Politecnico di Milano

MSc in Mathematical Engineering

Course in Bayesian Statistics

February 15, 2022

Bayesian Optimization

A comparative study on acquisition functions

Authors:
Raffaella D'Anna
Michele Di Sabato
Martina Garavaglia
Anna Iob
Veronica Mazzola

Supervisor:
Bruno Guindani

Abstract

In this report we will be presenting a brief introduction to Bayesian Optimization, which is an approach to finding the maximum of expensive cost functions. Furthermore we will explore in detail the most commonly used acquisition functions, used in order to guide the search for the optimum of the objective function. We will also compare their performances, with a keen interest to the Knowledge Gradient.

In particular our focus was implementing in Python some ad hoc functions to the already existing open-source library of Bayesian Optimization by Fernando Nogueira [1]. We developed a multi-dimensional version of the Knowledge Gradient function, together with some graphical interfaces for the visualization of all the acquisition functions already implemented in the library and the Knowledge Gradient. These interfaces help us better visualize our results and compare their convergence to the optimum.

Finally we will consider how to address new problems, such as the presence of Gaussian noise in the objective function.

Contents

1 Introduction

The main goal of Bayesian Optimization is solving the problem

$$\max_{\boldsymbol{x} \in A \subset \mathbb{R}^d} f(\boldsymbol{x}) \tag{1}$$

where

- $f(\boldsymbol{x})$ is the objective function
- A is a feasible set, typically A is a hyper-rectangle
- The objective function f is continuous.
- f is "expensive to evaluate" typically we only evaluate it a few hundred times because each evaluation is computationally expensive or has a monetary cost (e.g., from purchasing cloud computing power, or buying laboratory materials), or has a high opportunity cost (e.g., if evaluating f requires human computations).
- f lacks of a known special structure like concavity or linearity. Moreover we observe only f and no first- or second-order derivatives. This prevents the application of first- and second-order methods like Newton's method. In general, a Bayesian optimization algorithm works also if the analytical expression of f is unknown and we are only able to obtain its evaluations in specific points. We thus say that $f(\boldsymbol{x})$ is a "black box" function.
- Our focus is on finding a global rather than local optimum.

In order to achieve such goal, two main ingredients are necessary: a Bayesian statistical model for modeling the objective function, and an acquisition function to decide where to sample next.

The statistical model would be a Gaussian process, which provides a Bayesian posterior probability distribution that describes potential values for $f(\boldsymbol{x})$ at a candidate point x. Each time we observe f at a new point, this posterior distribution is updated. On the other hand, the acquisition function measures the added value that would be generated through the evaluation of the objective function at a new point x, based on the current posterior distribution over f. The mean can be interpreted as a point estimate of $f(\boldsymbol{x})$.

2 Gaussian process regression

A Gaussian process regression is a Bayesian statistical approach for modeling functions. From a general point of view, we have a finite collection of points $\boldsymbol{x}_1, ..., \boldsymbol{x}_k \in \mathbb{R}^d$ and a vector of evaluations of our objective function $[f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_k)]$ in these points that is unknown.

In particular, using a Gaussian Process as prior for f entails that the prior distribution for this vector is a multivariate normal with a particular mean vector and covariance matrix, i.e.:

$$f(\boldsymbol{x}_{1:k}) \sim \mathcal{N}(\mu_0(\boldsymbol{x}_{1:k}), \Sigma_0(\boldsymbol{x}_{1:k}, \boldsymbol{x}_{1:k})) \tag{2}$$

where $\mu_0(\boldsymbol{x}_{1:k})$ is the mean vector evaluated in $\boldsymbol{x}_{1:k}$ and $\Sigma_0(\boldsymbol{x}_{1:k}, \boldsymbol{x}_{1:k})$ is the covariance matrix or kernel evaluated in $\boldsymbol{x}_{1:k}$.
For our purposes, Gaussian processes are used to model a priori the objective function, which in this black-box optimization framework is assumed to be the unknown quantity of interest. Consequently, the posterior distribution of the vector $[f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_k)]$ is still a Gaussian distribution. In particular, to obtain such posterior, the Gaussian process is updated whenever we suggest a new point in which to evaluate the objective function next (i.e. at each iteration of the algorithms).

2.1 Mean function and covariance matrix

We construct a mean vector by evaluating a mean function μ_0 at each point $\boldsymbol{x}_i$. The most common choice for the mean function is a constant value $\mu_0(\boldsymbol{x}) = \mu$, but when f is believed to have a specific trend or some particular structure useful in applications, we can use a function such as:

$$\mu_0(\boldsymbol{x}) = \mu + \sum_{i=1}^{p} \beta_i \Psi_i(\boldsymbol{x}) \tag{3}$$

where each Ψ_i is a parametric function (most of the time a low order polynomial in $\boldsymbol{x}$).

The choice of the kernel depends on two properties, which need to be satisfied for the kernel to be used as a covariance structure:

- Points closer in the input space are more strongly correlated, i.e., taking some norm $||*||$, if $||\boldsymbol{x} - \boldsymbol{x}'|| < ||\boldsymbol{x} - \boldsymbol{x}''||$ then $\Sigma_0(\boldsymbol{x}, \boldsymbol{x}') > \Sigma_0(\boldsymbol{x}, \boldsymbol{x}'')$
- Kernels are positive semi-definite functions.

There exist different types of kernels. One of the most commonly used for its simplicity is the power exponential or Gaussian kernel:

$$\Sigma_0(\boldsymbol{x}, \boldsymbol{x}') = \alpha_0 \exp(-||\boldsymbol{x} - \boldsymbol{x}'||^2) \tag{4}$$

where $||\boldsymbol{x} - \boldsymbol{x}'||^2 = \sum_{i=1}^{d} \alpha_i (\boldsymbol{x}_i - \boldsymbol{x}_i')^2$ and α_0 are parameter of the kernel chosen by the user.
The mean function and kernel hyperparameters are chosen with three different approaches:

- Use the *maximum likelihood estimator* (MLE)
- Use the *maximum a posteriori* (MAP) estimator
- Use a *fully Bayesian* approach computing the posterior distribution of $f(\boldsymbol{x})$ marginalizing over all possible values of the hyperparameters.

