Bayesian optimisation using Stan

Anna Elisabeth Riha Adam Howes Seth Flaxman Elizaveta Semenova

September 13, 2024

Imperial College London

Aalto University School of Science

FCAI Finnish Center for Artificial Intelligence

Bayesian optimisation using Stan

(a nice BO image here)

In this tutorial, we will

- provide an overview of Bayesian optimisation (BO),
- demonstrate how Stan can be used within the BO procedure,
- demonstrate variable query cost and non-response propensity within BO.

Learning outcomes for today's tutorial

After this session, you will be able to

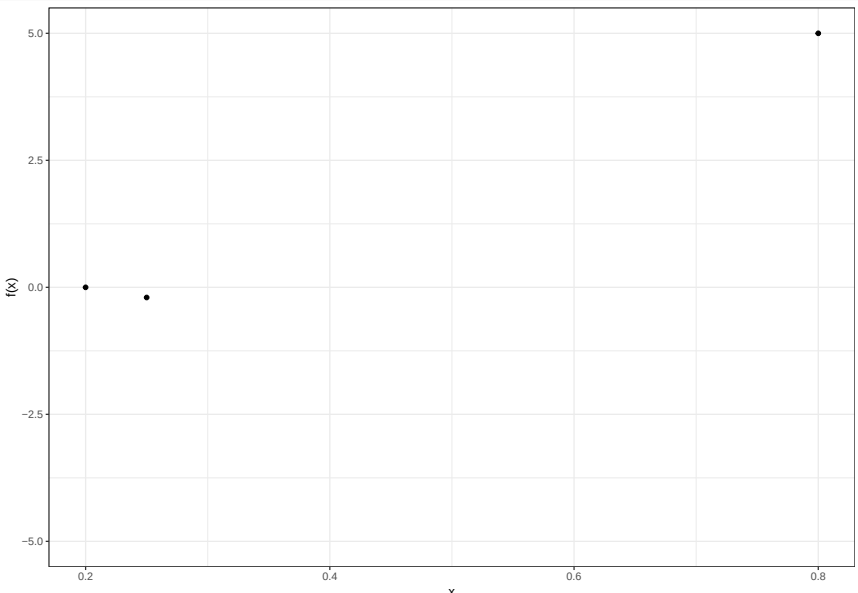
- formulate the main goal and components of BO,
- implement Gaussian process surrogates,
- use several acquisution functions within BO.

Schedule for today's tutorial

Time	Activity
5 min	Warmup
10 min	What does Bayesian optimisation solve?
15 min	Building blocks of Bayesian optimisation
20 min	GPs as Surrogates
10 min	Acquisition functions
10 min	Break
5 min	Computational tricks
10 min	Cost- and propensity-aware BO
30 min	BO Hands-on
5 min	Outro

Warmup

Where to sample next to find global minimum?



What does Bayesian optimisation solve?

Problem definition: global optimisation

The goal of **global optimisation** of a real-valued function $f : \mathcal{X} \to \mathbb{R}$ is to find a *minimiser* x^* (there may be more than one) in the search space $\mathcal{X}$, such that:

$$x^* = \arg\min_{x \in \mathcal{X}} f(x).$$

Problem definition: global optimisation

Today we focus on finding a minimum, but finding a maximum can be approached in the same way since

$$\max_{x \in \mathcal{X}} f(x) = -\min_{x \in \mathcal{X}} f(x).$$

Problem definition: global optimisation

The function f to be optimised is referred to as the **objective function**.

In contrast to *local optimisation*, global optimisation requires that

$$f(x^*) \leq f(x)$$

for **all** $x \in \mathcal{X}$ rather than only in some neighbourhood about x^*. Throughout this workshop, we assume that the search space $\mathcal{X}$ is a subset of $\mathbb{R}^d$ where $d \in \mathbb{N}$:

$$\mathcal{X} \subset \mathbb{R}^d$$

Group discussion

Given a function $f : \mathcal{X} \to \mathbb{R}$, how would you approach the search of its minimum?

