Lab Session 9

CS342 Optimization Method

Junjie Qiu

Department of Statistics and Data Science
Southern University of Science and Technology

2024-05-09

Task 1: Black Box Optimization

Algorithm Design

We use a simple implementation of Bayesian Optimizer to handle black box optimization problems, which makes use of a surrogate model to simulate the behavior of the black box fucntion. Here are descriptions for each API defined in the *bo.python* file which is contained in the attached *code.zip* file.

- **EI**: Calculate the Expected Improvement (EI) acquisition function value for Bayesian optimization.
- **acq_max**: Find the maximum of the acquisition function using random search.
- **bayesian_optimizer**: Perform Bayesian optimization.

Algorithm Design

Our bayesian optimizer that is set with n iterations will conduct $100 + n$ evaluations on the black box fucntion f with the default setting in Figure 1. The κ parameter is used to balance between exploration and exploitation, and we will show how it works later.

```
BAYESIAN_OPTIMIZER(f, n, κ, EI):
 1 initialize X_train and Y_train                    // default: 100
 2 for i in range(n):
 3     model.fit(X_train, Y_train)
 4     X_trails = random(1000)                       // default: 1000
 5     μ, σ = model(X_trails)
 6     while max(EI) < 0
 7         x_best = arg max_x EI(κ, Y_train, μ)
 8     y_best = f(x_best)
 9     add x_best and f(x_best) into train data
10 return min(Y_train)
```

Figure 1: Pseudocode of Bayesian Optimizer.

One Simple Example

Figure 2 shows that with the increasement of observations, the posterior function computed by Gaussian Process will approximate the Black Box Fucntion better with lower uncertainty.