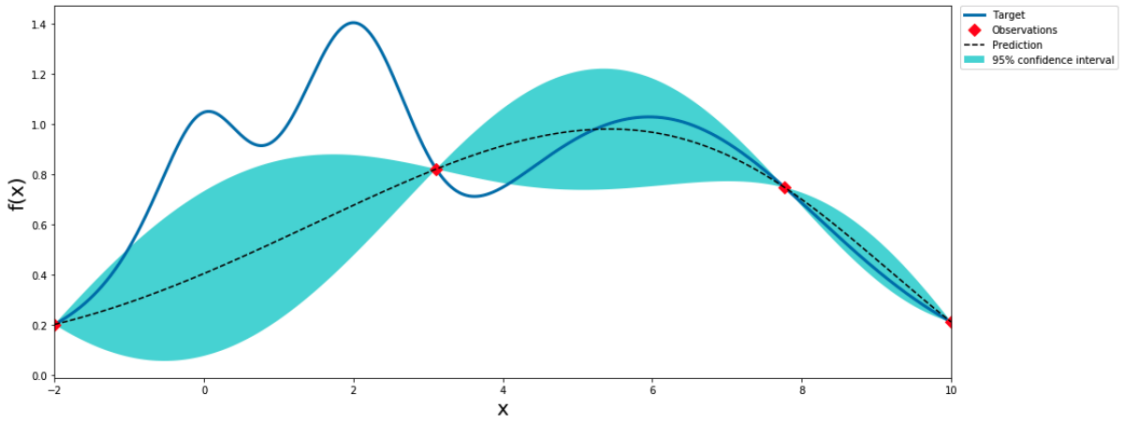

Figure 1: Example of a Gaussian process with four observations.
Target function: $f(x) = e^{-(x-2)^2} + e^{-\frac{(x-6)^2}{10}} + \frac{1}{x^2+1}$ *with* $x \in (-2, 10)$

3 Acquisition functions

Acquisition functions are used in order to guide the search for the optimum, and are typically defined in a way such that points which maximize the acquisition function are close to the point which maximizes the objective function. This can either be because the prediction is high, or because the uncertainty is great, or both. So the maximization of the acquisition function u is used to select the next point at which to evaluate the objective function:

$$\boldsymbol{x}_{n+1} = \arg\max_{\boldsymbol{x} \in A} u(\boldsymbol{x} | \mathcal{D}_{1:n}) \tag{5}$$

$\mathcal{D}_{1:n}$ is the collection of all the points in the dataset up to iteration n, i.e.

$$\mathcal{D}_{1:n} = \{(\boldsymbol{x}_i, f(\boldsymbol{x}_i))\}_{i=1}^{n}$$

3.1 Probability of Improvement

One strategy would be to mazimize the *probability of improvement* over the current $f_n^* = \arg\max_{i \in [1,n]} f(\boldsymbol{x}_i)$, so that

$$PI(\boldsymbol{x}) = P(f(\boldsymbol{x}) \geq f_n^*) = \Phi\left(\frac{\mu(\boldsymbol{x}) - f_n^*}{\sigma(\boldsymbol{x})}\right) \tag{6}$$

where Φ is the normal cumulative distribution function.
Other names under which this function is known are MPI (for "maximum probability of improvement") and "the P-algorithm" (since the utility is the probability of improvement).
However the main drawback of this formulation is that it is pure exploitation: points that have a high probability of being infinitesimally greater than f_n^* will be drawn over points that offer larger gains but less certainty.
Adding a trade-off parameter ξ can be a possible solution to tackle this problem;

in this way the *probability of improvement* selects the point most likely to offer an improvement of at least ξ, and its analytical expression becomes:

$$PI(\boldsymbol{x}) = P(f(\boldsymbol{x}) \geq f_n^* + \xi) = \Phi\left(\frac{\mu(\boldsymbol{x}) - f_n^* - \xi}{\sigma(\boldsymbol{x})}\right) \tag{7}$$

where the exact choice of the ξ is left to the users.
This trade-off, called exploitation and exploration trade-off, is important because an acquisition function should balance the will to evaluate regions in which the posterior mean is high and regions in which the variance is large.
A somewhat more satisfying alternative acquisition function would be one that takes into account not only the probability of improvement, but also the magnitude of the improvement a point can potentially yield.

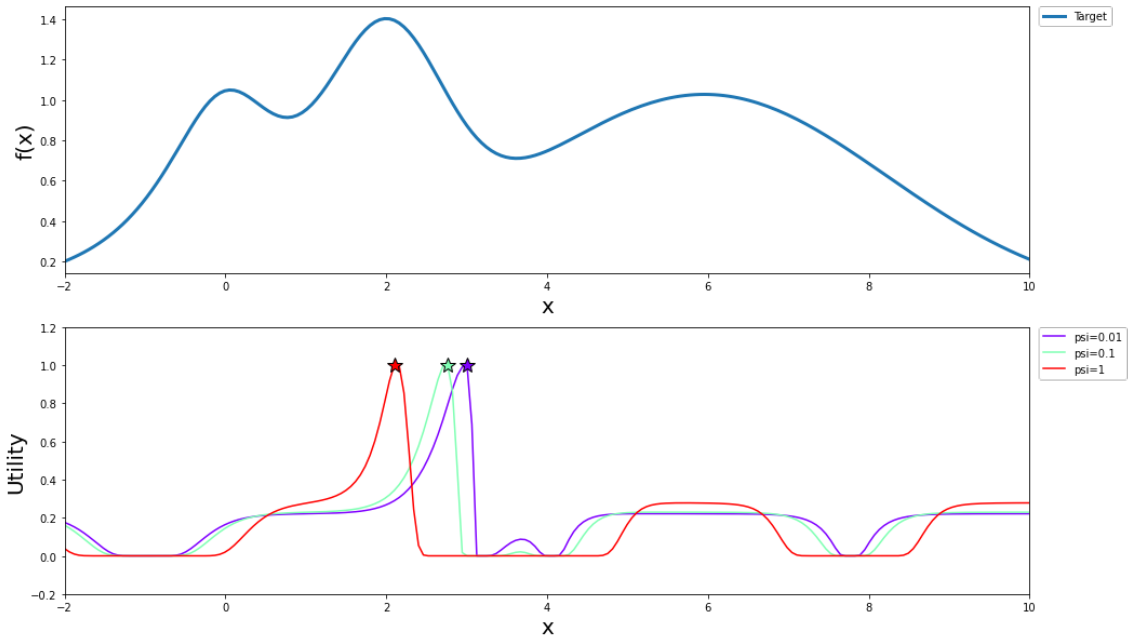

Figure 2: Probability of improvement after 5 iterations and 2 initial points with different parameter values.
Target function: as in Figure 1

3.2 Expected Improvement

Expected improvement is the most commonly used acquisition function since it is not expensive to evaluate and easy to use.
It selects as next point to evaluate, the one that guarantees the biggest improvement between $f(\boldsymbol{x})$, unknown until after the evaluation, and f_n^*, the largest observed value in the first n iterations. Its drawback is that it depends only on points that are already in the dataset up to a certain iteration.
The *expected improvement* is defined as:

$$EI_n(\boldsymbol{x}) := \mathbb{E}_n[\max\{0,\ f(\boldsymbol{x}) - f_n^*\}] \tag{8}$$

or, alternatively:

$$EI_n(\boldsymbol{x}) := [\Delta_n(\boldsymbol{x})]^+ + \sigma_n(\boldsymbol{x})\varphi\left(\frac{\Delta_n(\boldsymbol{x})}{\sigma_n(\boldsymbol{x})}\right) + |\Delta_n(\boldsymbol{x})|\,\Phi\left(\frac{\Delta_n(\boldsymbol{x})}{\sigma_n(\boldsymbol{x})}\right) \tag{9}$$

where $\Delta_n(\boldsymbol{x}):=\mu_n(\boldsymbol{x})$-$f_n^*$ is the expected difference in quality between the proposed point $\boldsymbol{x}$ and the previous best.
The expected improvement algorithm evaluates each next point choosing it according to the exploration-exploitation trade-off, meaning that points with largest expected improvement are those in which the posterior standard deviation or the posterior mean is higher.

