Spring 2024 Semester Recap

Ethan Hersch and Darian Nwankwo

Cornell University

Abstract. Throughout this semester, Ethan and Darian have engaged in extensive literature review, discovered real-world optimization problems to inspire research, and coded Bayesian optimization from the ground up. This paper explains this study throughout the semester. It also provides insight on a future research direction for following semesters.

Keywords: Bayesian optimization, Multi-Fidelity Bayesian optimization, Stellarator Optimization.

1 Introduction of Bayesian optimization

Bayesian optimization (BO) is a powerful technique for optimizing expensive, black-box functions. It treats the objective function as a probabilistic model and iteratively updates this model based on observations, typically using Gaussian Processes (GPs). The key idea is to balance exploration (sampling in regions of high uncertainty) and exploitation (sampling where the objective is expected to be optimal) to efficiently find the global optimum. The goal of this document is to detail the mathematics which drive Bayesian optimization and explain a Julia notebook titled "Multi-Fidelity Bayesian optimization: Investigating a Minimal Toy Problem" written by Ethan Hersch and Darian Nwankwo.

It is first useful to mention why Bayesian optimization is important to study. Here are several reasons.

Efficiency: Bayesian optimization is known for its efficiency in finding the optimal solution, especially in scenarios where evaluating the objective function is time-consuming or expensive. By intelligently selecting the next set of parameters to evaluate based on the current model of the objective function, Bayesian optimization reduces the number of evaluations required to find the optimal solution.

In practice, the optimal solution might be hard to find, but BO intelligently leverages data to find better solutions over time. If given a large enough budget, we can increase the chances of finding the global optimum. It is efficient about searching the parameter space, but, in practice, we don't search the entire parameter space since it can be quite large.

Flexibility: Bayesian optimization can handle various types of objective functions, including non-convex, noisy, and multimodal functions. This makes it applicable in a wide range of domains, such as hyperparameter tuning for machine learning models, experimental design in scientific research, and optimizing parameters in engineering design.

Uncertainty Quantification: Bayesian optimization provides a principled framework for quantifying uncertainty in the optimization process. By modeling the objective function as a probabilistic surrogate, Bayesian optimization can estimate the uncertainty associated with each candidate solution. This information is valuable for decision-making, as it allows users to balance exploration and exploitation effectively.

Adaptive Sampling: Bayesian optimization employs adaptive sampling, dynamically selecting the next set of parameters to evaluate based on the current knowledge of the objective function. While some traditional optimization methods rely on fixed grids or random sampling of parameter space, Bayesian optimization's adaptive approach allows it to focus resources on promising regions, often leading to faster convergence towards optimal solutions. However, it's essential to note that certain scenarios, such as optimization tasks with high experimental setup costs, may favor non-adaptive methods due to their ability to maximize information gain from each costly evaluation. For instance, in agricultural optimization like crop yield enhancement,

where experiments require extensive time and resources, non-adaptive approaches may be more suitable for maximizing efficiency over extended evaluation periods.

Global Optimization: Bayesian optimization aims to find the global optimum (within a specified domain) of the objective function rather than getting stuck in local optima. By maintaining a probabilistic model of the objective function and actively exploring regions of high uncertainty, Bayesian optimization can efficiently search the entire parameter space to find the global optimum.

Overall, the combination of efficiency, flexibility, uncertainty quantification, adaptive sampling, and global optimization makes Bayesian optimization a popular choice for optimizing complex, expensive, and black-box functions in various real-world applications.

1.1 Mathematical Foundation

At its core, BO leverages Bayesian inference to model the objective function as a Gaussian Process (GP). A GP defines a prior distribution over functions, which is updated with observed data to obtain a posterior distribution that represents our belief about the true objective function.

Given observations $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, where x_i represents input features and y_i is the corresponding function value (possibly noisy), the posterior distribution over the objective function f is distributed according to a GP:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')),$$

where $m(x)$ is the mean function and $k(x, x')$ is the covariance function (kernel). The conditional distribution (conditioned on $\mathcal{D}$) is defined by this predictive mean and predictive variance. The choice of kernel encodes assumptions about the function's smoothness and behavior.

1.2 Acquisition Functions

Acquisition functions play a crucial role in guiding the selection of the next point to evaluate in the optimization process. These functions balance exploration (sampling in regions of high uncertainty) and exploitation (sampling where the objective is expected to be optimal). Here's an explanation of some common acquisition functions:

Probability of Improvement (PI) The probability of improvement (PI) is defined as follows:

$$PI(x) = \Phi\left(\frac{\mu(x) - f^+ - \xi}{\sigma(x)}\right)$$

where f^+ denotes the best value (maximum) known, $\mu(x)$ is the predictive mean at x, $\sigma(x)$ is the predictive variance, ξ is our exploration parameter, and Φ is the standard normal cumulative distribution function.