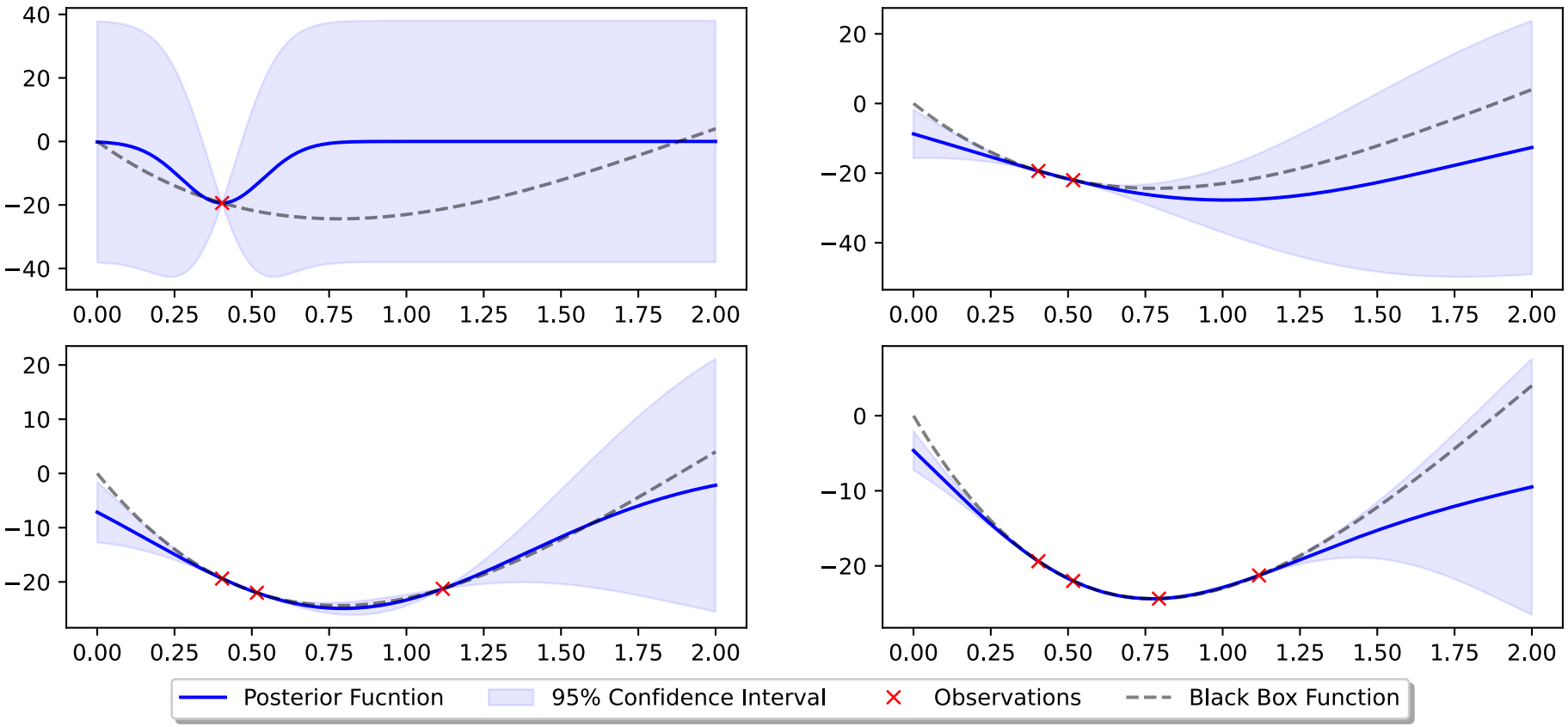

Figure 2: Simple 1D Minimization Example. Each plot shows the behavior of posterior function after adding one observation.

One Simple Example

In addition, the next point to pick is a random but good evaluated by the Expected Increasement (EI) fucntion. Next we will talk about the fucntion of κ in EI.

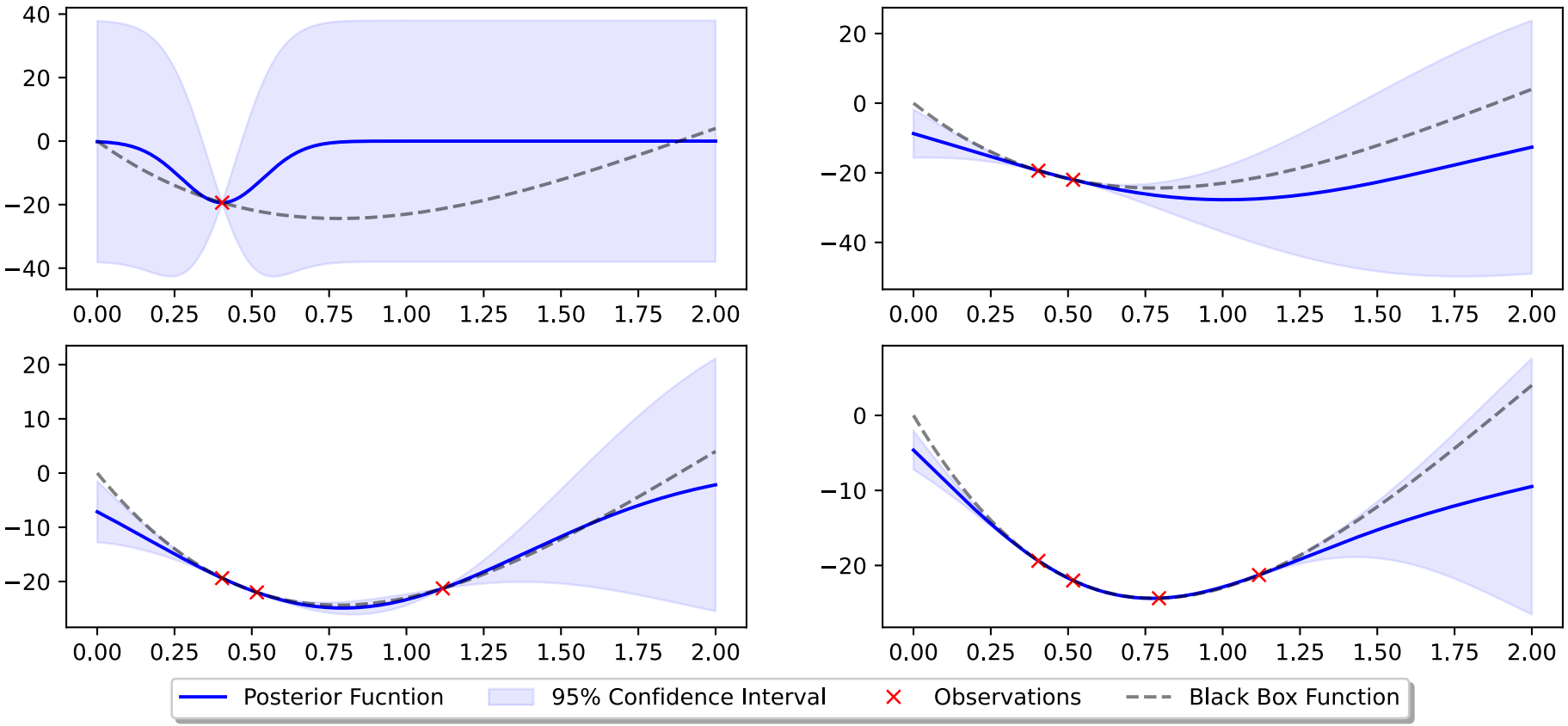

Figure 3: Simple 1D Example. Each plot shows the behavior of posterior function after adding one observation.