Problem definition: global optimisation

In practice, the objective function f may possess the following challenging properties:

1. *Non-linear*, *non-convex*,

Problem definition: global optimisation

In practice, the objective function f may possess the following challenging properties:

2. *Black-box:* A function is called **black-box** if it can only be viewed in terms of its inputs and outputs. If f is black-box then it does not have an analytic form or derivatives, such as the gradient ∇f or Hessian $\mathbf{H}$.

Problem definition: global optimisation

"Any sufficiently complex system acts as a black-box when it becomes easier to experiment with than to understand."

Golovin et al, "Google Vizier" (2017)

Problem definition: global optimisation

In practice, the objective function f may possess the following challenging properties:

3. *Expensive to evaluate*: The sampling procedure is computationally, economically or otherwise prohibitively expensive.

Problem definition: global optimisation

In practice, the objective function f may possess the following challenging properties:

4. *Noisy*: When f is evaluated at x, the value returned y is contaminated by noise ϵ, typically assumed to be Gaussian with zero mean and variance σ^2 such that

$$y = f(x) + \epsilon$$

.

Problem definition: global optimisation

- Perhaps, the global optimisation problem can be solved using sampling!

Sample designs

- How should points be queried to efficiently learn about x^*?

Sample designs

- How should points be queried to efficiently learn about x^*?
- Let's focus on finding a "good" solution or converging to a minimiser x^* in few evaluations, rather than in making theoretical guarantees about optimality.

Sample designs

The two relatively naive strategies:

- *grid-search*,
- *random-search*.

Sample designs: grid search

Grid-search:

- **How**: Samples are taken spaced evenly throughout the domain $\mathcal{X}$ at a resolution appropriate to the optimisation budget.
- **Pitfalls**: Although the whole domain is superficially covered, if few function evaluations can be afforded then this coverage is too sparse to reliably locate a minimiser.

Sample designs: random search

Random-search:

- **How**: random-search chooses inputs in the domain $\mathcal{X}$ to evaluate at random.
- **Pitfalls**: complete randomness lends itself to clumps of points and large areas left empty.

Sample designs: latin-hypercube

Latin-hypercube:

a grid with exactly one sample in each column and each row. This avoids the problem of collapsing, from which grid-search suffers.

Sample designs: static designs

An issue with the aforementioned strategies: information gained during the search is not used to better inform the next decision.

Sample designs: sequential decision making

Rather than choose all the points at once it makes sense instead to consider a *sequential decision making* problem where at each stage a point is carefully chosen to be evaluated next.

Sample designs: sequential decision making

So, how can previous information be used?

Sequential decision making: idea

- Make assumptions about f, e.g. that f is smooth.

Sequential decision making: idea

- Make assumptions about f, e.g. that f is smooth.
- Build a model which mimics behaviour of f. A model with uncertainty!

Sequential decision making: idea

- Make assumptions about f, e.g. that f is smooth.
- Build a model which mimics behaviour of f. A model with uncertainty!
- Sample in uncertain regions, not near to any previous evaluations, to *explore*.

Sequential decision making: idea

- Make assumptions about f, e.g. that f is smooth.
- Build a model which mimics behaviour of f. A model with uncertainty!
- Sample in uncertain regions, not near to any previous evaluations, to *explore*.
- Sample in promising regions, near to previous low evaluations, to *exploit*.

Sequential decision making: idea

This is exactly how Bayesian optimisation works.

Buildling blocks of Bayesian optimisation

What is bayesian optimisation?

So, what is Bayesian optimisation exactly?

What is bayesian optimisation?

- **Bayesian optimisation** (BO) (Mockus, Tiesis, and Zilinskas (1978)) is a statistical / machine learning technique for addressing the global optimisation task by treating the objective function $f(.)$ as a *random variable*.

What is bayesian optimisation?

- **Bayesian optimisation** (BO) (Mockus, Tiesis, and Zilinskas (1978)) is a statistical / machine learning technique for addressing the global optimisation task by treating the objective function $f(.)$ as a *random variable*.
- It enables the optimisation of "black-box" functions.

What is bayesian optimisation?

- **Bayesian optimisation** (BO) (Mockus, Tiesis, and Zilinskas (1978)) is a statistical / machine learning technique for addressing the global optimisation task by treating the objective function $f(.)$ as a *random variable*.
- It enables the optimisation of "black-box" functions.
- It is a *sequential*, *model-based* optimisation algorithm which learns from all available observations to make better informed choices about the next point to query.

