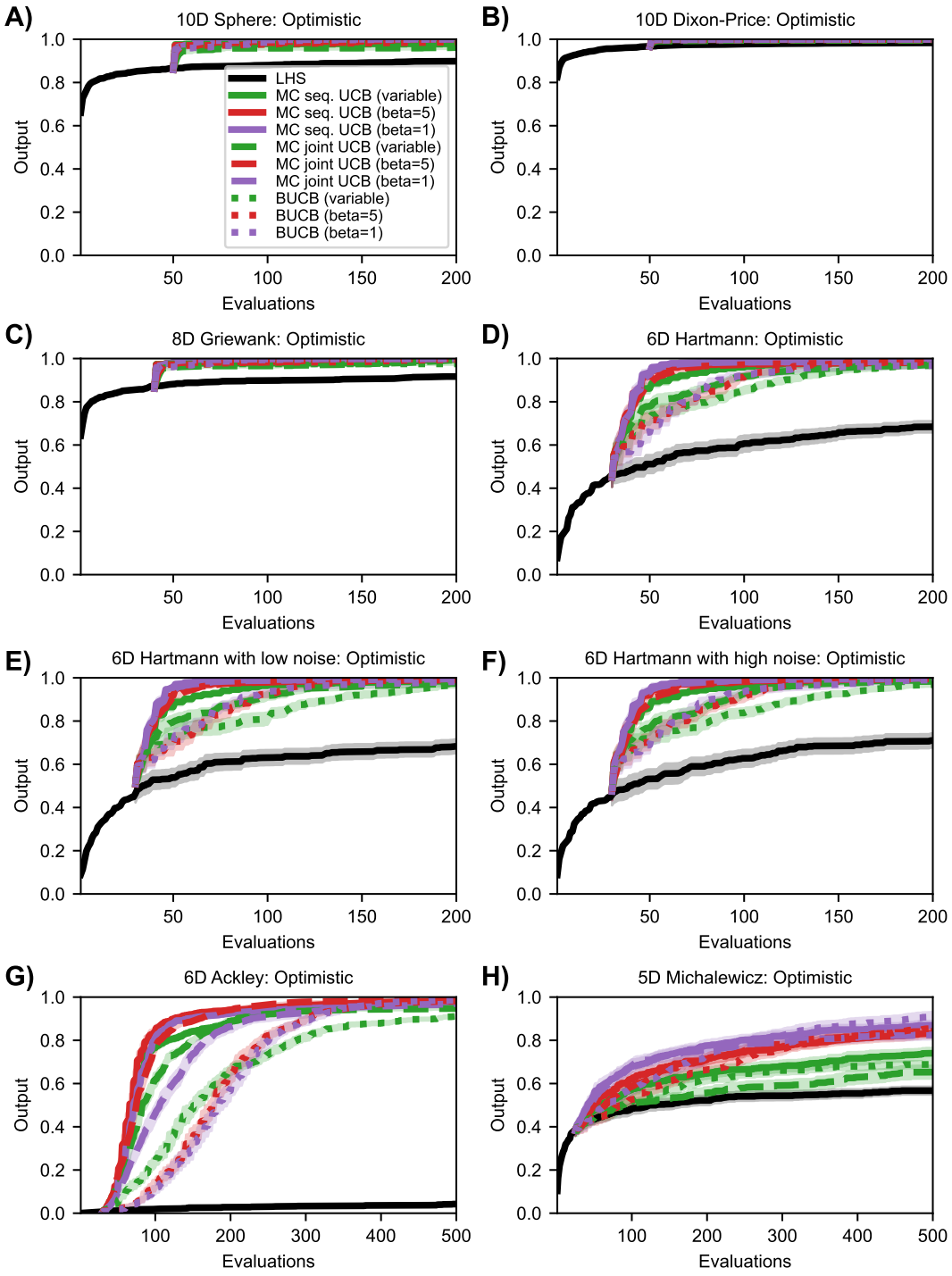

Figure A.7: Performance plots for optimistic multi-point acquisition functions with five initial training points per input dimension. Solid lines represent the mean over the 50 runs while the shaded area represents the 95% confidence intervals.

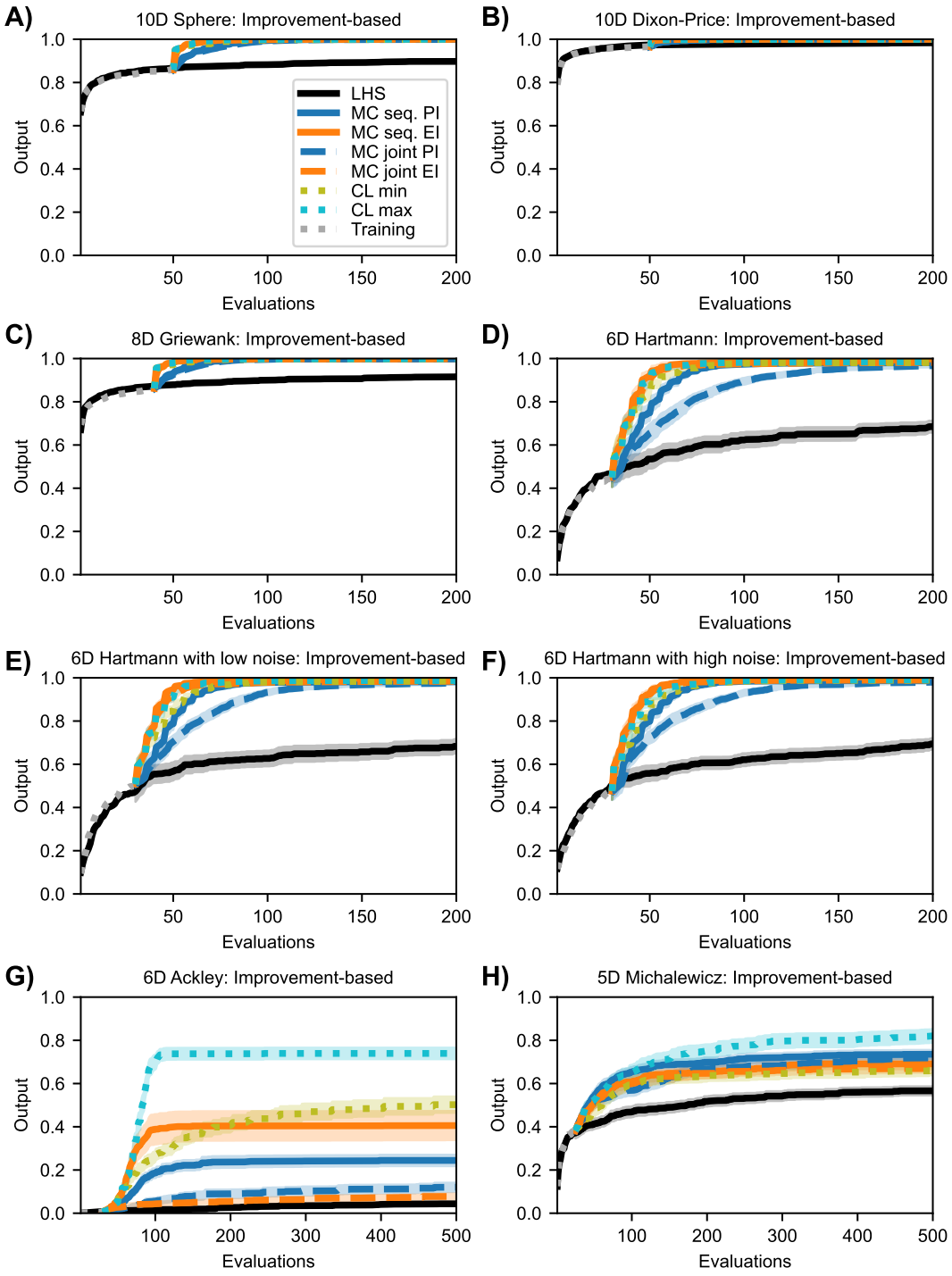

Figure A.8: Performance plots for improvement-based multi-point acquisition functions with five initial training points per input dimension. Solid lines represent the mean over the 50 runs while the shaded area represents the 95% confidence intervals.

A.4 Supplementary tables

Table A.2: Best solutions found for analytical single-point acquisition functions with five initial training points per input dimension.

Method	Sphere	Dixon-Price	Griewank	Hartmann	Hartmann low noise	Hartmann high noise	Michalewicz	Ackley
PI	**1.00**	**1.00**	**1.00**	0.97	0.99	**1.00**	0.78	0.88
EI	**1.00**	**1.00**	**1.00**	**0.99**	0.99	**1.00**	0.71	0.63
UCB (variable)	0.98	**1.00**	**1.00**	0.98	0.99	**1.00**	0.80	0.96
UCB (β=5)	**1.00**	**1.00**	**1.00**	**0.99**	**1.00**	**1.00**	0.85	**0.99**
UCB (β=1)	**1.00**	**1.00**	**1.00**	0.97	**1.00**	**1.00**	**0.89**	0.97
Hedge	**1.00**	**1.00**	**1.00**	0.98	0.99	**1.00**	0.85	0.91

Table A.3: Averaged AUC with standard error for analytical single-point acquisition functions with five initial training points per input dimension.

