Metadata of the chapter that will be visualized online

Chapter Title	Bayesian Decision Theory and Expected Improvement	
Copyright Year	2023	
Copyright Holder	Peng Liu	
Corresponding Author	Family Name	**Liu**
	Particle	
	Given Name	**Peng**
	Suffix	
	Division	
	Organization/University	
	Address	Singapore, Singapore
Abstract	The previous chapter used Gaussian processes (GP) as the surrogate model to approximate the underlying objective function. GP is a flexible framework that provides uncertainty estimates in the form of probability distributions over plausible functions across the entire domain. We could then resort to the closed-form posterior predictive distributions at proposed locations to obtain an educated guess on the potential observations.	

CHAPTER 3

Bayesian Decision Theory and Expected Improvement

The previous chapter used Gaussian processes (GP) as the surrogate model to approximate the underlying objective function. GP is a flexible framework that provides uncertainty estimates in the form of probability distributions over plausible functions across the entire domain. We could then resort to the closed-form posterior predictive distributions at proposed locations to obtain an educated guess on the potential observations.

However, it is not the only choice of surrogate model used in Bayesian optimization. Many other models, such as random forest, have seen increasing use in recent years, although the default and mainstream choice is still a GP. Nevertheless, the canonical Bayesian optimization framework allows any surrogate model as long as it provides a posterior estimate for the function, which then gets used by the acquisition function to generate a sampling proposal.

The acquisition function bears even more choices and is an increasingly crowded research space. Standard acquisition functions such as expected improvement and upper confidence bound have seen wide usage in many applications, and problem-specific acquisition functions incorporating domain knowledge, such as safe constraint, are constantly being proposed. The acquisition function assumes a more important role in the Bayesian optimization framework as it directly determines the sampling decision for follow-up data acquisition. A good acquisition function thus enables the optimizer to locate the (global) optimum as fast as possible, where the optimum is measured in the sense of the location that holds the optimum value or the optimum value across the whole domain.

© Peng Liu 2023
P. Liu, *Bayesian Optimization*, https://doi.org/10.1007/978-1-4842-9063-7_3

This chapter will dive into the Bayesian optimization pipeline using expected improvement, the most widely used acquisition function for sampling decisions. We will first characterize Bayesian optimization as a sequential decision process under uncertainty, followed by a thorough introduction of expected improvement. An intelligent selection of the following sampling location that involves uncertainty estimates is the key to achieving sample-efficient global optimization. Lastly, we will go through a case study using expected improvement to guide hyperparameter tuning.

Optimization via the Sequential Decision-Making

The Bayesian optimization framework sequentially determines the following sampling location in search of the global optimum, usually assumed to be maximum in a maximization setting. Based on a specific policy, the optimizer would collect observed data points, update the posterior belief about the probability distributions of the underlying functions, propose the next sampling point for probing, and finally collect the additional data point at the proposed location and repeat. This completes one iteration within the iterative and sequential process. Our knowledge of the underlying function constantly evolves and gets updated every time a new data point is incorporated under the guidance of the existing collection of observations.

At the end of the process, the policy will return either the location of the optimal value or the optimum itself, although we are often interested in the optimal location. This is often referred to as the *outer loop* in the Bayesian optimization framework. Besides, maximizing the acquisition function to generate the following sampling location constitutes the *inner loop* of Bayesian optimization. The acquisition function serves as a side computation within the inner loop to aid the subsequent sampling decision. Optimizing the acquisition function is usually considered fast and cheap due to its inexpensive evaluation and analytical differentiability. We can obtain the closed-form expression for some acquisition functions and access its gradient by using an off-the-shelf optimization procedure. We can also resort to approximation methods such as Monte Carlo estimation for more complex acquisition functions without closed-form expression.

In addition, the policy would also need to consider *when* to terminate the probing process, especially in the case of a limited probing budget. Upon termination, the optimizer would return the optimal functional value or location, which may or may not live in the observed locations and could exist anywhere within the domain.

The optimizer thus needs to trade off between calling off the query and performing additional sampling, which incurs an additional cost. Therefore, the action space of the optimizer contains not only the sampling location but also a binary decision on termination.

Figure 3-1 characterizes the sequential decision-making process that underpins Bayesian optimization. The policy would propose the following sampling location at each outer loop iteration or terminate the loop. Suppose it decides to propose an additional sampling action. In that case, we will enter the inner loop to seek the most promising location with the highest value of the prespecified acquisition function. We would then probe the most favorable location and append the additional observation in our data collection, which is then used to update the posterior belief on the underlying objective function through GP. On the other hand, if the policy believes the additional query is not worth the corresponding cost to improve our belief on the global optimum, it would decide to terminate the outer loop and return the current best estimate of the global optimum or its location. This also forms the *stopping rule* of the policy, which could be triggered upon exhausting the limited budget or assuming an adaptive mechanism based on the current progression.

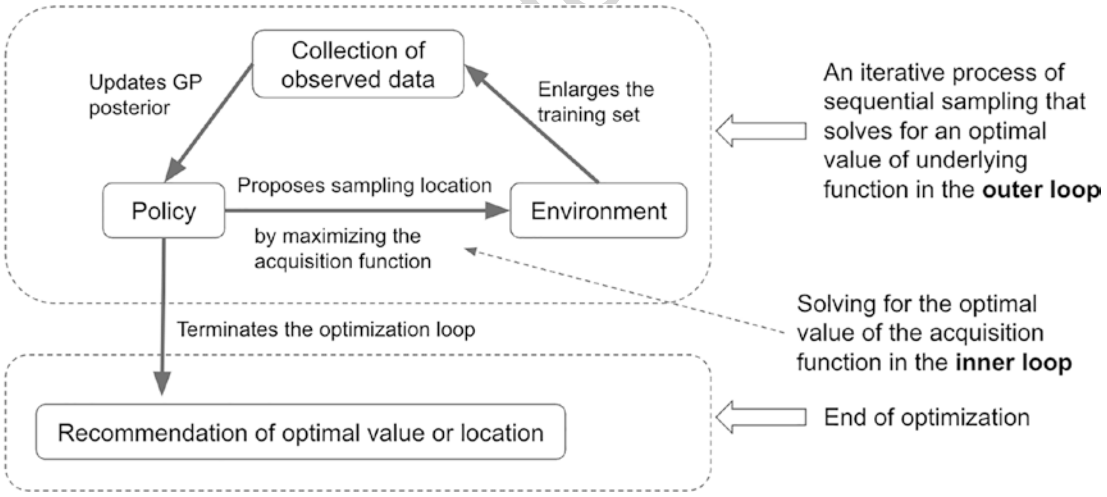
Collection of observed data
Updates GP posterior
Enlarges the training set
An iterative process of sequential sampling that solves for an optimal value of underlying function in the **outer loop**
Policy
Proposes sampling location
Environment
by maximizing the acquisition function
Solving for the optimal value of the acquisition function in the **inner loop**
Terminates the optimization loop
Recommendation of optimal value or location
End of optimization

Figure 3-1. *Sequential decision-making process in Bayesian optimization. The outer loop performs optimization in search of the global optimum by sequential sampling across the entire domain. Each iteration is based on the output of the inner loop, which involves another separate optimization*

Quantifying the improvement on the belief of the global optimum is reflected in the *expected marginal gain* on the *utility* of observed data, which is the core concept in the Bayesian decision theory used in Bayesian optimization. We will cover this topic in the following sections.

Seeking the Optimal Policy

The ultimate goal of BO is to develop an intelligent policy that performs sequential decision-making under uncertainty in a principled manner. When the policy is measured in terms of the quality of the collected data, BO would then seek the *optimal policy* that maximizes the expected quality of the collected data. In other words, the optimal policy would deliver the most informative dataset on average to assist the task of locating the global optimum while considering the posterior belief about the underlying objective function.

In this regard, the acquisition function is used to measure the data quality when considering the following sampling decision across the entire domain. The acquisition function maps each candidate location to a numeric score, which essentially encodes preferences over different candidate locations. It serves as the intermediate calculation that bridges the gap between updating the posterior belief and seeking the optimal policy. Specifically, the optimal decision based on the most updated posterior belief is made by choosing the location with the maximal score calculated using the specified acquisition function, which completes one round of inner optimization.

Mathematically, for each candidate location x in domain A, the self-defined acquisition function $\alpha(x)$ maps each x to a scalar value, that is, $\alpha : A \rightarrow \mathbb{R}$. Here, we assume x is single-dimensional without loss of generality and for the seek of notational convenience, although it can assume multiple features, that is, $\mathbf{x} \in \mathbb{R}^d$. Besides, the acquisition function is an evolving scoring function that also depends on the currently collected dataset $\mathcal{D}_n$ with n observations, thus writing $\alpha(x;\mathcal{D}_n)$ to indicate such dependence.

AU1

Therefore, for any arbitrary locations x_1 and x_2 within A, we would prefer x_1 over x_2 if $\alpha(x_1;\mathcal{D}_n) > \alpha(x_2;\mathcal{D}_n)$, and vice versa. Out of infinitely many candidate locations in the case of a continuous search domain, the optimal policy would then act greedily by selecting the single location x_{n+1}^* with the highest acquisition value as the following sampling action. We use the superscript $*$ to denote an optimal action and the subscript $n + 1$ to indicate the additional first future sampling location on top of the existing n observations. The addition of one thus means the lookahead horizon or time step into the future. The notation x_{n+1} denotes all possible candidate locations, including the observed ones, and is viewed as a random variable. The optimal decision is then defined as follows:

$$x_{n+1}^* = \text{argmax}_{x_{n+1} \in A} \alpha\left(x_{n+1}; \mathcal{D}_n\right)$$

Common acquisition functions such as expected improvement (to be introduced later) admit fast gradient-based optimization due to the availability of the closed-form analytic expression and the corresponding gradient. This means that we have converted the original quest for global optimization of a difficult and unknown objective function to a series of fast optimizations of a known acquisition function. However, as we will learn later, some acquisition functions, especially those featuring multi-step lookahead, may not be analytically differentiable, making the inner optimization a nontrivial problem. In such cases, the Monte Carlo approximation is often used to approximate the calculation.

