Bayesian Optimization

A comparative study on acquisition functions

Raffaella D'Anna, Michele Di Sabato,
Martina Garavaglia, Anna Iob, Veronica Mazzola
Supervisor: Bruno Guindani

February 15, 2022

Outline

A brief summary

OUR PROBLEM

We want to solve black box optimization problems such as:

$$\text{find } x^* \text{ s.t. } f(x^*) = \max_{\boldsymbol{x} \in A \subset \mathbb{R}^d} f(\boldsymbol{x})$$

Bayesian Optimization requires:

- **Gaussian process regression** as the prior distribution for the objective function f
- **Acquisition function** to decide where to evaluate the objective function

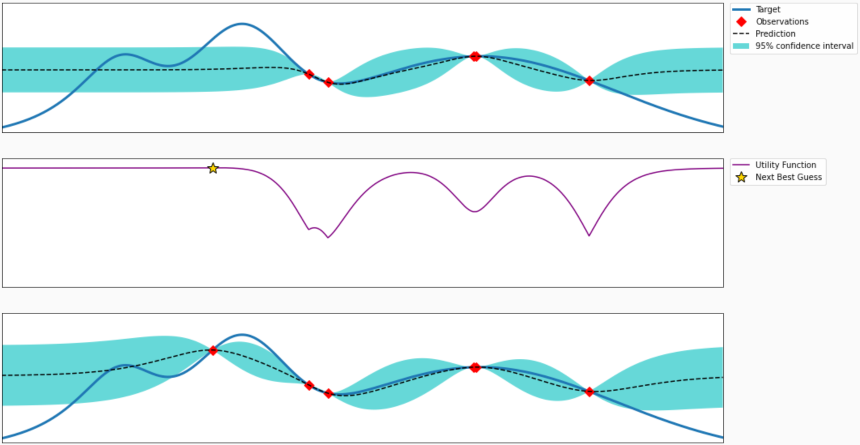
$$\boldsymbol{x}_{n+1} = \arg\max_{\boldsymbol{x} \in A} u(\boldsymbol{x} | \mathcal{D}_{1:n})$$

PSEUDO CODE

for $n = 1, 2, 3, ..., N$:

1. Find the new point $\boldsymbol{x}_{n+1}$ by maximizing the function u, called acquisition function: $\boldsymbol{x}_{n+1} = \arg\max_{\boldsymbol{x} \in A} u(\boldsymbol{x}|\mathcal{D}_{1:n})$
2. Evaluate the objective function f in the new point $\boldsymbol{x}_{n+1}$ and set $f_{n+1} := f(\boldsymbol{x}_{n+1})$
3. Augment the dataset: $\mathcal{D}_{1:n+1} = \{\mathcal{D}_{1:n}, (\boldsymbol{x}_{n+1}, f_{n+1})\}$.
4. Compute the posterior mean $\mu(\cdot|\mathcal{D}_{1:n+1})$ and posterior variance $\sigma^2(\cdot|\mathcal{D}_{1:n+1})$

EXAMPLE


Target
Observations
Prediction
95% confidence interval
Utility Function
Next Best Guess

Goal

The goal of our project is:

Conducting a comparative analysis of the performance of different acquisition functions applied to a variety of objective functions

Library and acquisition functions

BayesianOptimization library

- We started from the **BayesianOptimization** library by Fernando Nogueira, built in 2014
- It gives us a pure Python implementation of Bayesian global Optimization with Gaussian Processes and some known acquisition functions:
  - **PoI**: Probability of Improvement
  - **EI**: Expected Improvement
  - **UCB**: Upper Confidence Bound

Acquisition Functions

Setting $x_n^+ = \underset{i \in [1,n]}{\arg\max} f(x_i)$:

- **PoI**:

$$PI(x) = P(f(x) \geq f_n^* + \xi) = \Phi\left(\frac{\mu(x) - f_n^* - \xi}{\sigma(x)}\right) \tag{1}$$

where Φ is the normal cumulative distribution function and ξ is the trade-off parameter

- **EI**:

$$EI_n(x) := \mathbb{E}_n[\max\{0,\ f(x) - f(x_n^+)\}] \tag{2}$$

- **UCB**:

$$UCB(x) = \mu(x) + k\sigma(x) \tag{3}$$

where k is a trade-off parameter.