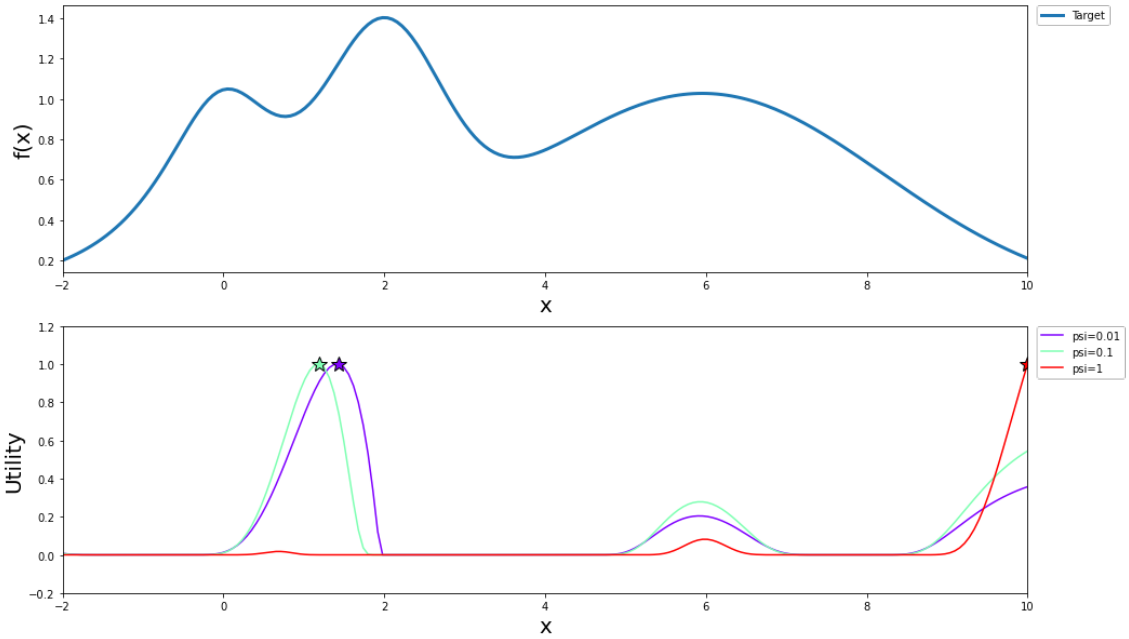

Figure 3: Expected improvement after 5 iterations and 2 initial points with different parameter values.
Target function: as in Figure 1

3.3 Upper Confidence Bound

Another typically used acquisition function is the Upper Confidence Bound, whose analytical expression is:

$$UCB(\boldsymbol{x}) = \mu(\boldsymbol{x}) + k\sigma(\boldsymbol{x}) \tag{10}$$

where the choice of parameter k is left to the user.
A higher value of k favors regions that are least explored, instead a lower value favors regions where the regression function is the highest.

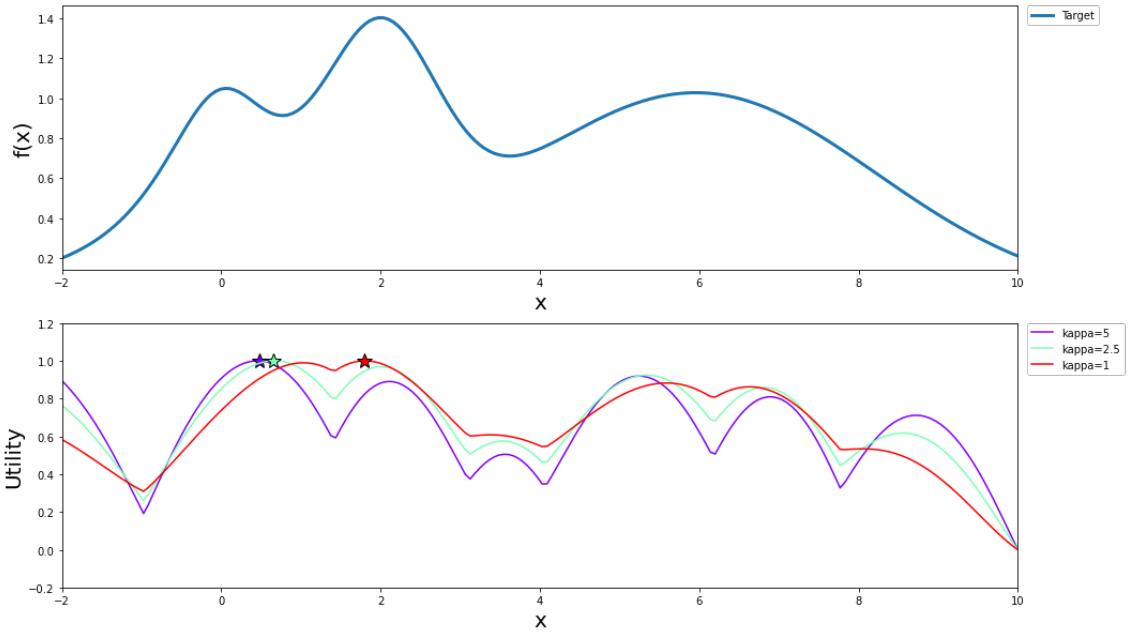

Figure 4: Upper Confidence Bound after 5 iterations and 2 initial points with different parameter values.
Target function: as in Figure 1

4 Knowledge Gradient

The Knowledge Gradient acquisition function is derived by eliminating the assumption made in EI's definition, which uses only previously evaluated points. Hence, the KG suggests a new point which is more uncertain than the points suggested by the other acquisition functions. The fact that KG takes into account more uncertainty makes it particularly suitable for the case of noisy measurements of the objective function.
The analytical expression of the Knowledge Gradient is the following:

$$KG_n(\boldsymbol{x}) := \mathbb{E}_n[\mu^*_{n+1} - \mu^*_n \mid x_{n+1} = x] \tag{11}$$

where

- $\mu_n^* := \max_{\boldsymbol{x'}} \mu_n(\boldsymbol{x'})$, $\mu_n(\boldsymbol{x'})$ being the posterior mean of the Gaussian Process given the observations up to time n
- $\mu_{n+1}^* := \max_{\boldsymbol{x'}} \mu_{n+1}(\boldsymbol{x'})$, $\mu_{n+1}(\boldsymbol{x'})$ being the posterior mean of the Gaussian Process with the new observation at time $n+1$

The difference between KG and EI is that KG is based on the increment in conditional expected value and not in the objective function. Although EI and KG's formulations look similar, coding-wise they are extremely different: since the Expected Improvement is based on the value of the function f in a specific point $\boldsymbol{x}$ and since f is the unknown quantity of interest, we are able to compute the posterior distribution of f and therefore obtain a closed form expression of the EI acquisition function, as show in (8) and (9). This entails that whenever we want to compute the value of the EI acquisition function in a point $\boldsymbol{x}$, we simply need to evaluate formula (9) in $\boldsymbol{x}$. On the contrary, there is no closed form expression for the Knowledge Gradient acquisition function: this means that each time we wish to compute $KG_n(\boldsymbol{x})$, we need to compute the maximum of the mean of the current gaussian process, update it with the new point $\boldsymbol{x}$, compute again its mean and maximize it. Then we need to approximate (using a Monte Carlo technique) the integral of $\mu_{n+1}^* - \mu_n^*$ with respect to the current posterior distribution, as shown in (11).
In particular this means that the augmentation of the conditional expected solution value $(\mu_{n+1}^* - \mu_n^*)$ is unknown before we sample at $\boldsymbol{x}_{n+1}$, but it is possible to compute its expected value given the observations up to time n.

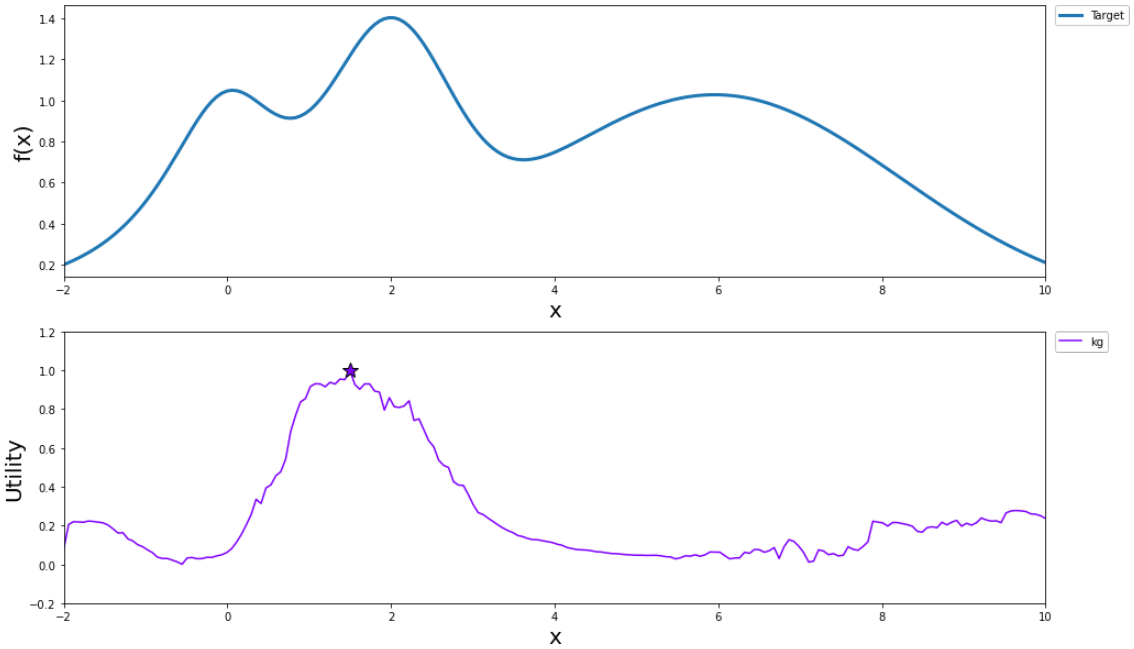

Figure 5: Knowledge gradient after 3 iterations and 5 initial points.
Target function: as in Figure 1

4.1 Knowledge Gradient algorithm

Firstly, we have tried to implement the KG acquisition function in Python via "vanilla" Monte Carlo estimation. In this way the computation was conceptually simple but it took too much time to execute.
Therefore, in order to speed up the iterations, we have implemented the KG using the ad hoc Stochastic Gradient Ascent algorithm, as described by Frazier [3].
In particular, we have implemented the following algorithms:

Algorithm 1 Computation of KG via simulation

Let $\mu_n^* = \max_{\boldsymbol{x'}} \mu_n(\boldsymbol{x'})$
for j in 1 to J **do**
  Generate $y_{n+1} \sim \mathcal{N}(\mu_n(\boldsymbol{x}), \sigma_n(\boldsymbol{x}))$
  Calculate $\mu_{n+1}(\boldsymbol{x'}; \boldsymbol{x}, \boldsymbol{y_{n+1}})$ by refitting the Gaussian Process
  $\mu_{n+1}^* = \max_{\boldsymbol{x'}} \mu_{n+1}(\boldsymbol{x'}; \boldsymbol{x}, \boldsymbol{y_{n+1}})$
  $\Delta_j = \mu_{n+1}^* - \mu_n^*$
end for
Estimate $KG_n(\boldsymbol{x})$ by $\frac{1}{J}\sum_{j=1}^{J} \Delta_j$

In the first algorithm the KG acquisition function is estimated via simulation: initially, the posterior mean of the Gaussian process is maximized over a multidimensional grid of points and, for every iteration, a new observation y_{n+1} is generated from a Gaussian distribution with mean and standard deviation equal to the posterior mean and to the posterior standard deviation of the Gaussian process, both evaluated in the new point $\boldsymbol{x}$. Then, the new posterior mean is computed by refitting the Gaussian process with the new observation $(\boldsymbol{x}, y_{n+1})$ and it is maximized over the grid of points. Finally the difference between the two expected values is computed and the KG is estimated with the average of these differences, as in a standard "vanilla" Monte Carlo framework.

Algorithm 2 Computation of ∇KG via simulation

for j in 1 to J **do**
  Generate $y_{n+1} \sim \mathcal{N}(\mu_n(\boldsymbol{x}), \sigma_n(\boldsymbol{x}))$
  Calculate $\mu_{n+1}(\boldsymbol{x'}; \boldsymbol{x}, \boldsymbol{y_{n+1}})$ by refitting the Gaussian Process
  Solve $\max_{\boldsymbol{x'}} \mu_{n+1}(\boldsymbol{x'}; \boldsymbol{x}, \boldsymbol{y_{n+1}})$ and let x^* be the maximizing x'
  Let G_j be the gradient of $\mu_{n+1}(\boldsymbol{x^*}; \boldsymbol{x}, \boldsymbol{y_{n+1}})$ with respect to x, holding x^* fixed
end for
Estimate $\nabla KG_n(\boldsymbol{x})$ by $\frac{1}{J}\sum_{j=1}^{J} G_j$

The second algorithm is based on the envelope theorem (Milgrom and Segal, 2002), which states that (under sufficient regularity conditions) the gradient with respect to $\boldsymbol{x}$ of a maximum of a collection of functions in $\boldsymbol{x}$ is given simply by first finding the maximum in this collection, and then differentiating this single function with respect to $\boldsymbol{x}$.

In our case the functions involved are $\mu_{n+1}(\boldsymbol{x'}; \boldsymbol{x}, \mu_n(\boldsymbol{x}) + \sigma_n(\boldsymbol{x})Z)$, where Z is a standard normal variable, so this idea is realized in the following equality:

$$\nabla \max_{\boldsymbol{x'}} \mu_{n+1}(\boldsymbol{x'}; \boldsymbol{x}, \mu_n(\boldsymbol{x}) + \sigma_n(\boldsymbol{x})Z) = \nabla \mu_{n+1}(\boldsymbol{x^*}; \boldsymbol{x}, \mu_n(\boldsymbol{x}) + \sigma_n(\boldsymbol{x})Z) \tag{12}$$

where $\boldsymbol{x^*}$ is assumed to be the point in which the maximum is reached and the gradient is estimated using a Monte Carlo method.

Algorithm 3 Optimization of KG based on multistart stochastic gradient ascent

for r in 1 to R **do**
  Choose $\boldsymbol{x}_0$ at random using the Latin Hypercube sampling algorithm
  for t in 1 to T **do**
    Let G be the estimate of $\nabla KG_n(\boldsymbol{x}^r_{t-1})$ from algorithm 2
    $\alpha_t = a/(a+t)$
    $\boldsymbol{x}^r_t = \boldsymbol{x}^r_{t-1} + \alpha_t G$
    if $\boldsymbol{x}^r_t$ is within the optimization bounds **then**
      $\boldsymbol{x}^r_{t-1} = \boldsymbol{x}^r_t$
    end if
  end for
  Estimate $KG_n(\boldsymbol{x}^r_T)$ using algorithm 1 and J replications
end for
return $\boldsymbol{x}^r_t$ with the largest estimated value of $KG_n(\boldsymbol{x}^r_T)$

The last algorithm describes an efficient approach for finding $\boldsymbol{x}$ with the largest $KG_n(\boldsymbol{x})$, based on Multistart Stochastic Gradient Ascent.
The algorithm iterates over starting points, indexed by r, and for each one of them constructs a sequence of iterates $\boldsymbol{x}^r_t$, indexed by t, that attempts at converging to the global maximum of the KG using a Gradient Ascent approach. The reason why multiple starting points are needed, is that there is no guarantee that the Stochastic Gradient Ascent converges to the true global maximum for any starting point.
In order to choose multidimensional initial guesses for the $\text{argmax}_{\boldsymbol{x}}\{KG_n(\boldsymbol{x})\}$, we have implemented the Latin Hypercube Design algorithm, so that the starting points could allow for a high degree of spatial exploration (Figure 6).
The next iterate is obtained by taking a step in the direction of the stochastic gradient G. The size of this step is determined by the magnitude of G and a decreasing step size α_t. Once Stochastic Gradient Ascent has run for T iterations for each starting point, the algorithm uses MC simulations (Algorithm 1) to evaluate the KG in each of the R final point obtained $\{\boldsymbol{x}^1_T, \boldsymbol{x}^2_T, ..., \boldsymbol{x}^R_T\}$, and selects the one which achieves the maximum value of the Knowledge Gradient.
In our implementation we set a low value for most of the parameters (e.g. the number of runs and the number of initial guesses of the Gradient Ascent algorithm) to speed up the computation.
We remark that we implemented all the aforementioned algorithms in a way that allows them to work for any dimension of the optimisation space. Finally, we note that the Gradient Ascent algorithm is more efficient than a simple grid search to find the maximum of the Knowledge Gradient, especially in higher dimensions. Indeed, we noticed that in these circumstances, because of the curse of dimensionality, not

only a grid search is expensive, but the quality of the maximum found with this approach is not satisfactory, even if the grid is quite dense.

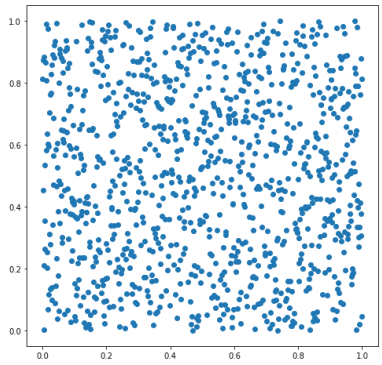

Figure 6: 1000 random points generated by our Latin Hypercube sampling in $[0, 1]^2$.

5 Case studies

We tested the performance of our model on different objective functions and we measured the convergence using the concept of *regret*.

5.1 Simple regret and convergence

Since we are dealing with a black-box optimization approach, in a real world scenario it is difficult to assess whether the algorithm has reached convergence. Indeed the objective function itself is unknown and so is its maximum. In particular, the optimal acquisition function is defined as the one which reaches first the closest point to the actual argmax of the objective function. Indeed the advantage of a Bayesian optimization approach is that the number of evaluation of the objective function needed is generally low. To measure the optimality of an acquisition function, we define as *regret* a function of the number of iteration, which represents the difference between the true maximum of the test function (which in our case studies was assumed to be known, for obvious reasons) and the maximum value of the test function, reached using all the points that the acquisition function under study has proposed, up to the selected iteration. The analytic formula is:

$$r(k) = \max_{\boldsymbol{x} \in A} \{f(\boldsymbol{x})\} - \max_{t \in [1,k]} \{f(\boldsymbol{x}_t)\} \tag{13}$$

where $\{\boldsymbol{x}_t\}_{t=1}^{k}$ are all the points that have been suggested by the acqusition function up to iteration k

5.2 Noisy measurements

So far we have assumed to have perfectly noise-free observations. In a real life scenario, this is rarely possible and instead of observing $f(\boldsymbol{x})$, we can often only

observe a noisy transformation of it.
However GP regression can be extended naturally to observations with independent normally distributed noise of known variance, that is the simplest transformation of $f(\boldsymbol{x})$ we could have. Indeed, in this settings:

$$y_i \sim f(\boldsymbol{x}_i) + \epsilon_i \quad \forall i = 1, .., n, \tag{14}$$

where

$$\boldsymbol{\epsilon} \sim \mathcal{N}_n(\mathbf{0}, \sigma^2_{noise}\mathbb{I}) \tag{15}$$

5.3 A toy example in 1D

Firstly we started our investigation with a one-dimensional test function as:

$$f(x) = e^{-(x-2)^2} + e^{-\frac{(x-6)^2}{10}} + \frac{1}{x^2+1} \quad \text{with} \quad x \in (-2, 10) \tag{16}$$

In order to have a general idea of the framework in which we are working on, we have plotted the target function and some iterations of the different acquisition functions:

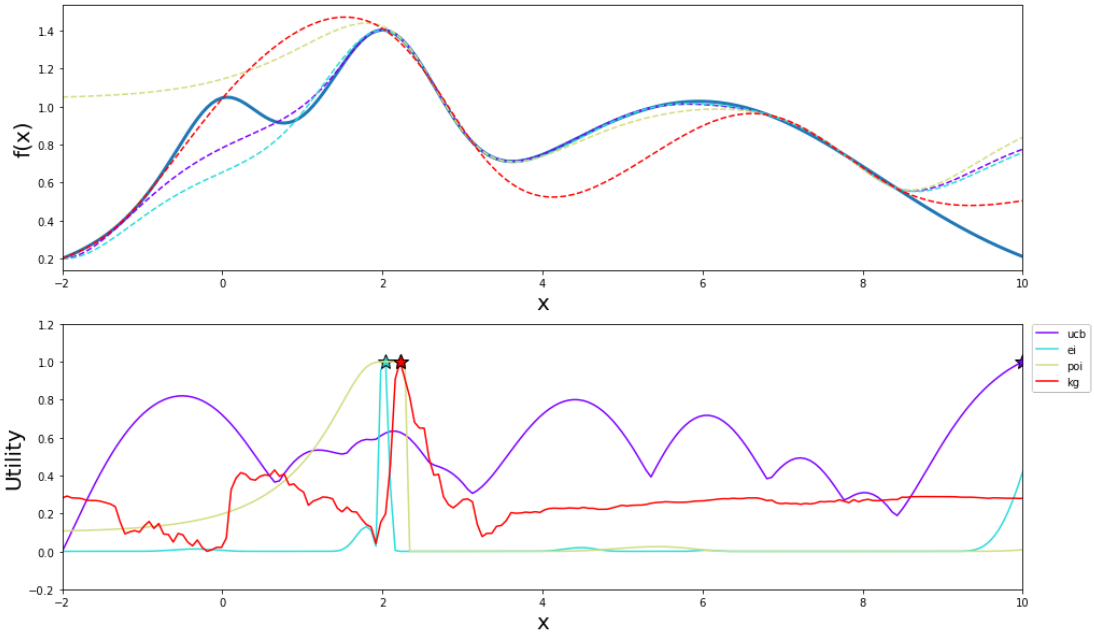

Figure 7: Target function and acquisition functions after 10 iterations and 2 initial points. Optimum to be achieved: $x = 2$

In Figure 8 we can notice that the Knowledge Gradient converges to the optimum after 10 iterations, then it moves away. This behaviour is not unexpected because this particular acquisition function doesn't perform very well in only one dimension.

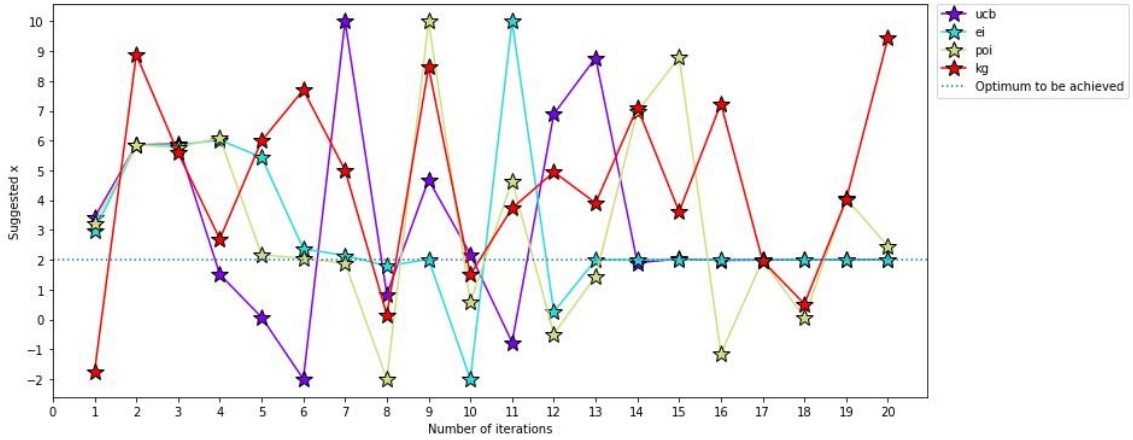

Figure 8: Convergence of the acquisition functions to the optimum in $x = 2$. In the x axis we plot the iteration number and in the y axis the position of the suggested point.

However the convergence of the Knowledge Gradient is still satisfying as we can see in Figure 9, which shows that KG proposes relatively quickly a point near the true argmax of the objective function, then tries to explore the space and therefore deviates from the true maximum. Still, KG has successfully proposed a good point at the sixth iteration.

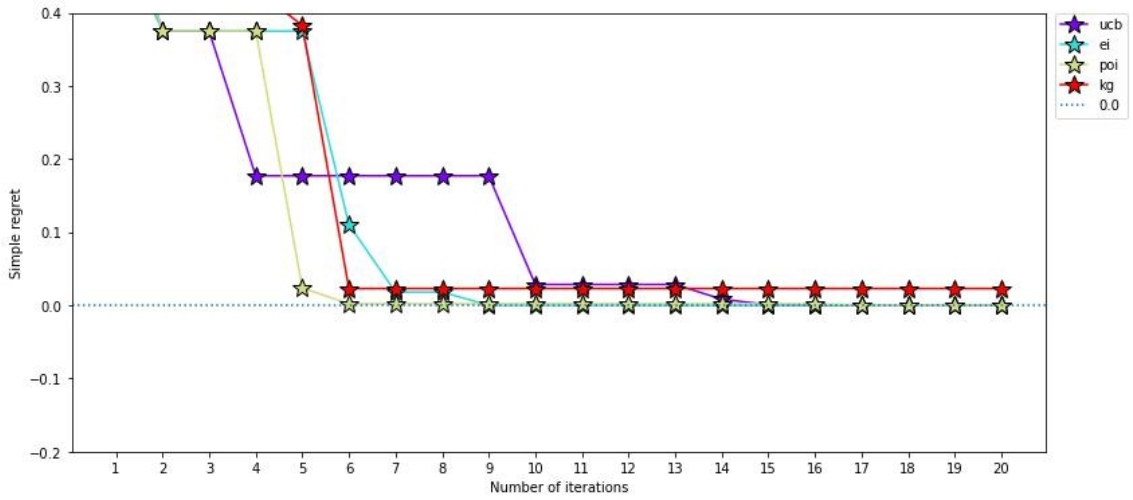

Figure 9: Plot of the regret of the acquisition functions.

5.4 Rosenbrock's valley in 2D

Then we have continued using some famous target function as the Rosenbrock's valley, with the following analytical expression:

$$f(x, y) = 10(y - x^2)^2 + (1 - x)^2 + \epsilon \quad \text{with} \quad (x, y) \in [-2, 2] \times [-1, 3] \tag{17}$$

where $\epsilon \sim \mathcal{N}(0, 0.5^2)$.
This function has a global minimum in $(0, 0)$, since we deal with maximization problem, we take its opposite.

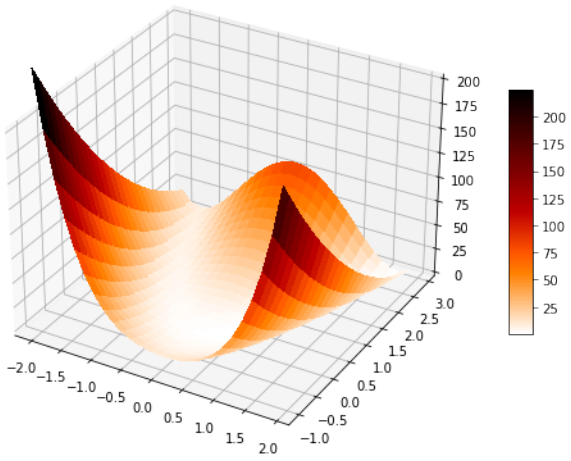

Figure 10: Rosenbrock's function in 2D

We can see in the following figures that with this target function EI, UCB and KG perform quite well in both cases without noise (Figure 11) and with noise (Figure 12), while POI remains far from the optimum value.

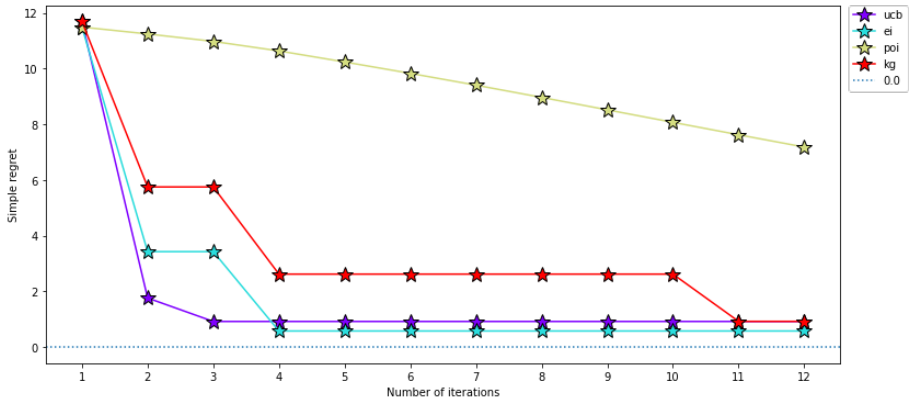

Figure 11: Simple regret of Rosenbrock after 12 iterations and 5 initial points

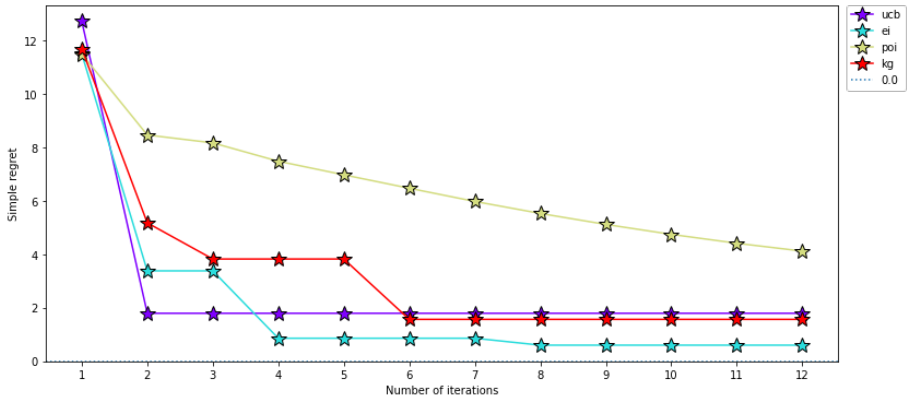

Figure 12: Simple regret of Rosenbrock with noise after 12 iterations and 5 initial points

5.5 Ackley's function in 2D with noise

The analytical expression for Ackley's function with noise is:

$$f(x,y) = -a\ e^{-b\ \sqrt{\frac{x^2+y^2}{2}}} - e^{\frac{\cos(cx)+\cos(cy)}{2}} + a + e + \epsilon$$

with $(x,y) \in [-32.728, 32.728] \times [-32.728, 32.728]$ and $\epsilon \sim \mathcal{N}(0, 0.01^2)$ is the gaussian noise.[1]. The global maximum of this function is in $(0,0)$.

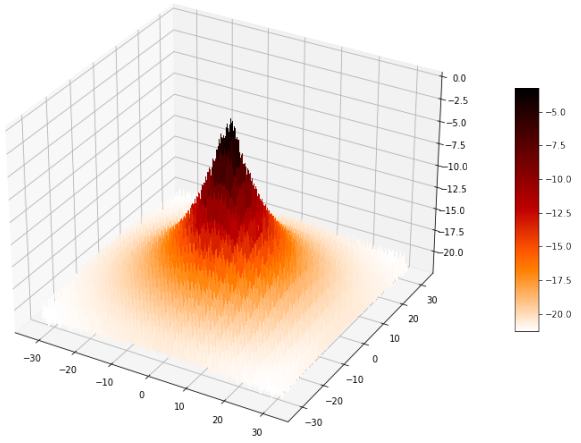

Figure 13: Ackley's function in 2D

From the plot of the simple regret (Figure 14), we can clearly see that the Knowledge Gradient acquisition function greatly outperforms POI: we can say that after just 7 iterations, KG has reached a satisfactory result.

[1]In this case to find the maximum of the Knowledge Gradient we used a grid search algorithm instead of the Gradient Ascent.

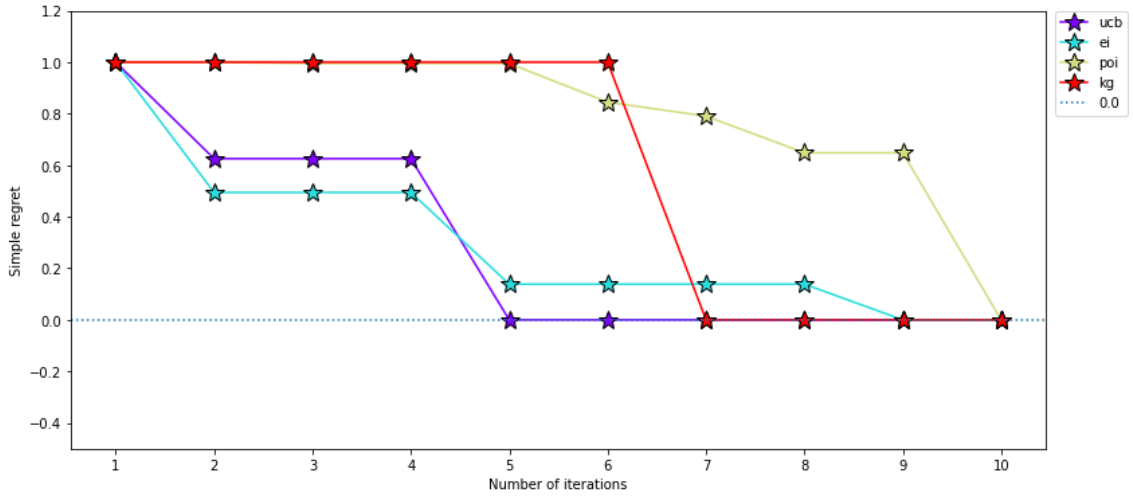

Figure 14: Plot of the regret of Ackley's function with Gaussian noise with standard deviation = 0.01

We tried increasing the standard deviation of the noise to 0.5 and 1.5. Figure 15 and 16 show that even though the performance of UCB, EI and POI becomes worse and worse, KG's results are consistently quite satisfactory.

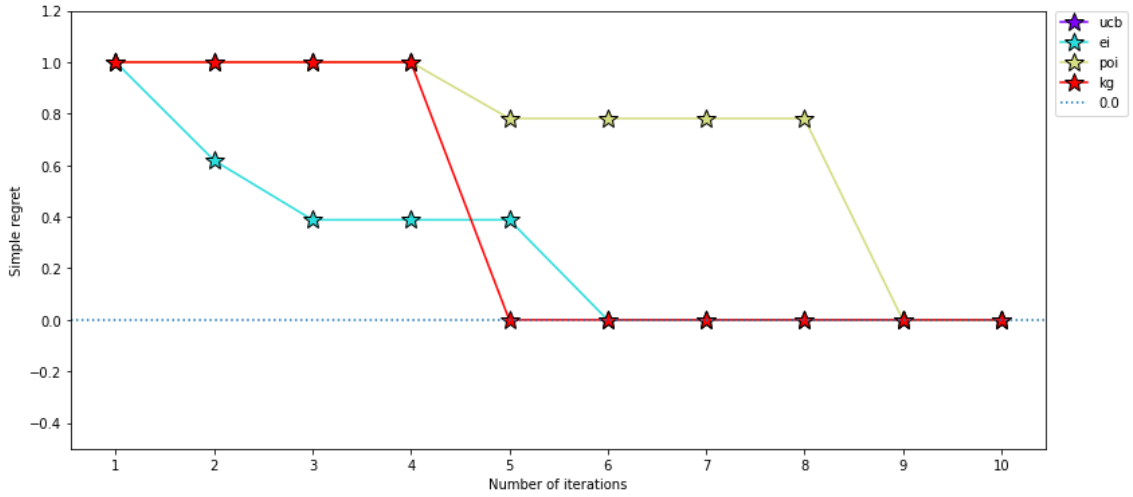

Figure 15: Plot of the regret of Ackley's functionv with Gaussian noise with standard deviation = 0.5. The plot of UCB is missing, as it hasn't reached convergence yet.

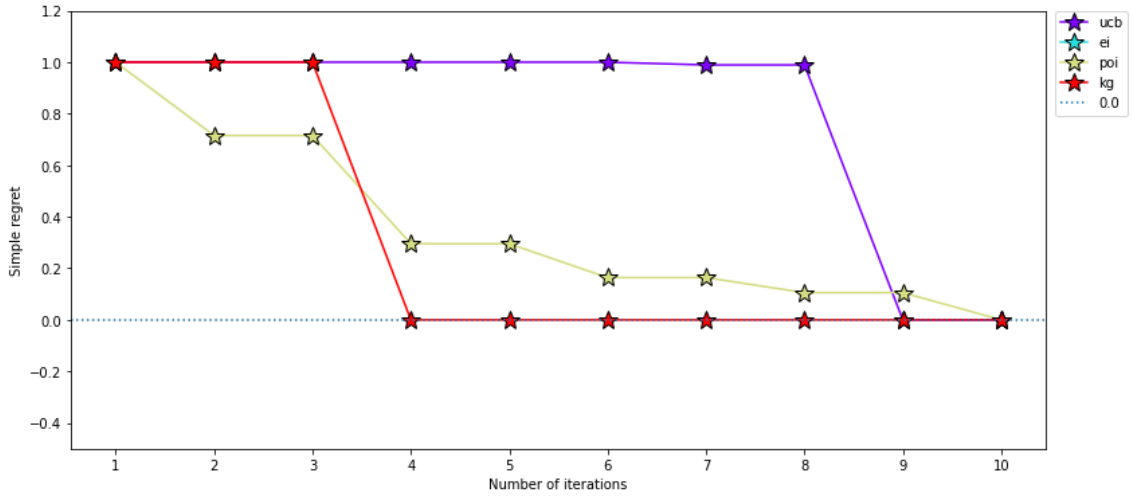

Figure 16: Plot of the regret of Ackley's function with Gaussian noise with standard deviation = 1.5. The plot of EI is missing, as it hasn't reached convergence yet.

6 Conclusion and Further Developments

Our contribution to the open-source library of Bayesian Optimization by Fernando Nogueira [1] is especially helpful when dealing with functions in higher dimensions, or when the presence of noise is significant. Indeed the computational cost is quite relevant, so the trade-off between the great performances and the cost is well balanced especially when working with complex functions. To tackle the computational issue, a possible solution would be switching to C++ and possibly adopting parallel computing to optimize specific tasks.
We emphasize that our improvements to the existing library perform well even in presence of Gaussian noise, which, as explained in [7] and in [1], is an exciting area of research.

Bibliography

[1] Fernando Nogueira, *Bayesian Optimization: Open source constrained global optimization tool for Python*
Github link: https://github.com/fmfn/BayesianOptimization

[2] Eric Brochu, Vlad M. Cora and Nando de Freitas, *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*, 2010;

[3] Peter I. Frazier, *A Tutorial on Bayesian Optimization*, 2018;

[4] Kawaguchi, Kenji, Leslie Pack Kaelbling, and Tomás Lozano-Pérez, *Bayesian Optimization with Exponential Convergence* (Advances in Neural Information Processing Systems 28), 2015;

[5] Candelieri Antonio, *Sequential model based optimization of partially defined functions under unknown constraints*, 2019;

[6] Marcin Molga, Czesław Smutnicki, *Test functions for optimization needs*, 2005;

[7] Jian Wu, Peter I. Frazier, *The Parallel Knowledge Gradient Method for Batch Bayesian Optimization*, 2016;

[8] Our Bayesian Optimization library extension:
Github link: https://github.com/MartinaGaravaglia/BayesianOptimization