We can now represent the high-level mathematical details of BO, ignoring the decision on termination for now. As shown in Figure 3-2, the whole BO (outer) loop consists of three major steps: proposing the following sampling location x_{n+1}^* as the maximizing location of the acquisition function $\alpha\left(x_{n+1}; \mathcal{D}_n\right)$ based on current dataset $\mathcal{D}_n$, probing the proposed location and appending the additional observation in the current dataset $\mathcal{D}_{n+1} = \mathcal{D}_n \cup \left\{\left(x_{n+1}^*, y_{n+1}^*\right)\right\}$, and finally updating the posterior belief assuming a GP surrogate model $p\left(f|\mathcal{D}_{n+1}\right)$. Here, y_{n+1}^* denotes the observation at the optimal next sampling location x_{n+1}^*. Choosing an appropriate acquisition function thus plays a crucial role in determining the quality of the sequential optimization process.

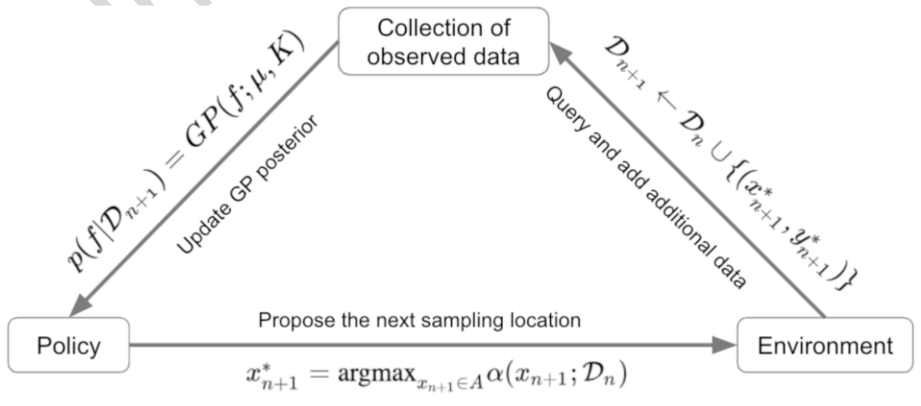
Collection of observed data
$p(f|\mathcal{D}_{n+1}) = GP(f; \mu, K)$
Update GP posterior
$\mathcal{D}_{n+1} \leftarrow \mathcal{D}_n \cup \{(x_{n+1}^*, y_{n+1}^*)\}$
Query and add additional data
Policy
Propose the next sampling location
$x_{n+1}^* = \text{argmax}_{x_{n+1} \in A} \alpha(x_{n+1}; \mathcal{D}_n)$
Environment

Figure 3-2. *Illustrating the entire BO loop by iteratively maximizing the current acquisition function, probing additional data, and updating posterior belief*

Although the BO loop could begin with an empty dataset, practical training often relies on a small dataset consisting of a few uniformly sampled observations. This accelerates the optimization process as it serves as a warm start and presents a more informed prior belief than a uniform one. The effect is even more evident when the initial dataset has good coverage of different locations of the domain.

Utility-Driven Optimization

The eventual goal of BO is to collect a valuable set of observations that are most informative about the global optimum. The value of a dataset is quantified by *utility*, a notion initially used in the Bayesian decision theory and used here to assist the sequential optimization in BO via the acquisition function. The acquisition function builds on top of the utility of the currently available dataset when assessing the value of candidate locations.

Since our goal is to locate the global maximum, a natural choice for the utility function is the maximum value of the current dataset, that is, $u(\mathcal{D}_n) = \max\{y_{1:n}\} = y_n^*$, assuming the case of noise-free observations. This is also called the incumbent of the current dataset and is used as a benchmark when evaluating all future candidate observations. As the most widely used acquisition function in practical applications, the expected improvement function uses this incumbent to award candidate locations whose putative observations are likely to be higher.

When assessing a candidate location x_{n+1}, we would require a fictional observation y to be able to calculate the utility *if* we were to acquire an additional observation at this location. Considering the randomness of the objective function, our best estimate is that y_{n+1} will follow a posterior normal distribution according to the updated GP posterior. Since y_{n+1} is a random variable, the standard approach is to integrate out its randomness by calculating the *expected utility* at the particular location, that is, $\mathbb{E}_{y_{n+1}}\left[u(x_{n+1}, y_{n+1}, \mathcal{D}_n) | x_{n+1}, \mathcal{D}_n\right]$, conditioned on the specific evaluation location x_{n+1} and current set of observations $\mathcal{D}_n$. This also corresponds to the expected utility when assuming we have an additional unknown observation (x_{n+1}, y_{n+1}), leading to $\mathbb{E}_{y_{n+1}}\left[u(\mathcal{D}_{n+1}) | x_{n+1}, \mathcal{D}_n\right] = \mathbb{E}_{y_{n+1}}\left[u(\mathcal{D}_n \cup (x_{n+1}, y_{n+1})) | x_{n+1}, \mathcal{D}_n\right] = \mathbb{E}_{y_{n+1}}\left[u(x_{n+1}, y_{n+1}, \mathcal{D}_n) | x_{n+1}, \mathcal{D}_n\right]$. We could then utilize the posterior predictive distribution $p(y_{n+1} | \mathcal{D}_n)$ to express the expected utility as an integration operation in the continuous case as follows:

$$\mathbb{E}_{y_{n+1}}\left[u(x_{n+1}, y_{n+1}, \mathcal{D}_n) | x_{n+1}, \mathcal{D}_n\right] = \int u(x_{n+1}, y_{n+1}, \mathcal{D}_n) p(y_{n+1} | x_{n+1}, \mathcal{D}_n) dy_{n+1}$$

This expression considers all possible values of y_{n+1} at location x_{n+1}. It weighs the corresponding utility based on the probability of occurrence. With access to the expected utility at each candidate location, the next following location could be determined by selecting the one with the largest expected utility:

$$x_{n+1}^{*} = \text{argmax}_{x_{n+1} \in A} \mathbb{E}_{y_{n+1}} \left[u\left(x_{n+1}, y_{n+1}, \mathcal{D}_n\right) | x_{n+1}, \mathcal{D}_n \right]$$

Therefore, we need to have an appropriately designed utility function when determining the next optimal location by maximizing the expected utility. Equivalently, each action taken by the policy is selected to maximize the improvement in the expected utility. This process continues until the stopping rule is triggered, at which point the quality of the final returned dataset $\mathcal{D}_N$ is evaluated using $u\left(\mathcal{D}_N\right)$.

Since we are concerned with the optimal one-step lookahead action, the preceding problem can be formulated as maximizing the *expected marginal gain* in the utility, which serves as the acquisition function to guide the search. The one-step lookahead policy using the expected marginal gain is thus defined as follows:

$$\alpha_1\left(x_{n+1}; \mathcal{D}_n\right) = \mathbb{E}_{y_{n+1}} \left[u\left(\mathcal{D}_{n+1}\right) | x_{n+1}, \mathcal{D}_n \right] - \mathbb{E}\left[u\left(\mathcal{D}_n\right) \right]$$

$$= \mathbb{E}_{y_{n+1}} \left[u\left(x_{n+1}, y_{n+1}, \mathcal{D}_n\right) | x_{n+1}, \mathcal{D}_n \right] - u\left(\mathcal{D}_n\right)$$

where the subscript 1 in $\alpha_1\left(x_{n+1}; \mathcal{D}_n\right)$ denotes the number of lookahead steps into the future. The second step follows since there is no randomness in the utility of the existing observations $u\left(\mathcal{D}_n\right)$.

The optimal action using the one-step lookahead policy is then defined as the maximizer of the expected marginal gain:

$$x_{n+1}^{*} = \text{argmax}_{x_{n+1} \in A} \alpha_1\left(x_{n+1}; \mathcal{D}_n\right)$$

Figure 3-3 illustrates this process. We start with the utility of collected observations $u\left(\mathcal{D}_n\right)$ as the benchmark for comparison when evaluating the expected marginal gain at a new candidate location. The evaluation needs to consider all possible values of the next observation based on updated posterior GP and thus leads to the expected utility term $\mathbb{E}_{y_{n+1}} \left[u\left(x_{n+1}, y_{n+1}, \mathcal{D}_n\right) | x_{n+1}, \mathcal{D}_n \right]$. Since we are considering one step ahead in the future, the acquisition function $\alpha_1\left(x_{n+1}; \mathcal{D}_n\right)$ becomes one-step lookahead policy, and our goal is to select the location that maximizes the expected marginal gain in the utility of the collected dataset.

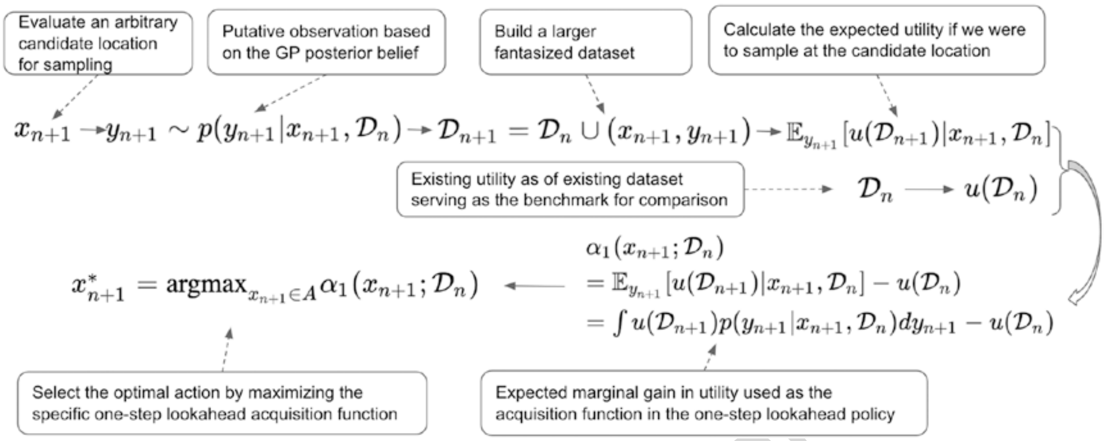
Evaluate an arbitrary candidate location for sampling
Putative observation based on the GP posterior belief
Build a larger fantasized dataset
Calculate the expected utility if we were to sample at the candidate location
$x_{n+1} \to y_{n+1} \sim p(y_{n+1}|x_{n+1}, \mathcal{D}_n) \to \mathcal{D}_{n+1} = \mathcal{D}_n \cup (x_{n+1}, y_{n+1}) \to \mathbb{E}_{y_{n+1}}[u(\mathcal{D}_{n+1})|x_{n+1}, \mathcal{D}_n]$
Existing utility as of existing dataset serving as the benchmark for comparison
$\mathcal{D}_n \longrightarrow u(\mathcal{D}_n)$
$x^*_{n+1} = \text{argmax}_{x_{n+1}\in A}\alpha_1(x_{n+1}; \mathcal{D}_n)$
$\alpha_1(x_{n+1}; \mathcal{D}_n) = \mathbb{E}_{y_{n+1}}[u(\mathcal{D}_{n+1})|x_{n+1}, \mathcal{D}_n] - u(\mathcal{D}_n) = \int u(\mathcal{D}_{n+1})p(y_{n+1}|x_{n+1}, \mathcal{D}_n)dy_{n+1} - u(\mathcal{D}_n)$
Select the optimal action by maximizing the specific one-step lookahead acquisition function
Expected marginal gain in utility used as the acquisition function in the one-step lookahead policy

Figure 3-3. *Deriving the one-step lookahead policy by maximizing the expected marginal gain in the utility of the acquired observations*

Multi-step Lookahead Policy

The sequential decision-making process using a one-step lookahead policy is a powerful and widely applied technique. By simulating possible future paths if we were to collect another observation, the policy becomes Bayes optimal due to maximizing the one-step expected marginal gain of the utility in the enlarged artificial dataset. However, the optimization process will continue until reaching a terminal point when the search budget is exhausted. The choice of the following sampling location, $x^*_{n+1} = \text{argmax}_{x_{n+1}\in A}\alpha_1\left(x_{n+1}; \mathcal{D}_n\right)$, thus impacts all remaining optimization decisions. That is, we need to consider all the future sampling steps until the stopping rule triggers, instead of only one step into the future.