Improvement

We have decided to improve the library adding:

- **Knowledge gradient** acquisition function
- Some **plot functions** to compare the acquisition functions and their convergence to the optimum

Knowledge Gradient

Knowledge Gradient

$$EI_n(\boldsymbol{x}) := \mathbb{E}_n[\max\{0, f(\boldsymbol{x}) - f(\boldsymbol{x}_n^+)\}]$$
$$KG_n(\boldsymbol{x}) := \mathbb{E}_n[\mu_{n+1}^* - \mu_n^* \mid \boldsymbol{x}_{n+1} = \boldsymbol{x}]$$

where

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

Knowledge Gradient

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

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

where $\Delta_n(x)$:=$\mu_n(x)$-f^*_n is the expected difference in quality between the proposed point x and the previous best

- Initially, we tried to maximize the KG using a naive **grid search** approach, but it turned out to be too computationally expensive, because of the curse of dimensionality and because at each point a MC estimation of the integral needs to be performed.
- In order to speed up the iterations, we switched to the ad hoc **multi-start Stochastic Gradient Ascent** algorithm.

PSEUDO CODE

Algorithm 1: estimate $KG_n(x)$

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

Pseudo code

Algorithm 1: estimate $KG_n(x)$

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

Algorithm 2: estimate $\nabla KG_n(x)$

for j in 1 to J **do**
  Generate $y_{n+1} \sim \mathcal{N}(\mu_n(x), \sigma_n(x))$
  Compute $\mu_{n+1}(x'; x, \mathbf{y_{n+1}})$ by refitting the Gaussian Process
  Compute $x^* := \operatorname{argmax}_{x'} \mu_{n+1}(x'; x, y_{n+1})$
  Let G_j be the gradient of $\mu_{n+1}(x^*; x, y_{n+1})$ w.r.t. x, with x^* fixed
end for
Estimate $\nabla KG_n(x)$ by $\frac{1}{J}\sum_{j=1}^{J} G_j$

PSEUDO CODE

Algorithm 1: estimate $KG_n(x)$

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

Estimate $KG_n(x)$ by $\frac{1}{J}\sum_{j=1}^{J} \Delta_j$

Algorithm 2: estimate $\nabla KG_n(x)$

for j in 1 to J **do**
  Generate $y_{n+1} \sim \mathcal{N}(\mu_n(x), \sigma_n(x))$
  Compute $\mu_{n+1}(x'; x, \mathbf{y_{n+1}})$ by refitting the Gaussian Process
  Compute $x^* := \mathrm{argmax}_{x'} \mu_{n+1}(x'; x, y_{n+1})$
  Let G_j be the gradient of $\mu_{n+1}(x^*; x, y_{n+1})$ w.r.t. x, with x^* fixed
end for

Estimate $\nabla KG_n(x)$ by $\frac{1}{J}\sum_{j=1}^{J} G_j$

Algorithm 3: maximize $KG_n(x)$ w.r.t x (SGA)

for r in 1 to R **do**
  Choose x_0 at random using Latin Hypercube Design
  for t in 1 to T **do**
    Let G be the estimate of $\nabla KG_n(x_{t-1}^r)$ from **Algorithm 2**
    $\alpha_t = a/(a+t)$
    $x_t^r = x_{t-1}^r + \alpha_t G$
    Check if x_t^r is within bounds
  end for
  Estimate $KG_n(x_T^r)$ using **Algorithm 1**
end for
return x_t^r with the largest estimated value of $KG_n(x_T^r)$

Case studies

A toy example in 1D - after 9 iterations

$$f(x) = e^{-(x-2)^2} + e^{-\frac{(x-6)^2}{10}} + \frac{1}{x^2+1}, \quad x \in [-2, 10]$$