Method	Sphere	Dixon-Price	Griewank	Hartmann	Hartmann low noise	Hartmann high noise	Michalewicz	Ackley
PI	0.99 (± 0.00)	**1.00** (± 0.00)	0.99 (± 0.00)	0.91 (± 0.04)	0.94 (± 0.02)	0.95 (± 0.02)	0.73 (± 0.10)	0.76 (± 0.08)
EI	**1.00** (± 0.00)	**1.00** (± 0.00)	**1.00** (± 0.00)	**0.97** (± 0.02)	**0.97** (± 0.02)	0.97 (± 0.02)	0.68 (± 0.08)	0.59 (± 0.15)
UCB (variable)	0.98 (± 0.01)	**1.00** (± 0.00)	0.98 (± 0.01)	0.94 (± 0.03)	0.95 (± 0.02)	0.95 (± 0.03)	0.69 (± 0.06)	0.85 (± 0.01)
UCB (β=5)	**1.00** (± 0.00)	**1.00** (± 0.00)	**1.00** (± 0.00)	**0.97** (± 0.02)	**0.97** (± 0.02)	**0.98** (± 0.02)	0.77 (± 0.08)	**0.90** (± 0.02)
UCB (β=1)	**1.00** (± 0.00)	**1.00** (± 0.00)	**1.00** (± 0.00)	0.95 (± 0.04)	**0.97** (± 0.03)	0.97 (± 0.02)	**0.80** (± 0.08)	0.88 (± 0.05)
Hedge	**1.00** (± 0.00)	**1.00** (± 0.00)	**1.00** (± 0.00)	0.95 (± 0.03)	0.96 (± 0.03)	0.97 (± 0.02)	0.76 (± 0.08)	0.77 (± 0.11)

Table A.4: Best solutions found for analytical single-point acquisition functions with one initial training point per input dimension.

Method	Sphere	Dixon-Price	Griewank	Hartmann	Hartmann low noise	Hartmann high noise	Michalewicz	Ackley
PI	**1.00**	**1.00**	**1.00**	0.98	0.99	**1.00**	0.78	0.88
EI	**1.00**	**1.00**	**1.00**	**0.99**	0.99	0.99	0.66	0.63
UCB (variable)	0.98	**1.00**	**1.00**	**0.99**	**1.00**	**1.00**	0.79	0.96
UCB (β=5)	**1.00**	**1.00**	**1.00**	**0.99**	**1.00**	**1.00**	0.81	**0.99**
UCB (β=1)	**1.00**	**1.00**	**1.00**	0.97	0.98	0.99	**0.89**	0.96
Hedge	**1.00**	**1.00**	**1.00**	0.96	**1.00**	0.99	0.84	0.94

Table A.5: Averaged AUC with standard error for analytical single-point acquisition functions with one initial training point per input dimension.

Method	Sphere	Dixon-Price	Griewank	Hartmann	Hartmann low noise	Hartmann high noise	Michalewicz	Ackley
PI	0.97 (± 0.01)	0.99 (± 0.00)	0.97 (± 0.01)	0.82 (± 0.09)	0.87 (± 0.07)	0.90 (± 0.06)	0.68 (± 0.07)	0.73 (± 0.11)
EI	**0.99** (± 0.00)	**1.00** (± 0.00)	**0.99** (± 0.00)	**0.95** (± 0.03)	**0.94** (± 0.04)	**0.95** (± 0.03)	0.62 (± 0.09)	0.58 (± 0.13)
UCB (variable)	0.96 (± 0.01)	**1.00** (± 0.00)	0.97 (± 0.01)	0.91 (± 0.04)	0.90 (± 0.05)	0.91 (± 0.03)	0.66 (± 0.06)	0.82 (± 0.01)
UCB (β=5)	0.98 (± 0.00)	**1.00** (± 0.00)	0.98 (± 0.00)	0.94 (± 0.03)	**0.94** (± 0.04)	**0.95** (± 0.03)	0.72 (± 0.05)	**0.87** (± 0.02)
UCB (β=1)	**0.99** (± 0.00)	**1.00** (± 0.00)	**0.99** (± 0.00)	0.93 (± 0.08)	0.92 (± 0.10)	0.93 (± 0.09)	**0.77** (± 0.07)	0.85 (± 0.06)
Hedge	0.98 (± 0.01)	0.99 (± 0.00)	0.98 (± 0.01)	0.89 (± 0.11)	0.91 (± 0.05)	0.92 (± 0.05)	0.73 (± 0.08)	0.76 (± 0.09)

Table A.6: Best solutions found for analytical single-point acquisition functions with ten initial training points per input dimension.

Method	Sphere	Dixon-Price	Griewank	Hartmann	Hartmann low noise	Hartmann high noise	Michalewicz	Ackley
PI	**1.00**	**1.00**	**1.00**	0.97	**0.99**	0.99	0.73	0.51
EI	**1.00**	**1.00**	**1.00**	**0.98**	**0.99**	0.99	0.68	0.53
UCB (variable)	0.99	**1.00**	**1.00**	**0.98**	0.98	0.99	0.77	0.96
UCB (β=5)	**1.00**	**1.00**	**1.00**	**0.98**	**0.99**	**1.00**	0.81	**0.98**
UCB (β=1)	**1.00**	**1.00**	**1.00**	**0.98**	**0.99**	**1.00**	**0.88**	0.95
Hedge	**1.00**	**1.00**	**1.00**	**0.98**	**0.99**	**1.00**	0.79	0.87

Table A.7: Averaged AUC with standard error for analytical single-point acquisition functions with ten initial training points per input dimension.

Method	Sphere	Dixon-Price	Griewank	Hartmann	Hartmann low noise	Hartmann high noise	Michalewicz	Ackley
PI	**1.00** (± 0.00)	**1.00** (± 0.00)	**1.00** (± 0.00)	0.93 (± 0.05)	0.96 (± 0.02)	0.96 (± 0.02)	0.69 (± 0.10)	0.43 (± 0.25)
EI	**1.00** (± 0.00)	**1.00** (± 0.00)	**1.00** (± 0.00)	**0.97** (± 0.02)	0.97 (± 0.02)	0.97 (± 0.02)	0.64 (± 0.09)	0.41 (± 0.07)
UCB (variable)	0.99 (± 0.01)	**1.00** (± 0.00)	**1.00** (± 0.00)	0.95 (± 0.03)	0.96 (± 0.03)	0.96 (± 0.03)	0.66 (± 0.07)	0.87 (± 0.01)
UCB (β=5)	**1.00** (± 0.00)	**1.00** (± 0.00)	**1.00** (± 0.00)	**0.97** (± 0.02)	**0.98** (± 0.02)	**0.98** (± 0.02)	0.74 (± 0.06)	**0.91** (± 0.03)
UCB (β=1)	**1.00** (± 0.00)	**1.00** (± 0.00)	**1.00** (± 0.00)	0.96 (± 0.03)	0.97 (± 0.02)	**0.98** (± 0.03)	**0.77** (± 0.07)	0.86 (± 0.08)
Hedge	**1.00** (± 0.00)	**1.00** (± 0.00)	**1.00** (± 0.00)	0.96 (± 0.03)	0.97 (± 0.02)	0.97 (± 0.02)	0.71 (± 0.09)	0.78 (± 0.13)

Table A.8: Best solutions found for Monte Carlo single-point acquisition functions with five initial training points per input dimension.

Method	Sphere	Dixon-Price	Griewank	Hartmann	Hartmann low noise	Hartmann high noise	Michalewicz	Ackley
MC PI	**1.00**	**1.00**	**1.00**	0.98	0.99	0.99	0.81	0.20
MC EI	**1.00**	**1.00**	**1.00**	**0.99**	0.99	0.99	0.68	0.07
MC UCB (variable)	0.98	**1.00**	**1.00**	0.98	0.99	0.99	0.77	0.96
MC UCB (β=5)	**1.00**	**1.00**	**1.00**	**0.99**	**1.00**	**1.00**	0.84	**0.99**
MC UCB (β=1)	**1.00**	**1.00**	**1.00**	0.97	**1.00**	**1.00**	**0.86**	0.97
MES	**1.00**	**1.00**	**1.00**	**0.99**	0.99	**1.00**	0.83	0.55

Table A.9: Averaged AUC with standard error for Monte Carlo single-point acquisition functions with five initial training points per input dimension.