BO components: surrogates

We need a model for the objectve “black-box” function. Where to get it?

BO components: surrogates

- Since f is black-box, there is uncertainty about it its values:

BO components: surrogates

- Since f is black-box, there is uncertainty about it its values:
  - where it has not been directly evaluated,

BO components: surrogates

- Since f is black-box, there is uncertainty about it its values:
  - where it has not been directly evaluated,
  - when f is noisy, then even at evaluated points as well.

BO components: surrogates

A perfect case for Bayesian inference and Stan to do it for us!

BO components: surrogates

- We can build an explicit probabilistic model of f, known as a **surrogate** (or **emulator**) $g_\theta(x)$.

BO components: surrogates

- We can build an explicit probabilistic model of f, known as a **surrogate** (or **emulator**) $g_\theta(x)$.
- The emulator provides a *predictive distribution* for f evaluated at new inputs.

BO components: surrogates

- We can build an explicit probabilistic model of f, known as a **surrogate** (or **emulator**) $g_\theta(x)$.
- The emulator provides a *predictive distribution* for f evaluated at new inputs.
- When data $(x_i, y_i), i = 1, ..., t$ is observed then the model can be sequentially updated and refined using the Bayes' rule.

BO components: surrogates

The surrogate $g_\theta(x)$ allows to replace the task of global optimisation of the objective function $f(.)$ with the task of global optimisation of the surrogate $g_\theta(.)$:

$$\arg\min_{x \in \mathcal{X}} f(x) \approx \arg\min_{x \in \mathcal{X}} g_\theta(x).$$

BO components: surrogates

Requirements to the surrogate:

- It should behave similarly to the objective function f,
- It should be cheaper to evaluate at a point x than the target function $f(x)$, and allow to quantify uncertainty of how well $g_\theta(x)$ approximates $f(x)$

BO components: surrogates

Based on noisy observations

$$y_i = f(x_i) + \epsilon_i$$

and collected are observation pairs (x_i, y_i) we fit the surrogate to the data

$$g_\theta(x_i) \approx y_i.$$

Acquisition functions

We have obtained a predictive distribution using $g_\theta(x)$. Now what?

Acquisition functions

At every iteration, an **acquisition function** $a(x)$ is used to decide which point x_{t+1} to query next as x_{t+1} is chosen as the optimum of $a(x)$:

$$x_{t+1} = \arg\min_{x \in \mathcal{X}} a(x)$$

.

BO components: summary

In summary, BO is a sequential *model-based* optimisation algorithm which learns from all available observations to make better informed choices about the next point to query, and uses a probabilistic surrogate model representing the objective function for it.

The BO loop

The Bayesian optimisation procedure

- *Initialization*: select initial points (x_0, y_0) to evaluate the objective function
- *Iterative Loop*:
  - update the **surrogate model** by fitting $g_\theta(x_i) \approx y_i, i = 0, ..., t$.
  - evaluate **acquisition function** $a(x)$
  - select the next point as the optimnum of the acquisition function to $x_{t+1} = \arg\min_{x \in \mathcal{X}} a(x)$.
  - evaluate the objective function: $y_{t+1} = f(x_{t+1}) + \epsilon_i$
  - Stopping Criterion: check if the stopping criterion is met (e.g., maximum number of iterations reached, convergence achieved, or budget exhausted).
- *Termination*: Return the best observed point or the point that minimises the surrogate model's prediction of the objective function.

Bayesian optimisation use cases

- Machine learning
- Drug development
- Chemical discovery
- Climate models
- Robotic control

Connection to active learning

- Active learning, on the other hand, focuses on selecting the most informative data points in order to learn the whole function.
- Put simply, BO is active learning for global optimisation.

Let's take a break! (10 min)

Some suggestions for recharging during breaks :)

- move your body
- open a window or go outside
- drink some water
- try to avoid checking e-mails, messengers, or social media

Gaussian Processes as surrogates

Choice of a surrogate

Choice of a surrogate depends on the assumptions that we make about the objective function f.