One Simple Example

Set initialization size to be 5, and iteration size n to be 5. We only need to evaluate 10 points to get the near-optimal result. Apprerently, this case is easy for Gaussian Process Regression to fit, indicating that Bayesian Optimizer is suitable for few-shots optimization problem.

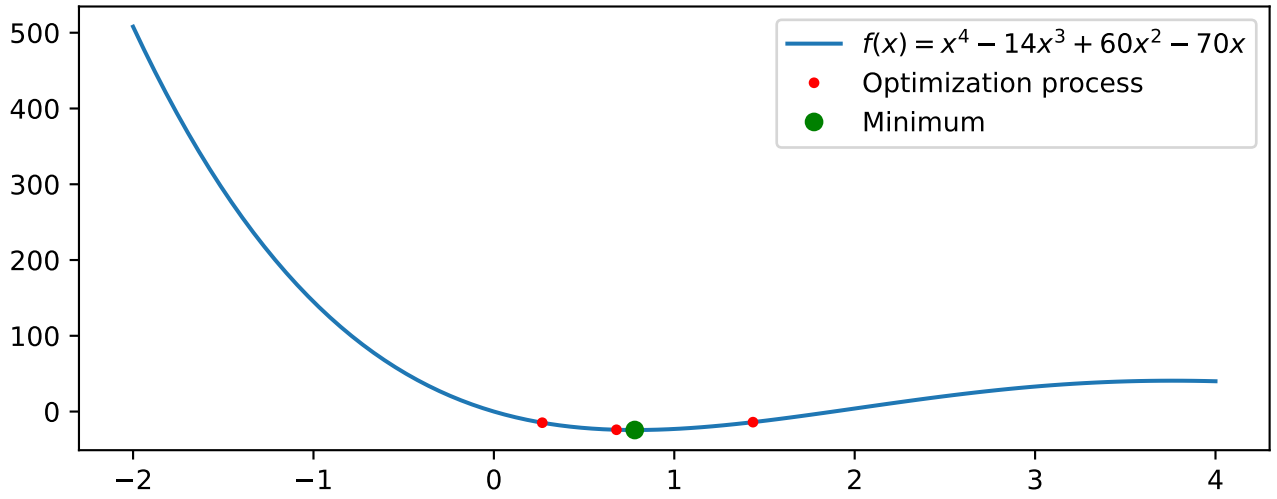

Figure 4: 1-D Case Optimized by Bayesian Optimizer.

EI and κ

For a **maximization problem**, the simplest EI of a newly sampled and unevaluated point y is defined as

$$\mathrm{EI} = E[(y - M_n)\mathbb{I}(y > M_n)]$$

where $M_{i=1}^{n_i} y_i$ is the best value observed so far.

Our implementation adds κ as a hyperparameter to balance between exploration and exploitation,

$$\mathrm{EI}(\kappa) = E[(y - M_n + \kappa)\mathbb{I}(y > M_n - \kappa)]$$

When κ is larger, the Bayesian Optimizer tends to explore more in points with poor estimation on surrogate model; vice versa, the Bayesian Optimizer will exploit the current optimum when κ is small.