Method	Sphere	Dixon-Price	Griewank	Hartmann	Hartmann low noise	Hartmann high noise	Michalewicz	Ackley
MC PI	0.99 (± 0.00)	**1.00** (± 0.00)	0.99 (± 0.00)	0.92 (± 0.06)	0.94 (± 0.02)	0.95 (± 0.04)	0.73 (± 0.09)	0.17 (± 0.12)
MC EI	**1.00** (± 0.00)	**1.00** (± 0.00)	**1.00** (± 0.00)	**0.97** (± 0.02)	0.97 (± 0.02)	0.97 (± 0.04)	0.64 (± 0.08)	0.05 (± 0.03)
MC UCB (variable)	0.98 (± 0.01)	**1.00** (± 0.00)	0.98 (± 0.01)	0.95 (± 0.02)	0.95 (± 0.02)	0.95 (± 0.04)	0.67 (± 0.06)	0.86 (± 0.01)
MC UCB (β=5)	**1.00** (± 0.00)	**1.00** (± 0.00)	**1.00** (± 0.00)	**0.97** (± 0.02)	**0.98** (± 0.02)	**0.98** (± 0.03)	0.75 (± 0.06)	**0.91** (± 0.01)
MC UCB (β=1)	**1.00** (± 0.00)	**1.00** (± 0.00)	**1.00** (± 0.00)	0.96 (± 0.07)	0.97 (± 0.03)	0.97 (± 0.07)	**0.77** (± 0.08)	0.89 (± 0.03)
MES	0.99 (± 0.00)	0.99 (± 0.00)	0.99 (± 0.00)	**0.97** (± 0.02)	0.97 (± 0.02)	0.97 (± 0.03)	0.74 (± 0.05)	0.49 (± 0.12)

Table A.10: Best solutions found for multi-point acquisition functions with five initial training points per input dimension.

Type	Method	Sphere	Dixon-Price	Griewank	Hartmann	Hartmann low noise	Hartmann high noise	Michalewicz	Ackley
Sequential Monte Carlo	PI	**1.00**	**1.00**	**1.00**	0.98	0.98	0.99	0.74	0.24
	EI	**1.00**	**1.00**	**1.00**	0.98	**0.99**	0.99	0.69	0.41
	UCB (variable)	0.98	**1.00**	0.99	0.98	0.98	0.99	0.74	0.97
	UCB (β=5)	**1.00**	**1.00**	**1.00**	0.98	**0.99**	**1.00**	0.83	0.98
	UCB (β=1)	**1.00**	**1.00**	**1.00**	0.98	**0.99**	**1.00**	0.87	0.97
Joint Monte Carlo	PI	**1.00**	**1.00**	**1.00**	0.97	0.98	0.98	0.73	0.12
	EI	**1.00**	**1.00**	**1.00**	0.98	0.98	0.99	0.67	0.08
	UCB (variable)	0.96	0.99	0.99	0.97	0.97	0.98	0.65	0.95
	UCB (β=5)	**1.00**	**1.00**	**1.00**	0.98	**0.99**	**1.00**	0.84	0.98
	UCB (β=1)	**1.00**	**1.00**	**1.00**	**0.99**	**0.99**	**1.00**	0.83	0.97
Analytical	CL min	**1.00**	**1.00**	**1.00**	0.98	**0.99**	0.99	0.66	0.50
	CL max	**1.00**	**1.00**	**1.00**	0.98	**0.99**	0.99	0.82	0.74
	BUCB (variable)	0.98	**1.00**	0.98	0.97	0.97	0.97	0.69	0.91
	BUCB (β=5)	0.99	**1.00**	0.99	0.98	**0.99**	**1.00**	**0.85**	0.98
	BUCB (β=1)	**1.00**	**1.00**	**1.00**	0.97	0.98	0.99	0.91	**0.99**

Table A.11: Averaged AUC with standard error for multi-point acquisition functions with five initial training points per input dimension.

Type	Method	Sphere	Dixon-Price	Griewank	Hartmann	Hartmann low noise	Hartmann high noise	Michalewicz	Ackley
Sequential Monte Carlo	PI	0.99 (± 0.01)	**1.00** (± 0.00)	0.99 (± 0.00)	0.92 (± 0.03)	0.93 (± 0.02)	0.94 (± 0.02)	0.69 (± 0.06)	0.22 (± 0.08)
	EI	**1.00** (± 0.00)	**1.00** (± 0.00)	**1.00** (± 0.00)	0.95 (± 0.03)	0.96 (± 0.02)	**0.97** (± 0.02)	0.64 (± 0.08)	0.37 (± 0.22)
	UCB (variable)	0.98 (± 0.01)	**1.00** (± 0.00)	0.98 (± 0.01)	0.94 (± 0.03)	0.94 (± 0.02)	0.94 (± 0.03)	0.64 (± 0.08)	0.84 (± 0.01)
	UCB (β=5)	0.99 (± 0.00)	**1.00** (± 0.00)	0.99 (± 0.00)	0.95 (± 0.02)	0.96 (± 0.02)	0.96 (± 0.02)	0.72 (± 0.06)	**0.88** (± 0.02)
	UCB (β=1)	**1.00** (± 0.00)	**1.00** (± 0.00)	**1.00** (± 0.00)	**0.96** (± 0.02)	**0.97** (± 0.02)	**0.97** (± 0.02)	**0.77** (± 0.07)	0.86 (± 0.02)
Joint Monte Carlo	PI	0.98 (± 0.01)	**1.00** (± 0.00)	0.99 (± 0.00)	0.86 (± 0.05)	0.89 (± 0.03)	0.89 (± 0.04)	0.65 (± 0.06)	0.09 (± 0.06)
	EI	**1.00** (± 0.00)	**1.00** (± 0.00)	**1.00** (± 0.00)	0.95 (± 0.03)	0.95 (± 0.02)	0.96 (± 0.02)	0.63 (± 0.08)	0.06 (± 0.03)
	UCB (variable)	0.96 (± 0.02)	0.99 (± 0.01)	0.97 (± 0.02)	0.89 (± 0.04)	0.90 (± 0.05)	0.90 (± 0.04)	0.56 (± 0.07)	0.81 (± 0.02)
	UCB (β=5)	0.99 (± 0.00)	**1.00** (± 0.00)	0.99 (± 0.00)	0.94 (± 0.03)	0.96 (± 0.02)	0.95 (± 0.03)	0.72 (± 0.05)	0.87 (± 0.02)
	UCB (β=1)	**1.00** (± 0.00)	**1.00** (± 0.00)	**1.00** (± 0.00)	**0.96** (± 0.02)	**0.97** (± 0.02)	**0.97** (± 0.02)	0.75 (± 0.07)	0.78 (± 0.06)
Analytical	CL min	0.99 (± 0.00)	**1.00** (± 0.00)	0.99 (± 0.00)	0.94 (± 0.02)	0.95 (± 0.02)	0.95 (± 0.02)	0.62 (± 0.08)	0.40 (± 0.08)
	CL max	**1.00** (± 0.00)	**1.00** (± 0.00)	**1.00** (± 0.00)	0.95 (± 0.03)	0.96 (± 0.02)	0.96 (± 0.03)	0.74 (± 0.10)	0.67 (± 0.09)
	BUCB (variable)	0.98 (± 0.01)	**1.00** (± 0.00)	0.98 (± 0.02)	0.85 (± 0.05)	0.84 (± 0.06)	0.85 (± 0.05)	0.61 (± 0.05)	0.65 (± 0.03)
	BUCB (β=5)	0.98 (± 0.01)	**1.00** (± 0.00)	0.98 (± 0.00)	0.88 (± 0.05)	0.89 (± 0.04)	0.90 (± 0.05)	0.69 (± 0.05)	0.66 (± 0.05)
	BUCB (β=1)	0.99 (± 0.01)	**1.00** (± 0.00)	0.99 (± 0.00)	0.88 (± 0.06)	0.89 (± 0.04)	0.90 (± 0.06)	0.74 (± 0.08)	0.65 (± 0.05)

Bibliography

[1] L. Acerbi and W. J. Ma. "Practical Bayesian Optimization for Model Fitting With Bayesian Adaptive Direct Search". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 30. NeurIPS Foundation. 2017.

[2] S. Ament, S. Daulton, D. Eriksson, M. Balandat, and E. Bakshy. "Unexpected Improvements to Expected Improvement for Bayesian Optimization". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 36. NeurIPS Foundation. 2023.

[3] C. Audet, J. Denni, D. Moore, A. Booker, and P. Frank. "A Surrogate-Model-Based Method for Constrained Optimization". In: *Multidisciplinary Analysis and Optimization Conference (MA&O)*. AIAA. 2000, p. 4891.

[4] C. Audet and J. E. Dennis Jr. "Mesh Adaptive Direct Search Algorithms for Constrained Optimization". In: *SIAM Journal on Optimization* 17.1 (2006), pp. 188–217.

[5] P. Auer. "Using Confidence Bounds for Exploitation-Exploration Trade-Offs". In: *Journal of Machine Learning Research* 3.Nov (2002), pp. 397–422.

[6] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. "Gambling in a Rigged Casino: The Adversarial Multi-Armed Bandit Problem". In: *Symposium on Foundations of Computer Science (FOCS)*. IEEE. 1995, pp. 322–331.

[7] M. Balandat, B. Karrer, D. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy. "BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. NeurIPS Foundation. 2020, pp. 21524–21538.

[8] P. Bartholomew, G. Deskos, R. A. Frantz, F. N. Schuch, E. Lamballais, and S. Laizet. "Xcompact3D: An Open-Source Framework for Solving Turbulence Problems on a Cartesian Mesh". In: *SoftwareX* 12 (2020), p. 100550.

[9] T. Bartz-Beielstein, C. W. Lasarczyk, and M. Preuß. "Sequential Parameter Optimization". In: *Congress on Evolutionary Computation (CEC)*. Vol. 1. IEEE. 2005, pp. 773–780.

[10] S. Belakaria, A. Deshwal, and J. R. Doppa. "Max-Value Entropy Search for Multi-Objective Bayesian Optimization". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 32. NeurIPS Foundation. 2019.

[11] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. "Algorithms for Hyper-Parameter Optimization". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 24. NeurIPS Foundation. 2011.

[12] D. A. Berry and B. Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. Monographs on Statistics and Applied Probability. Springer, 1985.

[13] M. Binois, D. Ginsbourger, and O. Roustant. "On the Choice of the Low-Dimensional Domain for Global Optimization via Random Embeddings". In: *Journal of Global Optimization* 76.1 (2020), pp. 69–90.

[14] S. Boersma, B. M. Doekemeijer, P. M. Gebraad, P. A. Fleming, J. Annoni, A. K. Scholbrock, J. A. Frederik, and J.-W. van Wingerden. "A Tutorial on Control-Oriented Modeling and Control of Wind Farms". In: *American Control Conference (ACC)*. IEEE. 2017, pp. 1–18.

[15] A. Booker. "Design and Analysis of Computer Experiments". In: *Multidisciplinary Analysis and Optimization Conference (MA&O)*. AIAA. 1998, p. 4757.

[16] P. Boyle. "Gaussian Processes for Regression and Optimisation". PhD thesis. Victoria University of Wellington, 2007.

[17] A. P. Bradley. "The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms". In: *Pattern recognition* 30.7 (1997), pp. 1145–1159.

[18] L. Breiman. "Random Forests". In: *Machine Learning* 45 (2001), pp. 5–32.

[19] E. Brochu, T. Brochu, and N. De Freitas. "A Bayesian Interactive Optimization Approach to Procedural Animation Design". In: *Symposium on Computer Animation (SCA)*. ACM. 2010, pp. 103–112.

[20] E. Brochu, V. M. Cora, and N. De Freitas. "A Tutorial on Bayesian Optimization of Expensive Cost Functions, With Application to Active User Modeling and Hierarchical Reinforcement Learning". In: *arXiv preprint arXiv:1012.2599* (2010).

[21] A. D. Bull. "Convergence Rates of Efficient Global Optimization Algorithms". In: *Journal of Machine Learning Research* 12.10 (2011).

[22] D. M. Bushnell and J. N. Hefner. *Viscous Drag Reduction in Boundary Layers*. Progress in Astronautics and Aeronautics. AIAA, 1990.

[23] R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth. "Bayesian Optimization for Learning Gaits Under Uncertainty: An Experimental Comparison on a Dynamic Bipedal Walker". In: *Annals of Mathematics and Artificial Intelligence* 76 (2016), pp. 5–23.

[24] P. B. Chang, B. J. Williams, K. S. B. Bhalla, T. W. Belknap, T. J. Santner, W. I. Notz, and D. L. Bartel. "Design and Analysis of Robust Total Joint Replacements: Finite Element Model Experiments With Environmental Variables". In: *Journal of Biomechanical Engineering* 123.3 (2001), pp. 239–246.

[25] P. B. Chang, B. J. Williams, T. J. Santner, W. I. Notz, and D. L. Bartel. "Robust Optimization of Total Joint Replacements Incorporating Environmental Variables". In: *Journal of Biomechanical Engineering* 121.3 (1999), pp. 304–310.

[26] I. Char, Y. Chung, W. Neiswanger, K. Kandasamy, A. O. Nelson, M. Boyer, E. Kolemen, and J. Schneider. "Offline Contextual Bayesian Optimization". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 32. NeurIPS Foundation. 2019.

[27] X. Chen, M. Diessner, K. J. Wilson, and R. D. Whalley. "Optimizing Wall Blowing for Global Skin-Friction Drag Reduction Using a Bayesian Optimization Framework". In: *Symposium on Turbulence and Shear Flow Phenomena (TSFP)*. TSFP. 2024.

[28] Y. Chen, H. Li, K. Jin, and Q. Song. "Wind Farm Layout Optimization Using Genetic Algorithm With Different Hub Height Wind Turbines". In: *Energy Conversion and Management* 70 (2013), pp. 56–65.

[29] C. Chevalier and D. Ginsbourger. "Fast Computation of the Multi-points Expected Improvement With Applications in Batch Selection". In: *International Conference on Learning and Intelligent Optimization (LION)*. Springer. 2013, pp. 59–69.

[30] Y. Chung, I. Char, W. Neiswanger, K. Kandasamy, A. O. Nelson, M. D. Boyer, E. Kolemen, and J. Schneider. "Offline Contextual Bayesian Optimization for Nuclear Fusion". In: *Workshop on Machine Learning and the Physical Sciences (NeurIPS)*. Vol. 32. NeurIPS Foundation. 2019.

[31] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization*. MOS-SIAM Series on Optimization. SIAM, 2009.

[32] 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 (ECML PKDD)*. Springer. 2013, pp. 225–240.

[33] C. Cortes and V. Vapnik. "Support-Vector Networks". In: *Machine Learning* 20 (1995), pp. 273–297.

[34] A. Criminisi, J. Shotton, E. Konukoglu, et al. "Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-supervised Learning". In: *Foundations and Trends in Computer Graphics and Vision* 7.2–3 (2012), pp. 81–227.

[35] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, 2000.

[36] S. Daulton, X. Wan, D. Eriksson, M. Balandat, M. A. Osborne, and E. Bakshy. "Bayesian Optimization over Discrete and Mixed Spaces via Probabilistic Reparameterization". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 35. NeurIPS Foundation. 2022.

[37] N. De Freitas, A. Smola, and M. Zoghi. "Exponential Regret Bounds for Gaussian Process Bandits With Deterministic Observations". In: *International Conference on Machine Learning (ICML)*. PLMR. 2012, pp. 1743–1750.

[38] T. Desautels, A. Krause, and J. W. Burdick. "Parallelizing Exploration-Exploitation Tradeoffs in Gaussian Process Bandit Optimization". In: *Jorunal of Machine Learning Research* 15.1 (2014), pp. 3873–3923.

[39] A. Deshwal, S. Ament, M. Balandat, E. Bakshy, J. R. Doppa, and D. Eriksson. "Bayesian Optimization over High-Dimensional Combinatorial Spaces via Dictionary-Based Embeddings". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR. 2023, pp. 7021–7039.

[40] M. Diessner, X. Chen, K. J. Wilson, and R. D. Whalley. "Optimising Active Flow Control Strategies for Random and Controlled Wind Speeds via Bayesian Optimisation". In: *Symposium on Turbulence and Shear Flow Phenomena (TSFP)*. TSFP. 2024.

[41] M. Diessner, J. O'Connor, A. Wynn, S. Laizet, Y. Guan, K. Wilson, and R. D. Whalley. "Investigating Bayesian Optimization for Expensive-to-Evaluate Black Box Functions: Application in Fluid Dynamics". In: *Frontiers in Applied Mathematics and Statistics* 8 (2022), p. 1076296.

[42] M. Diessner, K. Wilson, and R. D. Whalley. "NUBO: A Transparent Python Package for Bayesian Optimisation". In: *arXiv preprint arXiv:2305.06709* (2023). Accepted by Journal of Statistical Software.

[43] M. Diessner, K. J. Wilson, and R. D. Whalley. "On the Development of a Practical Bayesian Optimization Algorithm for Expensive Experiments and Simulations With Changing Environmental Conditions". In: *Data-Centic Engineering* 5 (2024), e45.

[44] J. Duris, D. Kennedy, A. Hanuka, J. Shtalenkova, A. Edelen, P. Baxevanis, A. Egger, T. Cope, M. McIntire, S. Ermon, et al. "Bayesian Optimization of a Free-Electron Laser". In: *Physical Review Letters* 124.12 (2020), p. 124801.