Choice of a surrogate

There exist some variety of surrogate models:

- support vector machines (Vapnik 1997),
- decision trees and random forests (Breiman 2001; Bergstra et al. 2011),
- Bayesian neural networks (Li, Rudner, and Wilson 2024).

Choice of a surrogate

However, for *continuous functions*, the most canonical choice is Gaussian processes (GPs).

GPs: multivariate Gaussian distribution

A random vector $x = (x_1, ..., x_d)^T \in \mathbb{R}^d$ has multivariate Gaussian distribution with mean vector m and covariance matrix Σ if its probability density is given by:

$$f(x) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-m)^T\Sigma^{-1}(x-m)\right)$$

This is denoted by

$$x \sim \mathcal{N}(m, \Sigma).$$

Multivariate Gaussian distribution: two attarctive properties

Consider the partitioning of x into $[x_1, x_2]$ where $x_1 \in \mathbb{R}^p$ and $x_2 \in \mathbb{R}^q$ with $p + q = d$, so that

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m_1 \\ m_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \right)$$

where m and Σ are partitioned naturally.

Multivariate Gaussian distribution: two attarctive properties

Then the *marginal* distribution of x_1 is:

$$p(x_1) = \mathcal{N}(x_1 | m_1, \Sigma_{11})$$

i.e. marginal distribution of any subset of x are a multivariate Gaussian.

Multivariate Gaussian distribution: two attarctive properties

Similarly, the *conditional* distribution of any subset of x conditioned on another subset is a multivariate Gaussian. The distribution of x_1 given that x_2 is known is given by:

$$p(x_1|x_2) = \mathcal{N}(x_1|m_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - m_2), \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}).$$

Gaussian process definition

A *Gaussian process* is defined to be a collection of random variables, any finite number of which have a joint Gaussian distribution (Rasmussen and Williams 2005). The random variables $f(x)$ are indexed by the elements x of some set $\mathcal{X}$.

Gaussian process definition

As we have seen, the multivariate Gaussian distribution is specified entirely by its mean vector m and covariance matrix Σ. Analogously, a GP is specified entirely by its prior *mean function* $\mu : \mathcal{X} \to \mathbb{R}$ and *covariance function* $k : \mathcal{X}^2 \to \mathbb{R}$, with:

$$\mu(x) = \mathbb{E}[f(x)]$$

$$k(x, x') = \mathrm{Cov}(f(x), f(x'))$$

This is commonly written as:

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

Gaussian process definition

GPs can be used as *priors* and posteriors *over functions*. This makes them a natural tool for sequential Bayesian learning.

Kernel functions

The covariance function k is also regularly called the *kernel* function. It is a positive-definite measure of the expected similarity of two inputs in the output space.

Kernel functions

The structure of a GP is primarily determined by the kernel as it specifies the likely properties of functions drawn from the process (Snelson 2008).

For simplicity, we will assume the mean to be zero.

Kernel functions: examples

The *squared exponential* (SE) kernel produces sample functions that are infinitely differentiable, and so can be used to model functions which are very smooth.

$$k_{\mathrm{SE}}(x, x') = \sigma_f^2 \exp\left(-\frac{\|x - x'\|^2}{2l^2}\right)$$

Kernel functions: examples

A more general *Matérn* family (Matérn 2013) allows the smoothness to be controlled:

$$k_{\mathrm{M}}(x, x') = \frac{\sigma_f^2}{2^{\nu-1}\Gamma(\varsigma)} \left(\frac{\sqrt{2\varsigma}\|x - x'\|}{l} \right)^{\nu} K_{\nu} \left(\frac{\sqrt{2\nu}\|x - x'\|}{l} \right)$$

Kernel functions: examples

For $\nu = 3/2$ we get

$$k_{\mathrm{M3/2}}(x, x') = \left(1 + \frac{\sqrt{3}|x - x'|}{l}\right) \exp\left(-\frac{\sqrt{3}|x - x'|}{l}\right)$$

and for $\nu = 5/2$ we get

$$k_{\mathrm{M5/2}}(x, x') = \left(1 + \frac{\sqrt{5}|x - x'|}{l} + \frac{5|x - x'|^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}|x - x'|}{l}\right)$$