2 Dimensional Example

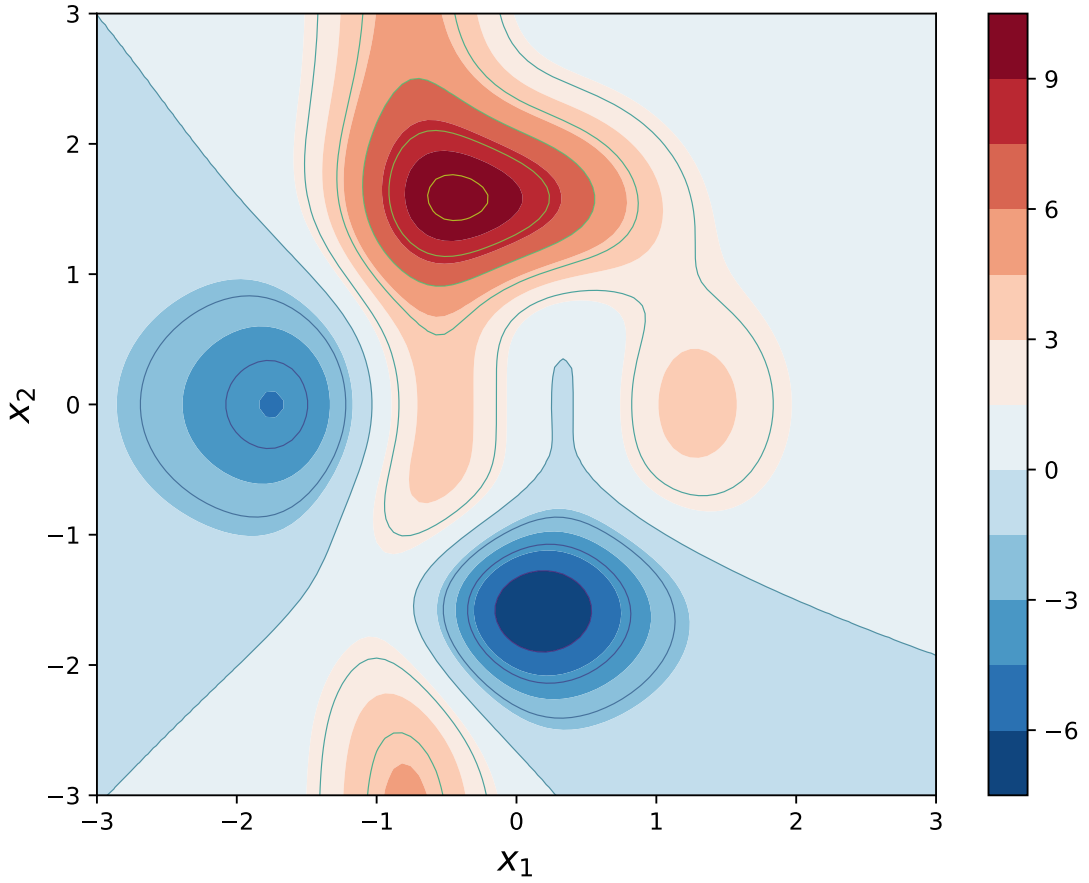

Figure 5: 2 Dimensiaonl Exmaple.

The plot of $3 \cdot (1-x)^2 \cdot e^{-x^2-(x+1)^2} - 10 \cdot \left(\frac{x}{5} - x^3 - y^5\right) \cdot e^{-x^2-y^2} - 3 \cdot e^{-(x+2)^2-y^2}$ is shown in Figure 5; this will be used as the 2-D problem in following contexts.

2 Dimensional Example

By applying the default setting shown in Figure 1 with 10 iterations, we only budget 110 evaluations and the result is near-perfect.