PI assesses the likelihood that sampling a particular point will result in an improvement over the current best value. It balances exploration and exploitation by favoring points with a higher chance of surpassing the current best value. Mathematically, it computes the probability that the improvement in the predicted mean exceeds a certain threshold, considering the uncertainty in the prediction.

Expected Improvement (EI) The expected improvement (EI) is defined as follows:

$$EI(x) = (\mu(x) - f^+ - \xi)\Phi\left(\frac{\mu(x) - f^+ - \xi}{\sigma(x)}\right) + \sigma(x)\Phi\left(\frac{\mu(x) - f^+ - \xi}{\sigma(x)}\right)$$

where Φ is the standard normal probability density function.

EI calculates the expected improvement over the current best value for each candidate point. It incorporates both the magnitude of improvement and the uncertainty associated with it. EI balances exploration and exploitation by favoring points with high expected improvement, taking into account both the mean improvement and the uncertainty in the prediction.

Lower Confidence Bound (LCB) The lower confidence bound (LCB) is defined as follows:

$$UCB(x) = \mu(x) - \beta\sigma(x)$$

where β is some exploration parameter.

LCB selects points based on their potential to be lower than the current best value with high confidence. It prioritizes exploration by favoring points with high uncertainty. When using LCB, unlike the other functions discussed, the goal is to minimize this acquisition function to determine the next point to evaluate. LCB encourages sampling in regions where the predicted mean is low relative to the uncertainty, allowing exploration of potentially promising but uncertain regions of the search space.

Upper Confidence Bound (UCB) The upper confidence bound (UCB) is defined as follows:

$$UCB(x) = \mu(x) + \beta\sigma(x)$$

where β is some exploration parameter.

UCB selects points based on their potential to be higher than the current best value with high confidence. It balances exploration and exploitation by favoring points with high expected value and low uncertainty. UCB encourages exploitation by sampling points with high predicted mean values and low uncertainty, exploiting regions likely to contain the optimal solution while still exploring other regions to refine the model's understanding of the search space.

These acquisition functions help BO algorithms decide where to sample next by evaluating the utility of sampling at each point based on the current GP model. By intelligently selecting points for evaluation, BO efficiently explores the parameter space and converges to the optimal solution with a minimal number of evaluations.

1.3 Multi-Fidelity Bayesian optimization (MFBO)

In scenarios where evaluating the objective function at high fidelity is costly, MFBO extends traditional BO by incorporating multiple levels of fidelity. This allows leveraging cheaper, lower-fidelity approximations to inform decisions about where to sample next.

Cokriging is a technique used in MFBO to model correlations between different fidelity levels. By jointly modeling high and low-fidelity data, cokriging improves the accuracy of the surrogate model and facilitates information transfer between fidelities. We often refer to cokriging as multi-output Gaussian process regresion.

The study of MFBO is still relatively similar to our research this semester. This will likely be a starting point for next semester.

2 Implementing Bayesian Optimization: A Dance of Theory with Practice

Here are some tactics which are useful when implementing Bayesian optimization in code.

2.1 Cholesky Decomposition

The Cholesky decomposition is a matrix factorization method that decomposes a positive definite matrix into the product of a lower triangular matrix and its conjugate transpose. Given a symmetric positive definite matrix A, the Cholesky decomposition expresses it as:

$$A = LL^T$$

where L is a lower triangular matrix. The Cholesky decomposition is particularly useful in numerical linear algebra and optimization, as it provides an efficient way to solve systems of linear equations, compute determinants, and invert matrices. The Cholesky decomposition is computed iteratively, and its complexity is $O(n^3)$, where n is the size of the matrix. It is more numerically stable than other matrix factorization methods like LU decomposition when dealing with positive definite matrices.

Applications of the Cholesky decomposition include solving linear least squares problems, simulating multivariate normal distributions, and preconditioning systems of linear equations to improve convergence of iterative solvers. Understanding the Cholesky decomposition is fundamental in numerical computations and optimization algorithms, as it underpins many computational techniques for solving linear algebraic problems efficiently and accurately.

3 Schur Complement

The Schur complement is a concept frequently encountered in linear algebra and optimization. In its essence, given a block matrix:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

where A and D are square matrices, the Schur complement of D with respect to A in this block matrix is defined as:

$$S = D - CA^{-1}B$$

This formulation appears in various optimization contexts, often associated with linear constraints. Notably, the posterior variance, a pivotal function used extensively in Bayesian optimization's acquisition functions, also aligns with the concept of a Schur complement.