To put things into context, let us assume that the *lookahead horizon*, that is, the number of steps to consider in the future, is τ. In other words, we would like to consider a putative dataset $\mathcal{D}_{n+\tau}$, which has additional τ artificial observations added to the existing dataset $\mathcal{D}_n$. Each observation involves selecting a candidate search location x and acquiring the corresponding observation value y, modeled as a random variable with updated posterior distribution based on previous observations (including both existing and putative ones). By expressing each addition of location and observation as a pair (x, y), the τ-step lookahead dataset $\mathcal{D}_{n+\tau}$ could be written as

$$\mathcal{D}_{n+\tau} = \mathcal{D}_n \cup \{(x_{n+1}, y_{n+1})\} \cup \{(x_{n+2}, y_{n+2})\} \cup \ldots \cup \{(x_{n+\tau}, y_{n+\tau})\}$$

Following the same mechanics as before, the multi-step lookahead policy would make the optimal sampling decision on x_{n+1}^* by maximizing the expected long-term terminal utility $\mathbb{E}\left[u(\mathcal{D}_{n+\tau})\right]$:

$$x_{n+1}^* = \text{argmax}_{x \in A} \mathbb{E}\left[u(\mathcal{D}_{n+\tau}) | x_{n+1}, \mathcal{D}_n\right]$$

where the expectation is taken with respect to randomness in future locations and observations. Equivalently, we can rely on the terminal expected marginal gain in the utility defined as follows:

$$\alpha_\tau(x_{n+1}; \mathcal{D}_n) = \mathbb{E}\left[u(\mathcal{D}_{n+\tau}) | x_{n+1}, \mathcal{D}_n\right] - u(\mathcal{D}_n)$$

which serves as the multi-step lookahead acquisition function to support the optimal sequential optimization:

$$x_{n+1}^* = \text{argmax}_{x_{n+1} \in A} \alpha_\tau(x_{n+1}; \mathcal{D}_n)$$

where the definition is only shifted downward by a constant value $u(\mathcal{D}_n)$ compared with maximizing the expected terminal utility $\mathbb{E}\left[u(\mathcal{D}_{n+\tau}) | x, \mathcal{D}_n\right]$ alone.

Now, if we expand the expectation in the definition of $\alpha_\tau(x_{n+1}; \mathcal{D}_n)$, we would need to consider all possible evolutions of future τ-step decisions on the locations $\{x_{n+i}, i = 2, \ldots, \tau\}$ and the associated realizations of the random variables $\{y_{n+i}, i = 1, \ldots, \tau\}$. Here, decisions on the locations $\{x_{n+i}, i = 2, \ldots, \tau\}$ start with $i = 2$ due to the fact that we are evaluating at location x_{n+1}. We can write the expanded form of the terminal expected marginal gain in utility as follows:

$$\alpha_\tau(x_{n+1}; \mathcal{D}_n) = \int \cdots \int u(\mathcal{D}_{n+\tau}) p(y_{n+1} | x_{n+1}, \mathcal{D}_n) \prod_{i=2}^{\tau} p(x_{n+i}, y_{n+i} | \mathcal{D}_{n+i-1}) dy_{n+1} d\{(x_{n+i}, y_{n+i})\} - u(\mathcal{D}_n)$$

where we explicitly write the posterior probability distribution of y_{n+1} as $p(y_{n+1} | x_{n+1}, \mathcal{D}_n)$ and the following joint probability distributions of $\{(x_{n+i}, y_{n+i}), i = 2, \ldots, \tau\}$ as $\prod_{i=2}^{\tau} p(x_{n+i}, y_{n+i} | \mathcal{D}_{n+i-1})$. Integrating out these random variables would give us the eventual multi-step lookahead marginal gain in the expected utility of the returned dataset.

Figure 3-4 summarizes the process of deriving the multi-step lookahead acquisition function. Note that the simulation of the next round of candidate locations and observations in $\{(x_{n+i}, y_{n+i}), i = 2, \ldots, \tau\}$ depends on all previously accumulated dataset $\mathcal{D}_{n+i-1}$, which is used to construct the updated posterior belief based on both observed and putative values.

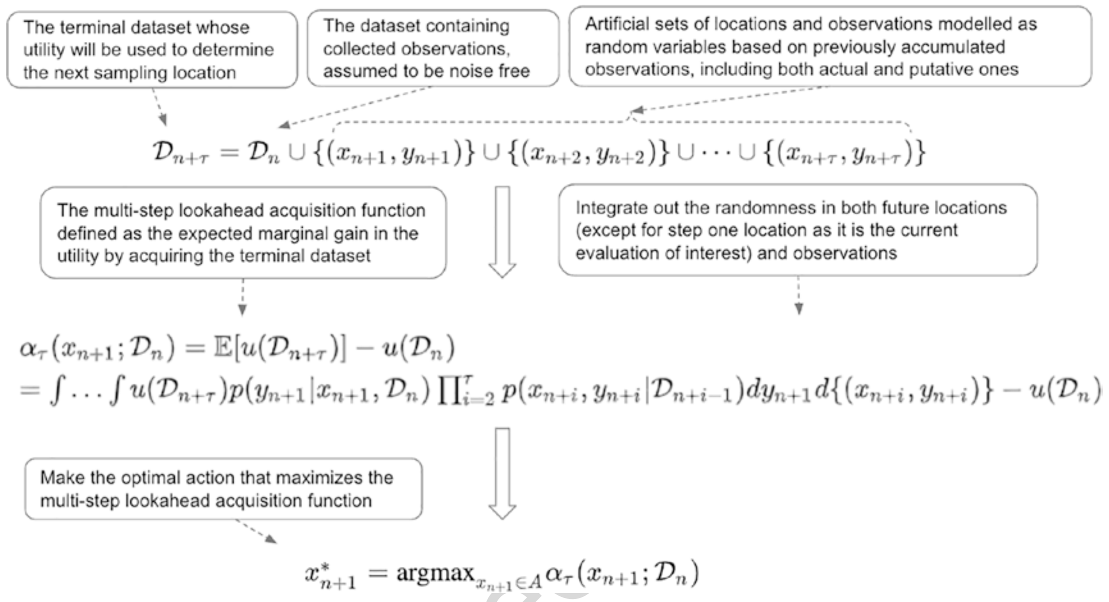
The terminal dataset whose utility will be used to determine the next sampling location
The dataset containing collected observations, assumed to be noise free
Artificial sets of locations and observations modelled as random variables based on previously accumulated observations, including both actual and putative ones
$\mathcal{D}_{n+\tau} = \mathcal{D}_n \cup \{(x_{n+1}, y_{n+1})\} \cup \{(x_{n+2}, y_{n+2})\} \cup \cdots \cup \{(x_{n+\tau}, y_{n+\tau})\}$
The multi-step lookahead acquisition function defined as the expected marginal gain in the utility by acquiring the terminal dataset
Integrate out the randomness in both future locations (except for step one location as it is the current evaluation of interest) and observations
$\alpha_\tau(x_{n+1}; \mathcal{D}_n) = \mathbb{E}[u(\mathcal{D}_{n+\tau})] - u(\mathcal{D}_n)$
$= \int \ldots \int u(\mathcal{D}_{n+\tau}) p(y_{n+1} | x_{n+1}, \mathcal{D}_n) \prod_{i=2}^{\tau} p(x_{n+i}, y_{n+i} | \mathcal{D}_{n+i-1}) dy_{n+1} d\{(x_{n+i}, y_{n+i})\} - u(\mathcal{D}_n)$
Make the optimal action that maximizes the multi-step lookahead acquisition function
$x_{n+1}^* = \text{argmax}_{x_{n+1} \in A} \alpha_\tau(x_{n+1}; \mathcal{D}_n)$

Figure 3-4. *The multi-step lookahead optimal policy that selects the best sampling location by maximizing the marginal expected utility of the terminal dataset*

We can glean more insight on the process of calculating this expression by drawing out the sequence of nested expectation and maximization operations. As shown in Figure 3-5, we start with the next sampling location x_{n+1} in a maximization operator, followed by y_{n+1} in an expectation operator. The same pattern continues at later stages, with a maximization operator in x_{n+2}, an expectation operator in y_{n+2}, and so on, until reaching the putative observation $y_{n+\tau}$ at the last stage. Each operator, be it maximization or expectation, involves multiple branches. Common strategy is to solve the maximization operation via a standard procedure such as L-BFGS and approximate the expectation operation via Gaussian quadrature.

$$x_{n+1}^{*} = \text{argmax}_{x_{n+1} \in \mathcal{X}} \alpha_\tau(x_{n+1}; \mathcal{D}_n)$$
$$= \int \dots \int u(\mathcal{D}_{n+\tau}) p(y_{n+1} | x_{n+1}, \mathcal{D}_n) \prod_{i=2}^{\tau} p(x_{n+i}, y_{n+i} | \mathcal{D}_{n+i-1})\, dy_{n+1} d\{(x_{n+i}, y_{n+i})\} - u(\mathcal{D}_n)$$

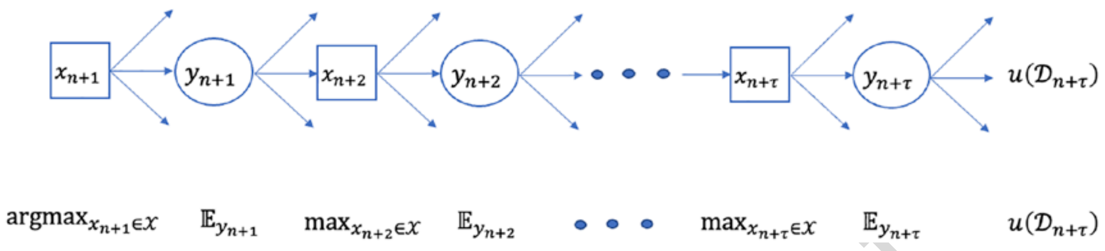
x_{n+1}
y_{n+1}
x_{n+2}
y_{n+2}
$x_{n+\tau}$
$y_{n+\tau}$
$u(\mathcal{D}_{n+\tau})$
$\text{argmax}_{x_{n+1} \in \mathcal{X}}$
$\mathbb{E}_{y_{n+1}}$
$\max_{x_{n+2} \in \mathcal{X}}$
$\mathbb{E}_{y_{n+2}}$
$\max_{x_{n+\tau} \in \mathcal{X}}$
$\mathbb{E}_{y_{n+\tau}}$
$u(\mathcal{D}_{n+\tau})$

Figure 3-5. *Visualizing the nested maximization and expectation operators*

Apparently, calculating a nested form of expectations that accounts for all possible future paths is computationally challenging. In addition, since our goal is to select an optimal sampling action by maximizing the acquisition function, we will add a reasonable assumption that all future actions will also be optimal given the current dataset, which may include putative realizations of the random variable on the objective value. Adding the optimality condition means that rather than considering all possible future paths of $\{(x_{n+i}, y_{n+i}), i = 1, \dots, \tau\}$, we will only focus on the optimal one $\left\{\left(x_{n+i}^{*}, y_{n+i}\right), i = 1, \dots, \tau\right\}$, which essentially removes the dependence on the candidate locations by choosing the maximizing location. The argument for selecting the optimal action by maximizing the long-term expected gain in utility follows the *Bellman principle of optimality*, as described in the next section.

Bellman's Principle of Optimality

Bellman's principle of optimality states that a sequence of optimal decisions starts with making the first optimal decision, followed by a series of optimal decisions conditioned on the previous outcome. This is a recursive expression in that, in order to make an optimal action at the current time point, we will need to act optimally in the future.

Let us build from the multi-step lookahead acquisition function from earlier. Recall that the τ-step expected marginal gain in utility at a candidate location x_{n+1} is defined as

$$\alpha_\tau\left(x_{n+1}; \mathcal{D}_n\right) = \mathbb{E}\left[u\left(\mathcal{D}_{n+\tau}\right) | x_{n+1}, \mathcal{D}_n\right] - u\left(\mathcal{D}_n\right)$$

which is the subject we seek to maximize. To explicitly connect with the one-step lookahead acquisition function and the remaining $\tau - 1$ steps of simulations into the future, we can introduce the one-step utility $u(\mathcal{D}_{n+1})$ by adding and subtracting this term in the expectation, as shown in the following:

$$\begin{aligned}
&\alpha_\tau(x_{n+1};\mathcal{D}_n)\\
&=\mathbb{E}\left[u(\mathcal{D}_{n+\tau})|x_{n+1},\mathcal{D}_n\right]-u(\mathcal{D}_n)\\
&=\mathbb{E}\left[u(\mathcal{D}_{n+\tau})-u(\mathcal{D}_{n+1})+u(\mathcal{D}_{n+1})|x_{n+1},\mathcal{D}_n\right]-u(\mathcal{D}_n)\\
&=\left(\mathbb{E}\left[u(\mathcal{D}_{n+1})|x_{n+1},\mathcal{D}_n\right]-u(\mathcal{D}_n)\right)+\mathbb{E}\left[u(\mathcal{D}_{n+\tau})-u(\mathcal{D}_{n+1})|x_{n+1},\mathcal{D}_n\right]\\
&=\alpha_1(x_{n+1};\mathcal{D}_n)+\mathbb{E}\left[\alpha_{\tau-1}(x_{n+2};\mathcal{D}_{n+1})|x_{n+1},\mathcal{D}_n\right]
\end{aligned}$$

Here, we have decomposed the long-term expected marginal gain in utility into the sum of an immediate one-step lookahead gain in utility and the expected lookahead gain for the remaining $\tau - 1$ steps.

Now, following Bellman's principle of optimality, all the remaining $\tau - 1$ actions will be made optimally. This means that instead of evaluating each candidate location for x_{n+2} when calculating $\alpha_{\tau-1}(x_{n+2};\mathcal{D}_{n+1})$, we would only be interested in the location with the maximal value, that is, $\alpha_{\tau-1}(x^*_{n+2};\mathcal{D}_{n+1})$, or equivalently $\alpha^*_{\tau-1}(\mathcal{D}_{n+1})$, removing dependence on the location x_{n+2}. The multi-step lookahead acquisition function under the optimality assumption thus becomes

$$\alpha_\tau(x_{n+1};\mathcal{D}_n)=\alpha_1(x_{n+1};\mathcal{D}_n)+\mathbb{E}\left[\alpha^*_{\tau-1}(\mathcal{D}_{n+1})|x_{n+1},\mathcal{D}_n\right]$$

As shown in the previous section, the optimal next sampling location x^*_{n+1} using the multi-step lookahead acquisition function is thus determined by maximizing $\alpha_\tau(x_{n+1};\mathcal{D}_n)$. The optimal multi-step lookahead acquisition function $\alpha^*_\tau(x_{n+1};\mathcal{D}_n)$ is thus defined as

$$\alpha^*_\tau(x_{n+1};\mathcal{D}_n)=\max_{x_{n+1}\in A}\alpha_\tau(x_{n+1};\mathcal{D}_n)$$

Plugging in the definition of $\alpha_\tau(x_{n+1};\mathcal{D}_n)$ gives

$$\begin{aligned}
&\alpha^*_\tau(x_{n+1};\mathcal{D}_n)\\
&=\max_{x_{n+1}\in A}\left\{\alpha_1(x_{n+1};\mathcal{D}_n)+\mathbb{E}\left[\alpha^*_{\tau-1}(\mathcal{D}_{n+1})|x_{n+1},\mathcal{D}_n\right]\right\}\\
&=\max_{x_{n+1}\in A}\left\{\alpha_1(x_{n+1};\mathcal{D}_n)+\mathbb{E}\left[\max_{x_{n+2}\in A}\alpha_{\tau-1}(x_{n+2};\mathcal{D}_{n+1})|x_{n+1},\mathcal{D}_n\right]\right\}
\end{aligned}$$

where we have plugged in the definition of $\alpha^*_{\tau-1}(\mathcal{D}_{n+1})$ as well to explicitly express the optimal policy value $\alpha^*_{\tau}(x_{n+1};\mathcal{D}_n)$ as a series of nested maximization and expectation operations. Such recursive definition is called the *Bellman equation*, which explicitly reflects the condition that all follow-up actions need to be made optimally to make an optimal action.

Figure 3-6 summarizes the process of deriving the Bellman equation for the multi-step lookahead policy. Again, calculating the optimal policy value requires calculating the expected optimal value of future subpolicies. Being recursive in nature, calculating the current acquisition function can be achieved by adopting a reverse computation, starting from the terminal step and performing the calculations backward. However, this would still incur an exponentially increasing burden as the lookahead horizon expands.

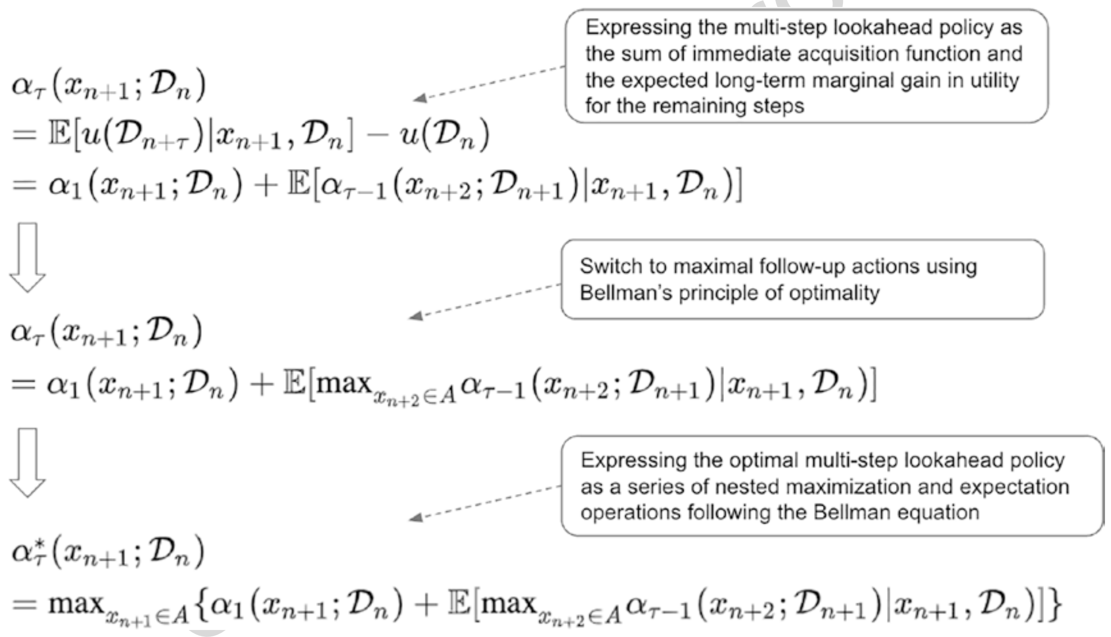
Expressing the multi-step lookahead policy as the sum of immediate acquisition function and the expected long-term marginal gain in utility for the remaining steps
$\alpha_\tau(x_{n+1};\mathcal{D}_n)$
$= \mathbb{E}[u(\mathcal{D}_{n+\tau})|x_{n+1},\mathcal{D}_n] - u(\mathcal{D}_n)$
$= \alpha_1(x_{n+1};\mathcal{D}_n) + \mathbb{E}[\alpha_{\tau-1}(x_{n+2};\mathcal{D}_{n+1})|x_{n+1},\mathcal{D}_n)]$
Switch to maximal follow-up actions using Bellman's principle of optimality
$\alpha_\tau(x_{n+1};\mathcal{D}_n)$
$= \alpha_1(x_{n+1};\mathcal{D}_n) + \mathbb{E}[\max_{x_{n+2}\in A}\alpha_{\tau-1}(x_{n+2};\mathcal{D}_{n+1})|x_{n+1},\mathcal{D}_n)]$
Expressing the optimal multi-step lookahead policy as a series of nested maximization and expectation operations following the Bellman equation
$\alpha^*_\tau(x_{n+1};\mathcal{D}_n)$
$= \max_{x_{n+1}\in A}\{\alpha_1(x_{n+1};\mathcal{D}_n) + \mathbb{E}[\max_{x_{n+2}\in A}\alpha_{\tau-1}(x_{n+2};\mathcal{D}_{n+1})|x_{n+1},\mathcal{D}_n)]\}$

Figure 3-6. *Illustrating the derivation process of the Bellman equation for the multi-step lookahead policy, where the optimal policy is expressed as a series of maximization and expectation operations, assuming all follow-up actions need to be made optimally in order to make the optimal action at the current step*

We will touch upon several tricks to accelerate the calculation of this *dynamic programming* (DP) problem later in the book and only highlight two common approaches for now. One approach is called limited lookahead, which limits the number of lookahead steps in the future. The other is to use a rollout approach with a base policy, which reduces the maximization operator into a quick heuristic-based exercise. Both approaches are called approximate dynamic programming (ADP) methods and are illustrated in Figure 3-7. See the recent book titled *Bayesian Optimization* by Roman Garnett for more discussion on this topic.

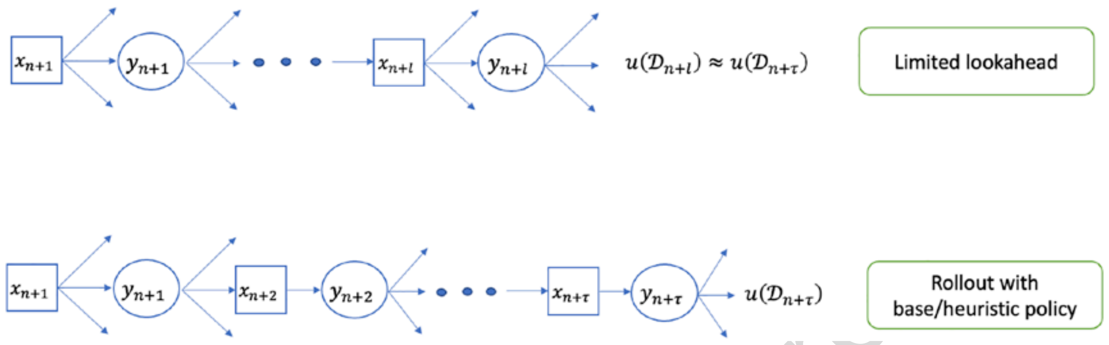
x_{n+1}
y_{n+1}
x_{n+l}
y_{n+l}
$u(\mathcal{D}_{n+l}) \approx u(\mathcal{D}_{n+\tau})$
Limited lookahead
x_{n+1}
y_{n+1}
x_{n+2}
y_{n+2}
$x_{n+\tau}$
$y_{n+\tau}$
$u(\mathcal{D}_{n+\tau})$
Rollout with base/heuristic policy

Figure 3-7. *Two approximate dynamic programming approaches commonly used to calculate the multi-step lookahead BO policies*

In the next section, we will introduce the expected improvement acquisition function, which is the most widely used and empirically performing acquisition function in practical Bayesian optimization applications.

Expected Improvement

Acquisition functions differ in multiple aspects, including the choice of the utility function, the number of lookahead steps, the level of risk aversion or preference, etc. Introducing risk appetite directly benefits from the posterior belief about the underlying objective function. In the case of GP regression as the surrogate model, the risk is quantified by the covariance function, with its credible interval expressing the level of uncertainty about the possible values of the objective.

When it comes to the utility of the collected observations, the expected improvement chooses the maximum of the observed value as the benchmark for comparison upon selecting an additional sampling location. It also implicitly assumes that only one sampling is left before the optimization process terminates. The expected marginal gain in utility (i.e., the acquisition function) becomes the expected improvement in the maximal observation, calculated as the expected difference between the observed maximum and the new observation after the additional sampling at an arbitrary sampling location.

These assumptions make the expected improvement a one-step lookahead acquisition function, also called *myopic* due to its short lookahead horizon. Besides, since the expectation of the posterior distribution is used, the expected improvement is also considered *risk neutral*, disregarding the uncertainty estimates across the whole domain.

Specifically, denote $y_{1:n} = \{y_1, \ldots, y_n\}$ as the set of collected observations at the corresponding locations $x_{1:n} = \{x_1, \ldots, x_n\}$. Assuming the noise-free setting, the actual observations are exact, that is, $y_{1:n} = f_{1:n}$. Given the collected dataset $\mathcal{D}_n = \{x_{1:n}, y_{1:n}\}$, the corresponding utility is $u(\mathcal{D}_n) = \max\{f_{1:n}\} = f_n^*$, where f_n^* is the incumbent maximum observed so far. Similarly, assuming we obtain another observation $y_{n+1} = f_{n+1}$ at a new location x_{n+1}, the resulting utility is $u(\mathcal{D}_{n+1}) = u(\mathcal{D}_n \cup \{x_{n+1}, f_{n+1}\}) = \max\{f_{n+1}, f_n^*\}$. Taking the difference of these two gives the increase in utility due to the addition of another observation:

$$u(\mathcal{D}_{n+1}) - u(\mathcal{D}_n) = \max\{f_{n+1}, f_n^*\} - f_n^* = \max\{f_{n+1} - f_n^*, 0\}$$

which returns the marginal increment in the incumbent if $f_{n+1} \geq f_n^*$ and zero otherwise, as a result of observing f_{n+1}. Readers familiar with the activation function in neural networks would instantly connect this form with the ReLU (rectified linear unit) function, which keeps the positive signal as it is and silences the negative one.

Due to randomness in y_{n+1}, we can introduce the expectation operator to integrate it out, giving us the expected marginal gain in utility, that is, the expected improvement acquisition function:

$$\begin{aligned}\alpha_{\text{EI}}(x_{n+1}; \mathcal{D}_n) &= \mathbb{E}\left[u(\mathcal{D}_{n+1}) - u(\mathcal{D}_n) | x_{n+1}, \mathcal{D}_n\right] \\ &= \int \max\{f_{n+1} - f_n^*, 0\} p(f_{n+1} | x_{n+1}, \mathcal{D}_n) df_{n+1}\end{aligned}$$

Under the framework of GP regression, we can obtain a closed-form expression of the expected improvement acquisition function, as shown in the following section.

Deriving the Closed-Form Expression

The expected improvement acquisition function admits a convenient closed-form expression, which could significantly accelerate its computation. Deriving the closed-form expression requires the scale-location transformation from a standard normal variable to an arbitrary normally distributed variable, as covered in a previous chapter. This is also called the *reparameterization trick* since we can convert the subject of interest into a standard normal variable to simplify mathematical analysis and practical computation.

Precisely, since the observation f_{n+1} at candidate location x_{n+1} follows a normal distribution with the corresponding posterior mean μ_{n+1} and variance σ_{n+1}^2, writing $f_{n+1} \sim N\left(f_{n+1}; \mu_{n+1}, \sigma_{n+1}^2\right)$, we can reparameterize it as $f_{n+1} = \mu_{n+1} + \sigma_{n+1}\varepsilon$, where $\varepsilon \sim N(\varepsilon; 0, 1)$.

Figure 3-8 gives the full derivation process that involves a few technical details such as linearity of expectation, integration by parts, the standard and cumulative standard normal distribution, and change of variable in differentiation. Assuming $\sigma_{n+1}^2 > 0$, the process starts by converting the max operator into an integral, which is then separated into two different and easily computable parts. These two parts correspond to exploitation and exploration, respectively. Exploitation means continuing sampling the neighborhood of the observed region with a high posterior mean, and exploration encourages sampling an unvisited area where the posterior uncertainty is high. The expected improvement acquisition function thus implicitly balances off these two opposing forces.

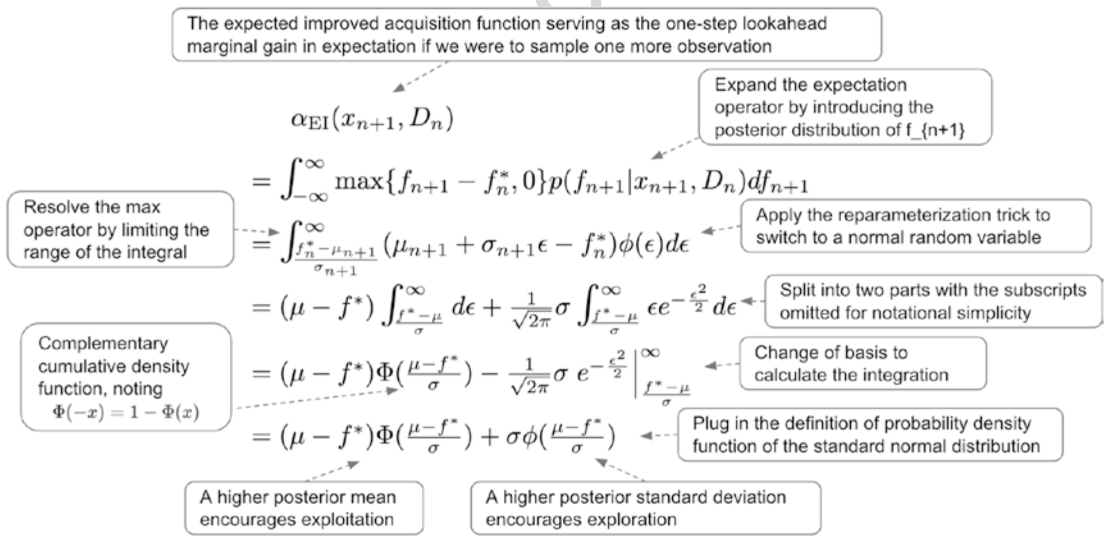
The expected improved acquisition function serving as the one-step lookahead marginal gain in expectation if we were to sample one more observation
Expand the expectation operator by introducing the posterior distribution of f_{n+1}
$\alpha_{\mathrm{EI}}(x_{n+1}, D_n)$
$= \int_{-\infty}^{\infty} \max\{f_{n+1} - f_n^*, 0\} p(f_{n+1}|x_{n+1}, D_n) df_{n+1}$
Resolve the max operator by limiting the range of the integral
Apply the reparameterization trick to switch to a normal random variable
$= \int_{\frac{f_n^* - \mu_{n+1}}{\sigma_{n+1}}}^{\infty} (\mu_{n+1} + \sigma_{n+1}\epsilon - f_n^*)\phi(\epsilon) d\epsilon$
$= (\mu - f^*) \int_{\frac{f^*-\mu}{\sigma}}^{\infty} d\epsilon + \frac{1}{\sqrt{2\pi}}\sigma \int_{\frac{f^*-\mu}{\sigma}}^{\infty} \epsilon e^{-\frac{\epsilon^2}{2}} d\epsilon$
Split into two parts with the subscripts omitted for notational simplicity
Complementary cumulative density function, noting $\Phi(-x) = 1 - \Phi(x)$
$= (\mu - f^*)\Phi(\frac{\mu - f^*}{\sigma}) - \frac{1}{\sqrt{2\pi}}\sigma\, e^{-\frac{\epsilon^2}{2}} \Big|_{\frac{f^*-\mu}{\sigma}}^{\infty}$
Change of basis to calculate the integration
$= (\mu - f^*)\Phi(\frac{\mu - f^*}{\sigma}) + \sigma\phi(\frac{\mu - f^*}{\sigma})$
Plug in the definition of probability density function of the standard normal distribution
A higher posterior mean encourages exploitation
A higher posterior standard deviation encourages exploration

Figure 3-8. *Deriving the closed-form expression of expected improvement, which automatically balances between the exploitation of promising areas given existing knowledge and exploration of uncertain areas*

To further establish the monotonic relationship between the posterior parameters (μ_{n+1} and σ_{n+1}^2) and the value of the expected improvement, we could examine the respective partial derivative. Concretely, we have the following:

$$\frac{\partial}{\partial \mu_{n+1}} \alpha_{\mathrm{EI}}\left(x_{n+1};\mathcal{D}_n\right) = \Phi\left(\frac{\mu_{n+1} - f^*}{\sigma_{n+1}}\right) > 0$$

$$\frac{\partial}{\partial \sigma_{n+1}} \alpha_{\mathrm{EI}}\left(x_{n+1};\mathcal{D}_n\right) = \phi\left(\frac{\mu_{n+1} - f^*}{\sigma_{n+1}}\right) > 0$$

Since the partial derivatives of the expected improvement with respect to μ_{n+1} and σ_{n+1} are both positive, an increase in either parameter will result in a higher expected improvement, thus completing the automatic trade-off between exploitation and exploration under the GP regression framework.

It is also worth noting that $\sigma_{n+1} = 0$ occurs when the posterior mean function passes through the observations. In this case, we have $\alpha_{\mathrm{EI}}\left(x_{n+1};\mathcal{D}_n\right) = 0$. In addition, a hyperparameter ξ is often introduced to control the amount of exploration in practical implementation. By subtracting ξ from $\mu_{n+1} - f_n^*$ in the preceding closed-form expression, the posterior mean μ_{n+1} will have less impact on the overall improvement compared to the posterior standard deviation σ_{n+1}. The closed-form expression of the expected improvement acquisition function thus becomes AU3

$$\alpha_{\mathrm{EI}}\left(x_{n+1};\mathcal{D}_n\right) = \begin{cases} \left(\mu_{n+1} - f_n^* - \xi\right)\Phi\left(z_{n+1}\right) + \sigma_{n+1}\phi\left(z_{n+1}\right);\ \sigma_{n+1} > 0 \\ 0;\ \sigma_{n+1} = 0 \end{cases}$$

where

$$z_{n+1} = \begin{cases} \dfrac{\mu_{n+1} - f^* - \xi}{\sigma_{n+1}};\sigma_{n+1} > 0 \\ 0;\sigma_{n+1} = 0 \end{cases}$$

The following section will implement the expected improvement acquisition function and use it to look for the global optimum of synthetic test functions.

Implementing the Expected Improvement

In this section, we will first implement the expected improvement acquisition function from scratch based on plain NumPy and SciPy packages, followed by using an off-the-shelf BO package to complete the same task. The true underlying objective function is given to us for comparison and unrevealed to the optimizer, whose goal is to approximate the objective function from potentially noise samples. The codes are adapted from a tutorial blog from Martin Krasser (http://krasserm.github.io/2018/03/21/bayesian-optimization/#Optimization-algorithm).

First, we will set up the environment by importing a few packages for Gaussian process regression from scikit-learn, numerical optimization from SciPy, and other utility functions on plotting. We also set the random seed to ensure reproducibility.

Listing 3-1. Setting up the coding environment

```
import numpy as np
import random
import matplotlib.pyplot as plt
from scipy.stats import norm
from scipy.optimize import minimize
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import ConstantKernel, Matern

SEED = 8
random.seed(SEED)
np.random.seed(SEED)
%matplotlib inline
```

Next, we will define the objective function and search domain. The objective function provides noise-perturbed observations upon sampling at an arbitrary location within the search domain. It will also be used to generate noise-free observations for reference during plotting.

As shown in the following code listing, we generate a random number from a standard normal distribution based on the dimension of the domain, accessed via the * sign to unpack the tuple into an acceptable format. The value is then multiplied by the prespecified noise level for the observation model. The search domain is specified as a nested list in bounds, where the inner list contains the upper and lower bounds for each dimension; in this case, we are looking at a single-dimensional search domain.

Listing 3-2. Defining the search domain and objective function

```
# search bounds of the domain
# each element in the inner list corresponds to one dimension
bounds = np.array([[-1.0, 2.0]])
# observation noise
noise = 0.2
# objective function used to reveal observations upon sampling, optionally
with noise
def f(x, noise=0):
    # use * to unpack the tuple from x.shape when passing into
    np.random.randn
    return -np.sin(3*x) - x**2 + 0.7*x + noise*np.random.randn(*x.shape)
```

Now we can visualize the objective function and generate two random noisy samples in `X_init` and `Y_init` to kick-start the optimization procedure. Note that plotting a function is completed by generating a dense grid of points/locations with the search bounds in `X_plot`, calculating the corresponding noise-free functional values `Y_plot` for each location, and connecting these values smoothly, as shown in the following code listing.

Listing 3-3. Visualizing the underlying objective function and initial noisy samples

```
# initial observations upon initiation
X_init = np.array([[-0.7], [1.6]])
Y_init = f(X_init, noise=noise)
# dense grid of points within bounds used for plotting
X_plot = np.arange(bounds[:, 0], bounds[:, 1], 0.01).reshape(-1, 1)
# noise-free objective function values used for plotting
Y_plot = f(X_plot, noise=0)
# Plot objective function with noisy observations
plt.plot(X_plot, Y_plot, 'y--', lw=2, label='Objective function')
plt.plot(X_init, Y_init, 'kx', mew=3, label='Initial noisy samples')
plt.legend()
```

The result is shown in Figure 3-9. Note that the two samples are selected to be sufficiently distant from each other. In practice, a good initial design should have good coverage of the whole search domain to promise a good GP prior before optimization starts.

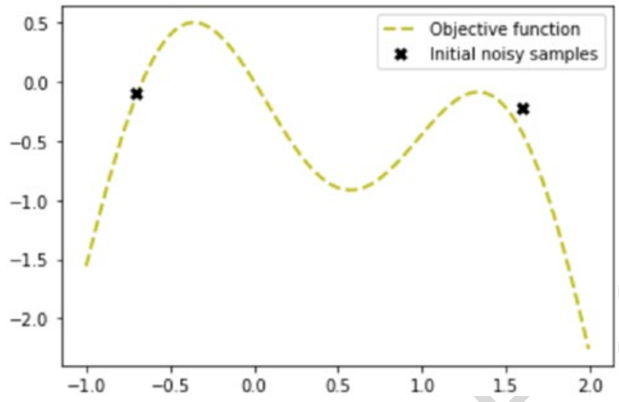

***Figure 3-9.** Visualizing the underlying objective function and two initial random noisy samples*

We now define the expected improvement acquisition function as our sampling policy. This function maps each sampling location input to a numeric scalar output, the expected marginal gain in utility. In the following code listing, other than the evaluation points in X, the inputs also include the previously observed locations X_sample and values Y_sample, along with a GP regressor gpr fitted to the training samples. Besides, we also include the hyperparameter xi to control the level of exploration with a default value of 0.01.

Listing 3-4. Defining the expected improvement acquisition function

```
def expected_improvement(X, X_sample, Y_sample, gpr, xi=0.01):
    # posterior mean and sd at proposed location
    mu, sigma = gpr.predict(X, return_std=True)
    # posterior mean at observed location
    mu_sample = gpr.predict(X_sample)
    # reshape to make one sd per each proposed location
    sigma = sigma.reshape(-1, 1)
    # use maximal posterior mean instead of actual observations due
    to noise
    mu_sample_opt = np.max(mu_sample)
    # ignore divide by zero warning if any
    with np.errstate(divide='warn'):
        # calculate ei if sd>0
        imp = mu - mu_sample_opt - xi
        Z = imp / sigma
        ei = imp * norm.cdf(Z) + sigma * norm.pdf(Z)
        # set zero if sd=0
        ei[sigma == 0.0] = 0.0
    return ei
```

Note that we start by plugging in the definition of expected improvement assuming a nonzero standard deviation for the posterior distribution at the proposed location, followed by setting the entries with zero standard deviation to be zero. Since directly dividing by zero gives an error, as needed when calculating `Z = imp / sigma`, the calculation is moved within the context of `np.errstate(divide='warn')`, which is a particular arrangement to tell Python to temporarily ignore such error because of the follow-up treatment via `ei[sigma == 0.0] = 0.0`.

At this stage, we can calculate the expected improvement of any candidate location, and our goal is to find the optimal location with the biggest value in expected improvement. To achieve this, we will use a particular off-the-shelf optimizer called "L-BFGS-B" provided by the `minimize` function from `SciPy`, which utilizes the approximate second-order derivative to solve for the optimum of a given function, that is, the expected improvement. The location of the optimum can be retrieved at the end of the optimization procedure.

The following code listing defines a function called `propose_location()` that performs optimization for a total of `n_restarts` rounds so as to avoid local optima. By keeping a running minimum `min_val` and its location `min_x`, each round of optimization returns an optimal solution via minimizing the negative of the acquisition function value via the `min_obj()` function; maximizing a positive value is equivalent to minimizing its negative. At last, we decide if the current running minimum and the location need to be replaced by comparing it with the optimization solution. This function also completes the *inner loop* of BO, as introduced earlier.

Listing 3-5. Proposing the next sampling point by optimizing the acquisition function

```
def propose_location(acquisition, X_sample, Y_sample, gpr, bounds,
n_restarts=25):
    # dimension of search domain
    dim = X_sample.shape[1]
    # temporary running best minimum
    min_val = 1
    # temporary location of best minimum
    min_x = None

    # map an arbitrary location to the negative of acquisition function
    def min_obj(X):
        # Minimization objective is the negative acquisition function
        return -acquisition(X.reshape(-1, dim), X_sample, Y_sample, gpr)

    # iterate through n_restart different random points and return most
    promising result
    for x0 in np.random.uniform(bounds[:, 0], bounds[:, 1],
    size=(n_restarts, dim)):
        # use off-the-shelf solver based on approximate second order
        derivative
        res = minimize(min_obj, x0=x0, bounds=bounds, method='L-BFGS-B')
        # replace running optimum if any
        if res.fun < min_val:
            min_val = res.fun[0]
            min_x = res.x

    return min_x.reshape(-1, 1)
```

Before entering BO's outer loop to seek the global optimum, we will define a few utility functions that plot the policy performance across iterations. This includes the `plot_approximation()` function that plots the GP posterior mean and 95% confidence interval along with the collected samples and objective function, the `plot_acquisition()` function that plots the expected improvement across the domain along with the location of the maximum, and the def `plot_convergence()` function that plots the distances between consecutive sampling locations and the running optimal value as optimization proceeds. All three functions are defined in the following code listing. AU4

Listing 3-6. Proposing the next sampling point by optimizing the acquisition function

```
def plot_approximation(gpr, X_plot, Y_plot, X_sample, Y_sample,
X_next=None, show_legend=False):
    # get posterior mean and sd across teh dense grid
    mu, std = gpr.predict(X_plot, return_std=True)
    # plot mean and 95% confidence interval
    plt.fill_between(X_plot.ravel(),
                     mu.ravel() + 1.96 * std,
                     mu.ravel() - 1.96 * std,
                     alpha=0.1)
    plt.plot(X_plot, Y_plot, 'y--', lw=1, label='Noise-free objective')
    plt.plot(X_plot, mu, 'b-', lw=1, label='Surrogate function')
    plt.plot(X_sample, Y_sample, 'kx', mew=3, label='Noisy samples')
    # plot the next sampling location as vertical line
    if X_next:
        plt.axvline(x=X_next, ls='--', c='k', lw=1)
    if show_legend:
        plt.legend()

def plot_acquisition(X_plot, acq_value, X_next, show_legend=False):
    # plot the value of acquisition function across the dense grid
    plt.plot(X_plot, acq_value, 'r-', lw=1, label='Acquisition function')
    # plot the next sampling location as vertical line
    plt.axvline(x=X_next, ls='--', c='k', lw=1, label='Next sampling
    location')
    if show_legend:
        plt.legend()

def plot_convergence(X_sample, Y_sample, n_init=2):
    plt.figure(figsize=(12, 3))
    # focus on sampled queried by the optimization policy
    x = X_sample[n_init:].ravel()
    y = Y_sample[n_init:].ravel()
    r = range(1, len(x)+1)
    # distance between consecutive sampling locations
    x_neighbor_dist = [np.abs(a-b) for a, b in zip(x, x[1:])]
    # best observed value until the current time point
    y_max = np.maximum.accumulate(y)
    # plot the distance between consecutive sampling locations
    plt.subplot(1, 2, 1)
    plt.plot(r[1:], x_neighbor_dist, 'bo-')
    plt.xlabel('Iteration')
    plt.ylabel('Distance')
    plt.title('Distance between consecutive x\'s')
    # plot the evolution of observed maximum so far
    plt.subplot(1, 2, 2)
    plt.plot(r, y_max, 'ro-')
    plt.xlabel('Iteration')
    plt.ylabel('Best Y')
    plt.title('Value of best selected sample')
```

Now we can move into the main outer loop to look for the global optimum by maximizing the expected improvement at each stage. In the following code listing, we first instantiate a GP regressor with a Matérn kernel, which accepts two hyperparameters that can be estimated by maximizing the marginal likelihood of the observed samples. In this case, we fix these hyperparameters to simplify the process. The GP regressor also accepts the unknown noise level via the `alpha` argument to incorporate noise in the observations.

Listing 3-7. The main BO loop

```
# Gaussian process with Matern kernel as surrogate model
# kernel parameters could be optimized using MLE
m52 = ConstantKernel(1.0) * Matern(length_scale=1.0, nu=2.5)
# specify observation noise term, assumed to be known in advance
gpr = GaussianProcessRegressor(kernel=m52, alpha=noise**2)
# initial samples before optimization starts
```

```
X_sample = X_init
Y_sample = Y_init
# number of optimization iterations
n_iter = 20
# specify figure size
plt.figure(figsize=(12, n_iter * 3))
plt.subplots_adjust(hspace=0.4)
# start of optimization
for i in range(n_iter):
    # update GP posterior given existing samples
    gpr.fit(X_sample, Y_sample)
    # obtain next sampling point from the acquisition function (expected_
    improvement)
    X_next = propose_location(expected_improvement, X_sample, Y_sample,
    gpr, bounds)
    # obtain next noisy sample from the objective function
    Y_next = f(X_next, noise)
    # plot samples, surrogate function, noise-free objective and next
    sampling location
    plt.subplot(n_iter, 2, 2 * i + 1)
    plot_approximation(gpr, X_plot, Y_plot, X_sample, Y_sample, X_next,
    show_legend=i==0)
    plt.title(f'Iteration {i+1}')
    plt.subplot(n_iter, 2, 2 * i + 2)
    plot_acquisition(X_plot, expected_improvement(X_plot, X_sample, Y_
    sample, gpr), X_next, show_legend=i==0)
    # append the additional sample to previous samples
    X_sample = np.vstack((X_sample, X_next))
    Y_sample = np.vstack((Y_sample, Y_next))
```

Here, we use X_sample and Y_sample to be the running dataset augmented with additional samples as optimization continues for a total of 20 iterations. Each iteration consists of updating the GP posterior, locating the maximal expected improvement, observing at the proposed location, and incorporating the additional sample to the training set.

The codes also generate plots using `plot_approximation()` and `plot_acquisition()` to show more details on the optimization in each iteration. Figure 3-10 shows the first three iterations, where the optimizer exhibits an exploratory attribute by proposing samples relatively distant from each other. In other words, regions with high uncertainty are encouraged at the initial stage of optimization using the expected improvement acquisition function.

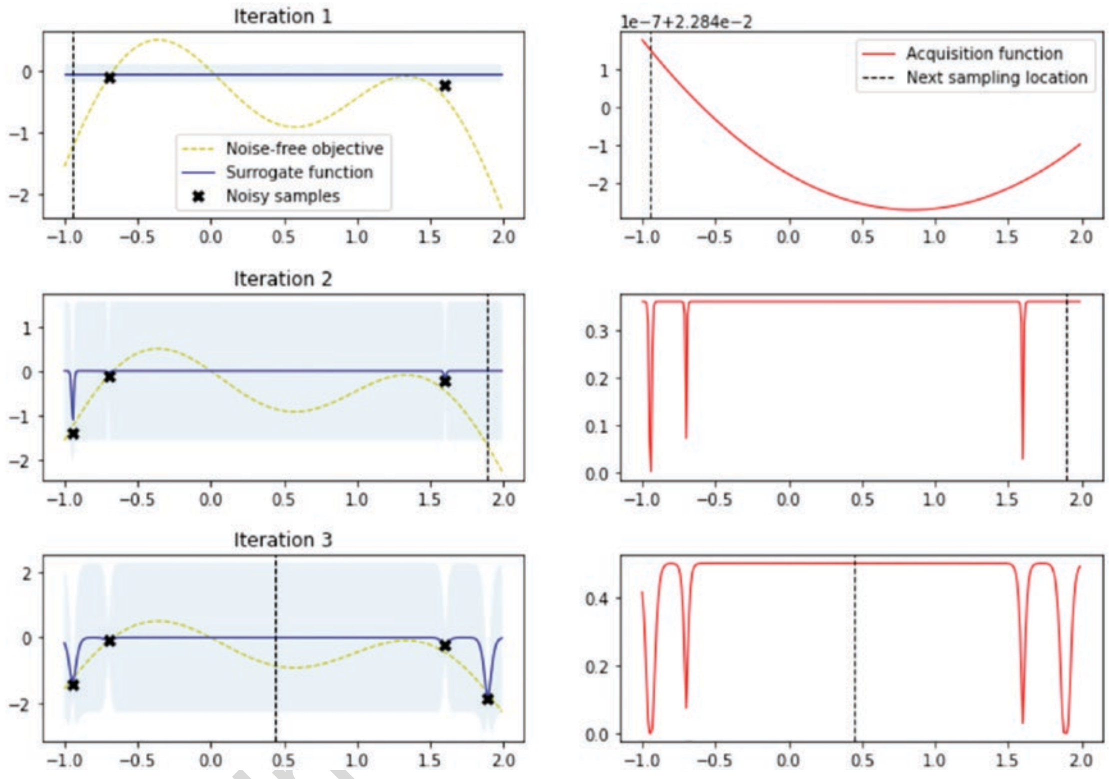

Figure 3-10. *Plotting the first three iterations, in which the EI-based BO performs more exploration at regions with high uncertainty*

As the optimization proceeds, the optimizer gradually resolves the uncertainty at distant locations and starts to reply more on exploitation of promising regions based on existing knowledge. This is reflected by the concentration of sampling locations at the left peak of the objective function, as shown by the last three iterations in Figure 3-11. Given that the last three sampling proposals occur at similar locations, we can roughly sense that the optimization process has converged, and the task of locating the global maximum is successfully completed.

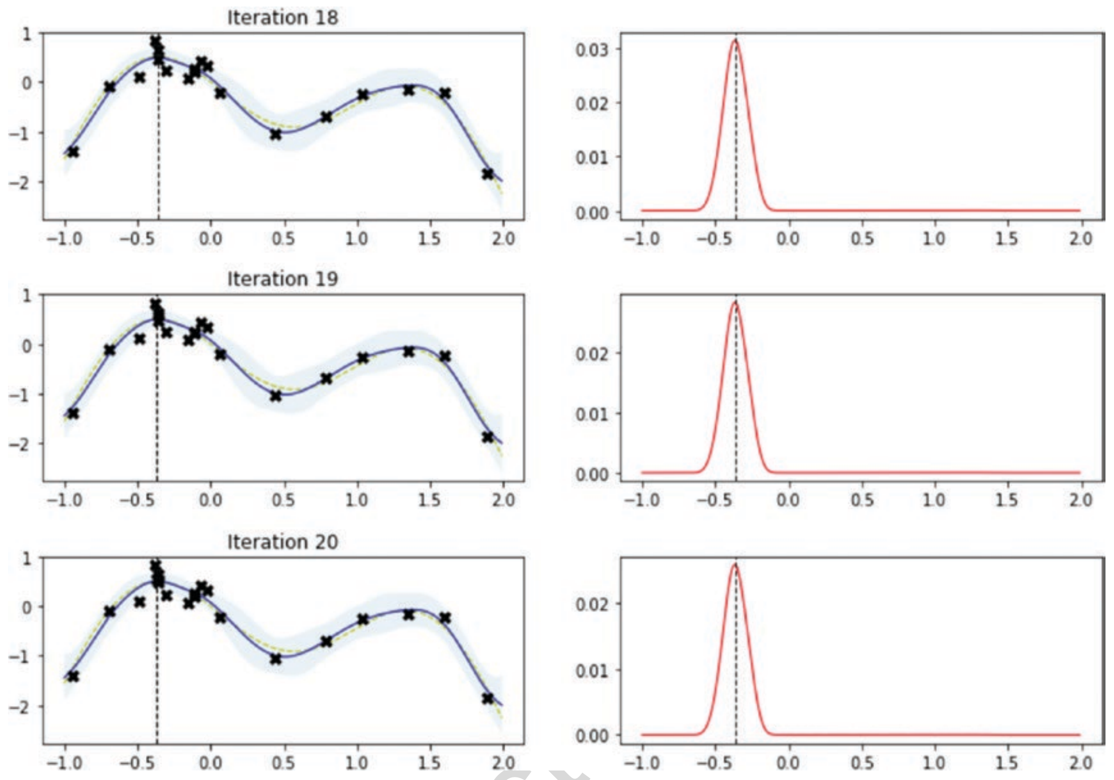

Figure 3-11. *Concentration of sampling locations at the left peak of the objective function, a sign of exploitation as the optimization process converges*

For the full list of intermediate plots across iterations, please visit the accompanying notebook for this chapter at `https://github.com/jackliu333/bayesian_optimization_theory_and_practice_with_python/blob/main/Chapter_3.ipynb`.

Once the optimization completes, we can examine its convergence using the `plot_convergence()` function. As shown in the left plot in Figure 3-12, a larger distance corresponds to more exploration, which occurs mostly at the initial stage of optimization as well as iterations 17 and 18 even when the optimization seems to be converging. Such exploration nature is automatically enabled by expected improvement and helps jumping out of local optima in search of a potentially higher global optimum. This is also reflected in the right plot, where a higher value is obtained at iteration 17 due to exploration.

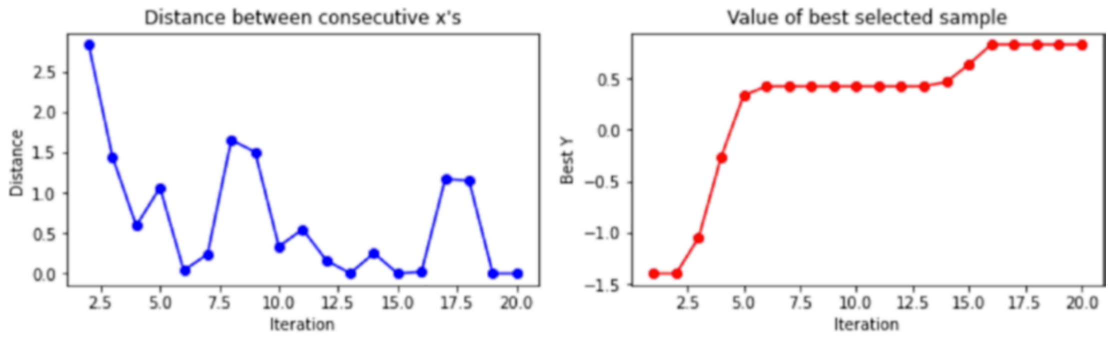

Figure 3-12. *Plotting the distance between consecutive proposed locations and the value of the best-selected sample as optimization proceeds*

At this point, we have managed to implement the full BO loop using expected improvement from scratch. Next, we will look at a few BO libraries that help us achieve the same task.

Using Bayesian Optimization Libraries

In this section, we will use two public Python-based libraries that support BO: `scikit-optimize` and `GPyOpt`. Both packages provide utility functions that perform BO after specifying the relevant input arguments. Let us look at optimizing the same function as earlier using `gp_minimize`, a function from `scikit-optimize` used to perform BO using GP.

In the following code listing, we specify the same kernel and hyperparameters setting for the GP instance `gpr`, along with the function `f` that provides noisy samples, search bounds `dimensions`, acquisition function `acq_func`, initial samples, exploration and exploitation trade-off parameter `xi`, number of iterations `n_calls`, as well as initial samples in `x0` and `y0`. At the end of optimization, we show the approximation plot to observe the locations of the proposed samples.

Listing 3-8. BO using scikit-optimize

```
from sklearn.base import clone
from skopt import gp_minimize
from skopt.learning import GaussianProcessRegressor
from skopt.learning.gaussian_process.kernels import ConstantKernel, Matern
```

```
# use custom kernel and estimator to match previous example
m52 = ConstantKernel(1.0) * Matern(length_scale=1.0, nu=2.5)
g = GaussianProcessRegressor(kernel=m52, alpha=noise**2)
# start BO
r = gp_minimize(func=lambda x: -f(np.array(x), noise=noise)[0], # function
to minimize
                dimensions=bounds.tolist(),    # search bounds
                base_estimator=gpr, # GP prior
                acq_func='EI',      # expected improvement
                xi=0.01,            # exploitation-exploration trade-off
                n_calls=n_iter,     # number of iterations
                n_initial_points=0, # initial samples are provided
                x0=X_init.tolist(), # initial samples
                y0=-Y_init.ravel())
# fit GP model to samples for plotting
gpr.fit(r.x_iters, -r.func_vals)
# Plot the fitted model and the noisy samples
plot_approximation(gpr, X_plot, Y_plot, r.x_iters, -r.func_vals, show_
legend=True)
```

Running the preceding code will generate Figure 3-13, which shows a concentration of samples around the global maximum at the left peak. Note that the samples are not exactly the same as in our previous example due to the nondeterministic nature of the optimization procedure as well as the randomness in the observation model.

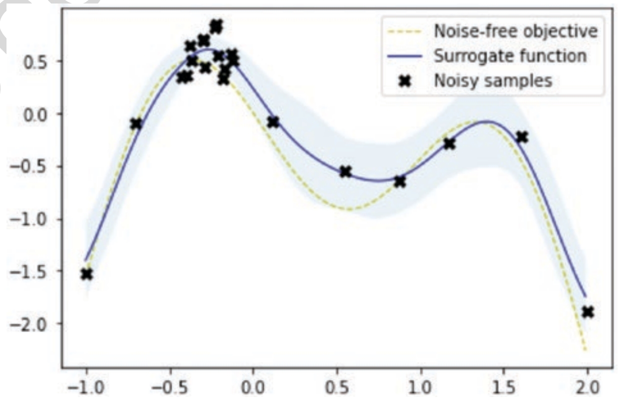

Figure 3-13. *Visualizing the proposed samples using the* `gp_minimize()` *function*

We can also show the plots on the distances of consecutive proposals and the best-observed value. As shown in Figure 3-14, even though the optimizer obtains a high value at the second iteration, it continues to explore promising regions with high uncertainty, as indicated by the two peaks in the distance plot.

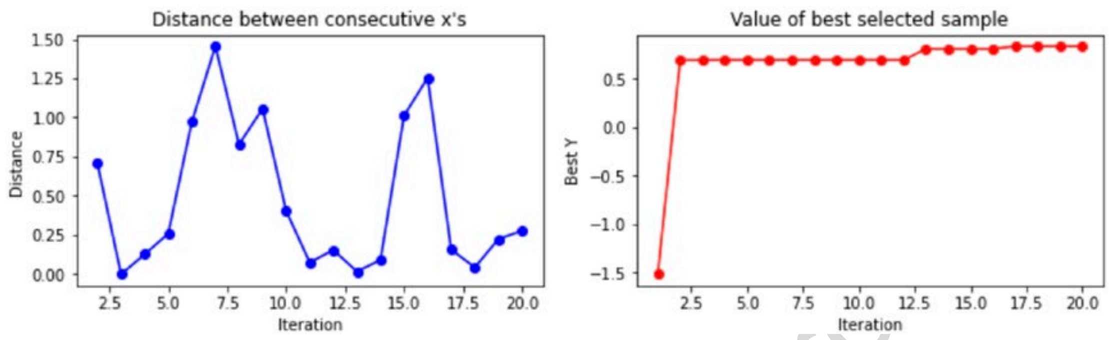

Figure 3-14. *Visualizing the convergence plots*

Summary

Bayesian optimization is an extension of the classic Bayesian decision theory. The extension goes into its use and choice of surrogate and acquisition functions. In this chapter, we covered the following list of items:

- Bayesian optimization requires defining a utility function that measures the value of the returned dataset in seeking the global optimum.
- The inner BO loop involves seeking the location that maximizes the acquisition function, and the outer BO loop seeks the location of the global optimum.
- The acquisition function is defined as the expected marginal gain in utility, which can be myopic (one-step lookahead) or nonmyopic (multi-step lookahead).
- Optimizing the multi-step lookahead expected marginal gain in utility follows Bellman's principle of optimality and can be expressed as a recursive form, that is, a sum of the immediate expected marginal gain in utility and the maximal expected marginal gain from all future evolutions.

- Expected improvement is a widely used one-step lookahead acquisition function that recommends the best-observed value upon optimization terminates and has a friendly closed-form expression to support fast computation. AU6

In the next chapter, we will revisit the Gaussian process and discuss GP regression using a widely used framework: GPyTorch.

Author Queries

Chapter No.: 3 0005551214

Queries	Details Required	Author's Response
AU1	Please check if "for the seek of notational convenience" is correct here.	
AU2	Please check if "In the case of GP regression" is okay as edited.	
AU3	Please check if "The closed-form expression" is okay as edited.	
AU4	Please check if "def plot_convergence() function" should be changed to "plot_convergence() function".	
AU5	Please check if "to reply more on exploitation" is correct here.	
AU6	Please check if "upon optimization terminates" should be changed to "when optimization terminates".	