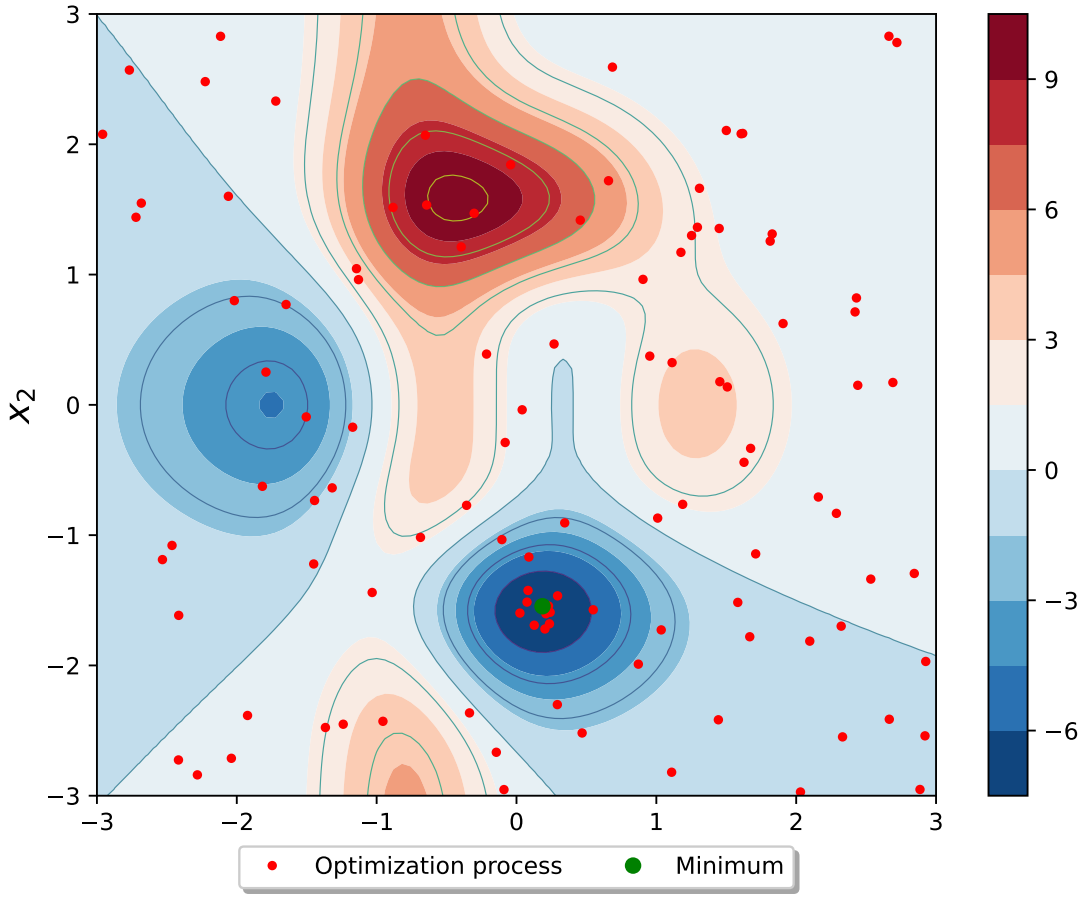

Figure 6: 2-D Case Optimized by Bayesian Optimizer.

Comparsion with SA and GA

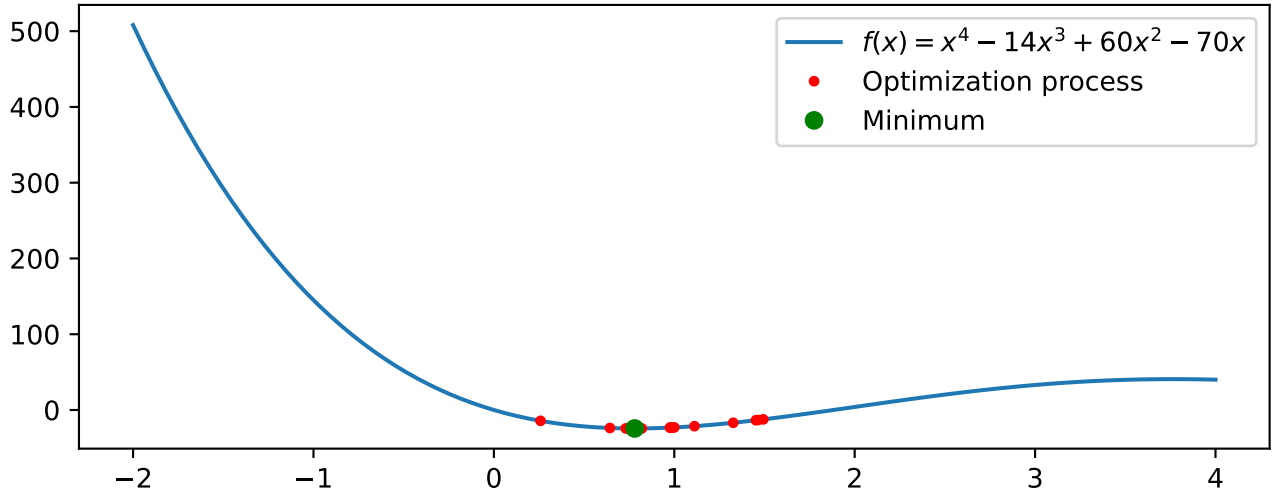

Figure 7: 1-D Case Optimized by Simulated Annealing (SA).

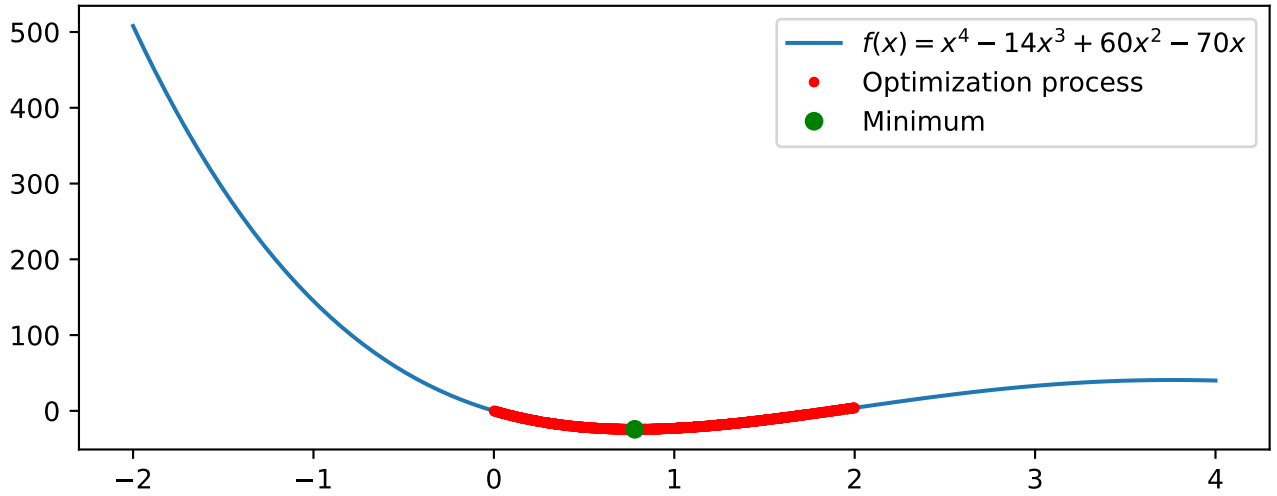

Figure 8: 1-D Case Optimized by Genetic Algorithms (GA).

Comparsion with SA and GA

Simulated Annealing (SA): Figure 7 shows the 1-D case optimized by simulated annealing after 1000 evaluations. The temperature function is 100×0.8^n, thus a lot of those points which are evaluated behind the first few ones are clustered around minimum.

Genetic Algorithms (GA): Figure 8 shows the 1-D case optimized by genetic algorithm after 1000 evaluations (20 points a generation with 98 gernerations). The evolution strategy includes single-point crossover and mutation by normal distribution sampling.

Conclusion: 1-D case is too easy for SA, GA, and Bayesian Optimizer. We need a more complex case to differentiate their performances.

Comparsion with SA and GA

With the same parameter setting to 1-D case, we leverage SA to optimize the aforementioned 2-D case.

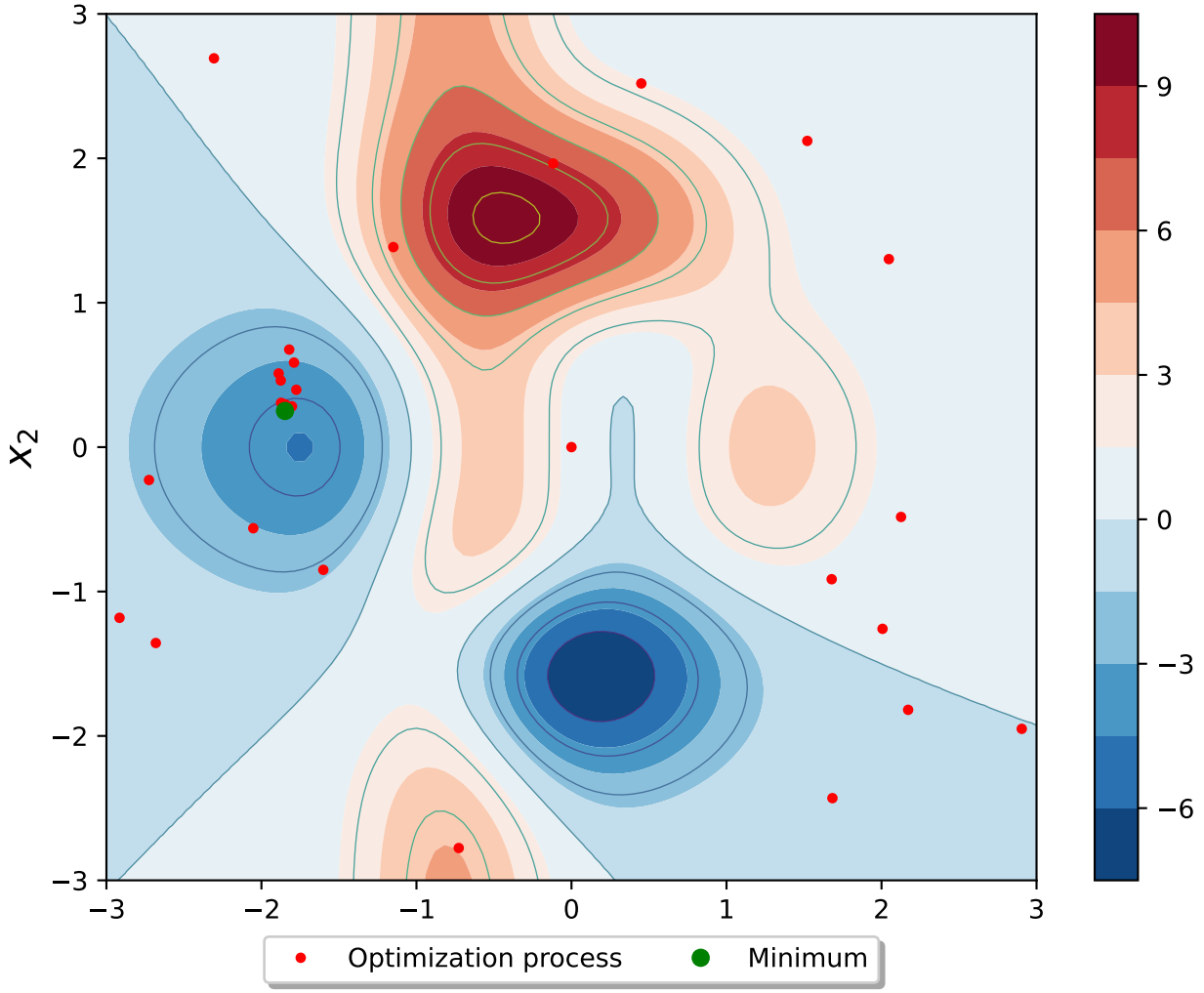

Figure 9: 1-D Case Optimized by Simulated Annealing (SA).

Comparsion with SA and GA

With the same parameter setting to 1-D case, we leverage GA to optimize the aforementioned 2-D case.

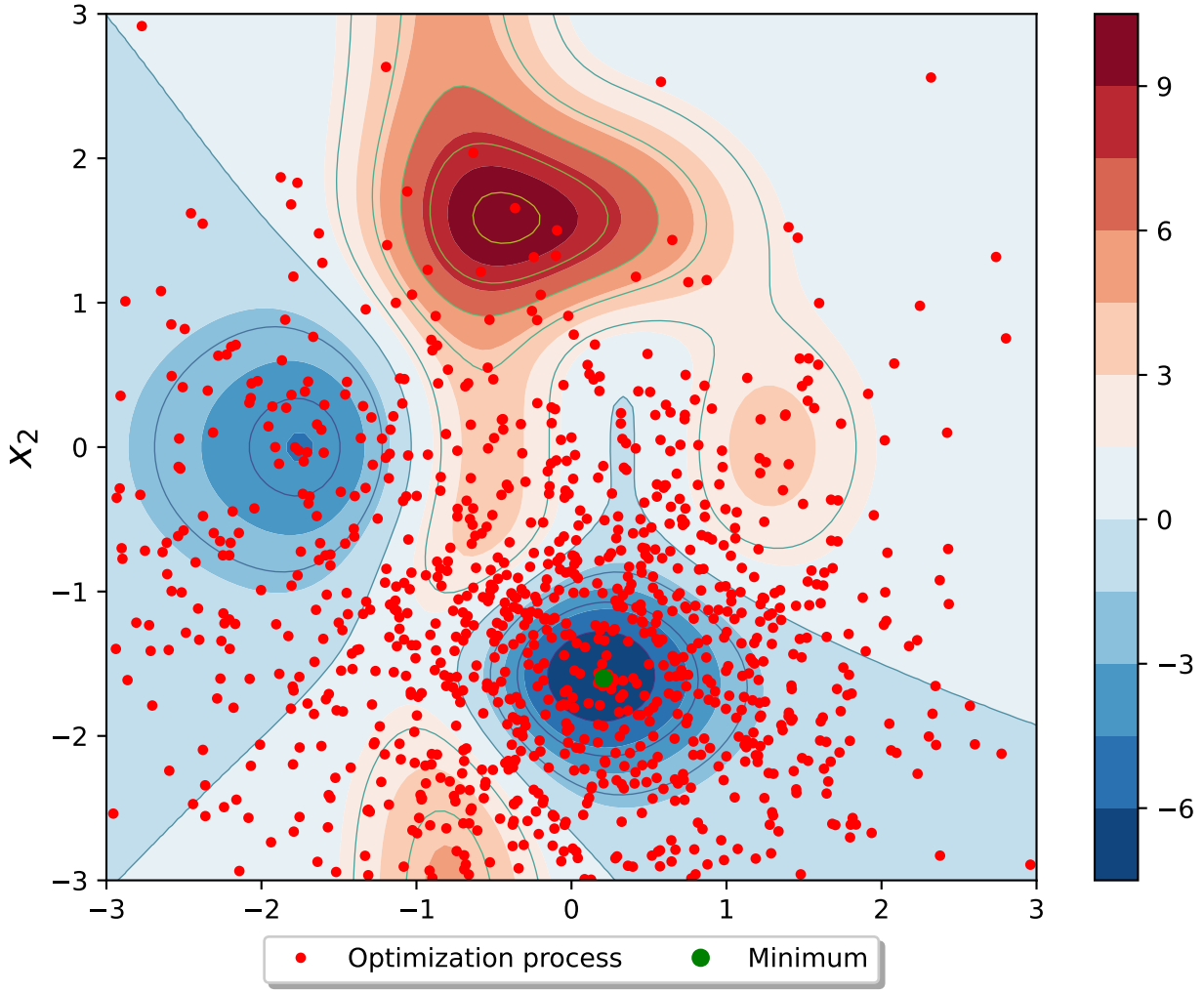

Figure 10: 1-D Case Optimized by Genetic Algorithm (GA).

Comparsion with SA and GA

Simulated Annealing (SA): As Figure 9 shows, the optimal point of SA falls into a local minimum and it never approaches the global optimal point. It probably has to do with the temperature and the initial value.

Genetic Algorithms (GA): As Figure 10 shows, the evaluated points of GA scatter largely but also have to trend to cluster around the global minimum, by which they finally reach a near perfect result.

Bayesian Optimizer: As Figure 6 shows, only budget 10 points, the Bayesian Optimizer can easily optimize to the optimal point.

Conclusion: Bayesian optimizers also show excellent optimization with a few shots on 2-dimensional problems, *however, is it really without drawbacks*?

Further Discussion

In every iteration, the Bayesian Optimier will make use of all evaluated points $(\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x}))$ to conduct inference on newly sampled points $\boldsymbol{x}^*$. We would like to estimate the $\boldsymbol{y}^*$ by Gaussian Process Regression. The prior distribution of $\boldsymbol{y}^*$ is set as $N(\boldsymbol{\mu_y}, K_{\text{yy}})$.

Assume that $(\boldsymbol{f}(\boldsymbol{x}),\boldsymbol{y}^*) \sim$ Multivariate Gaussian Distribution.

$$\begin{bmatrix} f(\boldsymbol{x}) \\ \boldsymbol{y}^* \end{bmatrix} \sim N\left(\begin{pmatrix} \boldsymbol{\mu_f} \\ \boldsymbol{\mu_y} \end{pmatrix}, \begin{pmatrix} K_{\text{ff}} & K_{\text{fy}} \\ K_{\text{fy}}^T & K_{\text{yy}} \end{pmatrix} \right)$$

And $(\boldsymbol{y}^*|f(\boldsymbol{x})) \sim N\left(K_{\text{fy}}^T K_{\text{ff}}^{-1} f(\boldsymbol{x}), K_{\text{yy}} - K_{\text{fy}}^T K_{\text{ff}}^{-1} K_{\text{fy}}\right)$. Let n be the number of all evaluated points and m be the dimension of input space. We can know from above equation that the computation comsumption of $(\boldsymbol{\mu}, \boldsymbol{\sigma})$ of $\boldsymbol{y} *$ is $O(n^3 + m^2 \times n^2)$, which means the Bayesian Optimizer will be hard to fit when it comes to high dimentional problems or when the iteration time is too large.