When exploring Bayesian optimization, particularly in the context of Gaussian processes, the posterior variance plays a central role. By incorporating observed data, Gaussian processes provide estimates of the objective function's behavior along with associated uncertainty. Mathematically, the posterior variance emerges as a Schur complement within the Gaussian process framework. This connection underscores the utility of the Schur complement beyond its traditional applications in optimization theory.

Understanding the Schur complement is crucial for analyzing the structure of optimization problems, particularly when dealing with large-scale systems where direct inversion of the entire matrix may be computationally prohibitive. Instead, leveraging the structure of the Schur complement can facilitate the development of more efficient algorithms and computational strategies, both within Bayesian optimization and across diverse optimization domains.

4 Literature Review

Nearly twenty papers were considered throughout the semester's literature review process. Many were used to develop a baseline understanding of BO and MFBO. To summarize results from these papers, two of the most influential are explained below. These papers will drive research for the future, yet other papers considered are referenced at the bottom of this writeup.

4.1 "Non-Myopic Multifidelity Bayesian optimization"

One of the most significant motivating papers discovered is one on "Non-Myopic Multifidelity Bayesian optimization" by Francesco Di Fiore and Laura Mainini.

The paper introduces a non-myopic multifidelity Bayesian optimization (MFBO) framework that enhances optimization by utilizing a two-step lookahead multifidelity acquisition function. This approach aims to improve long-term rewards from future optimization steps, reducing computational costs. The non-myopic MFBO outperforms standard MFBO methods across various benchmark problems by efficiently exploring the input space and exploiting information for closer approximations to the optimum. This strategy showcases potential for real-world applications where balancing exploration and exploitation is critical for optimization under computational constraints.

The paper presents a comprehensive study on a novel non-myopic multifidelity Bayesian optimization (MFBO) framework designed to efficiently solve optimization problems by leveraging information across different fidelities. The core contribution is the development of a two-step lookahead strategy that anticipates the impact of current decisions on future optimization steps, aiming to minimize overall computational costs while maximizing the accuracy of the optimization process.

Mathematically, the framework is built upon the Gaussian Process (GP) model to represent the unknown objective function across fidelities. The GP model is used to predict the objective function's value at unobserved points, with the uncertainty in these predictions quantified by the GP's variance. The two-step lookahead strategy is formulated through an acquisition function that guides the selection of the next point to evaluate, considering both the expected improvement in the objective function and the expected reduction in uncertainty across the search space.

The authors introduce a novel acquisition function, which is a key innovation of their method. This function is designed to balance exploration and exploitation by weighing the potential improvement in objective function value against the cost of evaluation at different fidelities. The acquisition function is optimized to determine not only the next sampling point but also the fidelity at which to evaluate it, enabling efficient allocation of computational resources. Extensive experimental results demonstrate the superiority of the proposed non-myopic MFBO approach over existing methods. The authors show that their method achieves higher optimization accuracy with fewer high-fidelity evaluations across a range of synthetic and real-world benchmark problems. This efficiency is particularly evident in scenarios where high-fidelity evaluations are significantly more expensive than low-fidelity ones, highlighting the practical benefits of the proposed approach in reducing computational costs.

Given the understanding of a two-step look-ahead with plain, single-objective Bayesian optimization, this paper provides a unique application of this technique to solving a more involved problem. This paper serves as a key artifact and motivating algorithm for this continued study of multifidelity BO (Di Fiore et al.).

4.2 "Remarks on Multi-Output Gaussian Process Regression"

The article "Remarks on multi-output Gaussian process regression" explores the application of multi-output Gaussian processes (MOGPs) in addressing multi-output regression problems, which are prevalent in modern engineering. It categorizes MOGPs into symmetric and asymmetric types, the former improving predictions for all outputs simultaneously, and the latter focusing on enhancing high-fidelity outputs by leveraging related low-fidelity outputs. The characteristics of existing MOGPs are analyzed, including covariance functions, modeling processes, information transfer, and hyperparameter inference. Performance evaluations of ten representative MOGPs on various scenarios highlight their effectiveness and provide insights for their application. Recommendations are made for using MOGPs and identifying potential research directions, such as addressing issues like positive and negative transfer, handling big data, non-stationarity, and optimizing MOGP-assisted tasks like optimization and uncertainty quantification.

The paper on multi-output Gaussian process regression offers valuable insights for studies in multi-fidelity Bayesian optimization. Firstly, it provides a thorough examination of asymmetric MOGPs, particularly those relevant to multi-fidelity modeling. This is particularly useful as multi-fidelity Bayesian optimization often involves dealing with multiple levels of fidelity in simulators, where the lower fidelity models provide quick but less accurate predictions, and the higher fidelity models offer more accurate but computationally expensive predictions. Understanding how asymmetric MOGPs transfer knowledge from low-fidelity outputs to improve predictions of high-fidelity outputs can inform the development of efficient multi-fidelity Bayesian optimization algorithms.

Secondly, the paper discusses various evaluation metrics and scenarios for assessing the performance of MOGPs, including different training data, training sizes, output correlations, and output sizes. Such considerations are essential in designing experiments to evaluate the efficacy of multi-fidelity Bayesian optimization algorithms across diverse problem domains and data settings.

Moreover, the paper highlights potential research directions for enhancing MOGPs, such as addressing issues of positive and negative transfer across outputs and developing sampling strategies for sequential updating of MOGPs in asymmetric scenarios. These directions are relevant to advancing the capabilities of multi-fidelity Bayesian optimization methods, especially in scenarios where transferring information between fidelity levels is crucial for efficient optimization.

This paper can guide the development and improvement of multi-fidelity Bayesian optimization algorithms by offering a comprehensive understanding of modeling strategies, evaluation methodologies, and avenues for further research in the context of multi-output regression problems (Liu et al.).

5 Explanation of Code

In this section, we delve into a comprehensive explanation of the codebase, dissecting its various components, functionalities, and design principles. This code was developed by Darian and Ethan. It is written in Julia. This section will focus on the parts Ethan contributed to. The notebook `toy_problem_darian.ipynb` was written solely by Darian so it will not be covered here. From the initialization of the toy problem to the implementation of surrogate models and acquisition functions, each aspect of the code is meticulously examined to provide a deeper understanding of its inner workings. By elucidating the rationale behind specific design choices, encapsulation strategies, and architectural decisions, this section aims to demystify the intricacies of the Bayesian optimization notebook. This exposition serves as a valuable resource for navigating the details of modern optimization algorithms.

The Github repository we collaborated on throughout this semester can be found at this footnote[1].

5.1 toy_problem.ipynb

The code in this file demonstrates a Bayesian optimization process applied to a toy problem represented by the function `get_toy_problem`. Bayesian optimization is a sequential model-based optimization technique used for optimizing expensive-to-evaluate functions. Thus, `get_toy_problem` randomly selects one of these expensive-to-evaluate functions. Throughout writing this code, I discovered Gramacy Lee was an easy one of these functions to use (in terms of computation time and iterations), so by default I have chosen to work with this. This code outlines the 7 steps of BO. Before the first step of gathering initial samples, it is necessary to import any useful packages. These include LinearAlgebra, Distributions, and Optim.

The first part of the code initializes the toy problem and prints its name, along with plotting the function for visualization. The next step involves defining a Gaussian process (GP) surrogate model using the Squared Exponential kernel, which serves as a probabilistic surrogate for the true objective function. The user of the notebook can also change which kernel will by used by changing this line. The kernels available are listed in

[1] https://github.com/DarianNwankwo/Multi-Fidelity-Bayesian-Optimization

`kernels.jl`. The surrogate model is then updated iteratively using the Schur complement update method with the observed data points (X_1, y_1), where X_1 is the input and y_1 is the corresponding output obtained by optimizing the acquisition function. Using the Schur complement allows for more numerically stable and efficient updates as many compuation must be done with each update.

The code iterates over a predefined budget, optimizing different acquisition functions such as Expected Improvement (EI), Probability of Improvement (POI), Lower Confidence Bound (LCB), and Upper Confidence Bound (UCB) in each iteration. These acquisition functions guide the exploration-exploitation trade-off by selecting the next input point `Xopt` to evaluate the true objective function. The acquisition functions are created in step 3. In step 4, a function is declared to actually optimize the acquisition function at hand and provide the optimal x location. In steps 5 and 6, the code repeats this process of continuously sampling at new locations until the budget is exhausted. Finally, the code prints the recommended input (`Xopt`) and its corresponding output (`testfn.f(Xopt)`), representing the optimal solution found by the Bayesian optimization process. The results are visualized by plotting the final recommendations on the same plot as the original toy problem function. Overall, this code demonstrates the application of Bayesian optimization for optimizing complex objective functions efficiently.

5.2 testfns.jl

Test functions play a crucial role in the field of optimization, providing standardized benchmarks for evaluating optimization algorithms. The code `testfns.jl` embodies good software design principles to facilitate the creation and manipulation of such test functions. It adopts a modular approach, encapsulating each test function within its own function definition. This modularity promotes code reusability and maintainability by allowing users to easily extend or modify individual test functions without affecting other parts of the codebase. Additionally, test functions are represented as structs with associated functions, providing a clear and intuitive interface for users to interact with. This abstraction enhances code readability and makes it easier for users to understand and utilize the provided test functions.

Test functions were discovered in the "Virtual Library of Simulation Experiments"[2]. We chose over twenty of these functions. Some of the notable ones are the Ackley functions, the McCormick function, the Brainhoo function, and the Gramacy & Lee function.

Moreover, the code `testfns.jl` demonstrates flexibility in its design, offering customizable parameters for creating test functions tailored to specific optimization scenarios. Operations such as addition and multiplication are defined as methods on the `TestFunction` struct, enabling users to combine multiple test functions to create more complex scenarios. Furthermore, the inclusion of detailed comments throughout the codebase enhances its documentation, making it easier for users to understand the purpose of each function and its parameters. This design of `testfns.jl` adheres to best practices in software engineering, providing a robust and extensible framework for working with test functions in optimization problems.

5.3 kernels.jl

This document provides an explanation of the Julia code for kernel functions and utility functions. The code implements several kernel functions commonly used in Gaussian Processes (GPs) modeling:

- **Squared Exponential Kernel (SE):** Models smooth functions and is characterized by lengthscale parameters.
- **Periodic Kernel:** Represents functions with periodic behavior and takes lengthscale and period parameters.
- **Exponential Kernel:** Models functions that decay exponentially with distance, controlled by a lengthscale parameter.
- **Gamma Exponential Kernel:** Similar to the exponential kernel, with an additional shape parameter.

[2] https://www.sfu.ca/ ssurjano/optimization.html

- **Rational Quadratic Kernel:** A mixture of SE kernels with different lengthscale parameters.
- **Matern Kernels (1/2, 3/2, 5/2):** A family of kernels characterized by lengthscale and smoothness parameters.
- **White Noise Kernel:** Represents uncorrelated noise with a noise level parameter.

These kernels were found in the "Kernel Cookbook" from the University of Toronto[3].

The code also provides utility functions for working with kernel objects:

- **Kernel Construction:** Generic constructors for creating kernel objects with specified hyperparameters.
- **Inorder Traversal and Expression Tree:** Constructs expression trees to represent kernel combinations using addition and multiplication.
- **Gram Matrix and Covariance Matrix:** Functions for computing the Gram matrix and covariance matrix given a kernel and input data matrices.
- **Kernel Vector:** Computes the vector of kernel values between a single input vector and a matrix of input vectors.

The code is designed with modularity, abstraction, and flexibility in mind:

- **Modularity:** Each kernel function is implemented as a separate function, allowing easy addition or modification of kernels.
- **Abstraction:** Generic constructors and utility functions abstract away kernel construction details, allowing users to focus on specifying hyperparameters and combining kernels.
- **Flexibility:** The code enables the construction of complex kernel functions by combining simpler ones using expression trees, providing flexibility in modeling different types of functions with GPs.

Note: It's worth mentioning that kernels used in Gaussian Processes are symmetric and positive definite, which is necessary for proper functioning within the framework of Gaussian Processes.

5.4 acquisitions.jl

The file comprises several components, including abstract types, struct definitions, implementations of acquisition functions, utility functions, and a plotting function.

1. **Abstract Types**:
   - `AbstractAcquisitionFunction`: Abstract type defining the common interface for acquisition functions.
   - `DifferentiableAcquisitionFunction <: AbstractAcquisitionFunction`: Abstract type for differentiable acquisition functions.
   - `AcquisitionFunctionOnly <: AbstractAcquisitionFunction`: Abstract type for acquisition functions that may not be differentiable.
2. **Struct**:
   - `AcquisitionFunctionT <: Function, V <: AbstractVector`: Struct representing an acquisition function, parameterized by a function f, lower bounds *lowerbounds*, and upper bounds *upperbounds*.
3. **Acquisition Functions**:
   - `UCB`: Implements the Upper Confidence Bound acquisition function.
   - `LCB`: Implements the Lower Confidence Bound acquisition function.
   - `POI`: Implements the Probability of Improvement acquisition function.
   - `EI`: Implements the Expected Improvement acquisition function.
4. **Utility Functions**:
   - `optimize_acquisition`: Optimizes the acquisition function over the search space.
   - `get_acquisition_functions`: Returns a dictionary of predefined acquisition functions.
   - `get_random_acquisitionfn`: Randomly selects an acquisition function from the predefined set.
5. **Plotting Function**:

[3] https://www.cs.toronto.edu/ duvenaud/cookbook/

– `plotaf1d`: Plots the 1D acquisition function over a specified interval.

The code is designed to be modular, promoting code reuse and maintainability. Each acquisition function is encapsulated within its own function, and abstract types allow for flexibility in defining new acquisition functions. From a mathematical perspective, each acquisition function is derived from Bayesian optimization principles, utilizing Gaussian process models for surrogate modeling. These functions leverage statistical properties such as mean and standard deviation predictions to guide the search toward promising regions of the objective function space.

5.5 surrogates.jl

Gaussian Process Definition

A Gaussian Process (GP) is represented by a mean function $\mu(x)$ and a covariance function $k(x, x')$. Given a set of input points X and their corresponding output values y, the GP defines a distribution over functions $f(X)$. The mean and covariance functions are typically defined as:

$$\mu(x) = E[f(x)], k(x, x') = E[(f(x) - \mu(x))(f(x') - \mu(x'))].$$

Gaussian Process Regression

Given a set of observed data points $X = \{x_1, x_2, \ldots, x_N\}$ and their corresponding output values $y = \{y_1, y_2, \ldots, y_N\}$, the goal of GP regression is to predict the output value y^* at a new input point x^*. This is done by computing the predictive distribution $p(y^*|x^*, X, y)$, which is Gaussian with mean μ^* and variance σ^2:

$$\mu^* = k(x^*, X)(K + \sigma_n^2 I)^{-1} y,$$
$$\sigma^2 = k(x^*, x^*) - k(x^*, X)(K + \sigma_n^2 I)^{-1} k(x^*, X)^T,$$

where $k(x^*, X)$ is a vector of covariance values between the test point x^* and the training points, K is the Gram matrix of covariances between all training points, and σ_n^2 is the noise variance.

Kernel Functions

The covariance function $k(x, x')$ is often defined in terms of a kernel function. Commonly used kernel functions include the Radial Basis Function (RBF) kernel, linear kernel, polynomial kernel, etc. These kernels capture the similarity or correlation between input points.

Cholesky Decomposition

The code utilizes the Cholesky decomposition to efficiently compute the inverse of the covariance matrix $(K + \sigma_n^2 I)^{-1}$. This is done by decomposing the covariance matrix as LL^T, where L is a lower triangular matrix. The Cholesky decomposition helps in solving linear systems and computing determinants efficiently.

Gaussian Process Update

The code implements two methods for updating the Gaussian process with new data points:

– **Naive Update**: Directly updates the covariance and Cholesky matrices by recomputing them from scratch.
– **Schur Update**: Utilizes the Schur complement to efficiently update the covariance matrix. This method avoids re-computing the entire covariance matrix and is more computationally efficient, especially for large datasets.

6 Multi-Fidelity Bayesian optimization

Multi-Fidelity Bayesian optimization (MFBO) has emerged as a powerful tool in optimizing complex systems with varying levels of fidelity. Each level of fidelity has a varying certainty of measure. With this comes a higher evaluation cost. For example, the highest fidelity regions are incredibly expensive to measure but give the true underlying value of your search space. Lower fidelity regions are cheap to evaluate but their measurements may be offset by large error. MFBO is applicable in regimens where there is a finite evaluation budget and evaluation is constrained by cost. In this collaborative research proposal, we explore the applications of MFBO to the optimization of a stellarator, specifically addressing the challenge of managing alpha particle loss. This originates after a conversation with Michael Czekanski. The high-fidelity space corresponds to continuous operation iterations, and we propose the use of MFBO to determine optimal stop times based on information about alpha particle loss. Additionally, we discuss potential research areas such as leveraging cokriging techniques and other methodologies to enhance the efficiency and effectiveness of MFBO in this context.

It is also important to discuss a prevalent technique which comes up often in MFBO research. Cokriging, a technique originating from geostatistics, has found significant applications in multi-fidelity optimization. In the context of optimization, cokriging serves as a surrogate modeling approach that leverages information from multiple sources or levels of fidelity to efficiently approximate expensive-to-evaluate objective functions. By incorporating data from both high and low-fidelity models, cokriging enhances the accuracy of the surrogate model, leading to more effective optimization algorithms. This integration of multiple fidelity levels enables practitioners to balance the trade-off between accuracy and computational cost, ultimately improving the efficiency of the optimization process.

6.1 Application to Stellarator Optimization

Managing Alpha Particle Loss

To model and estimate the proportion of trajectories in a stellarator that lead to alpha particle loss, we consider a toy optimization problem defined by the standard map:

$$\begin{aligned} p_{\text{new}} &= \text{mod}(p + K\sin(q), 2\pi) \\ q_{\text{new}} &= \text{mod}(q + p_{\text{new}}, 2\pi) \end{aligned} \tag{1}$$

where q and p represent the position and momentum of the particles, and K is a parameter controlling the map behavior. In this context, a particle leaves the "device" if, at some point, $p > 5$.

We aim to estimate the proportion of trajectories that result in particle loss using the Monte Carlo method. The `mc_loss_estimate` function simulates the evolution of particles through the standard map, running a specified number of orbits for a given number of time steps.

```
# Toy optimization problem
function standard_map(q, p, K=0.6)
    pnew = mod(p + K*sin(q), 2*pi)
    qnew = mod(q + pnew, 2*pi)
    return qnew, pnew
end

# Parameters
K = 0.5
tmax = 1024
norb = 256

# Monte Carlo loss estimate
mc_loss_estimate(K, tmax, norb)
```

Here, the function `mc_loss_estimate` initializes particle positions and momenta, evolves them through the standard map, and estimates the proportion of particles leaving the device. This Monte Carlo estimate provides insights into the particle loss behavior for a given map defined by K. The goal is to minimize the total runtime, which is the sum of observation time steps for each particle. Minimizing this quantity will lead to runtime improvements in stellarator optimization.

Determining Optimal Stop Times

Using Multi-Fidelity Bayesian optimization (MFBO), we can determine optimal stop times for the stellarator optimization. By integrating information about alpha particle loss obtained from the toy optimization problem, MFBO enables us to make decisions on when to halt iterations to achieve efficient and optimal performance. However, it's essential to clarify that the discussion of optimal stop times refers to determining when to stop simulating a given trajectory, rather than real-time control decisions.

This is the broad setup of such a real-world application fo MFBO which can be a possible focus for future research. Our knowledge on this topic can still be more focused, but for now this is the information we are familiar with.

7 Research Directions

Here are possible applications of our research for next semester. We can focus on applying MFBO, cost-constrained BO, multi-objective BO, or vanilla BO. These subsets of BO can also be used as tools to solve other problems and be applied to various fields.

7.1 Further Investigations with Cokriging

Expanding on the use of cokriging in MFBO, we will explore its applicability to the stellarator optimization context. Cokriging, with its ability to handle correlated data, may offer improved accuracy in modeling the complex interactions leading to alpha particle loss.

7.2 Applications to Other Plasma Physics Phenomena

Beyond alpha particle loss, we will investigate the broader applications of MFBO to various plasma physics phenomena. This includes exploring different optimization objectives and constraints relevant to stellarator performance. In conclusion, our collaborative research aims to leverage MFBO and advanced simulation techniques to optimize stellarator operation, with a specific focus on managing alpha particle loss. Through the proposed research directions, we anticipate contributing to the advancement of plasma physics and optimization methodologies in fusion research. This research can be done in close collaboration with other members of David's research group who have more knowledge of the physics principles driving this research.

7.3 Applications to Finance

Bayesian optimization has gained significant traction in the field of finance due to its ability to tackle complex optimization problems efficiently while incorporating uncertainty quantification—a crucial aspect in financial decision-making. Below are some prominent applications of Bayesian optimization in finance:

Portfolio Optimization Portfolio optimization involves selecting a combination of assets to maximize returns while minimizing risk. Bayesian optimization offers a sophisticated approach to this problem by optimizing portfolio allocations based on historical data, market conditions, and investor preferences. By modeling the returns and volatility of different assets using Gaussian Processes, Bayesian optimization can find an optimal asset allocation strategy that balances risk and return objectives.

Algorithmic Trading In algorithmic trading, Bayesian optimization plays a vital role in developing and optimizing trading strategies. By treating trading strategy parameters as variables to be optimized, Bayesian optimization can search the space of possible strategies to maximize profitability while accounting for transaction costs, market impact, and risk constraints. This approach enables traders to adapt quickly to changing market conditions and improve the performance of their trading algorithms.

Risk Management Risk management is paramount in finance to mitigate potential losses and ensure the stability of financial institutions. Bayesian optimization aids in risk management by optimizing risk-adjusted returns, determining optimal hedging strategies, and stress testing portfolios under different market scenarios. By incorporating uncertainty quantification into risk models, Bayesian optimization provides more robust risk assessments and helps financial institutions make informed decisions in turbulent market environments.

Option Pricing and Hedging Option pricing and hedging are fundamental tasks in derivative markets. Bayesian optimization can be applied to calibrate option pricing models, such as Black-Scholes or stochastic volatility models, to market data effectively. Moreover, Bayesian optimization can optimize hedging strategies by minimizing the risk exposure of option positions while maximizing potential profits. This approach allows financial institutions to manage their options portfolios more efficiently and reduce the impact of market fluctuations.

Acknowledgement. David Bindel, Michael Czekanski

References

1. Abdolshah, Majid, et al. "Cost-aware Multi-objective Bayesian optimization." 2019. arXiv preprint arXiv:1909.03600.
2. Allmendinger, Richard, Julia Handl, and Joshua Knowles. "Multiobjective Optimization: When Objectives Exhibit Non-uniform Latencies." European Journal of Operational Research, vol. 243, no. 2, 2015, pp. 497-513, ISSN 0377-2217, https://doi.org/10.1016/j.ejor.2014.09.033.
3. Buathong, Poompol, et al. "Bayesian optimization of Function Networks with Partial Evaluations." 2023. arXiv preprint arXiv:2311.02146.
4. Coello, Carlos, David Veldhuizen, and Gary Lamont. "Evolutionary Algorithms for Solving Multi-Objective Problems Second Edition." Kluwer Academic, 2007.
5. Di Fiore, Francesco, and Mainini, Laura. "Non-Myopic Multifidelity Bayesian optimization." arXiv preprint arXiv:2207.06325 (2022).
6. Galuzio, Paulo Paneque, et al. "MOBOpt — Multi-objective Bayesian optimization." SoftwareX, vol. 12, 2020, article 100520, ISSN 2352-7110, https://doi.org/10.1016/j.softx.2020.100520.
7. Konakovic Lukovic, Mina, Yunsheng Tian, and Wojciech Matusik. "Diversity-Guided Multi-Objective Bayesian optimization With Batch Evaluations." Advances in Neural Information Processing Systems, edited by H. Larochelle et al., vol. 33, 2020, Curran Associates, Inc., pp. 17708-17720, https://proceedings.neurips.cc/paper_files/paper/2020/file/cd3109c63bf4323e6b987a5923becb96-Paper.pdf.
8. Lee, Eric Hans, et al. "A Nonmyopic Approach to Cost-Constrained Bayesian optimization." 2021. arXiv preprint arXiv:2106.06079.
9. Li, Shibo, et al. "Multi-Fidelity Bayesian optimization via Deep Neural Networks." 2020. arXiv preprint arXiv:2007.03117.
10. Liu, H., Cai, J., & Ong, Y.-S. (2018). "Remarks on Multi-Output Gaussian Process Regression." Knowledge-Based Systems, 144, 102-121. ISSN 0950-7051.
11. Poloczek, Matthias, Jialei Wang, and Peter I. Frazier. "Multi-Information Source Optimization." 2016. arXiv preprint arXiv:1603.00389.
12. Snoek, Jasper, Hugo Larochelle, and Ryan P. Adams. "Practical Bayesian optimization of Machine Learning Algorithms." 2012. arXiv preprint arXiv:1206.2944.
13. Wang, Xilu, et al. "Alleviating Search Bias in Bayesian Evolutionary Optimization with Many Heterogeneous Objectives." 2022. arXiv preprint arXiv:2208.12217.
14. Wang, Xilu, et al. "Recent Advances in Bayesian optimization." 2022. arXiv preprint arXiv:2206.03301.
15. Wang, Xilu, et al. "Transfer Learning Based Co-surrogate Assisted Evolutionary Bi-objective Optimization for Objectives with Non-uniform Evaluation Times." 2021. arXiv preprint arXiv:2108.13339.
16. Wu, Jian, and Peter Frazier. "Practical Two-Step Lookahead Bayesian optimization." Advances in Neural Information Processing Systems, edited by H. Wallach et al., vol. 32, 2019, Curran Associates, Inc., https://proceedings.neurips.cc/paper_files/paper/2019/file/2e6d9c6052e99fcdfa61d9b9da273ca2-Paper.pdf.
17. Zanjani Foumani, Zahra, et al. "Multi-fidelity Cost-aware Bayesian optimization." Computer Methods in Applied Mechanics and Engineering, vol. 407, Mar. 2023, article 115937. Elsevier BV, ISSN 0045-7825, doi:10.1016/j.cma.2023.115937.