Kernels and function properties

Let $r = \|x - x'\|$. Some covariance kernels and the types of functions they can model:

Name	Definition	Type of functions
Squared Exponential	$\alpha \exp\left(-\frac{r^2}{2\ell^2}\right)$	Infinitely differentiable functions
Mat'ern 1/2	$\alpha \exp\left(-\frac{r}{\ell}\right)$	Continuous but not differentiable
Mat'ern 3/2	$\alpha \left(1 + \frac{\sqrt{3}r}{\ell}\right) \exp\left(-\frac{\sqrt{3}r}{\ell}\right)$	1 time differentiable functions
Mat'ern 5/2	$\alpha \left(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}r}{\ell}\right)$	2 time differentiable functions
Linear Kernel	$\sigma_b^2 + \sigma_v^2 (x - c)(x' - c)$	Linear functions
Periodic Kernel	$\alpha \exp\left(-\frac{2\sin^2(\pi r/p)}{\ell^2}\right)$	Periodic functions

Sampling from Gaussian process

We want to samples from a zero-mean GP at a finite collection of points

$$X := [x_1, \ldots, x_n]^T \in \mathcal{X}^n \subseteq \mathbb{R}^{n \times d}.$$

The vector of random variables

$$f := f(X) := [f(x_1), \ldots, f(x_n)]^T$$

corresponding to this matrix by definition has a joint multivariate Gaussian distribution.

Sampling from Gaussian process

The distribution of f is

$$\begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \dots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \dots & k(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \dots & k(x_n, x_n) \end{bmatrix} \right)$$

Or, put more succinctly:

$$f \sim \mathcal{N}(0, K)$$

Gaussian process regression

Recall that out observations are noisy:

$$y = f(x) + \epsilon.$$

We are interested in the value of the GP $f_* := f(x_*)$ at a new input x_*.

The joint distribution of the values y and f_* is:

$$\begin{bmatrix} y \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K + \sigma^2 I & k_*^T \\ k_* & k(x_*, x_*) \end{bmatrix} \right)$$

Gaussian process regression

This can be conditioned to show that the predictive distribution $p(f_*|y)$ is Gaussian with mean and variance given by:

$$\mu(x_*) = k_*^T (K + \sigma^2 I)^{-1} y$$

$$\sigma^2(x_*) = k(x_*, x_*) - k_*^T (K + \sigma^2 I)^{-1} k_*$$

Acquisition functions

Acquisition functions

- Acquisition functions provide a heuristic measure of the utility of prospective query points.
- At stage t to determine the next point to query x_t, the chosen acquisition function a is maximized over the design space

$$x_t = \text{argmax}_{x \in \mathcal{X}} a(x).$$

Acquisition functions: probability of improvement (POI)

Probability of improvement acquisition function (Kushner (1964)) under the GP surrogate can be computed analytically as

$$a_{\text{POI}}(x) = \Phi\left(\frac{\hat{y} - \mu(x)}{\sigma(x)}\right) = \Phi(z(x)).$$

Here

- $\hat{y}$ is "the best value found so far",
- $\Phi(\cdot)$ is the standard normal cumulative distribution function,
- $z(x) = (\hat{y} - \mu(x))/\sigma(x)$ is the standardization score,
- $\mu(x)$ and $\sigma(x)$ are point-wise predicted mean and standard deviation of $f(x)$.

Acquisition functions: Expected improvement (EI)

Expected improvement (EI) (Mockus 1974):

$$a_{\mathrm{EI}}(x) = \sigma(x)[z(x)\Phi(z(x)) + \phi(z(x))].$$

Here

- $\Phi(\cdot)$ and $\phi(\cdot)$ are the CDF and the PDF of the standard normal probability density function.

Acquisition functions: lower confidence bound (LCB)

Lower confidence bound (LCB; or upper, when considering maximization) (Srinivas et al. (2009)):

$$a_{\text{LCB}}(x) = \mu(x) - \kappa * \sigma(x)$$

Here

- κ is a user-defined parameter allowing to balance exploration and exploitation.

Computational tricks

Tricks for GPs

GPs scale cubically leading to poor scalability. There is a expanding range of techniques available to bypass this problem. For instance,

- on a multivariate grid, the Kronecker product trick (Saatçi 2012),
- Hilbert Space Gaussian Process (HSGP) approximations Riutort-Mayol et al. (2023),
- approximations using variational techniques.