[45] N. Ebrahimzade, J. Portoles, M. Wilkes, P. Cumpson, and R. D. Whalley. "Optical MEMS Sensors for Instantaneous Wall-Shear Stress Measurements in Turbulent Boundary-Layer Flows". In: *Symposium on Turbulence and Shear Flow Phenomena (TSFP)*. TSFP. 2022, pp. 1–6.

[46] M. T. Emmerich, K. C. Giannakoglou, and B. Naujoks. "Single- and Multiobjective Evolutionary Optimization Assisted by Gaussian Random Field Metamodels". In: *IEEE Transactions on Evolutionary Computation* 10.4 (2006), pp. 421–439.

[47] D. Eriksson and M. Jankowiak. "High-Dimensional Bayesian Optimization With Sparse Axis-Aligned Subspaces". In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. PMLR. 2021, pp. 493–503.

[48] D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek. "Scalable Global Optimization via Local Bayesian Optimization". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 32. NeurIPS Foundation. 2019.

[49] D. Fernández-Sánchez, E. C. Garrido-Merchán, and D. Hernández-Lobato. "Improved Max-Value Entropy Search for Multi-objective Bayesian Optimization With Constraints". In: *Neurocomputing* 546 (2023), p. 126290.

[50] A. I. Forrester, A. Sóbester, and A. J. Keane. "Multi-Fidelity Optimization via Surrogate Modelling". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 463.2088 (2007), pp. 3251–3269.

[51] P. Frazier and W. Powell. "The Knowledge Gradient Policy for Offline Learning With Independent Normal Rewards". In: *International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE. 2007, pp. 143–150.

[52] P. Frazier, W. Powell, and S. Dayanik. "The Knowledge-Gradient Policy for Correlated Normal Beliefs". In: *INFORMS Journal on Computing* 21.4 (2009), pp. 599–613.

[53] P. I. Frazier. "A Tutorial on Bayesian Optimization". In: *arXiv preprint arXiv:1807.02811* (2018).

[54] P. I. Frazier and J. Wang. "Bayesian Optimization for Materials Design". In: *Information Science for Materials Discovery and Design* (2016), pp. 45–75.

[55] J. Gardner, C. Guo, K. Weinberger, R. Garnett, and R. Grosse. "Discovering and Exploiting Additive Structure for Bayesian Optimization". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR. 2017, pp. 1311–1319.

[56] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson. "GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference With GPU Acceleration". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 31. NeurIPS Foundation. 2018.

[57] J. R. Gardner, M. J. Kusner, Z. E. Xu, K. Q. Weinberger, and J. P. Cunningham. "Bayesian Optimization With Inequality Constraints." In: *International Conference on Machine Learning (ICML)*. PMLR. 2014, pp. 937–945.

[58] R. Garnett. *Bayesian Optimization*. Cambridge University Press, 2023.

[59] R. Garnett, M. A. Osborne, and S. J. Roberts. "Bayesian Optimization for Sensor Set Selection". In: *International Conference on Information Processing in Sensor Networks (IPSN)*. ACM/IEEE. 2010, pp. 209–219.

[60] E. C. Garrido-Merchán and D. Hernández-Lobato. "Dealing With Categorical and Integer-Valued Variables in Bayesian Optimization With Gaussian Processes". In: *Neurocomputing* 380 (2020), pp. 20–35.

[61] M. Gasca and T. Sauer. "Polynomial Interpolation in Several Variables". In: *Advances in Computational Mathematics* 12 (2000), pp. 377–410.

[62] M. A. Gelbart, J. Snoek, and R. P. Adams. "Bayesian Optimization With Unknown Constraints". In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. PMLR. 2014, pp. 250–259.

[63] D. Ginsbourger, J. Baccou, C. Chevalier, F. Perales, N. Garland, and Y. Monerie. "Bayesian Adaptive Reconstruction of Profile Optima and Optimizers". In: *SIAM/ASA Journal on Uncertainty Quantification* 2.1 (2014), pp. 490–510.

[64] D. Ginsbourger, R. Le Riche, and L. Carraro. "Kriging Is Well-Suited to Parallelize Optimization". In: *Computational Intelligence in Expensive Optimization Problems*. Springer, 2010, pp. 131–162.

[65] F. Glover and M. Laguna. *Tabu Search*. Springer, 1998.

[66] P. Goldberg, C. Williams, and C. Bishop. "Regression With Input-Dependent Noise: A Gaussian Process Treatment". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 10. NeurIPS Foundation. 1997.

[67] J. González, Z. Dai, P. Hennig, and N. Lawrence. "Batch Bayesian Optimization via Local Penalization". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR. 2016, pp. 648–657.

[68] S. Grady, M. Y. Hussaini, and M. M. Abdullah. "Placement of Wind Turbines Using Genetic Algorithms". In: *Renewable Energy* 30.2 (2005), pp. 259–270.

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

[70] R. B. Gramacy, G. A. Gray, S. L. Digabel, H. K. H. Lee, P. Ranjan, G. Wells, and S. M. Wild. "Modeling an Augmented Lagrangian for Blackbox Constrained Optimization". In: *Technometrics* 58.1 (2016), pp. 1–11.

[71] R. B. Gramacy and H. K. H. Lee. "Optimization Under Unknown Constraints". In: *Bayesian Statistics 9*. Oxford University Press, Oct. 2011.

[72] R. B. Gramacy and H. K. Lee. "Cases for the Nugget in Modeling Computer Experiments". In: *Statistics and Computing* 22 (2012), pp. 713–722.

[73] P. Groot, A. Birlutiu, and T. Heskes. "Bayesian Monte Carlo for the Global Optimization of Expensive Functions". In: *European Conference on Artificial Intelligence (ECAI)*. ECAI. 2010, pp. 249–254.

[74] H.-M. Gutmann. "A Radial Basis Function Method for Global Optimization". In: *Journal of Global Optimization* 19.3 (2001), pp. 201–227.

[75] J. A. Hanley and B. J. McNeil. "The Meaning and Use of the Area Under a Receiver Operating Characteristic (ROC) Curve". In: *Radiology* 143.1 (1982), pp. 29–36.

[76] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, et al. "Array Programming With NumPy". In: *Nature* 585.7825 (2020), pp. 357–362.

[77] Havard University, University of Toronto, and Socpra Sciences et Génie S.E.C. *Spearmint*. 2014. URL: `https://github.com/HIPS/Spearmint`.

[78] P. Hennig and C. J. Schuler. "Entropy Search for Information-Efficient Global Optimization". In: *Journal of Machine Learning Research* 13.6 (2012).

[79] D. Hernández-Lobato, J. Hernandez-Lobato, A. Shah, and R. Adams. "Predictive Entropy Search for Multi-objective Bayesian Optimization". In: *International Conference on Machine Learning (ICML)*. PMLR. 2016, pp. 1492–1501.

[80] J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani. "Predictive Entropy Search for Efficient Global Optimization of Black-Box Functions". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 27. NeurIPS Foundation. 2014.

[81] J. M. Hernández-Lobato, J. Requeima, E. O. Pyzer-Knapp, and A. Aspuru-Guzik. "Parallel and Distributed Thompson Sampling for Large-Scale Accelerated Exploration of Chemical Space". In: *International Conference on Machine Learning (ICML)*. PMLR. 2017, pp. 1470–1479.

[82] T. N. Hoang, Q. M. Hoang, R. Ouyang, and K. H. Low. "Decentralized High-Dimensional Bayesian Optimization With Factor Graphs". In: *Conference on Artificial Intelligence (CAI)*. Vol. 32. 1. AAAI. 2018.

[83] M. Hoffman, E. Brochu, N. De Freitas, et al. "Portfolio Allocation for Bayesian Optimization". In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. PMLR. 2011, pp. 327–336.

[84] D. Huang, T. T. Allen, W. I. Notz, and R. A. Miller. "Sequential Kriging Optimization Using Multiple-Fidelity Evaluations". In: *Structural and Multidisciplinary Optimization* 32 (2006), pp. 369–382.

[85] D. Huang, T. T. Allen, W. I. Notz, and N. Zeng. "Global Optimization of Stochastic Black-Box Systems via Sequential Kriging Meta-Models". In: *Journal of Global Optimization* 34 (2006), pp. 441–466.

[86] B. G. Husslage, G. Rennen, E. R. Van Dam, and D. Den Hertog. "Space-Filling Latin Hypercube Designs for Computer Experiments". In: *Optimization and Engineering* 12 (2011), pp. 611–630.

[87] F. Hutter, H. H. Hoos, and K. Leyton-Brown. "Sequential Model-Based Optimization for General Algorithm Configuration". In: *International Conference on Learning and Intelligent Optimization (LION)*. Springer. 2011, pp. 507–523.

[88] F. Hutter, H. H. Hoos, K. Leyton-Brown, and K. P. Murphy. "An Experimental Investigation of Model-Based Parameter Optimisation: SPO and Beyond". In: *Genetic and Evolutionary Computation Conference (GECCO)*. ACM. 2009, pp. 271–278.

[89] D. Hwang. "Review of Research into the Concept of the Microblowing Technique for Turbulent Skin Friction Reduction". In: *Progress in Aerospace Sciences* 40.8 (2004), pp. 559–575.

[90] J. Jiménez and J. Ginebra. "pyGPGO: Bayesian Optimization for Python". In: *Journal of Open Source Software* 2.19 (2017), p. 431.

[91] D. R. Jones. "A Taxonomy of Global Optimization Methods Based on Response Surfaces". In: *Journal of Global Optimization* 21 (2001), pp. 345–383.

[92] D. R. Jones. "Direct Global Optimization Algorithm". In: *Encyclopedia of Optimization*. Springer, 2001, pp. 431–440.

[93] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. "Lipschitzian Optimization Without the Lipschitz Constant". In: *Journal of Optimization Theory and Applications* 79 (1993), pp. 157–181.

[94] D. R. Jones, M. Schonlau, and W. J. Welch. "Efficient Global Optimization of Expensive Black-Box Functions". In: *Journal of Global Optimization* 13 (1998), pp. 455–492.

[95] D. A. Juangarcia, I. Eguinoa, and T. Knudsen. "Derating a Single Wind Farm Turbine for Reducing Its Wake and Fatigue". In: *Journal of Physics: Conference Series (JPCS)*. Vol. 1037. 3. IOP Publishing. 2018.

[96] L. P. Kaelbling, M. L. Littman, and A. W. Moore. "Reinforcement Learning: A Survey". In: *Journal of Artificial Intelligence Research* 4 (1996), pp. 237–285.

[97] K. Kandasamy, G. Dasarathy, J. B. Oliva, J. Schneider, and B. Póczos. "Gaussian Process Bandit Optimisation With Multi-Fidelity Evaluations". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 29. NeurIPS Foundation. 2016.

[98] K. Kandasamy, G. Dasarathy, J. Schneider, and B. Póczos. "Multi-Fidelity Bayesian Optimisation With Continuous Approximations". In: *International Conference on Machine Learning (ICML)*. PMLR. 2017, pp. 1799–1808.

[99] K. Kandasamy, A. Krishnamurthy, J. Schneider, and B. Póczos. "Parallelised Bayesian Optimisation via Thompson Sampling". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR. 2018, pp. 133–142.

[100] K. Kandasamy, J. Schneider, and B. Póczos. "High Dimensional Bayesian Optimisation and Bandits via Additive Models". In: *International Conference on Machine Learning (ICML)*. PMLR. 2015, pp. 295–304.

[101] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. "Physics-Informed Machine Learning". In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440.

[102] T. Kathuria, A. Deshpande, and P. Kohli. "Batched Gaussian Process Bandit Optimization via Determinantal Point Processes". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 29. NeurIPS Foundation. 2016.

[103] D. Khatamsaz, R. Neuberger, A. M. Roy, S. H. Zadeh, R. Otis, and R. Arróyave. "A Physics Informed Bayesian Optimization Approach for Material Design: Application to NiTi Shape Memory Alloys". In: *npj Computational Materials* 9.1 (2023), p. 221.

[104] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[105] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi. "Optimization by Simulated Annealing". In: *Science* 220.4598 (1983), pp. 671–680.

[106] V. I. Kornilov and A. V. Boiko. "Efficiency of Air Microblowing Through Microperforated Wall for Flat Plate Drag Reduction". In: *AIAA Journal* 50.3 (2012), pp. 724–732.

[107] D. Kraft. "Algorithm 733: TOMP–Fortran Modules for Optimal Control Calculations". In: *ACM Transactions on Mathematical Software (TOMS)* 20.3 (1994), pp. 262–281.

[108] A. Krause and C. Ong. "Contextual Gaussian Process Bandit Optimization". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 24. NeurIPS Foundation. 2011.

[109] A. Kulesza, B. Taskar, et al. "Determinantal Point Processes for Machine Learning". In: *Foundations and Trends in Machine Learning* 5.2–3 (2012), pp. 123–286.

[110] H. J. Kushner. "A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise". In: *Journal of Basic Engineering* (1964).

[111] H. J. Kushner. "A Versatile Stochastic Model of a Function of Unknown and Time Varying Form". In: *Journal of Mathematical Analysis and Applications* 5.1 (1962), pp. 150–167.

[112] T. L. Lai and H. Robbins. "Asymptotically Efficient Adaptive Allocation Rules". In: *Advances in Applied Mathematics* 6.1 (1985), pp. 4–22.

[113] J. Larson, M. Menickelly, and S. M. Wild. "Derivative-Free Optimization Methods". In: *Acta Numerica* 28 (2019), pp. 287–404.

[114] T. Lattimore and C. Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

[115] B. Letham, R. Calandra, A. Rai, and E. Bakshy. "Re-Examining Linear Embeddings for High-Dimensional Bayesian Optimization". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 33. NeurIPS Foundation. 2020, pp. 1546–1558.

[116] E. Levina and P. Bickel. "Maximum Likelihood Estimation of Intrinsic Dimension". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 17. NeurIPS Foundation. 2004.

[117] Y. Li, K. G. Reyes, J. Vazquez-Anderson, Y. Wang, L. M. Contreras, and W. B. Powell. "A Knowledge Gradient Policy for Sequencing Experiments To Identify the Structure of RNA Molecules Using a Sparse Additive Belief Model". In: *INFORMS Journal on Computing* 30.4 (2018), pp. 750–767.

[118] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. "SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization". In: *Journal of Machine Learning Research* 23.54 (2022), pp. 1–9.

[119] D. C. Liu and J. Nocedal. "On the Limited Memory BFGS Method for Large Scale Optimization". In: *Mathematical Programming* 45.1 (1989), pp. 503–528.

[120] S. Liu, Q. Feng, D. Eriksson, B. Letham, and E. Bakshy. "Sparse Bayesian Optimization". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR. 2023, pp. 3754–3774.

[121] D. J. Lizotte. "Practical Bayesian Optimization". PhD thesis. University of Alberta, 2008.

[122] W. Lyu, P. Xue, F. Yang, C. Yan, Z. Hong, X. Zeng, and D. Zhou. "An Efficient Bayesian Optimization Approach for Automated Optimization of Analog Circuits". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 65.6 (2017), pp. 1954–1967.

[123] O. Mahfoze, A. Moody, A. Wynn, R. Whalley, and S. Laizet. "Reducing the Skin-Friction Drag of a Turbulent Boundary-Layer Flow With Low-Amplitude Wall-Normal Blowing Within a Bayesian Optimization Framework". In: *Physical Review Fluids* 4.9 (2019), p. 094601.

[124] F. Mallor, G. Semprini-Cesari, T. Mukha, S. Rezaeiravesh, and P. Schlatter. "Bayesian Optimization of Wall-Normal Blowing and Suction-Based Flow Control of a Naca 4412 Wing Profile". In: *Flow, Turbulence and Combustion* (2023), pp. 1–26.

[125] A. Maraval, M. Zimmer, A. Grosnit, R. Tutunov, J. Wang, and H. B. Ammar. "Sample-Efficient Optimisation With Probabilistic Transformer Surrogates". In: *arXiv preprint arXiv:2205.13902* (2022).

[126] S. Marmin, C. Chevalier, and D. Ginsbourger. "Differentiating the Multipoint Expected Improvement for Optimal Batch Design". In: *Machine Learning, Optimization and Big Data (MOD)*. Lecture Notes in Computer Science. Springer. 2015, pp. 37–48.

[127] R. Martinez-Cantin, N. De Freitas, E. Brochu, J. Castellanos, and A. Doucet. "A Bayesian Exploration-Exploitation Approach for Optimal Online Sensing and Planning With a Visually Guided Mobile Robot". In: *Autonomous Robots* 27 (2009), pp. 93–103.

[128] G. Matheron. *The Theory of Regionalized Variables and Its Applications*. Paris School of Mines, 1971.

[129] M. D. McKay, R. J. Beckman, and W. J. Conover. "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output From a Computer Code". In: *Technometrics* 42.1 (2000), pp. 55–61.

[130] M. McLeod, M. A. Osborne, and S. J. Roberts. "Practical Bayesian Optimization for Variable Cost Objectives". In: *arXiv preprint arXiv:1703.04335* (2017).

[131] J. Močkus. "Application of Bayesian Approach to Numerical Methods of Global and Stochastic Optimization". In: *Journal of Global Optimization* 4 (1994), pp. 347–365.

[132] J. Močkus. *Bayesian Approach to Global Optimization: Theory and Applications*. Vol. 37. Mathematics and Its Applications. Springer, 1989, pp. 125–156.

[133] J. Močkus. "Bayesian Methods of Search for an Extremum". In: *Avtomatika i Vychislitel'naya Tekhnika (Automatic Control and Computer Sciences)* 6.3 (1972), pp. 53–62.

[134] J. Močkus. "On Bayesian Methods for Seeking the Extremum". In: *Technical Conference on Optimization Techniques*. Vol. 27. Lecture Notes in Computer Science. IFIP. 1974, pp. 400–404.

[135] R. Moriconi, M. P. Deisenroth, and K. Sesh Kumar. "High-Dimensional Bayesian Optimization Using Low-Dimensional Feature Spaces". In: *Machine Learning* 109 (2020), pp. 1925–1943.

[136] Y. Morita, S. Rezaeiravesh, N. Tabatabaei, R. Vinuesa, K. Fukagata, and P. Schlatter. "Applying Bayesian Optimization With Gaussian Process Regression to Computational Fluid Dynamics Problems". In: *Journal of Computational Physics* 449 (2022), p. 110788.

[137] G. Mosetti, C. Poloni, and B. Diviacco. "Optimization of Wind Turbine Positioning in Large Windfarms by Means of a Genetic Algorithm". In: *Journal of Wind Engineering and Industrial Aerodynamics* 51.1 (1994), pp. 105–116.

[138] M. Mutny and A. Krause. "Proceedings: Efficient High Dimensional Bayesian Optimization With Additivity and Quadrature Fourier Features". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 31. NeurIPS Foundation. 2018.

[139] Y. Nabae and K. Fukagata. "Bayesian Optimization of Traveling Wave-Like Wall Deformation for Friction Drag Reduction in Turbulent Channel Flow". In: *Journal of Fluid Science and Technology* 16.4 (2021).

[140] A. Nayebi, A. Munteanu, and M. Poloczek. "A Framework for Bayesian Optimization in Embedded Subspaces". In: *International Conference on Machine Learning (ICML)*. PMLR. 2019, pp. 4752–4761.

[141] R. M. Neal. *Bayesian Learning for Neural Networks*. Lecture Notes in Statistics. Springer Science & Business Media, 1996.

[142] D. M. Negoescu, P. I. Frazier, and W. B. Powell. "The Knowledge-Gradient Algorithm for Sequencing Experiments in Drug Discovery". In: *INFORMS Journal on Computing* 23.3 (2011), pp. 346–363.

[143] J. A. Nelder and R. Mead. "A Simplex Method for Function Minimization". In: *The Computer Journal* 7.4 (1965), pp. 308–313.

[144] F. Nogueira. *bayes_opt: Open Source Constrained Global Optimization Tool for Python*. 2014. URL: `https://github.com/fmfn/BayesianOptimization`.

[145] J. O'Connor, M. Diessner, K. Wilson, R. D. Whalley, A. Wynn, and S. Laizet. "Optimisation and Analysis of Streamwise-Varying Wall-Normal Blowing in a Turbulent Boundary Layer". In: *Flow, Turbulence and Combustion* 110.4 (2023), pp. 993–1021.

[146] C. Oh, E. Gavves, and M. Welling. "BOCK: Bayesian Optimization With Cylindrical Kernels". In: *International Conference on Machine Learning (ICML)*. PMLR. 2018, pp. 3868–3877.

[147] D. Packwood. *Bayesian Optimization for Materials Science*. SpringerBriefs in the Mathematics of Materials. Springer, 2017.

[148] A. Papoulis. *Probability, Random Variables and Stochastic Processes*. McGraw-Hill, 1965.

[149] L. Parada, C. Herrera, P. Flores, and V. Parada. "Wind Farm Layout Optimization Using a Gaussian-Based Wake Model". In: *Renewable Energy* 107 (2017), pp. 531–541.

[150] B. Paria, K. Kandasamy, and B. Póczos. "A Flexible Framework for Multi-Objective Bayesian Optimization Using Random Scalarizations". In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. PMLR. 2020, pp. 766–776.

[151] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 32. NeurIPS Foundation. 2019.

[152] M. Pearce and J. Branke. "Continuous Multi-Task Bayesian Optimisation With Correlation". In: *European Journal of Operational Research* 270.3 (2018), pp. 1074–1085.

[153] M. Pearce and J. Branke. "Efficient Information Collection on Portfolios". In: *Working Paper. University of Warwick* (2017).

[154] M. M. Pedersen, A. M. Forsting, P. van der Laan, R. Riva, L. A. A. Romàn, J. C. Risco, M. Friis-Møller, J. Quick, J. P. S. Christiansen, R. V. Rodrigues, B. T. Olsen, and P.-E. Réthoré. *PyWake 2.5.0: An Open-Source Wind Farm Simulation Tool*. DTU Wind, Technical University of Denmark. 2023. URL: `https://gitlab.windenergy.dtu.dk/TOPFARM/PyWake`.

[155] V. Picheny, D. Ginsbourger, Y. Richet, and G. Caplin. "Quantile-Based Optimization of Noisy Computer Experiments With Tunable Precision". In: *Technometrics* 55.1 (2013), pp. 2–13.

[156] R. Planas, N. Oune, and R. Bostanabad. "Extrapolation With Gaussian Random Processes and Evolutionary Programming". In: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC-CIE)*. Vol. 84003. American Society of Mechanical Engineers. 2020.

[157] W. Ponweiser, T. Wagner, D. Biermann, and M. Vincze. "Multiobjective Optimization on a Limited Budget of Evaluations Using Model-Assisted-Metric Selection". In: *International Conference on Parallel Problem Solving from Nature (PPSN)*. Springer. 2008, pp. 784–794.

[158] W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley Series in Probability and Statistics. John Wiley & Sons, 2007.

[159] Python Software Foundation. *Python Package Index - PyPI*. URL: `https://pypi.org/` (visited on 06/17/2024).

[160] T. A. Qureshi and V. Warudkar. "Wind Farm Layout Optimization Through Optimal Wind Turbine Placement Using a Hybrid Particle Swarm Optimization and Genetic Algorithm". In: *Environmental Science and Pollution Research* 30.31 (2023), pp. 77436–77452.

[161] E. Raponi, H. Wang, M. Bujny, S. Boria, and C. Doerr. "High Dimensional Bayesian Optimization Assisted by Principal Component Analysis". In: *Parallel Problem Solving from Nature (PPSN)*. Springer. 2020, pp. 169–183.

[162] C. E. Rasmussen and C. K. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2005.

[163] R. G. Regis and C. A. Shoemaker. "Constrained Global Optimization of Expensive Black Box Functions Using Radial Basis Functions". In: *Journal of Global Optimization* 31 (2005), pp. 153–171.

[164] R. G. Regis and C. A. Shoemaker. "Improved Strategies for Radial Basis Function Methods for Global Optimization". In: *Journal of Global Optimization* 37 (2007), pp. 113–135.

[165] R. G. Regis and C. A. Shoemaker. "Parallel Radial Basis Function Methods for the Global Optimization of Expensive Functions". In: *European Journal of Operational Research* 182.2 (2007), pp. 514–535.

[166] P. Rolland, J. Scarlett, I. Bogunovic, and V. Cevher. "High-Dimensional Bayesian Optimization via Additive Models With Overlapping Groups". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. PMLR. 2018, pp. 298–307.

[167] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. "Design and Analysis of Computer Experiments". In: *Statistical Science* 4.4 (1989), pp. 409–423.

[168] V. R. Šaltenis. "One Method of Multiextremum Optimization". In: *Avtomatika i Vychislitel'naya Tekhnika (Automatic Control and Computer Sciences)* 5.3 (1971), pp. 33–38.

[169] M. J. Sasena. "Flexibility and Efficiency Enhancements for Constrained Global Design Optimization With Kriging Approximations". PhD thesis. University of Michigan, 2002.

[170] P. Schlatter and R. Örlü. "Assessment of Direct Numerical Simulation Data of Turbulent Boundary Layers". In: *Journal of Fluid Mechanics* 659 (2010), pp. 116–126.

[171] M. Schonlau. "Computer Experiments and Global Optimization". PhD thesis. University of Waterloo, 1997.

[172] M. Schonlau, W. J. Welch, and D. R. Jones. "Global Versus Local Search in Constrained Optimization of Computer Models". In: *Lecture Notes-Monograph Series* (1998), pp. 11–25.

[173] W. Scott, P. Frazier, and W. Powell. "The Correlated Knowledge Gradient for Simulation Optimization of Continuous Parameters Using Gaussian Process Regression". In: *SIAM Journal on Optimization* 21.3 (2011), pp. 996–1026.

[174] A. Shah and Z. Ghahramani. "Parallel Predictive Entropy Search for Batch Global Optimization of Expensive Objective Functions". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 28. NeurIPS Foundation. 2015.

[175] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. "Taking the Human out of the Loop: A Review of Bayesian Optimization". In: *Proceedings of the IEEE* 104.1 (2015), pp. 148–175.

[176] B. Shahriari, Z. Wang, M. W. Hoffman, A. Bouchard-Côté, and N. de Freitas. "An Entropy Search Portfolio for Bayesian Optimization". In: *arXiv preprint arXiv:1406.4625* (2014).

[177] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Vol. 26. Monographs on Statistics and Probability. Chapman and Hall/CRC, 1986.

[178] J. Snoek, H. Larochelle, and R. P. Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 25. NeurIPS Foundation. 2012.

[179] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams. "Scalable Bayesian Optimization Using Deep Neural Networks". In: *International Conference on Machine Learning (ICML)*. PMLR. 2015, pp. 2171–2180.

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

[181] J. R. Snoek. "Bayesian Optimization and Semiparametric Models With Applications to Assistive Technology". PhD thesis. University of Toronto, 2013.

[182] A. Sóbester, S. J. Leary, and A. J. Keane. "A Parallel Updating Scheme for Approximating and Optimizing High Fidelity Computer Simulations". In: *Structural and Multidisciplinary Optimization* 27 (2004), pp. 371–383.

[183] J. T. Springenberg, A. Klein, S. Falkner, and F. Hutter. "Bayesian Optimization With Robust Bayesian Neural Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 29. NeurIPS Foundation. 2016.

[184] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. "Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design". In: *International Conference on Machine Learning (ICML)*. PMLR. 2010, pp. 1015–1022.

[185] N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger. "Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting". In: *IEEE Transactions on Information Theory* 58.5 (2012), pp. 3250–3265.

[186] D. Sterling, T. Sterling, Y. Zhang, and H. Chen. "Welding Parameter Optimization Based on Gaussian Process Regression Bayesian Optimization Algorithm". In: *International Conference on Automation Science and Engineering (CASE)*. IEEE. 2015, pp. 1490–1496.

[187] R. Storn and K. Price. "Differential Evolution–a Simple and Efficient Heuristic for Global Optimization Over Continuous Spaces". In: *Journal of Global Optimization* 11 (1997), pp. 341–359.

[188] S. Surjanovic and D. Bingham. *Virtual Library of Simulation Experiments: Test Functions and Datasets*. URL: `https://www.sfu.ca/~ssurjano/optimization.html` (visited on 06/17/2024).

[189] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

[190] K. Swersky, J. Snoek, and R. P. Adams. "Multi-Task Bayesian Optimization". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 26. NeurIPS Foundation. 2013.

[191] S. Takeno, H. Fukuoka, Y. Tsukada, T. Koyama, M. Shiga, I. Takeuchi, and M. Karasuyama. "Multi-Fidelity Bayesian Optimization With Max-Value Entropy Search and Its Parallelization". In: *International Conference on Machine Learning (ICML)*. PMLR. 2020, pp. 9334–9345.

[192] C. Talnikar, P. Blonigan, J. Bodart, and Q. Wang. "Parallel Optimization for Large Eddy Simulations". In: *arXiv preprint arXiv:1410.8859* (2014).

[193] The GPyOpt authors. *GPyOpt: A Bayesian Optimization Framework in Python*. 2016. URL: `http://github.com/SheffieldML/GPyOpt`.

[194] The pip developers. *pip: Package Installer for Python*. 2023. URL: `https://pip.pypa.io/en/stable/`.

[195] W. R. Thompson. "On the Likelihood That One Unknown Probability Exceeds Another in View of the Evidence of Two Samples". In: *Biometrika* 25.3-4 (1933), pp. 285–294.

[196] L. C. Tiao, A. Klein, M. W. Seeger, E. V. Bonilla, C. Archambeau, and F. Ramos. "BORE: Bayesian Optimization by Density-Ratio Estimation". In: *International Conference on Machine Learning (ICML)*. PMLR. 2021, pp. 10289–10300.

[197] A. Törn and A. Žilinskas. *Global Optimization*. Springer, 1989.

[198] S. Toscano-Palmerin and P. I. Frazier. "Bayesian Optimization With Expensive Integrands". In: *SIAM Journal on Optimization* 32.2 (2022), pp. 417–444.

[199] I. Vernica, K. Ma, and F. Blaabjerg. "Optimal Derating Strategy of Power Electronics Converter for Maximum Wind Energy Production With Lifetime Information of Power Devices". In: *IEEE Journal of Emerging and Selected Topics in Power Electronics* 6.1 (2017), pp. 267–276.

[200] J. Villemonteix, E. Vazquez, and E. Walter. "An Informational Approach to the Global Optimization of Expensive-To-Evaluate Functions". In: *Journal of Global Optimization* 44 (2009), pp. 509–534.

[201] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17.3 (2020), pp. 261–272.

[202] K. K. Vu, C. d'Ambrosio, Y. Hamadi, and L. Liberti. "Surrogate-Based Methods for Black-Box Optimization". In: *International Transactions in Operational Research* 24.3 (2017), pp. 393–424.

[203] K. Wang and A. W. Dowling. "Bayesian Optimization for Chemical Products and Functional Materials". In: *Current Opinion in Chemical Engineering* 36 (2022), p. 100728.

[204] Z. Wang and S. Jegelka. "Max-Value Entropy Search for Efficient Bayesian Optimization". In: *International Conference on Machine Learning (ICML)*. PMLR. 2017, pp. 3627–3635.

[205] Z. Wang and N. de Freitas. "Theoretical Analysis of Bayesian Optimisation With Unknown Gaussian Process Hyper-Parameters". In: *arXiv preprint arXiv:1406.7758* (2014).

[206] Z. Wang, F. Hutter, M. Zoghi, D. Matheson, and N. De Feitas. "Bayesian Optimization in a Billion Dimensions via Random Embeddings". In: *Journal of Artificial Intelligence Research* 55 (2016), pp. 361–387.

[207] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. De Freitas. "Bayesian Optimization in High Dimensions via Random Embeddings". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI. 2013, pp. 1778–1784.

[208] C. White, W. Neiswanger, and Y. Savani. "BANANAS: Bayesian Optimization With Neural Architectures for Neural Architecture Search". In: *Conference on Artificial Intelligence (CAI)*. Vol. 35. 12. AAAI. 2021, pp. 10293–10301.

[209] B. J. Williams, T. J. Santner, and W. I. Notz. "Sequential Design of Computer Experiments To Minimize Integrated Response Functions". In: *Statistica Sinica* 10.4 (2000), pp. 1133–1152.

[210] B. J. Williams. "Sequential Design of Computer Experiments To Minimize Integrated Response Functions". PhD thesis. The Ohio State University, 2000.

[211] J. Wilson, F. Hutter, and M. Deisenroth. "Maximizing Acquisition Functions for Bayesian Optimization". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 31. NeurIPS Foundation. 2018.

[212] J. T. Wilson, R. Moriconi, F. Hutter, and M. P. Deisenroth. "The Reparameterization Trick for Acquisition Functions". In: *arXiv preprint arXiv:2402.03006* (2017).

[213] S. Wilson. *ParBayesianOptimization: Parallel Bayesian Optimization of Hyperparameters*. 2020. URL: `https://cran.r-project.org/package=ParBayesianOptimization`.

[214] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, and S.-H. Deng. "Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization". In: *Journal of Electronic Science and Technology* 17.1 (2019), pp. 26–40.

[215] J. Wu and P. Frazier. "The Parallel Knowledge Gradient Method for Batch Bayesian Optimization". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 29. NeurIPS Foundation. 2016.

[216] 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: *Annual Allerton Conference on Communication, Control, and Computing*. IEEE. 2012, pp. 406–413.

[217] Y. Yan. *rBayesianOptimization: Bayesian Optimization of Hyperparameters*. 2016. URL: `https://cran.r-project.org/web/packages/rBayesianOptimization/index.html`.

[218] K. Yang, M. Emmerich, A. Deutz, and T. Bäck. "Efficient Computation of Expected Hypervolume Improvement Using Box Decomposition Algorithms". In: *Journal of Global Optimization* 75 (2019), pp. 3–34.

[219] K. Yang, M. Emmerich, A. Deutz, and T. Bäck. "Multi-Objective Bayesian Global Optimization Using Expected Hypervolume Improvement Gradient". In: *Swarm and Evolutionary Computation* 44 (2019), pp. 945–956.

[220] Z.-H. Zhan, L. Shi, K. C. Tan, and J. Zhang. "A Survey on Evolutionary Computation for Complex Continuous Optimization". In: *Artificial Intelligence Review* 55.1 (2022), pp. 59–110.

[221] Y. Zhang, T. N. Hoang, B. K. H. Low, and M. Kankanhalli. "Information-Based Multi-Fidelity Bayesian Optimization". In: *Workshop on Bayesian Optimization (NeurIPS)*. Vol. 49. NeurIPS Foundation. 2017.

[222] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. "Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization". In: *ACM Transactions on Mathematical Software (TOMS)* 23.4 (1997), pp. 550–560.

[223] A. G. Žilinskas. "Single-Step Bayesian Search Method for an Extremum of Functions of a Single Variable". In: *Cybernetics* 11.1 (1975), pp. 160–166.