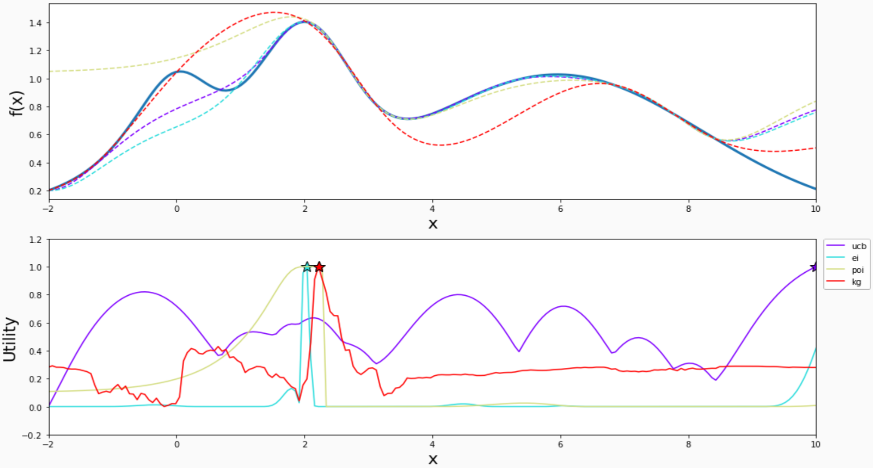


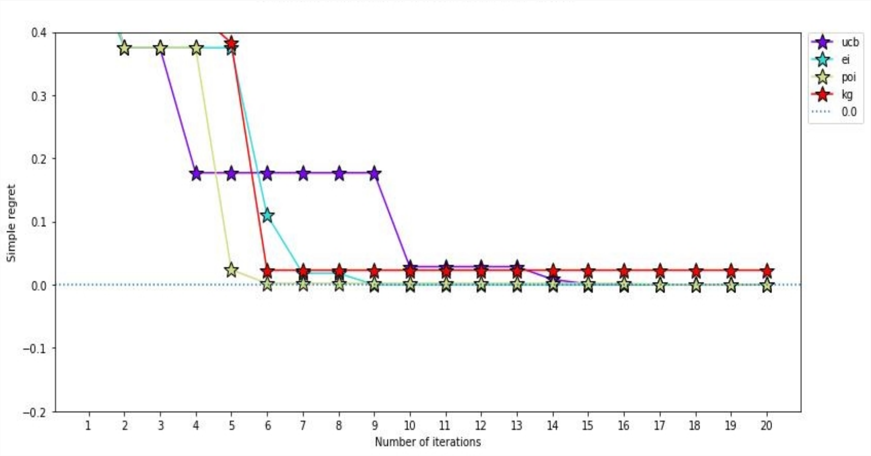
Figure 1: Argmax to be achieved: $x = 2$

Simple Regret

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

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

Regret of a toy example in 1D



2D-Rosenbrock function

$$f(x,y) = 10(y - x^2)^2 + (1-x)^2, \quad x \in [-3,1], y \in [-2,2]$$

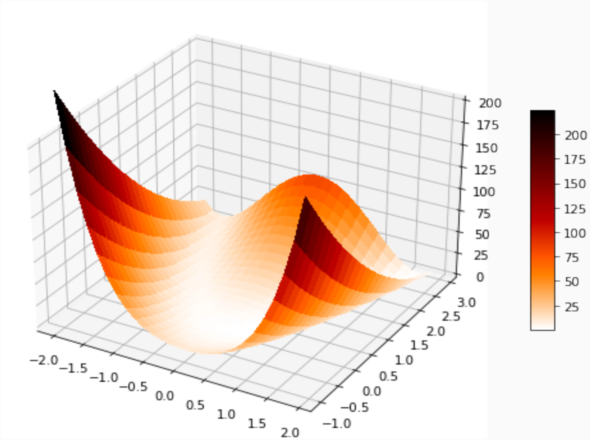
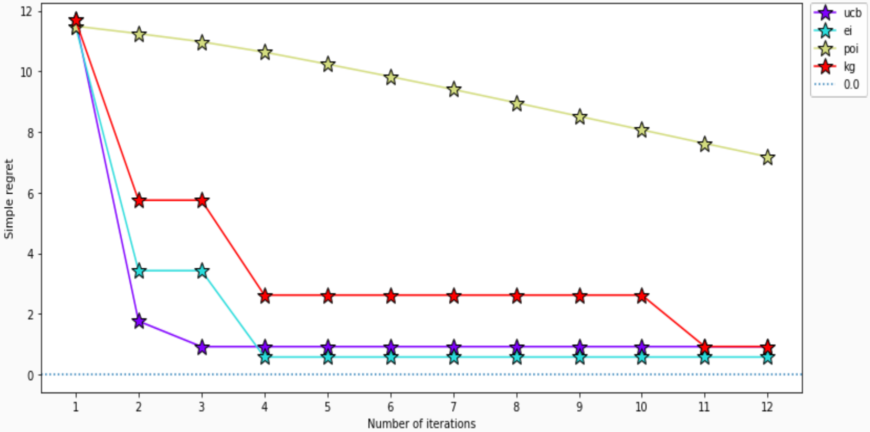


Figure 2: The minimum is 0 and is reached in $(0,0)$

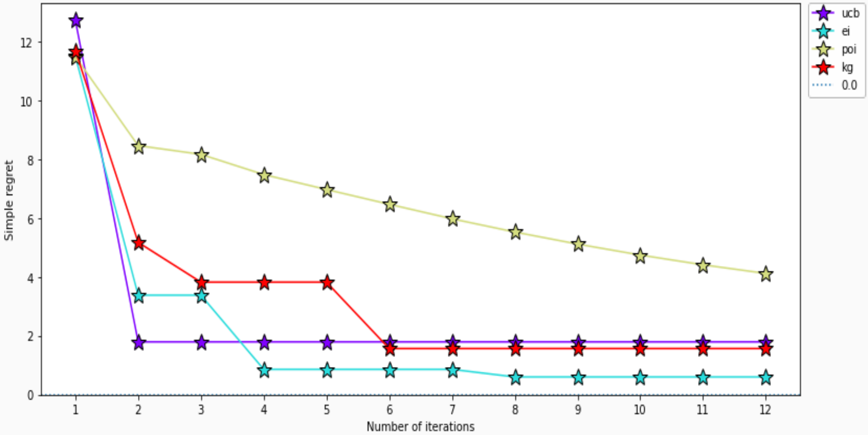
Regret of Rosenbrock function in 2D



Regret of Rosenbrock function with noise in 2D

$$f(x,y) = 10(y-x^2)^2 + (1-x)^2 + \epsilon, \quad x \in [-3,1], y \in [-2,2]$$

where $\epsilon \sim \mathcal{N}(0, 0.5^2)$



2D - Ackley's function

$$f(x,y) = -20\,e^{-0.2\sqrt{\frac{x^2+y^2}{2}}} - e^{\frac{\cos(2\pi x)+\cos(2\pi y)}{2}} + 20 + e, \quad (x,y) \in [-32.7, 32.7]^2$$

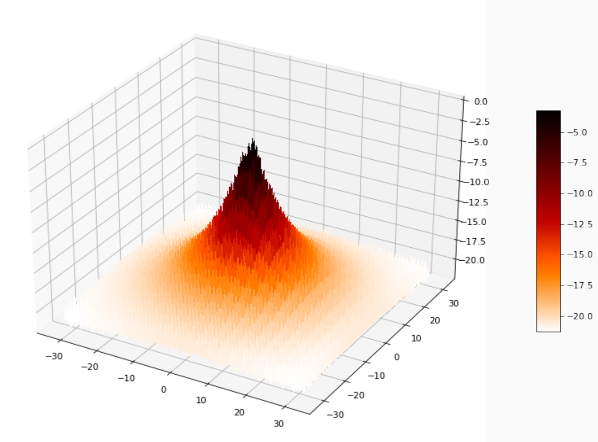
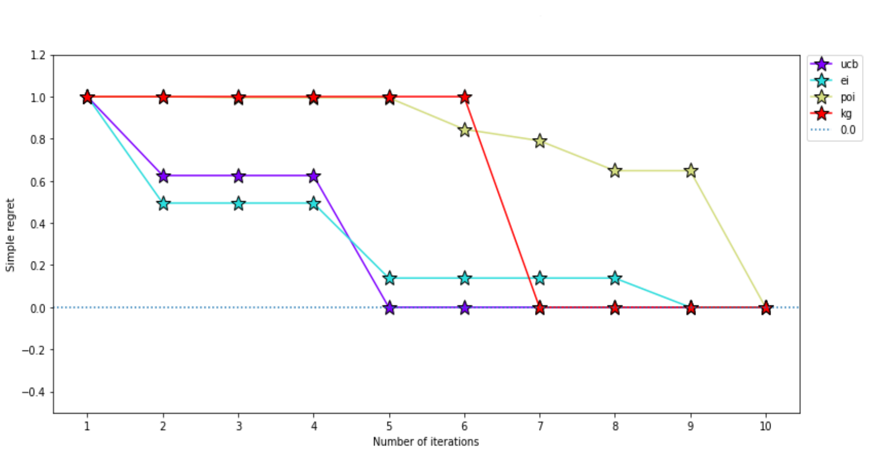


Figure 3: The maximum is 0 and is reached in $(0,0)$

Regret of Ackley's function with noise in 2D

$$f(x,y) = -20\, e^{-0.2\sqrt{\frac{x^2+y^2}{2}}} - e^{\frac{\cos(2\pi x)+\cos(2\pi y)}{2}} + 20 + e + \epsilon, \;\; (x,y) \in [-32.7, 32.7]^2$$

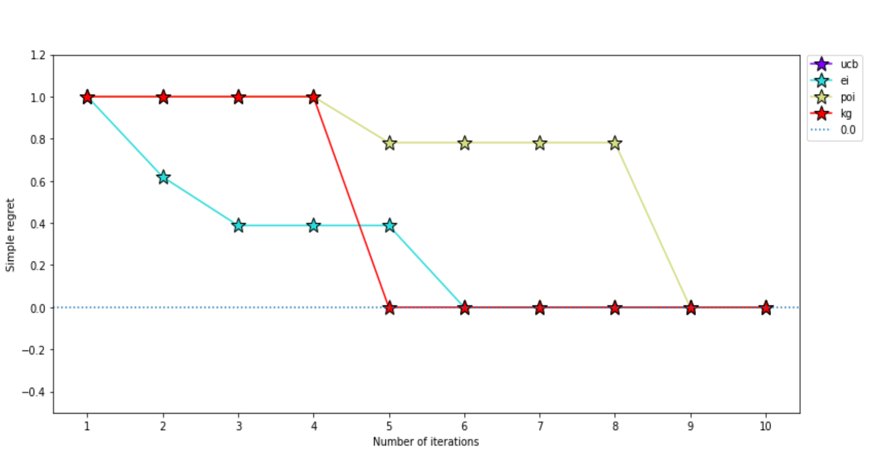
where $\epsilon \sim \mathcal{N}(0, 0.01^2)$



Regret of Ackley's function with noise in 2D

$$f(x,y) = -20\, e^{-0.2\sqrt{\frac{x^2+y^2}{2}}} - e^{\frac{\cos(2\pi x)+\cos(2\pi y)}{2}} + 20 + e + \epsilon, \ \ (x,y) \in [-32.7, 32.7]^2$$

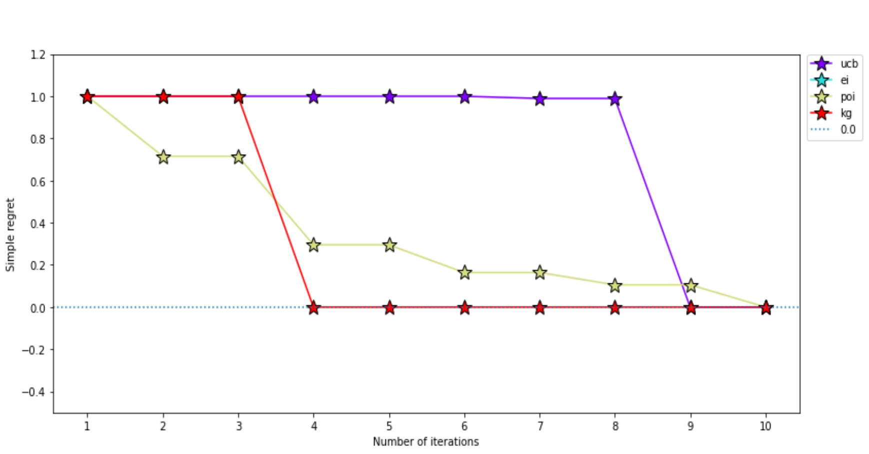
where $\epsilon \sim \mathcal{N}(0, 0.5^2)$



Regret of Ackley's function with noise in 2D

$$f(x,y) = -20\, e^{-0.2\sqrt{\frac{x^2+y^2}{2}}} - e^{\frac{\cos(2\pi x)+\cos(2\pi y)}{2}} + 20 + e + \epsilon,\ \ (x,y) \in [-32.7, 32.7]^2$$

where $\epsilon \sim \mathcal{N}(0, 1.5^2)$



Conclusion

Conclusion and Further Developments

- Our implementation is especially helpful when dealing with functions in higher dimensions and affected to significant noise
- However it bears computational costs that might be reduced using C++ and Parallel Programming

References

- Fernando Nogueira, *Bayesian Optimization: Open source constrained global optimization tool for Python* https://github.com/fmfn/BayesianOptimization
- Eric Brochu, Vlad M. Cora and Nando de Freitas, *A Tutorial on Bayesian Optimization of Expensive Cost Functions*, 2010;
- Peter I. Frazier, *A Tutorial on Bayesian Optimization*, 2018;
- Kawaguchi, Kenji, Leslie Pack Kaelbling, and Tomás Lozano-Pérez, *Bayesian Optimization with Exponential Convergence*, 2015;
- Candelieri Antonio, *Sequential model based optimization of partially defined functions under unknown constraints*, 2019;
- Marcin Molga, Czesław Smutnicki, *Test functions for optimization needs*, 2005;

Appendix

Regret of Rosenbrock function with noise in 2D

$$f(x,y) = 10(y - x^2)^2 + (1 - x)^2 + \epsilon, \quad x \in [-3, 1], y \in [-2, 2]$$

where $\epsilon \sim \mathcal{N}(0, 2^2)$

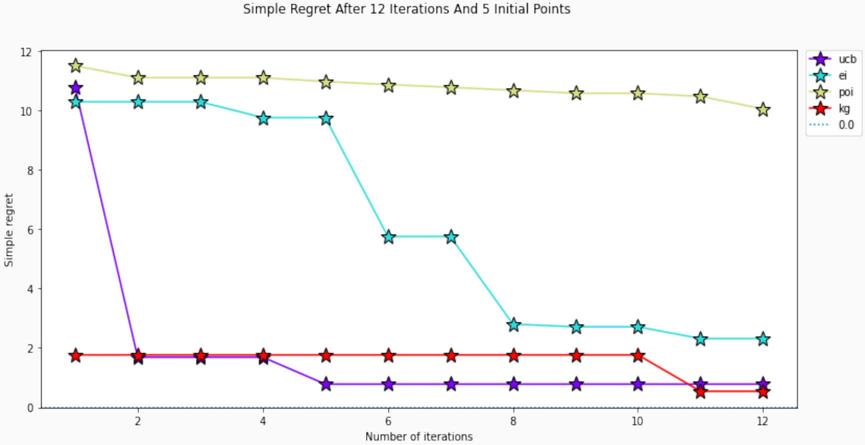


Figure 4: standard deviation of the noise $= 2$

Regret of Rosenbrock function with noise in 2D

$$f(x,y) = 10(y - x^2)^2 + (1-x)^2 + \epsilon, \quad x \in [-3,1], y \in [-2,2]$$
where $\epsilon \sim \mathcal{N}(0, 5^2)$

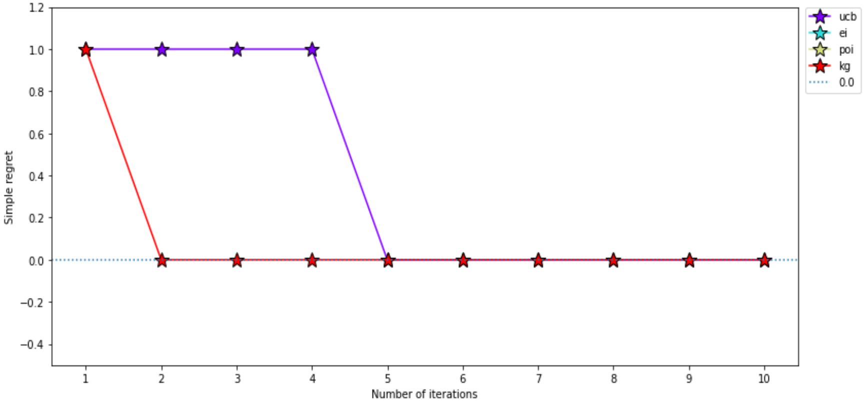


Figure 5: standard deviation of noise $= 5$

Regret of Rosenbrock function with noise in 2D

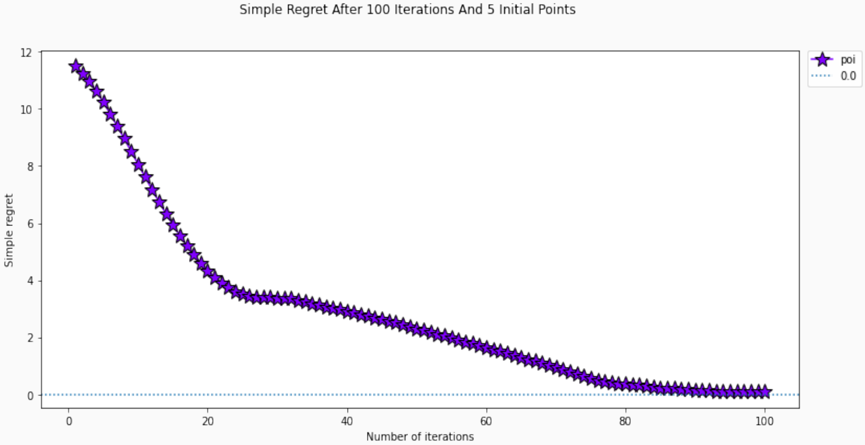


Figure 6: simple regret of POI; standard deviation of noise $= 0.5$