Thompson samping

Thompson sampling (Thompson 1933) selects the next evaluation point by using *one sample* from the posterior distribution of the objective function.

Query cost and response propensity

Query cost and response propensity

The default version of BO implicitly assumes

- constant cost of each query,
- a guaranteed response every time a query is made.

Variable cost

It is important to prioritize searching in areas with lower evaluation costs.

The cost can be included into the acquisition function by dividing the acquisition function by the cost:

$$a_{\text{cost}}(x) = \frac{a(x)}{c(x)}.$$

Response propensity

Propensity of response can be accounted for by introducing a corresponding propensity of response function. A propensity-aware acquisition function is then given by combining an acquisition function with a propensity function, denoted $r(x)$.

Cost-cooling

The cost-cooled (Lee et al. 2020) acquisition function is given by

$$a_{\text{cool}}(x) = \frac{a(x)}{c(x)^{\alpha}}.$$

Combining cost-cooling and propensity of response

$$a_{\text{prop,cool}}(x) = a_{\text{cool}}(x) r(x) = \frac{a(x)}{c(x)^{\alpha}} r(x)$$

In connection with a propensity of response and given a total budget τ, including the cost of queries allows to directly obtain estimates for the wasted budget due to the expected number of non-responses.

Implementation in R and Stan

Implementation of acquisition functions: probability of improvement

```
probability_improvement <- function(m, s, y_best) {
  if (s == 0) return(0)
  else {
    poi <- pnorm((y_best - m) / s)
    # if maximizing: poi <- 1 - poi
    return(poi)
  }
}
```

Implementation of acquisition functions: expected improvement

```r
expected_improvement <- function(m, s, y_best) {
  ei <- rep(NA, length(m))
  for (i in 1:length(ei)){
    if (s[i] == 0) {
      ei[i] <- 0
      next
    }
    gamma <- (y_best - m[i]) / s[i]
    phi <- pnorm(gamma)
    ei[i] <- s[i] * (gamma * phi + dnorm(gamma))
  }
  return(ei)
}
```

Implementation of acquisition functions: lower confidence bound

```
gp_lower <- function(m, s, kappa=2){
  lower_confidence_bound <- m - kappa * s
  # if maximizing: upper_confidence_bound <- mu + kappa * sigma
  return(lower_confidence_bound)
}
```

An initial example

For an initial example, assume that the unknown function is in fact the Forrester function (Forrester, Sobester, and Keane 2008):

$$f(x) = (6x - 2)^2 \sin(12x - 4).$$

Implementation

Coding time ...

Outro

Summary

What we learnt today:

- What problem is solved by Bayesian optimisation,

Summary

What we learnt today:

- What problem is solved by Bayesian optimisation,
- BO components: surrogates and acquisition functions,

Summary

What we learnt today:

- What problem is solved by Bayesian optimisation,
- BO components: surrogates and acquisition functions,
- Cost- and response propensity-aware BO,

Summary

What we learnt today:

- What problem is solved by Bayesian optimisation,
- BO components: surrogates and acquisition functions,
- Cost- and response propensity-aware BO,
- Relevant implementation.

Stay tuned

We will soon release a write-up based on the materials of this workshop, with more details.

References

Bergstra, James, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. "Algorithms for Hyper-Parameter Optimization." In *Advances in Neural Information Processing Systems*. Vol. 24. Curran Associates, Inc. https://papers.nips.cc/paper_files/paper/2011/hash/86e8f7ab32cfd12577bc2619bc635690-Abstract.html.

Breiman, Leo. 2001. "Random Forests." *Machine Learning* 45 (1): 5–32. https://doi.org/10.1023/A:1010933404324.

Forrester, Alexander, András Sobester, and Andy Keane. 2008. "Engineering Design via Surrogate Modelling: A Practical Guide | Wiley." *Wiley.com*. https://www.wiley.com/en-gb/Engineering+Design+via+Surrogate+Modelling%3A+A+Practical+Guide-p-9780470060681.

Kushner, Harold J. 1964. "A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise."