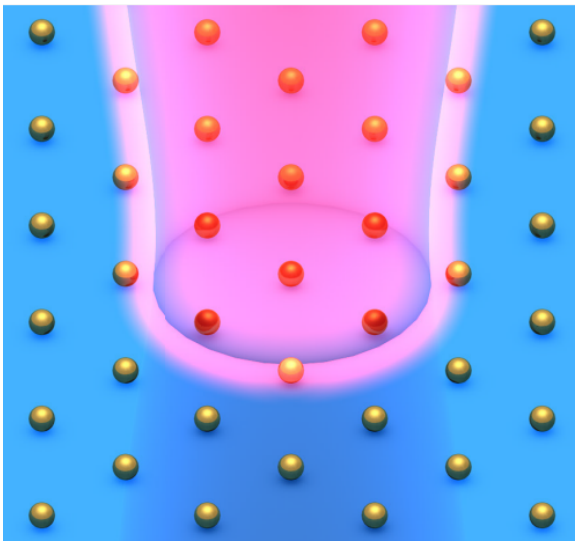
Optimizing the optical response of NP arrays using Bayesian Learning

Tijs L. Wiegman
UvA: 13865617, tijs.wiegman@student.uva.nl
VU: 2753040, t.l.wiegman@student.vu.nl



Bachelor Thesis
Double Bachelor Physics and Astronomy & Mathematics

Uva - Korteweg de Vries Instituut
Supervisor dr. Tim van Erven
Second grader prof. dr. Rob Stevenson

VU - PhotoConversion Materials Section
Supervisor dr. Sven Askes
Second grader dr. Andrea Baldi

June 30, 2024

Abstract

The heating of an illuminated nanoparticle (NP) array is influenced by multiple factors, including the array geometry. In this thesis, we consider silver cylindrical NPs in an air medium and on a glass substrate under a 532 nm laser. Testing various geometries is best done using simulations, therefore this is a computational study, employing the COMSOL software. Maximizing the geometry for heat generation is greatly useful in practical applications but difficult to estimate a priori. Moreover, running simulations is computationally expensive and should thus be limited. It is thus imperative to intelligently select which simulations to run to stay within a reasonable computational budget. Optimizing an unknown function with a restricted number of function observations can be achieved through Bayesian Optimization. Constructing a hypothesis of the objective function with a Gaussian Process (GP), each new observation can be used to update the GP using inference. The GP hypothesis, in turn, informs what location is best sampled next using an acquisition function. In this work, the GP is defined by a Matérn 5/2 covariance function, and we use Latin Hypercube Sampling to construct the initial GP. We compare the performance of our algorithm with different exploration parameters for the Expected Improvement (EI) class of acquisition functions, where we found moderate exploration to work best. We also compare EI with the Probability of Improvement (PoI) and Upper Confidence Bound (UCB) acquisition functions, with only UCB proving competitive.

Contents

Introduction

[This document is a compressed version and does not contain animation. For full version contact tijs.wiegman@student.uva.nl]

At each sufficiently distinct scale, a plethora of completely unique physical phenomena are observed. Just as physics at the galactic, planetary, human, atomic and quantum scales are completely different, interesting effects unique to the nanoscale (anything between 1μm and 1nm in size) are being studied. Because of this, there is a lot of excitement around nanoscale research. At the Photo-Conversion Materials department at the VU, research is being done into the optical response of nanoparticles (NPs) to control and detect physical processes at the nanoscale. By optical response, we mean the response of the system under illumination by laser. In this work, we are studying NP arrays of specifically cylindrical silver NPs placed in a 2D square lattice. The laser induces a special Electron-Magnetic (EM) field resonance called a Localized Surface Plasmon Resonance (LSPR), which causes the NPs to absorb more energy from the laser, resulting in enhanced heating. By how much the heating is enhanced depends on the array geometry as this increases or decreases the effect of LSPR. For most applications, the objective is to find the NP array configuration with the highest temperature increase. This can then be used for applications like chemical catalysis, where the activation energy of a reaction is lowered with LSPR or where the thermal energy increases reactivity [1]. It can also be used for cancer photothermal therapy, where NP arrays can be placed on cancer cells and then heated using lasers to target and kill the cancerous cells without killing healthy cells [2].

It is near impossible to test many different array geometries in a lab, as each of the many configurations would need to be manufactured. Thus, the NP array is modeled using the COMSOL simulation software [3], where we can control the array geometry using simple input parameters. When the simulation is run, each taking about 1 minute, the software returns a value computed by the physics engine, and from this, the increase in temperature is calculated. The relevant explanation of the setup of this problem is described in Chapter 1. This whole system can be reduced to a simple function, with the inputs being the parameters defining the array geometry and the output being the temperature increase. Our objective then becomes finding the maximum of this function. Yet, due to the vast number of possible inputs and the simulation duration, we can only know a very limited number of function values. Thus we need to optimize an unknown function.

As more and more data and computational resources are available, Artificial Intelligence (AI) models have gotten bigger and better. In the last couple of years especially, AI has gotten a boost thanks to Machine Learning, where models learn to replicate a response and making predictions, i.a. At the Korteweg de Vries Institute (KdVI) - Machine Learning division at the UvA, researchers

are working on how to use sophisticated maths to let computer systems learn unknown functions and distributions. In this work, we don't need to know the entire function, we want to find a good approximation of the maximum of the function. Bayesian Learning has proven a fitting machine learning framework for this problem. Optimization with Bayesian Learning is called Bayesian Optimization (BO). It works by treating the unknown objective function as a random variable described by a Gaussian Process (GP), which is like a continuous extension of a Gaussian distribution. The GP will form the hypothesis of the objective function, and by gathering points we will update the hypothesis using Bayesian inference. Then using the hypothesis, an acquisition function scores each point in the domain, with its maximum becoming the next observation point. By doing this iteratively, the hypothesis becomes stronger and our suggested sample locations should return higher values. This algorithm is the BO algorithm, and there are many variations of it. For example, how to define the GP that models the objective function, which acquisition functions to pick, and how to deal with the trade-off between exploiting the hypothesis and exploring new parts of the search space. Each of these choices affects the performance of the BO algorithm, and the better we understand these effects, the better algorithms we can design. Thus, we were not so much interested in finding an optimum of the physical system, but more so in how to construct an algorithm with the best performance [4].

This topic has been worked on before by my predecessor Martijn Kievit [5], supervised by Sven Askes from the VU and Boukje de Gooijer, with Sven giving input on the nanophysics and Boukje on the optimization algorithm. I started where they left off, working on questions that were left unanswered and gaining more insight and confidence in the algorithm. We changed the model in COMSOL [3] to speed up the simulation process. Along with Sven and Boukje, I was also assisted by Tim van Erven for the Mathematics side. Martijn implemented the BO algorithm in Python using the BO package [6] however many of his choices were not optimized and were poorly motivated.

In this work, we worked through some of the theory behind BO, mainly following the book Bayesian Optimization (2023) by Roman Garnett [4]. How to model the objective function using a GP, how GP inference works and different types of acquisition functions among other things are described in Chapter 2. In Chapter 3, we discuss how we can formally analyze the performance of an algorithm and give some known bounds. We then go into some experiments in Chapter 4. The first experiments are aimed at understanding what values for the exploration parameter of the Expected Improvement (EI) acquisition function results in the best performance. Then we compare the performance of EI with that of other acquisition functions, namely Probability of Improvement (PoI) and Upper Confidence Bound (UCB). To gain more intuition and understanding, we plot the algorithm in for 1D version of the problem. We then found the implementation using the BO package might be flawed, so I wrote the algorithm using the pyGPGO package [7]. We then analyze the behavior of the new algorithm and compare the implementations with the BO package and the pyGPGO package. Throughout this project, a lot of simulations have been done, and the simulation with the highest target value was analyzed, to understand why it was so extraordinary high and whether it was valid. In Chapter 5 we discuss our work and look to what future research can learn from this project, what mistakes were made and which questions are left.

The image on the cover is adapted from [8]

Chapter 1

Getting to know the problem

1.1 Metallic nanoparticles

The nanoparticles (NPs) used in the aforementioned research at the VU are usually made up of gold or silver, so-called noble metallic NPs. There are two main ways of producing NPs: Bottom-up and top-down. Bottom-up approaches start with atoms and/or molecules and seek to conglomerate these to form nanostructures. This method will not be explored further here. Top-down approaches of metallic NP synthesis, which start with a larger mass and reduce this to the desired shape and size, allow for flexible and relatively inexpensive fabrication with precise control, and thus this is the method we will consider.

Electron beam lithography methods, whereby an electron beam creates the desired features, are especially good at creating precise periodic arrays with controlled interparticle spacing. Because we can also control the height of the features, the results are 2.5D periodic particle arrays (PPAs) with features that can be as small as 15nm. The low throughput of the electron beam can be mitigated by using multiple beams in parallel, increasing production speed [9].

In this work, we will restrict ourselves to the study of PPAs constructed with the limitations of electron beam lithography. That is, we will assume we can produce PPAs with gold and/or silver NPs, with the PPAs having feature sizes no smaller than 15nm. We will further restrict ourselves to studying PPAs consisting of upright cylinders, i.e. the particles are cylindrical shaped and oriented vertically. In total, the considered PPAs have interparticle distance not smaller than 100nm, radius not smaller than 15nm and uniform height not smaller than 15nm.

1.2 LSPR & temperature

When light hits a metallic NP smaller than the wavelength of the light, the surface-level conduction electrons of the NP can oscillate collectively, called a surface plasmon. When this surface plasmon is restricted to the particle and does not propagate, it is called a localized surface plasmon. When the oscillation of a localized surface plasmon resonates with an incoming light wave, Localized Surface Plasmon Resonance (LSPR) can occur. As a result, the EM-field is greatly enhanced near

the particle surface and the absorption cross-section is larger, sometimes becoming larger than the surface of the NP itself. This is called the optical response of the NP. The large absorption cross section means the NP can absorb more of the laser's energy, thus creating more heat. When NPs are put in an array, the heat sources combine, increasing the heat generated. Also, LSPRs can couple, influencing the optical response further. The geometry of the NP array has a great effect on how the LSPR coupling, meaning it also greatly affects the strength of the optical response. This results in the laser-induced heating also being greatly influenced, possibly increasing past what would be achieved by a simple combination of heat sources. The NP array we use is placed on a substrate, meaning the medium is not homogeneous and there is no analytical formulation to predict LSPR strength, hence we have to use numerical methods using simulations to test them [10], [8].

1.3 COMSOL

For this work, no new lab work was done as this is a computational study, meaning all of the experiments were computational experiments. The software used, COMSOL, is a finite element solver that can solve coupled phenomena, like fluid and chemical or optical and heat phenomena, by solving the coupled systems of partial differential equations (PDEs). My predecessor [5] used the 'electromagnetic wave frequency domain' (ewfd) and 'heat physics' packages, solving optical and heat PDEs consecutively. This use of multiphysics is computationally expensive compared to using a single physics module. Thus we choose to use COMSOL only to solve the optical PDEs and use the result to compute an approximation of the temperature. Note that when repeating a COMSOL simulation, the answer can have deviations of 1/1000 or much smaller. We will ignore this effect and not investigate where it comes from, instead using these as perfectly identitical, which is reasonable as the differences are so small.

1.4 Temperature proxy

The temperature increase of the center NP of an NP array is noted ΔT_0, and can be decomposed as follows:

$$\Delta T_0 = \Delta T_0^{\mathrm{s}} + \Delta T_0^{\mathrm{ext}}$$

Here ΔT_0^{s} stands for *self* contribution and $\Delta T_0^{\mathrm{ext}}$ stands for *external* contribution, i.e. the temperature increase due to other NPs, we are only interested in $\Delta T_0^{\mathrm{ext}}$.

For the uniform circular illumination of an infinite array of identical NPs, Baffou et al. [10] gives the following approximation:

$$\Delta T_0^{\mathrm{ext}} \approx \frac{\sigma_{\mathrm{abs}}\, I\, D}{4\,\overline{\kappa}\, A}\left(1 - \frac{2\sqrt{A}}{\sqrt{\pi}\, D}\right) \approx \frac{\sigma_{\mathrm{abs}}\, I}{2\,(\kappa + \kappa_s)\, A}\left(D - \frac{2\sqrt{A}}{\sqrt{\pi}}\right) \tag{1.1}$$

Here, σ_{abs} [m^2] denotes the NP absorption cross-section, I [W m^{-2}] denotes the irradiance of the illumination, κ [W m^{-1} K^{-1}] and κ_s [W m^{-1} K^{-1}] denote the thermal conductivity of the medium and the substrate respectively, A [m^2] denotes the are of the unit cell and D [m] denotes the diameter of the illuminated area.

In our model, we consider an infinite array of NPs with a circular uniform illumination with diameter $D = 3$mm. We compute σ_{abs} for the case where the entire infinite array is under illumination, use

this as an approximation for the σ_{abs} for the 3mm laser case, and plug it in the formula to find ΔT_0^{ext}. Researchers are usually interested in optimizing ΔT_0, but in this case, ΔT_0^{ext} is a good approximation of ΔT_0.

Chapter 2

Bayesian Optimization

Bayesian Optimization (BO) is used to learn information about a function, the *objective function* denoted f^* (the * is used to differentiate functions from the objective function), using a very limited amount of data points sampled from that function, with the goal of finding its optimum. With 'sample from f^*', we mean given x evaluating $f^*(x)$, and with BO it is assumed that the number of samples is restricted.

Our objective function has as inputs the input parameters of the COMSOL simulation, and the output is the external temperature increase ΔT_0^{ext} computed using σ_{abs} returned by COMSOL. Then finding the NP geometry with the largest ΔT_0^{ext} comes down to finding the global maximum of this objective function.

In the setting of optimization, the goal is to find the global maximum value of the objective function, $f^*_{\max}$ by finding the optimizer $x_{\max}$, with $f^*(x_{\max}) = f^*_{\max}$:

$$x_{\max} \in \arg\max_{x \in \mathcal{X}} f^*(x). \qquad f^*_{\max} = \max_{x \in \mathcal{X}} f^*(x)$$

It's highly likely we won't find the actual optimizer within reasonable computation time, so our goal comes down to finding an approximation x of $x_{\max}$, where its location is not of interest except that $f^*(x)$ is as large as possible. Because we have no concrete knowledge about f, in BO we reason about the objective function f^* as a random variable. To model f^*, we use a Gaussian Process (GP) which allows us to use some of the nice properties of Gaussian distributions and depending on our choice of kernel allows us to model a wide range of underlying functions and behaviors.

The methods of BO are based on Bayesian Inference, which uses Bayes' rule from Stochastics to incorporate evidence to update some hypothesis. We begin with some prior belief about some random variable, and then we use data to update our belief about that random variable, giving us our posterior.

BO is a form of sequential optimization, whereby we execute a version of the following iterative algorithm: (1) We have some prior belief of f^*. (2) We use this prior belief to find an optimal place where to sample from f^*. (3) We add this to our data set. (4) We update our prior belief resulting

in the posterior belief of f^*. (5) We repeat these steps, with the current posterior acting as the prior.

Given some prior belief of the objective function, different choices in where to sample next can be made. This decision is dictated by our choice from a variety of acquisition functions, and which is appropriate depends on the specific problem. The acquisition function gives a value to each possible sampling point reflecting how favorable it is to look at the objective function there. The acquisition function should be easy and cheap to compute so that we can find its maximum. Thus optimizing the expensive objective function is reduced to (among other things) repeatedly optimizing a cheaper function.

In the theory, we will assume our domain, denoted $\mathcal{X}$, is a convex and compact subset of $\mathbb{R}^d$. In reality our objective function, the COMSOL simulation, can only be sampled at discrete inputs, but this assumption on the domain is pretty reasonable for the small step size and very useful. We will further model the observations of the objective function as exact, since COMSOL is deterministic and repeating the sampling will give (almost) the exact same outcome, meaning given x, the value $f^*(x)$ is (almost) certain.

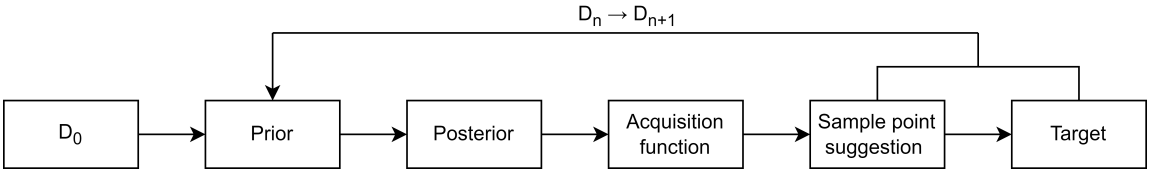


Figure 2.1: Diagram giving a general overview of the Bayesian Optimization algorithm. Here D_0 indicates the initial data set, and $D_n \rightarrow D_{n+1}$ indicates the updating of the data step at the nth step by adding the newest sample location and acquired target value.

2.1 Motivation

Optimization has been the goal of much of academia and as a result, there are many, many optimization algorithms and routines in the literature. So why from this vast collection choose Bayesian Optimization? It boils down to the qualities of the objective function.

Our objective function, the COMSOL simulation is a black box with computationally very expensive queries. We can ask for the specific function value at a specific location, but that is all we can get out of it and it's computationally expensive. If you have the analytical expression of the objective function, analytical tools can be used and optimization is easy. We don't have that luxury. If the objective function was less expensive, you could sample many many points, thousands, tens of thousands, or even more, with low computational cost. Our objective function being so expensive however, we need to bring the amount of samplings to a few hundred, and Bayesian Optimization guiding us to the smartest place to sample makes sure we use those few hundred samplings as effectively as possible. Many optimization algorithms use the gradient of the objective function to guide them, however using the COMSOL simulations we can only ever know the function values. To approximate or calculate the gradient numerically, we would need to do multiple samplings, at least one in each direction/feature. Since one of the main concerns is keeping the amount of points sampled down, this is also a non-starter.

With all of these restrictions, many options for optimization algorithms have been ruled out. Out of the remaining few, BO is one of the most sophisticated with deep ties to multiple mathematical fields, making it a good choice for a mathematics project as well.

2.2 Bayesian Inference

In probability theory, the conditional probability (the chance of one given the other), can be computed using Bayes' rule:

$$\mathbb{P}[H \mid E] = \frac{\mathbb{P}[H]\,\mathbb{P}[E \mid H]}{\mathbb{P}[E]}.$$

Depending on the context, we can view this as updating our belief about some *hypothesis* H. Then $\mathbb{P}[H]$ denotes our previous belief about the probability of the hypothesis H, $\mathbb{P}[E]$ denotes the probability of the *evidence* E, and $\mathbb{P}[E|H]$ denotes the probability of some evidence E given our current hypothesis. With this rule, we use these probabilities to compute our updated belief about the probability of the hypothesis denoted $\mathbb{P}[H|E]$. If the evidence and the hypothesis are independent, the updated hypothesis is identical to the previous hypothesis, in agreement with intuition. This framework of inference can be used to learn about quantities ruled by randomness and is one of the cornerstones of Stochastics.

We can extend this law of inference to continuous probability distributions. For some random variable y and evidence $\mathcal{D}$ (noted $\mathcal{D}$ for *data set*), we can write:

$$p(y \mid \mathcal{D}) = \frac{p(y)\,p(\mathcal{D} \mid y)}{p(\mathcal{D})}.$$

Here we call $p(y)$ the *prior* probability distributions and $p(y|\mathcal{D})$ the *posterior* probability distributions. In BO, we use the posterior belief to update the data set by adding one new point. We then use inference again to arrive at a new posterior, Doing this and applying the inference iteratively is where the power of inference really comes to shine.

In our case, we wish to learn information about some function using data points collected by the simulation. This means we treat our function as a random variable for which we must choose a suitable model. Before we explain our choice of model, the Gaussian Process (GP), we first introduce the Gaussian distribution and some it's relevant properties.

2.3 Bayesian inference on Gaussians

A *univariate Gaussian* in $\mathbb{R}$ is denote $\mathcal{N}(\mu, \sigma^2)$ and is completely defined by the two quantities, $\mu \in \mathbb{R}$ and $\sigma^2 \in \mathbb{R}_{>0}$. For $x \sim \mathcal{N}(\mu, \sigma^2)$ a Gaussian random variable taking values in $\mathbb{R}$, these quantities related by

$$\mu = \mathbb{E}[x], \qquad \sigma^2 = \mathrm{var}[x]$$

We can write the probability density function (PDF) for this distrubition, denoted $p(x)$ or $\mathcal{N}(x; \mu, \sigma^2)$, in terms of the *z-score*:

$$z = \frac{x - \mu}{\sigma} \implies p(x) = \mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{1}{2}z^2}$$

When $\sigma = 0$, we say $\mathcal{N}(x; \mu, 0^2) = \delta(\mu - x)$, where δ denotes the delta distribution.

A very useful univariate Gaussian is the standard normal, with PDF $\phi(x) := \mathcal{N}(x; 0, 1^2)$ (we have already used notation ϕ so we will minimize its use as this PDF). The cumulative distribution function (CDF) of this distribution is expressed $\Phi(x') = \mathbb{P}[x < x'] = \int_{-\infty}^{x'} \phi(x)\,\mathrm{d}x$, and CDFs of any univariate Gaussian can be expressed as an affine transformation of Φ. To illustrate, if we have $p(y) = \mathcal{N}(y; \mu, \sigma^2)$, we can write $\mathbb{P}[y < y'] = \Phi\left(\frac{y'-\mu}{\sigma}\right)$. Because $\phi(x)$ is non-zero everywhere, Φ is a strict monotonically increasing function and thus bijective.

A *multivariate Gaussian* (also called multivariate normal, from here on just *Gaussian*) in $\mathbb{R}^d$ is denoted $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ and is completely defined by the two quantities, the *mean vector* $\boldsymbol{\mu} \in \mathbb{R}^d$ and the *covariance matrix* $\boldsymbol{\Sigma} \in \mathbb{R}^{d \times d}$, with $\boldsymbol{\Sigma}$ symmetric, meaning $\boldsymbol{\Sigma} = \boldsymbol{\Sigma}^\top$, and positive definite, meaning it has strictly positive eigenvalues so the determinant is positive and thus $\boldsymbol{\Sigma}$ is invertible. For $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ a random variable (or rather a vector of ranom variables) taking values in $\mathbb{R}^d$, these quantities are related by

$$\boldsymbol{\mu} = \mathbb{E}[\mathbf{x}], \qquad \boldsymbol{\Sigma}_{i,j} = \mathrm{cov}[\mathbf{x}_i, \mathbf{x}_j]$$

Note that $\mathrm{cov}[\mathbf{x}_i, \mathbf{x}_i] = \mathrm{var}[\mathbf{x}_i]$, so the diagonal of $\boldsymbol{\Sigma}$ contains variances.

To fully understand the role of the covariance matrix, it's important to first understand covariance. The covariance between two random variables is defined as the expectation of the product of the deviations from the means:

$$\mathrm{cov}[x, y] = \mathbb{E}\big[(x - \mathbb{E}[x]) \cdot (y - \mathbb{E}[y])\big] = \mathrm{cov}[y, x]$$

The covariance tells us something about how two random variables x and y change in relation to each other. If the covariance is positive, it means there is a positive correlation between x and y, so large positive values of x coincide with large positive values of y. If the covariance is negative, there is a negative correlation between x and y, so large positive values of x coincide with large negative values of y. This means that if the covariance is large (positive or negative), the conditional uncertainty of x given y is smaller and vice versa, as knowing one informs about the other. When the covariance is 0 there is no correlation.

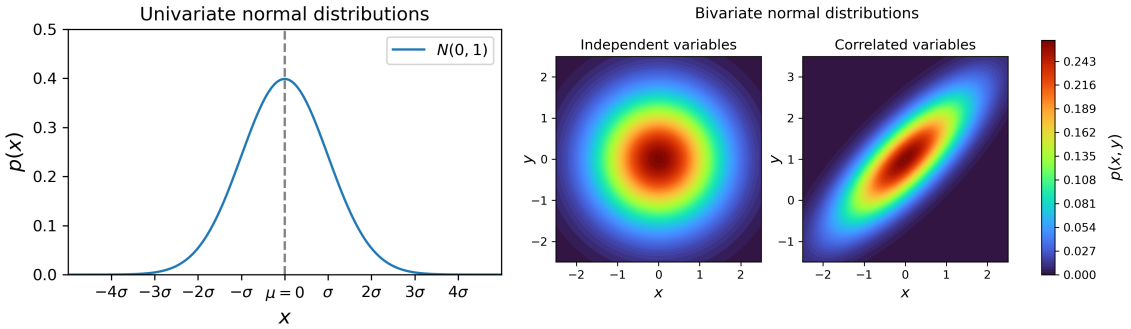


Figure 2.2: (Left) A plot of the PDF of a univariate Gaussian with $\mu = 0$ and $\sigma = 1$. (Middle) A heat map of the PDF of a multivariate Gaussian in 2 dimensions (bivariate) where the random variables x and y are independent ($\mathbf{\Sigma} = \mathbf{I}$). Red indicates high probability. (Right) A similar heat map, but where x and y have positive correlation ($\text{var}[x] = \text{var}[y] = 1$, $\text{cov}[x, y] = 0.8$). All plots made by me adopted from [11].

(In the non-degenrate case) We can write the PDF for this distribution, denoted $p(\mathbf{x})$ or $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \mathbf{\Sigma})$, in terms of the *Mahalanobis distance*:

$$\Delta^2 := (\mathbf{x} - \boldsymbol{\mu})^\top \mathbf{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \implies p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \mathbf{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d \det(\mathbf{\Sigma})}} e^{-\frac{1}{2}\Delta^2}$$

Note that when $d = 1$, we retrieve the univariate case.

To understand the expressions of inference on Gaussians, there are a few things we need to tackle first.

First up is the joint distribution. Given two Gaussians, the joint distribution is again a Gaussian, constructed as follows:

$$\begin{aligned} p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_1, \mathbf{\Sigma}_{11}), \qquad p(\mathbf{y}) = \mathcal{N}(\mathbf{y}; \boldsymbol{\mu}_2, \mathbf{\Sigma}_{22}) \\ \implies p(\mathbf{x}, \mathbf{y}) = \mathcal{N}\left(\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}; \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{bmatrix} \mathbf{\Sigma}_{11} & \mathbf{\Sigma}_{12} \\ \mathbf{\Sigma}_{21} & \mathbf{\Sigma}_{22} \end{bmatrix} \right) \end{aligned} \tag{2.1}$$

Here $\mathbf{\Sigma}_{12}$ is called the *cross covariance matrix* and is defined by $\mathbf{\Sigma}_{12} = \text{cov}[\mathbf{x}, \mathbf{y}]$, also defined as $(\mathbf{\Sigma}_{12})_{i,j} = \text{cov}[\mathbf{x}_i, \mathbf{y}_j]$. Then $\mathbf{\Sigma}_{21} = \text{cov}[\mathbf{y}, \mathbf{x}]$, which means that $\mathbf{\Sigma}_{21} = \mathbf{\Sigma}_{12}^\top$. If $\mathbf{x}$ and $\mathbf{y}$ have the same dimension, then $\mathbf{\Sigma}_{12} = \mathbf{\Sigma}_{21}$ is symmetric. If $\mathbf{x}$ and $\mathbf{y}$ are independent, we would have $\text{cov}[\mathbf{x}_i, \mathbf{y}_j] = 0$ for all i, j, meaning $\mathbf{\Sigma}_{12} = \mathbf{0}$ (the 0-matrix).

If $\mathbf{z} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}$ and $\mathbf{A} = \begin{bmatrix} \mathbf{\Sigma}_{11} & \mathbf{\Sigma}_{12} \\ \mathbf{\Sigma}_{21} & \mathbf{\Sigma}_{22} \end{bmatrix}$, then $\mathbf{A}_{i,j} = \text{cov}[\mathbf{z}_i, \mathbf{z}_j]$ with $\mathbf{A} = \mathbf{A}^\top$. If additionally, $\mathbf{b} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}$, we would have $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{b}, \mathbf{A})$. Thus it works both ways: The joint distrubtion of $\mathbf{x}, \mathbf{y}$ is Gaussian for any $\mathbf{x}, \mathbf{y}$ Gaussian, and for $\mathbf{z}$ Gaussian, any *partition* on $\mathbf{z}$, splitting it up in multiple random variables, each of those constituents would be Gaussian as well.

This leads us naturally to *marginalization*, i.e. ignoring some of the information. For $\mathbf{x}$ a vector of random variabls, if we partition $\mathbf{x}$ we can write:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} \implies p(\mathbf{x}) = p(\mathbf{x}_1, \mathbf{x}_2) = \mathcal{N}\left(\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix}; \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix} \right)$$

$$\implies p(\mathbf{x}_1) = \mathcal{N}(\mathbf{x}_1; \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}) \quad \& \quad p(\mathbf{x}_2) = \mathcal{N}(\mathbf{x}_2; \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22})$$

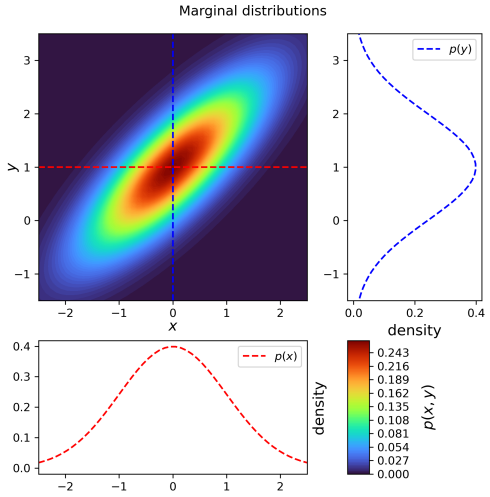


Figure 2.3: A heat map of the PDF of the same bivariate Gaussian as in (2.2) with correlation, as well as the marginal PDFs of both x and y. The red and blue dotted lines in the heat map indicate how the marginal distributions relate to the heat map. In fact, any such 'slice' that goes through the mean will be a Gaussian again. All plots made by me adopted from [11]

Then we have *conditioning*, i.e. incorporating information. If we continue with the above partition on $\mathbf{x}$, and suppose we know $\mathbf{x}_2 = \mathbf{a}$. Then for the probability of $\mathbf{x}_1$ conditional on $\mathbf{x}_2 = a$, we can write

$$p(\mathbf{x}_1 \,|\, \mathbf{x}_2 = \mathbf{a}) = \mathcal{N}(\mathbf{x}_1; \boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{11|2}) \tag{2.2}$$

The *posterior mean vector* $\boldsymbol{\mu}_{1|2}$ and the *posterior covariance matrix* $\boldsymbol{\Sigma}_{11|2}$ are given by

$$\boldsymbol{\mu}_{1|2} = \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{a} - \boldsymbol{\mu}_2), \qquad \boldsymbol{\Sigma}_{11|2} = \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}$$

It can be easily checked that $\boldsymbol{\mu}_{1|2}$ and $\boldsymbol{\Sigma}_{11|2}$ conform to the requirements and are indeed a mean vector and covariance matrix respectively. Thus, if the prior $p(\mathbf{x}_1)$ is Gaussian then posterior $p(\mathbf{x}_1)$ is also Gaussian. Note that if $\mathbf{x}_1$ and $\mathbf{x}_2$ are independent, then $\boldsymbol{\Sigma}_{12} = \mathbf{0}$ and the posterior mean and covariance are the same as their priors.

A special case we will come back to later is when we have a partition like $\mathbf{x} = \begin{bmatrix} x_1 \\ \mathbf{x}_2 \end{bmatrix}$ and want to find the conditional probability of x_1 given $\mathbf{x}_2$. Here x_1 is a single random variable and $\mathbf{x}_2$ again is a vector of random variables (and thus so is $\mathbf{x}$). Suppose $p(x_1) = \mathcal{N}(x_1; \mu, \Sigma_{11})$ and $p(\mathbf{x}_2) = \mathcal{N}(\mathbf{x}_2; \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22})$, then equation (2.1) becomes:

$$p(\mathbf{x}) = p(x_1, \mathbf{x}_2) = \mathcal{N}\left(\begin{bmatrix} x_1 \\ \mathbf{x}_2 \end{bmatrix}; \begin{bmatrix} \mu_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix} \right)$$

Here $\boldsymbol{\Sigma}_{12}$ is the cross-covariance *vector* with $\boldsymbol{\Sigma} = \text{cov}[x_1, \mathbf{x}_2]$, i.e. $(\boldsymbol{\Sigma}_{12})_i = \text{cov}[x_1, (\mathbf{x}_2)_i]$. Again we have $\boldsymbol{\Sigma}_{21} = \boldsymbol{\Sigma}_{12}^\top$. If we look at the probability of x_1 conditional on $\mathbf{x}_2 = \mathbf{a}$, equation (2.2) becomes

$$p(x_1 \,|\, \mathbf{x}_2 = \mathbf{a}) = \mathcal{N}(x_1; \boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{11|2})$$

$$\mu_{1|2} = \mu_1 + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\mathbf{a} - \boldsymbol{\mu}_2), \qquad \Sigma_{11|2} = \Sigma_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}$$

2.4 Gaussian Process Modeling

Sources: Roman Garnett, Stochastics 1

Treating the objective function as a random variable, we need to pick a suitable modeling strategy. We will first define our choice of model, the *Gaussian Process*. Then we will show what choices can be made within this model and explain how inference works.

For $\mathcal{X}$ an arbitrary domain and $f^* : \mathcal{X} \to \mathbb{R}$, a stochastic process on f^* is a Gaussian Process, with $p(f) = \mathcal{GP}(f; \mu, K)$ as PDF, if each finite collection of function values $\{f(x_1), \ldots, f(x_k)\}$ has a Gaussian distribution:

$$p(\boldsymbol{\phi} \,|\, \mathbf{x}) = \mathcal{N}(\boldsymbol{\phi}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$$

Here $\mathbf{x} = (x_1, \ldots, x_k)^\top$ is a vecotr of points (no uncertainty) and $\boldsymbol{\phi} = \big(f(x_1), \ldots, f(x_k)\big)^\top$ is a vector of random variables. The definition also implies that any single function value $\phi = f(x)$ given x has a univariate Gaussian distribution.

$$p(\phi \,|\, x) = \mathcal{N}(\phi; \mu, \sigma^2)$$

For f a sample from a GP, denoted $f \sim \mathcal{GP}(f; \mu, K)$, we call f a *sample path*. Instead of the random variable living in a space of points, it lives in a space of functions, and a GP can be thought of as a continous extension of the Gaussian. Just like a Gaussian is defined by the mean vector and covariance matrix (both discrete), a GP is specified by two functions: the *mean function* $\mu : \mathcal{X} \to \mathbb{R}$ and the *covariance function* $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. For $p(f) = \mathcal{GP}(f; \mu, K)$, we find the following connection between the GP's mean and covariance functions and the Gaussian distribution of $\boldsymbol{\phi}$ given $\mathbf{x}$:

$$p(\boldsymbol{\phi} \,|\, \mathbf{x}) = \mathcal{N}\big(\boldsymbol{\phi}; \boldsymbol{\mu}, \boldsymbol{\Sigma}\big) \implies \boldsymbol{\mu} = \mu(\mathbf{x}) \quad \& \quad \boldsymbol{\Sigma} = K(\mathbf{x}, \mathbf{x})$$

The mean function and covariance function can thus be viewed as continuous extensions of the mean vector and covariance matrix.

The mean function $\mu(x)$ determines the expected value of the function value, so

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

The covariance function $K(x, x')$ determines how deviations from the mean are structured and is given by

$$K(x, x') = \text{cov}[f(x), f(x')]$$

Note that $K(x, x) = \text{var}[f(x)]$.

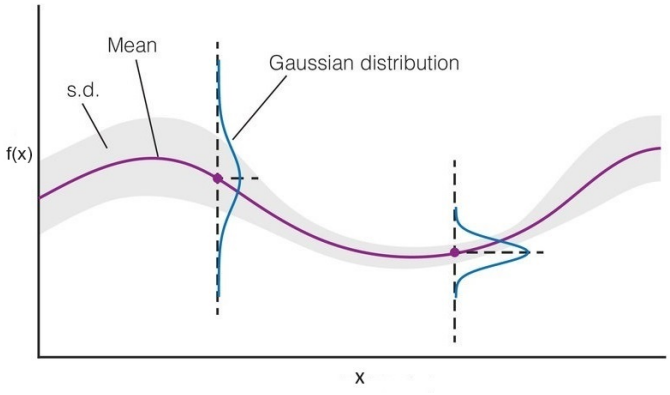


Figure 2.4: A GP *prior*, i.e. there has been no inference done yet. The purple line indicates the mean function $\mu(x)$. The gray area indicates 1 standard deviation of the variable ϕ at that location, with $\sigma = \sqrt{K(x, x)}$. Each blue Gaussian distribution indicates a univariate Gaussian of $f(x)$ at that point, i.e. for an unknown sample path f, the value $f(x)$ has this distribution. Taking multiple such slices at once gives a multivariate Gaussian. Adapted from [12].

Different mean functions correspond to different point-wise translations of the sample paths and most of the interesting properties are controlled by the covariance function instead. Because we don't know and don't want to assume too much about the function we are trying to model, we set the mean function to a constant function. Usually, this constant is 0, in which case we call the GP a *centered* GP.

The matrix $\mathbf{\Sigma} = K(\mathbf{x}, \mathbf{x})$ with entries $\mathbf{\Sigma}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ is called the *Gram matrix*. For the covariance function to be admissable, its Gram matrix must adhere to two requirements. First off, it must be *symmetric*. As we have seen, the covariance function is always symmetrical in its inputs, thus by construction so is the Gram matrix,. Second, the Gram matrix must be *positive definite*, i.e. it must have strictly positive eigenvalues. This is consistent with, but not implied by, $K(x, x) = \text{var}[f(x)] \geq 0$. If the Gram matrix is positive definite, we say the covariance function is positive definite as well.

The covariance function obviously has similar properties to covariance: If $K(x, x')$ is large (positive) for some x, x', then the values of $f(x)$ and $f(x')$ are strongly (positively) correlated and if $K(x, x')$ is small instead, then $f(x)$ and $f(x')$ have almost no correlation. If the covariance is negative,

there is a negative correlation between $f(x)$ and $f(x')$, so large positive values of $f(x)$ coincide with large negative values of $f(x')$ and vice versa. This means that if the covariance is large (positive or negative), the conditional uncertainty of $f(x$ given $f(x')$ is smaller and vice versa, as knowing one informs about the other. If $K(x, x') = 0$, there is no correlation at all. It can thus be loosely interpreted as a measure for how similar points $x, x' \in \mathcal{X}$ in the domain are.

2.5 Gaussian Process Inference

Performing inference on GPs is central to BO techniques. We've seen that GPs are closely linked to Gaussian distributions, and now we will see that exact inference, meaning inference with exact data points, is also closely linked to the conditioning on Gaussians. We write $\mathbf{x} = (x_1, \ldots, x_k)^\top$ for the vector of sampled locations, $\boldsymbol{\phi} = f(\mathbf{x})$ for the vector of random variables of the corresponding function values. Even though we have observed the value of f^* and thus $\boldsymbol{\phi}$ is not really unknown, we are interested in the probability of the evidence under the prior belief on f^*, thus we first treat $\boldsymbol{\phi}$ as a random variable. We will write $\boldsymbol{\phi}^*$ for the actual, totally determined observed values, and $\mathcal{D} = (\mathbf{x}, \boldsymbol{\phi}^*)$ for the total data set, which also does not contain any uncertaint. Suppose our prior on f is $p(f) = \mathcal{GP}(f; \mu, K)$. Appealing to this GP assumption on f, we can write

$$p(\boldsymbol{\phi} \,|\, \mathbf{x}) = \mathcal{N}(\boldsymbol{\phi}; \mu(\mathbf{x}), K(\mathbf{x}, \mathbf{x}))$$

Here $\mu(x)$ and $K(x, x')$ are the prior mean and prior covariance function respectively. For any $x \in \mathcal{X}$, for the cross-variance vector we write

$$K(\mathbf{x}, x) = \big(K(x_1, x), \ldots, K(x_k, x)\big)^\top$$

Note that this is also a function of x and that $K(\mathbf{x}, x)^\top = K(x, \mathbf{x})$. The joint distribution of f and $\boldsymbol{\phi}$ is given by

$$p(f, \boldsymbol{\phi} \,|\, \mathbf{x}) = \mathcal{GP}\left(\begin{bmatrix} f \\ \boldsymbol{\phi} \end{bmatrix}; \begin{bmatrix} \mu(x) \\ \mu(\mathbf{x}) \end{bmatrix}, \begin{bmatrix} K(x, x') & K(x, \mathbf{x}) \\ K(\mathbf{x}, x') & K(\mathbf{x}, \mathbf{x}) \end{bmatrix}\right)$$

We can then write

$$p(f \,|\mathcal{D}) = p(f \,|\, \mathbf{x}, \boldsymbol{\phi} = \boldsymbol{\phi}^*) = \mathcal{GP}(f; \mu_\mathcal{D}, K_\mathcal{D})$$

The *posterior mean function* $\mu_\mathcal{D}$ and the *posterior covariance function* $\boldsymbol{\Sigma}_\mathcal{D}$ are given by

$$\begin{aligned} \mu_\mathcal{D}(x) &= \mu(x) + K(x, \mathbf{x}) K(\mathbf{x}, \mathbf{x})^{-1} \big(\boldsymbol{\phi}^* - \mu(\mathbf{x})\big) \\ K_\mathcal{D}(x, x') &= K(x, x') - K(x, \mathbf{x}) K(\mathbf{x}, \mathbf{x})^{-1} K(\mathbf{x}, x') \end{aligned} \tag{2.3}$$

Here $K(\mathbf{x}, \mathbf{x})^{-1}$ indicates the inverse of the matrix $K(\mathbf{x}, \mathbf{x})$. Recalling the inference on Gaussians, we can see clear similarities. Taking x_1 as a single Gaussian random variable and $\mathbf{x}_2$ as a vector (collection) of Gaussian random variables:

$$p(x_1, \mathbf{x}_2) = \mathcal{N}\left(\begin{bmatrix} x_1 \\ \mathbf{x}_2 \end{bmatrix}; \begin{bmatrix} \mu_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}\right)$$

Then, as we have seen, the probability of x_1 given $\mathbf{x}_2 = \mathbf{a}$ is given by

$$p(x_1 | \mathbf{x}_2 = \mathbf{a}) = \mathcal{N}(x_1; \boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{11|2})$$

$$\mu_{1|2} = \mu_1 + \mathbf{\Sigma}_{12}\mathbf{\Sigma}_{22}^{-1}(\mathbf{a} - \boldsymbol{\mu}_2), \qquad \Sigma_{11|2} = \Sigma_{11} - \mathbf{\Sigma}_{12}\mathbf{\Sigma}_{22}^{-1}\mathbf{\Sigma}_{21}$$

The notable difference between this Gaussian case and the GP case is of course that f is a random variable of functions instead of points.

We will call $\mathcal{GP}(f; \mu_{\mathcal{D}}, K_{\mathcal{D}})$ the posterior, which we will use to intelligently add one point to $\mathcal{D}$, upon which we repeat this process.

Suppose at the very start we have a prior $\mathcal{GP}(f; \mu_{\mathcal{D}_0}, K_{\mathcal{D}_0})$ and some data set $\mathcal{D}_n$ and the same data set with one additional point $\mathcal{D}_{n+1}$. Note that conditioning the prior $\mathcal{GP}(f; \mu_{\mathcal{D}_n}, K_{\mathcal{D}_n})$ on $\mathcal{D}_{n+1}$ is the same as conditioning the original prior $\mathcal{GP}(f; \mu_{\mathcal{D}_0}, K_{\mathcal{D}_0})$ on $\mathcal{D}_{n+1}$. In general, the order of conditioning does not matter, so using a data set instead of an order data tuple is completely valid.

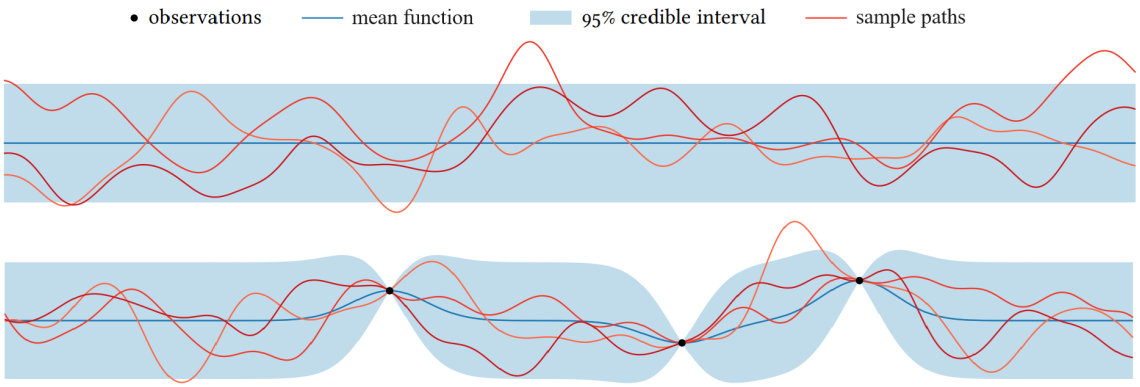


Figure 2.5: The top shows a centered prior GP with the Matérn 5/2 covariance function. The bottom show the posterior GP after inference with 3 observation points. The lines colored in shades of red show what a sample path from the corresponding GPs typically look like. Note that after incorparating the observations, the uncertainty near the observations is smaller [4].

2.6 Covariance functions

Covariance functions can be categorized by the following characteristics. When a covariance function $K(x, x')$ is only dependent on the difference $x - x'$, we call it *stationary*, and we write $K(x - x')$. This means the function is invariant under translation: $K(x + a, x' + a) = K(x + a - x' - a) = K(x - x') = K(x, x')$. If additionally, the covariance function depends only on the Euclidean distance $d = \|x - x'\|$, we call the covariance function *isotropic*, and we write $K(d)$. The function is then not only invariant under translation but also under rotation.

2.6.1 Matérn Family

The Matérn family of covariance functions, denoted K_{M}, contains isotropic functions and can model any degree of differentiability of the corresponding sample paths. The family is parametrized by $\nu \in (0, \infty]$, with centered GP sample paths being $\lceil \nu \rceil - 1$ times continuously differentiable *almost*

surely (a.s.). Recall a.s. means with probability 1. If $\nu \in \{k + \frac{1}{2} : k \in \mathbb{N}\}$, the covariance function is given by

$$K_{\mathrm{M}}(d; \nu) = \frac{2^{1-\nu}}{\Gamma(\nu)} \cdot (\sqrt{2\nu}\, d)^{\nu} \cdot K_{\nu}(\sqrt{2\nu}\, d).$$

Here, Γ is the Gamma function and K_{ν} is the modified Bessel function of the second kind. There are two special cases, $\nu = \frac{1}{2}$ and $\nu = \infty$.

For $\nu = \frac{1}{2}$, centered GP sample paths are (merely) a.s. continuous. The covariance function reduces to the *exponential covariance*:

$$K_{\mathrm{M}}(d; \nu = \tfrac{1}{2}) = e^{-d}.$$

For $\nu = \infty$, centered GP sample paths are a.s. infinitely continuously differentiable. The covariance function reduces to (using the above expression and the limit of $\nu \to \infty$) the *squared exponential covariance*, noted K_{SE}:

$$K_{\mathrm{M}}(d; \nu = \infty) = K_{\mathrm{SE}}(d) = e^{-\frac{1}{2}d^2}.$$

Other common values for ν are $\nu = \frac{3}{2}$ and $\nu = \frac{5}{2}$. Note that values of ν are usually expressed as $\frac{n}{2}$ for some n, because the covariance function is easily expressed when this is the case.

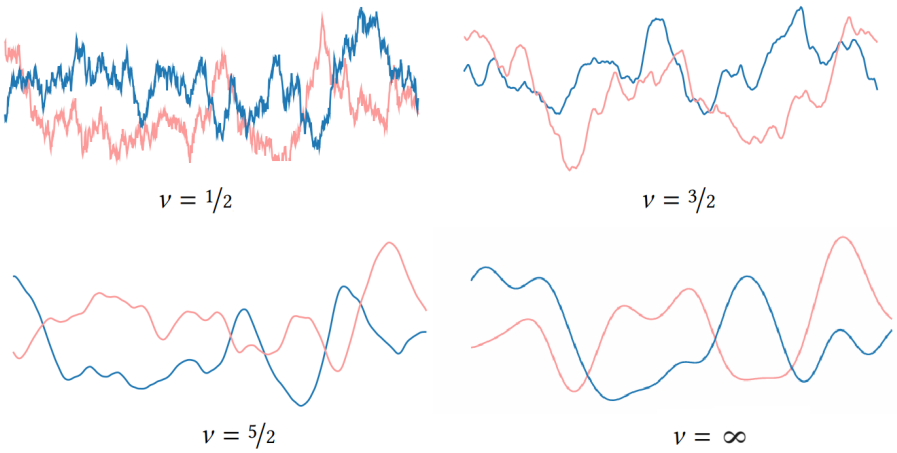


Figure 2.6: Sample paths from centered GPs with standard Matérn covariance functions, with the corresponding ν-value displayed underneath. The larger ν, the smoother the function is. [4]

As they are defined now, the Matérn covariance functions have arbitrary scale, $K_{\mathrm{M}}(x, x) = K(0) = 1$ (scale of output) and $K_{\mathrm{M}}(1) \approx 0.5$ (scale of characteristic length) for example. We can use *output scaling* and *dilation*, which are linear maps that transform the scale of the output and characteristic length respectively. We can easily compute for any random function $f : \mathcal{X} \to \mathbb{R}$ with covariance function K_f that

$$K_{\lambda f}(x, x') = \mathrm{cov}[\lambda f(x), \lambda f(x')] = \lambda^2 \mathrm{cov}[f(x), f(x')] = \lambda^2 K_f(x, x') \tag{2.4}$$

Thus any valid covariance function can be scaled and still be a valid covariance function. We can thus rescale a Matérn covariance function:

$$K'_{\mathrm{M}}(d;\nu) = \lambda^2 K_{\mathrm{M}}(d;\nu).$$

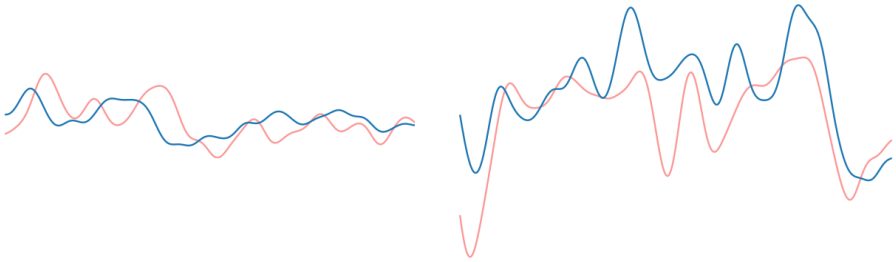


Figure 2.7: All lines depict sample paths from centered GPs with the same covariance function, except for the output scale, which is smaller on the left and bigger on the right. [4]

We call $\ell \in \mathbb{R}$ the *characteristic length*, which we can set to a desired value using dilation. The dilation map is then given by $\delta : x \mapsto x/\ell$ such that $\tilde{d} = \|\delta(x) - \delta(x')\| = \|x/\ell - x/\ell'\| = \|x - x'\|/\ell = d/\ell$. First applying dilation and then the covariance function yields

$$K'(d) = K(\tilde{d}) = K(d/\ell).$$

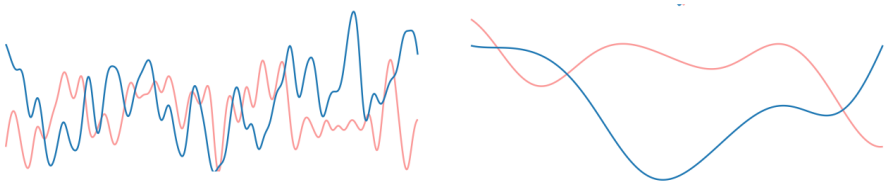


Figure 2.8: All lines depict sample paths from centered GPs with the same covariance function, except for the characteristic length scale, which is smaller on the left and bigger on the right. [4]

Thus the Matérn covariance function can be extended to a family parametrized by three quantities: the degree of smoothness ν, the output scale λ & the characteristic length scale ℓ. For $\nu = \frac{1}{2}$ and $\nu = \infty$, we get

$$K'_{\mathrm{M}}(d, \lambda, \ell; \nu = \tfrac{1}{2}) = \lambda^2 e^{-d/\ell}, \qquad K'_{\mathrm{M}}(d, \lambda, \ell; \nu = \infty) = K'_{\mathrm{SE}}(d, \lambda, \ell) = \lambda^2 e^{-\frac{1}{2}d^2/\ell^2}$$

The same qualities still hold regarding continuity and differentiability.

If we expect linear behavior in the objective function f, we use the linear regression model,

$$f(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}^\top \mathbf{x}$$

Treating f as a random variable then reduces to treating $\beta, \boldsymbol{\beta}$ as random variables. Let the priors on β and $\boldsymbol{\beta}$ be given by

$$p(\beta) = \mathcal{N}(\beta; a, b^2), \qquad p(\boldsymbol{\beta}) = \mathcal{N}(\beta; \mathbf{a}, \mathbf{B})$$

The covariance function that models this behavior is the linear covariance function, in this case given by

$$K_{\text{LIN}}(\mathbf{x}, \mathbf{x}'; b, \mathbf{B}) = b^2 + \mathbf{x}^\top \mathbf{B} \mathbf{x}'$$

For a GP with a linear covariance function, $\mathcal{GP}(f; \mu, K_{\text{LIN}})$, a sample path is always a linear function.

A periodic function $f : \mathbb{R} \to \mathbb{R}$ with period p can be modeled by warping the input to the unit circle with radius $r = p/(2\pi)$, for example with the map $x \mapsto \begin{bmatrix} r\cos(x) & r\sin(x) \end{bmatrix}^\top$. Employing a Matérn covariance function after this warping gives a somewhat periodic function. Other formulations of periodic covariance functions exist [David Mackay via The Kernel Cookbook].

Both the linear and periodic covariance functions are not often used by themselves, but they can be combined to create more interesting covariance functions.

2.7 Combining covariance functions

For any two random, independent functions $f, g : \mathcal{X} \to \mathbb{R}$ (so not necessarily from GPs), we have $\text{cov}[f(x), g(x')] = \text{cov}[g(x), f(x')] = 0 \ \forall x \in \mathcal{X}$. If we write $K_f(x, x') := \text{cov}[f(x), f(x')]$ and $K_g(x, x') := \text{cov}[g(x), g(x')]$ for the corresponding covariance functions, we have the following result for the pointwise sum $(f + g)(x) = f(x) + g(x)$:

$$\begin{aligned} K_{f+g}(x, x') &= \text{cov}[f(x) + g(x), f(x') + g(x')] \\ &= \text{cov}[f(x), f(x')] + \text{cov}[g(x), g(x')] + \cancel{\text{cov}[f(x), g(x')]} + \cancel{\text{cov}[g(x), f(x')]} \\ &= \text{cov}[f(x), f(x')] + \text{cov}[g(x), g(x')] \\ &= K_f(x, x') + K_g(x, x') \end{aligned}$$

If additionally f, g are centered, i.e. $\mathbb{E}[f(x)] = 0 = \mathbb{E}[g(x)] \ \forall x \in \mathcal{X}$, then we have the following result for the pointwise product $(fg)(x) = f(x)g(x)$:

$$\begin{aligned} K_{fg}(x, x') &= \text{cov}[f(x)g(x), f(x')g(x')] \\ &= \cancel{\mathbb{E}[f(x)]} \cdot \cancel{\mathbb{E}[f(x')]} \cdot \text{cov}[g(x), g(x')] + \cancel{\mathbb{E}[g(x)]} \cdot \cancel{\mathbb{E}[g(x')]} \cdot \text{cov}[f(x), f(x')] \\ &\quad + \text{cov}[f(x), f(x')]\text{cov}[g(x), g(x')] \\ &= K_f(x, x') \cdot K_g(x, x') \end{aligned}$$

Together with result (2.4) we conclude that any polynomial of valid covariance functions is a valid covariance function. We can combine covariance functions to model different behaviors simultaneously, for example to encode additive behavior at different length scales.

If the two covariance functions are nonnegative and have roughly the same length scales, the addition can be thought of as a logical disjunction ($a \vee b$): the sum will be nontrivial when any of the two originals aren't. The product of two covariance functions can be thought of as logical conjuction ($a \wedge b$): the product will be nontrivial when both of the originals aren't.

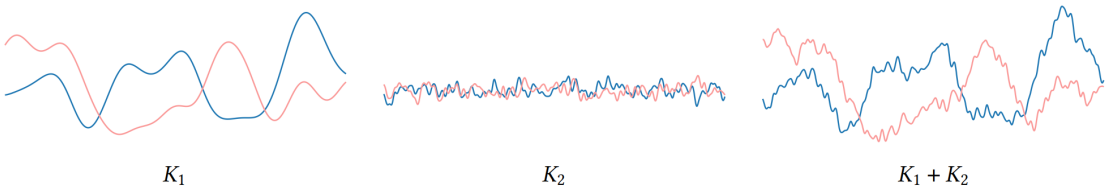


Figure 2.9: This figure shows samples from centered GPs with three covariance functions, the third being the sum of the previous two as indicated. This illustrates how two covariance functions (both Matérn with $\nu = \infty$), one modeling larger scale characteristics and one modeling smaller scale characteristics, can come together to model both simultaneously. [4]

2.8 Utility functions

A decision policy guides which actions should be taken at any point in the optimization. We use a *utility function*, whose role is to express our preference for certain outcomes over others. In optimization, the goal is to return a final recommendation, based on the data set at termination, $\mathcal{D}_\tau$. Thus, for optimization, a natural way of defining the utility is in terms of the data set $\mathcal{D}$, were we set $D(\varnothing) = -\infty$. Our goal is to optimize the utility of the final data set as it contains our final recommendation. Because BO is based on probabilities, we resort to thinking about *expected utility*.

Optimal policies are those that maximize the expected utility, specifically, the expected utility of the terminal recommendation should be maximized. Basically they are optimal in the average case. For us to execute an optimal policy, we would need every step to be optimal. Because at any step we don't know how the remaining steps will unfold, we could assume all future steps to be optimal as well. This is called the *principle of optimality*. The result would be the optimal policy, but it's generally too expensive to compute (you would need to look many steps ahead). Thus we need to make an approximation to the optimal policy. We do this with *limited look ahead*, which puts a limit on the number of future decisions considered. We define a decision horizon ℓ, which determines how many future steps we consider to be optimal. In the most optimal case, we would have $\ell = \tau$, as we optimize for the utility of $\mathcal{D}_\tau$. In limited look ahead, instead of optimizing for the expected utility of $\mathcal{D}_\tau$, we optimize for the expected utility after an additional ℓ steps, $\mathcal{D}_\ell$. In light of computational restrictions, the decision horizon used is usually some smaller number, often one or two. We will consider one-step look ahead only, in which case we note $\mathcal{D}_{\ell=1}$ as $\mathcal{D}'$.

We treat our objective function as a random variable which in most places has uncertainty. The expected utility of one location may be higher, but the uncertainty at that location may also be large, i.e. there is a large *risk*. If the approach does not consider the uncertainty, it is *risk neutral*.

In our case, we choose for our final recommendation to always be a point we have already sampled. We won't recommend a point we are not 100% sure about, even if our final hypothesis thinks somewhere we've not sampled probably has higher target. In this way, our final recommendation should be risk-free.

2.9 Acquisition functions

Acquisition functions $\alpha(x; \mathcal{D})$ are functions that for data set $\mathcal{D}$ use the corresponding posterior belief of the objective function $p(f \mid \mathcal{D})$ to assign a score to each possible next sampling point x in the domain $\mathcal{X}$. Wherever in the domain the acquisition function attains its maximum will be the suggested next sample point. Note that the maximum is not necessarily unique, in which case a random choice is made, making working with an acquisition function (and thus BO) inherently random. The acquisition function should be easy to work with, we might even have an analytical expression of it and its gradient, and the maximum can usually be found by inexpensive numerical techniques.

One point of attention is the difference in cost when sampling in different places. With the same cost, sampling at multiple cheap locations could be more beneficial than sampling at fewer expensive locations. Thus, if sampling at one location is much cheaper compared to some other location, it might be more favorable to sample there. In our use case, we expect the COMSOL model to take longer if the for example the radius is larger, as we've seen this behavior empirically. However, the cost also seems somewhat random, almost unpredictable. Each parameter almost certainly affects the average computation time. However, we just don't know how exactly the computation time scales with certain single parameters, not even if it scales monotonically. We also don't know how changing all parameters simultaneously affects the computation time, etcetera. With this many unknowns and no way to anticipate the relative cost between points, it's reasonable to disregard the cost entirely, which also considerably simplifies the material.

2.10 Exploration vs Exploitation

One of the strengths of BO is that at each step we are making a (somewhat) informed decision on where to sample next, based on the posterior. One open question that remains is that of exploration vs exploitation. Suppose we have sampled a point with a rather high target. The posterior will then also be high in its immediate area. An exploitation-focused policy would advise to keep sampling points similar to (similar in terms of the covariance function, for Matérn this would be 'close to') the already known point and exploit the fact that targets are very likely to be high. After all, the true maximum could very well be similar to this point too. An exploration-focused policy would advise to keep exploring the entire space, even if these other regions are less promising. Just because you hit gold here, does not mean there won't be more gold somewhere else. Both exploration and exploitation have their merits, and both have their downsides. When pursuing an exploitation-focused policy, there is a high chance you get stuck in the local optimum and never find the true optimum, which might be many times higher. When pursuing an exploration-focused policy, there is a high chance you sample points with low targets, which in hindsight can be seen as wasteful. Whenever choosing an acquisition function, it should be considered how it handles the exploration-exploitation dilemma.

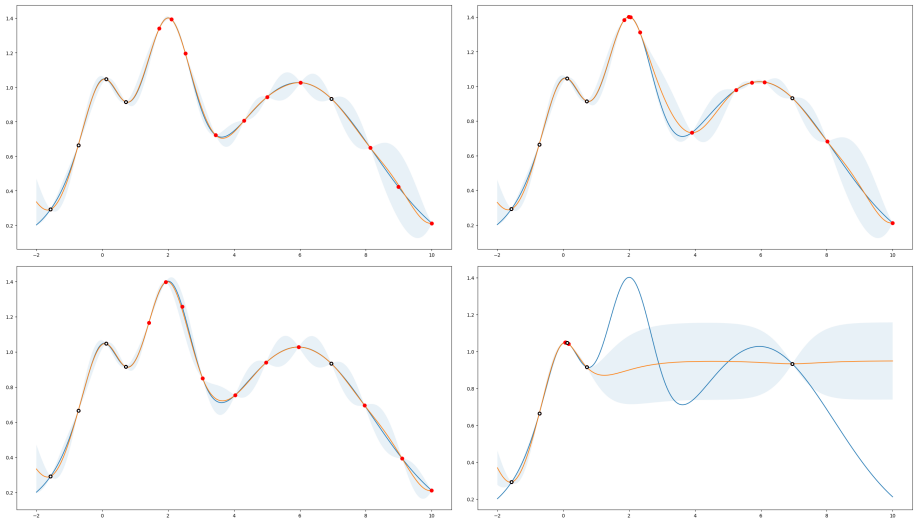


Figure 2.10: Exploration (Left column) vs Exploitation (Right column). All show BO of 10 points with the black and white dots representing the initial data set, and a centerd GP prior with the Matérn 5/2 covariance function. The objective function is in blue, the posterior mean in orange, and the shaded area is the posterior 95% credible interval. The BO decision policy is based on the EI acquisition function (Top) and UCB acquisition function (Bottom) [6].

Figure 2.10 shows an example to illustrate how either one can succeed and fail. Note that though the acquisition function is different in each case, this is not the deciding factor and for the sake of illustration can be ignored. You can see the exploration-focused policies spread their sampled points evenly, whereas the exploitation-focused policies bunch their sampled points close to local optima. In the top case, exploitation has performed better, as its highest target is higher than exploration. In the bottom case, exploitation was really stuck around the local optima found by the initial data set as all sampled points are close here. As a result, exploration performed much better in the second case.

2.11 Improvement based policies

Let $\mathcal{D} = (\mathbf{x}, \boldsymbol{\phi})$ be the current data set, and for the next sample point and resulting observation, note x and ϕ respectively. Then the data set after the an observation is noted $\mathcal{D}' = (\mathbf{x}', \boldsymbol{\phi}') = \mathcal{D} \cup (x, \phi)$. With one step look ahead, we want to maximize the increase in utility looking one step ahead, i.e. maximize the increase in utility between data set $\mathcal{D}$ and data set $\mathcal{D}'$. We call this increase the *expected marginal gain*. Because we don't yet know $\mathcal{D}'$, we again resort to the expected value. With one step look ahead, an acquisition function based on the expected marginal gain is then defined as:

$$\alpha(x; \mathcal{D}) = \mathbb{E}[u(\mathcal{D}') \,|x, \mathcal{D}] - u(\mathcal{D}) = \mathbb{E}[u(\mathcal{D}') - u(\mathcal{D}) \,|x, \mathcal{D}] \tag{2.5}$$

Here α is a function of x, and it is treated as a given in the expectation. The real unknown is thus the resulting value ϕ at x. Depending on how you define the utility, different acquisition functions emerge, and some of the most common ones will be introduced below.

2.11.1 Simple reward & Expected Improvement

As we've said before, we want our final recommendation to be one of the sampled points. Thus it is natural to define the utility of a data set in terms of the sampled points it contains, where our preference goes out to the highest target value. The *simple reward* utility function expressed precisely this idea. If we define $\widetilde{\phi} = \max_i \boldsymbol{\phi}_i$, it is given by

$$u(\mathcal{D}) = \max_i \boldsymbol{\phi}_i = \widetilde{\phi}, \qquad u(\mathcal{D}') = \max\{\phi, u(\mathcal{D})\} = \max\{\phi, \widetilde{\phi}\}$$

This utility function is risk-neutral per construction.

One-step look ahead with the simple reward utility function results in the *Expected Improvement* (EI) acquisition function. Denoted α_{EI}, this acquisition function is given by

$$\begin{aligned}
\alpha_{\text{EI}}(x; \mathcal{D}) &= \mathbb{E}[\max\{\phi, \widetilde{\phi}\} - \widetilde{\phi} \,|\, x, \mathcal{D}] \\
&= \mathbb{E}[\max\{\phi - \widetilde{\phi}, 0\} | \, x, \mathcal{D}] \\
&= \int_{-\infty}^{\infty} \max\{\phi - \widetilde{\phi}, 0\} \cdot p(\phi \,|\, x, \mathcal{D}) \, \mathrm{d}\phi \\
&= \int_{-\infty}^{\infty} \max\{\phi - \widetilde{\phi}, 0\} \cdot \mathcal{N}\big(\phi; \mu_{\mathcal{D}}(x), K_{\mathcal{D}}(x,x)\big) \, \mathrm{d}\phi \\
&= \int_{\widetilde{\phi}}^{\infty} (\phi - \widetilde{\phi}) \cdot \mathcal{N}\big(\phi; \mu_{\mathcal{D}}(x), K_{\mathcal{D}}(x,x)\big) \, \mathrm{d}\phi \\
&= \int_{\widetilde{\phi}}^{\infty} \phi \cdot \mathcal{N}\big(\phi; \mu_{\mathcal{D}}(x), K_{\mathcal{D}}(x,x)\big) \, \mathrm{d}\phi - \widetilde{\phi} \int_{\widetilde{\phi}}^{\infty} \mathcal{N}\big(\phi; \mu_{\mathcal{D}}(x), K_{\mathcal{D}}(x,x)\big) \, \mathrm{d}\phi
\end{aligned}$$

The first expression is an expected value shifted by $\widetilde{\phi}$, and the second expression is a complementary CDF (i.e. 1 - CDF) scaled by $\widetilde{\phi}$. Note that $1 - \Phi(x') = \int_{x'}^{\infty} \mathcal{N}(x; 0, 1^2) \, \mathrm{d}x$, and that the standard normal is symmetric around 0, so $1 - \Phi(x') = \Phi(-x')$.

We first consider $K_{\mathcal{D}}(x,x) > 0$:

$$\alpha_{\text{EI}}(x; \mathcal{D}) = \big(\mu_{\mathcal{D}}(x) - \widetilde{\phi}\big) \cdot \Phi\left(\frac{\mu_{\mathcal{D}}(x) - \widetilde{\phi}}{\sqrt{K_{\mathcal{D}}(x,x)}}\right) + \sqrt{K_{\mathcal{D}}(x,x)} \cdot \mathcal{N}\left(\frac{\mu_{\mathcal{D}}(x) - \widetilde{\phi}}{\sqrt{K_{\mathcal{D}}(x,x)}}; 0, 1^2\right)$$

In canonical notation, where $\phi(x) = \mathcal{N}(x; 0, 1^2)$ (the overlap in notation is unfortunate), this would be

$$\alpha_{\text{EI}}(x; \mathcal{D}) = \big(\mu_{\mathcal{D}}(x) - \widetilde{\phi}\big) \cdot \Phi\left(\frac{\mu_{\mathcal{D}}(x) - \widetilde{\phi}}{\sqrt{K_{\mathcal{D}}(x,x)}}\right) + \sqrt{K_{\mathcal{D}}(x,x)} \cdot \phi\left(\frac{\mu_{\mathcal{D}}(x) - \widetilde{\phi}}{\sqrt{K_{\mathcal{D}}(x,x)}}\right)$$

Expected improvement is remarkable in the way it handles the exploration-exploitation dilemma. At points similar to an observed point with a high target, the GP posterior will naturally have a higher mean function and smaller uncertainty bands. The high mean function could suggest there is a good opportunity for improvement, but because of the small uncertainty band, the size of that improvement is expected to be rather small. As a result, EI will never over-exploit in the sense that it will always sample somewhere at least somewhat dissimilar from the observations, as the uncertainty bands and thus the possible improvement are higher, but still considers how the

observations have shaped our belief to find a good sampling point. Empirically, this has proven to be a good balance in the exploration-exploitation trade-off, being somewhat exploitation-heavy.

As $K_{\mathcal{D}}(x,x) \to 0$, then $\Phi\left(\frac{\mu_{\mathcal{D}}(x)-\widetilde{\phi}}{\sqrt{K_{\mathcal{D}}(x,x)}}\right) \to \mathbb{1}_{\{\mu_{\mathcal{D}}(x)-\widetilde{\phi}>0\}}$, so if $K_{\mathcal{D}}(x,x) = 0$, the first term becomes $\max\{\mu_{\mathcal{D}}(x) - \widetilde{\phi}, 0\}$, and the second term falls away, so we get $\alpha_{\text{EI}} = \max\{\mu - \widetilde{\phi}, 0\}$.

We can analyze the gradient of α_{EI} to gain a deeper understanding of why it works great. Let us use the short hands $\mu = \mu_{\mathcal{D}}(x)$ and $\sigma = \sqrt{K_{\mathcal{D}}(x,x)}$ so that we can write

$$\alpha_{\text{EI}}(x;\mathcal{D}) = (\mu - \widetilde{\phi}) \cdot \Phi\left(\frac{\mu - \widetilde{\phi}}{\sigma}\right) + \sigma \cdot \phi\left(\frac{\mu - \widetilde{\phi}}{\sigma}\right)$$

$$\implies \frac{\partial \alpha_{\text{EI}}}{\partial \mu} = \Phi\left(\frac{\mu - \widetilde{\phi}}{\sigma}\right) > 0, \qquad \frac{\partial \alpha_{\text{EI}}}{\partial \sigma} = \phi\left(\frac{\mu - \widetilde{\phi}}{\sigma}\right) > 0.$$

Employing the chain rule yields

$$\frac{\partial \alpha_{\text{EI}}}{\partial x} = \frac{\partial \alpha_{\text{EI}}}{\partial \mu}\frac{\partial \mu}{\partial x} + \frac{\partial \alpha_{\text{EI}}}{\partial \sigma}\frac{\partial \sigma}{\partial x} = \Phi\left(\frac{\mu - \widetilde{\phi}}{\sigma}\right)\frac{\partial \mu}{\partial x} + \phi\left(\frac{\mu - \widetilde{\phi}}{\sigma}\right)\frac{\partial \sigma}{\partial x}$$

The EI acquisition function is thus monotonically increasing in both μ and σ. This perfectly shows the built-in balance in the exploration-exploitation dilemma. Where the gradient $\frac{\partial \alpha_{\text{EI}}}{\partial x}$ points to (i.e. is largest) is where it's more likely to give a recommendation. The direction where the expected value μ increases, i.e. $\frac{\partial \mu}{\partial x}$ is largest, is in the direction where it's more likely points get recommended, which points to exploitation, as we are using the belief of our posterior function. The direction where the uncertainty σ increases, i.e. $\frac{\partial \sigma}{\partial x}$ is largest, is also in the direction where it's more likely points get recommended, which points to exploration, as were sampling points with high uncertainty [4].

Putting this back in terms of $\mu_{\mathcal{D}}$ and $K_{\mathcal{D}}$, we get

$$\begin{aligned}\frac{\partial \alpha_{\text{EI}}}{\partial x} &= \Phi\left(\frac{\mu_{\mathcal{D}}(x) - \widetilde{\phi}}{\sqrt{K_{\mathcal{D}}(x,x)}}\right)\frac{\partial \mu_{\mathcal{D}}(x)}{\partial x} + \phi\left(\frac{\mu_{\mathcal{D}}(x) - \widetilde{\phi}}{\sqrt{K_{\mathcal{D}}(x,x)}}\right)\frac{\partial \sqrt{K_{\mathcal{D}}(x,x)}}{\partial x} \\ &= \Phi\left(\frac{\mu_{\mathcal{D}}(x) - \widetilde{\phi}}{\sqrt{K_{\mathcal{D}}(x,x)}}\right)\frac{\partial \mu_{\mathcal{D}}(x)}{\partial x} + \phi\left(\frac{\mu_{\mathcal{D}}(x) - \widetilde{\phi}}{\sqrt{K_{\mathcal{D}}(x,x)}}\right)\frac{1}{2\sqrt{K_{\mathcal{D}}(x,x)}}\frac{\partial K_{\mathcal{D}}(x,x)}{\partial x}\end{aligned}$$

Let us define tuning parameters

$$\rho(x) = \Phi\left(\frac{\mu_{\mathcal{D}}(x) - \widetilde{\phi}}{\sqrt{K_{\mathcal{D}}(x,x)}}\right) > 0, \qquad \kappa(x) = \phi\left(\frac{\mu_{\mathcal{D}}(x) - \widetilde{\phi}}{\sqrt{K_{\mathcal{D}}(x,x)}}\right)\frac{1}{2\sqrt{K_{\mathcal{D}}(x,x)}} > 0$$

$$\implies \frac{\partial \alpha_{\text{EI}}}{\partial x} = \rho(x)\frac{\partial \mu_{\mathcal{D}}(x)}{\partial x} + \kappa(x)\frac{\partial K_{\mathcal{D}}(x,x)}{\partial x}$$

The smaller $K_{\mathcal{D}}(x,x)$ is, the bigger $\kappa(x)$ and the more exploitative the policy becomes. Of course, the reverse is true when $K_{\mathcal{D}}(x,x)$ is larger. This points to a connection between the output scale of

the chosen covariance function and the balance between exploration and exploitation in Expected Improvement.

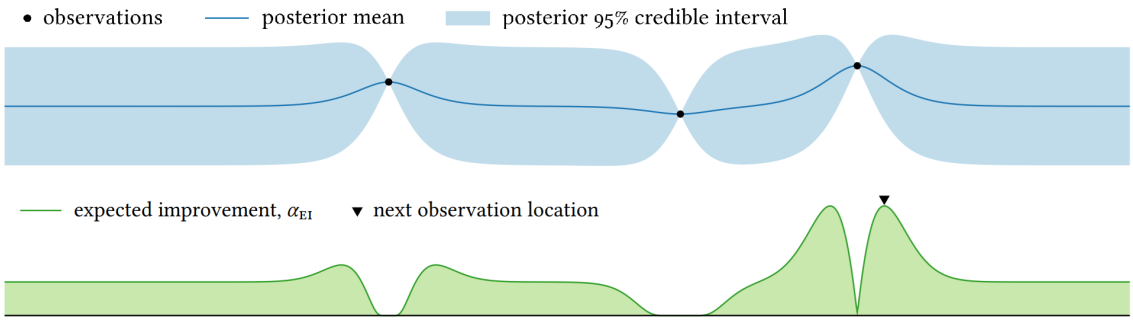


Figure 2.11: This shows the EI acquisition function (Bottom) based on a centered GP prior with the Matérn 5/2 covariance function with 3 data points (Top). Immediately next to the observed points, the acquisition function takes on small values, but the maximum is still quite close to the found maximum.

2.11.2 Modified expected improvement

One extension to EI that has been proposed and successfully implemented is Modified Expected Improvement (MEI). The goal is to gain more control over the exploration-exploitation tradeoff. We define a modified simple reward utility function $u(\mathcal{D}) = \max\{\phi, \widetilde{\phi} + \varepsilon\}$, with new controlling parameter ε. This means any improvement must exceed the inflated value $\widetilde{\phi} + \varepsilon$ instead of $\widetilde{\phi}$, with no increased utility if $\widetilde{\phi} + \varepsilon$ is not exceeded. The resulting acquisition function is then

$$\alpha_{\text{MEI}}(x, \varepsilon; \mathcal{D}) = \left(\mu_{\mathcal{D}}(x) - \widetilde{\phi} - \varepsilon\right) \cdot \Phi\left(\frac{\mu_{\mathcal{D}}(x) - \widetilde{\phi} - \varepsilon}{\sqrt{K_{\mathcal{D}}(x,x)}}\right) + \sqrt{K_{\mathcal{D}}(x,x)} \cdot \phi\left(\frac{\mu_{\mathcal{D}}(x) - \widetilde{\phi} - \varepsilon}{\sqrt{K_{\mathcal{D}}(x,x)}}\right)$$

The larger ε is, the more suggestions in regions with large uncertainty are encouraged, as these regions have larger potential gains. Thus increasing ε encourages exploration. In this framework, the traditional EI acquisition function coincides with the MEI acquisition function with $\varepsilon = 0$, as $\alpha_{\text{MEI}}(x, \varepsilon = 0; \mathcal{D}) = \alpha_{\text{EI}}(x;)$

2.11.3 Dynamic modified expected improvement

Intuitively, early on the algorithm should explore as many regions are unknown and the current belief on the objective function is weak, whereas later on the algorithm should exploit as we should have built a strong hypothesis about the objective function and can use this to sample efficiently. To achieve an algorithm that follows these principles, the dynamic setting of the exploration parameter ε in MEI has been proposed [MOCKUS]. There is no precise formulation so our implementation of this is somewhat guesswork. We propose a Dynamic Modified Expected Improvement, where the exploration parameter is linearly dependent on the number of iterations, where it starts with

maximum exploration by setting ε as large as feasible, and ends with maximum exploitation, so $\varepsilon = 0$. Our acquisition function would then be

$$\alpha_{\text{DMEI}}(x, \varepsilon, n; \mathcal{D}) = \big(\mu_{\mathcal{D}}(x) - \widetilde{\phi} - \varepsilon(n)\big) \cdot \Phi\left(\frac{\mu_{\mathcal{D}}(x) - \widetilde{\phi} - \varepsilon(n)}{\sqrt{K_{\mathcal{D}}(x,x)}}\right) + \sqrt{K_{\mathcal{D}}(x,x)} \cdot \phi\left(\frac{\mu_{\mathcal{D}}(x) - \widetilde{\phi} - \varepsilon(n)}{\sqrt{K_{\mathcal{D}}(x,x)}}\right)$$

$$\varepsilon(n) = \varepsilon_{\max} \cdot \frac{(n_{\max} - n)}{n_{\max}} \implies \varepsilon(0) = \varepsilon_{\max} \qquad \varepsilon(n_{\max}) = 0$$

2.11.4 Global reward & Knowledge Gradient

Instead of judging a data set based on the achieved targets, we could also judge it on the maximum of the mean function of the induced posterior. This would mean we are no longer risk-neutral. The utility function that captures this idea is the *global reward* utility function, given by:

$$u(\mathcal{D}) = \max_{x \in \mathcal{X}} \mu_{\mathcal{D}}(x)$$

Plugging this into the expected marginal gain acquisition function, we get the *Knowledge Gradient* acquisition function:

$$\alpha_{\text{KG}}(x; \mathcal{D}) = \mathbb{E}[\max_{x \in \mathcal{X}} \mu_{\mathcal{D}'}(x) \,|\, x, \mathcal{D}]$$

This alternative is very interesting, especially if we would accept risk in our final recommendation. We won't explore the KG acquisition function any further here.

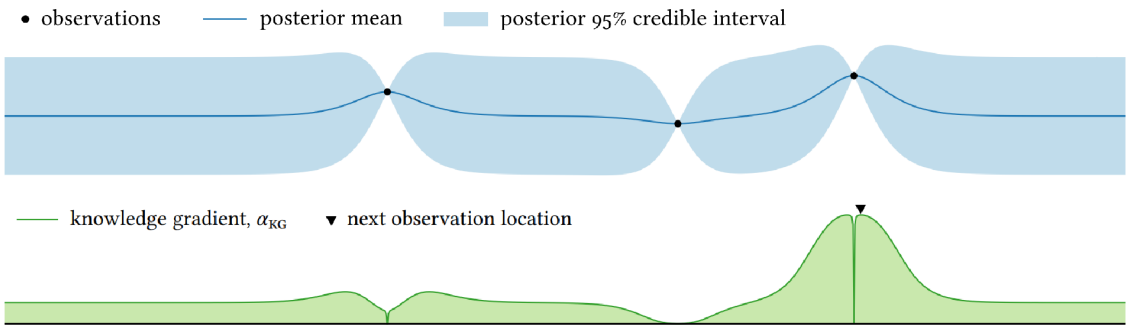


Figure 2.13: This shows the KG acquisition function (Bottom) based on a centered GP prior with the Matérn 5/2 covariance function with 3 data points (Top). Compared with EI, this has an (even) stronger exploitation bias.

2.11.5 Probability of Improvement

Instead of looking at the size of the improvement (like with EI and KG), we could also look at the *probability* of improvement, regardless of its size, as long as it's not too small. This leads to the aptly named *Probability of Improvement* (PoI or PI) acquisition function. It also is defined in terms of the by-now familiar simple reward

$$u(\mathcal{D}) = \max_i \boldsymbol{\phi}_i = \widetilde{\phi}, \qquad u(\mathcal{D}') = \max\{\phi, u(\mathcal{D})\} = \max\{\phi, \widetilde{\phi}\}$$

Given some *margin* $\varepsilon > 0$, the point with the highest probability that an observation improves the utility by at least ε is recommended. We denote the *utility threshold* by $\tau = u(\mathcal{D}) + \varepsilon = \widetilde{\phi} + \varepsilon$, also called the improvement threshold. The PoI acquisition function is then equal to the probability that $u(\mathcal{D}')$ is greater than τ:

$$\alpha_{\text{PI}}(x; \mathcal{D}, \tau) = \mathbb{P}[u(\mathcal{D}') > \tau \,|\, x, \mathcal{D}] = \mathbb{P}[\max\{\phi, \widetilde{\phi}\} > \tau \,|\, x, \mathcal{D}\}]$$

Looking at this expression, we can also view this policy as an application of (2.5), but with unit reward when the utility exceeds ε. Note that $\widetilde{\phi} = \tau + \varepsilon$, by construction, so we never have $\widetilde{\phi} > \tau$. Thus, we can simplify

$$\alpha_{\text{PI}}(x; \mathcal{D}, \tau) = \mathbb{P}[\phi > \tau \,|\, x, \mathcal{D}\}]$$

Noting that $p(\phi \,|\, x, \mathcal{D}) = \mathcal{N}(x; \mu_{\mathcal{D}}(x), K_{\mathcal{D}}(x, x))$ and that the above expression again is a complementary CDF of the standard normal, we can simply write

$$\alpha_{\text{PI}}(x; \mathcal{D}, \tau) = \Phi\left(\frac{\mu_{\mathcal{D}}(x) - \tau}{\sqrt{K_{\mathcal{D}}(x, x)}}\right)$$

Probability of improvement is set up in such a way that modest improvement with high certainty is preferable to a potentially huge improvement with lower certainty, making it more risk-averse than say EI, as can be seen in (**??**).

As $K_{\mathcal{D}}(x, x) \to 0$, then $\Phi\left(\frac{\mu_{\mathcal{D}}(x) - \tau}{\sqrt{K_{\mathcal{D}}(x,x)}}\right) \to \mathbb{1}_{\{\mu_{\mathcal{D}}(x) - \tau > 0\}}$, so if $K_{\mathcal{D}}(x, x) = 0$ we get $\alpha_{\text{PI}} = \mathbb{1}_{\{\mu_{\mathcal{D}}(x) > \tau\}}$.

We consider $K_{\mathcal{D}}(x, x) > 0$. Because Φ is a monotonically increasing function, to study the gradient of α_{PI} is equivalent to studying the gradient of the argument of Φ, which we'll denote $\widetilde{\alpha}_{\text{PI}}$:

$$\widetilde{\alpha}_{\text{PI}}(x; \mathcal{D}) = \frac{\mu_{\mathcal{D}}(x) - \tau}{\sqrt{K_{\mathcal{D}}(x, x)}}$$

Let us again use the short hands $\mu = \mu_{\mathcal{D}}(x)$ and $\sigma = \sqrt{K_{\mathcal{D}}(x, x)}$, so we get

$$\widetilde{\alpha}_{\text{PI}}(x; \mathcal{D}) = \frac{\mu - \tau}{\sigma} \implies \frac{\partial \widetilde{\alpha}_{\text{PI}}}{\partial \mu} = \frac{1}{\sigma} \quad \& \quad \frac{\partial \widetilde{\alpha}_{\text{PI}}}{\partial \sigma} = \frac{\tau - \mu}{\sigma^2}$$

$$\implies \frac{\partial \widetilde{\alpha}_{\text{PI}}}{\partial x} = \frac{\partial \widetilde{\alpha}_{\text{PI}}}{\partial \mu}\frac{\partial \mu}{\partial x} + \frac{\partial \widetilde{\alpha}_{\text{PI}}}{\partial \sigma}\frac{\partial \sigma}{\partial x} = \frac{1}{\sigma}\frac{\partial \mu}{\partial x} + \frac{\tau - \mu}{\sigma^2}\frac{\partial \sigma}{\partial x}$$

Since $\sigma > 0$, α_{PI} is monotonically increasing with μ everywhere, so exploitation is encouraged everywhere and scales with $\frac{1}{\sigma} = \frac{1}{\sqrt{K_{\mathcal{D}}(x,x)}}$. When the target value is greater than the predicted mean, $\tau > \mu$, α_{PI} is also increasing with σ, so exploration is encouraged. Interestingly, when $\tau < \mu$ we see α_{PI} is decreasing with σ, meaning in this case, exploration is discouraged. This is counter to our expectation: When the predicted mean is large, we could exploit this belief to find a large improvement. Instead in this regime, PoI looks for locations with smaller variance and more certainty, despite the probably smaller gain.

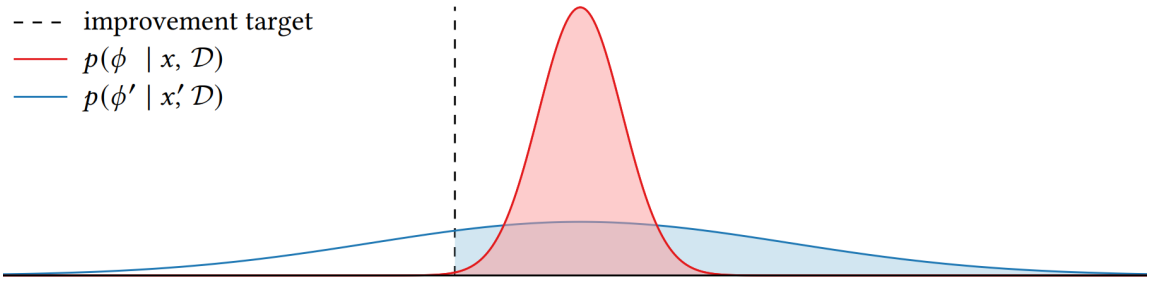


Figure 2.14: The distribution of the target at two observation locations x (red) and x' (blue). These distributions have the same mean but different variances. The dashed line is the improvement target $\tilde{\phi}$. The probability that the target is larger than the improvement target by some margin is quite large for x and smaller for x', thus PoI will prefer x over x'. However, the potential reward is larger for x' compared to x, and thus x' is preferable to EI.

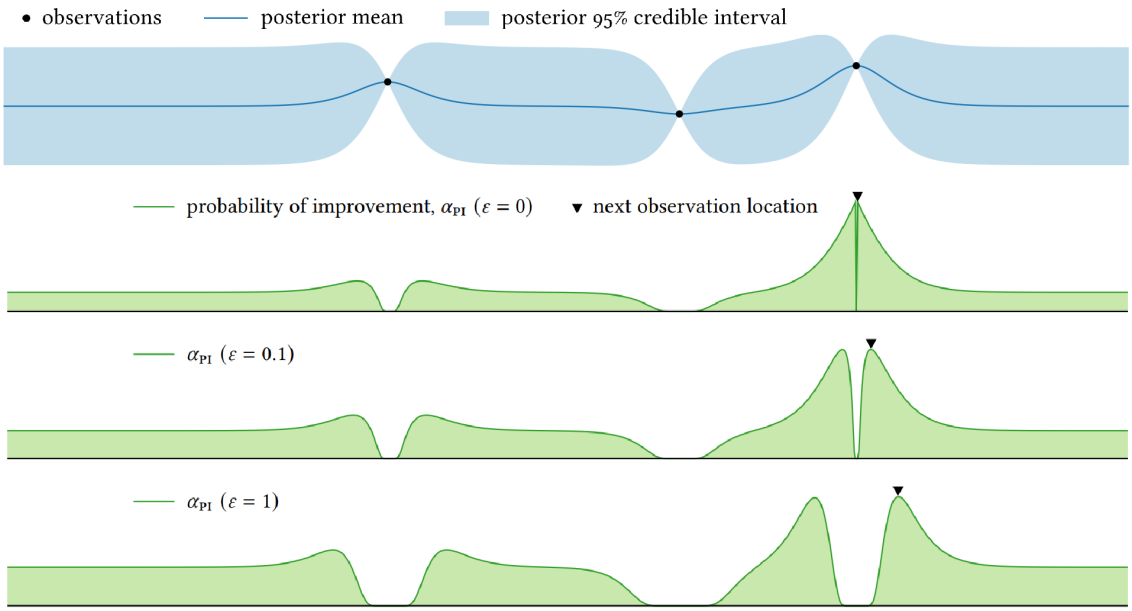


Figure 2.15: This shows the PoI acquisition function for 3 different margins (Bottom) based on a centered GP prior with the Matérn 5/2 covariance function with 3 data points (Top). The margin ε strongly influences how close the recommendation is to observed points, with larger values of ε corresponding to recommendations further from the observations, i.e. more exploratory behavior. Still, the recommended point is quite close to the found maximum every time.

The choice of ε has a strong effect on how PoI deals with the exploration-exploitation dilemma, as illustrated in figure 2.15. Even though the most attractive location remains close observed maxi-

mum, we can force it further and further out by increasing the margin and thus the improvement threshold.

2.12 Optimistic policies

Optimistic policies take a different approach to improvement-based policies. The key idea here is to allow for the benefit of the doubt: Whatever point has the highest plausible reward is recommended. At a point with a large uncertainty band, the reward has a wide range of plausible values, ranging from quite low to quite high. Optimistic policies will favor such an uncertain region due to the plausible high reward, thus favoring exploration.

Because the uncertainty bands are technically non-zero no matter how far you go out, we need a maximum on the amount of uncertainty tolerated: an upper bound. We will require the bound to hold with probability π, which will be our tuning parameter. Previously, we plotted the uncertainty bands as the 95% confidence interval, so if we were to use the top line of these bands as the bound, we would have $\pi = 0.05$. We can define this *upper confidence bound* using the quantile function. The quantile function q of a probability distribution p sends a probability π to the infimum among inputs such that the cumulative probability is lower or equal to the probability bound: $q(\pi) = \inf\{x' \,|\, \mathbb{P}[x < x'] \geq \pi\}$ (recall $\mathbb{P}[x < x'] = \int_{-\infty}^{x} p(x)\,\mathrm{d}x$). Applying this to $p(\phi \,|\, x, \mathcal{D})$ gives

$$q(\pi; x, \mathcal{D}) = \inf\big\{\phi' \,|\, \mathbb{P}[\phi < \phi' \,|\, x, \mathcal{D}] \geq \pi\big\}$$

The decision policy we are constructing should thus maximize the quantile function, so we define the *Upper Confidence Bound* acquisition function as follows

$$\alpha_{\mathrm{UCB}}(x; \mathcal{D}, \pi) = q(\pi; x, \mathcal{D})$$

In our case, $p(\phi \,|\, x, \mathcal{D}) = \mathcal{N}\big(\phi; \mu_{\mathcal{D}}(x), K_{\mathcal{D}}(x, x)\big)$, so we can use Φ, the standard normal CDF, to express this. Let us use shorthand $\mu(x) = \mu_{\mathcal{D}}(x)$ and $\sigma(x) = \sqrt{K_{\mathcal{D}}(x, x)}$. First note since Φ is bijective, $q(\pi)$ is defined in such a way that

$$\pi = \mathbb{P}[x < q(\pi; x, \mathcal{D}) \,|\, x, \mathcal{D}] = \Phi\left(\frac{q(\pi; x, D) - \mu(x)}{\sigma(x)}\right)$$

$$\iff \Phi^{-1}(\pi) = \frac{q(\pi) - \mu(x)}{\sigma(x)} \iff q(\pi; x, \mathcal{D}) = \mu + \Phi^{-1}(\pi)\sigma$$

$$\implies \alpha_{\mathrm{UCB}}(x; \mathcal{D}, \pi) = q(\pi; x, \mathcal{D}) = \mu(x) + \Phi^{-1}(\pi)\sigma(x)$$

Note that in the first two lines, x is a given and fixed, whereas in the definition of our acquisition function, x is the variable.

Thus, when using a GP model, the acquisition function can be expressed as $\alpha_{\mathrm{UCB}}(x; \mathcal{D}, \pi) = \mu_{\mathcal{D}}(x) + \Phi^{-1}(\pi)\sqrt{K_{\mathcal{D}}(x, x)}$ which is simple and insight. The parameter $\Phi^{-1}(\pi)$ can be seen as a tuning parameter in its own right, controlled by our choice of π. Note that $0 < \pi < 1$ and thus $-\infty < \Phi^{-1}(\pi) < \infty$ can be any real number. The gradient of α_{UCB} can be computed as

$$\frac{\partial \alpha_{\mathrm{UCB}}}{\partial x} = \frac{\partial \mu(x)}{\partial x} + \Phi^{-1}(\pi)\frac{\partial \sigma(x)}{\partial x}$$

We can see $\Phi^{-1}(\pi)$ as an exploration parameter, sometimes noted $\Phi^{-1}(\pi) = \beta$, in which case we write $\alpha_{\text{UCB}}(x) = \mu(x) + \beta\sigma(x)$. The larger $\Phi^{-1}(\pi)$, the larger $\frac{\partial\sigma}{\partial x}$ relative to $\frac{\partial\mu(x)}{\partial x}$ and thus the more exploration. If instead $\Phi^{-1}(\pi)$ is smaller, then $\frac{\partial\sigma}{\partial x}$ is smaller relative to $\frac{\partial\mu(x)}{\partial x}$ which points to exploitation.

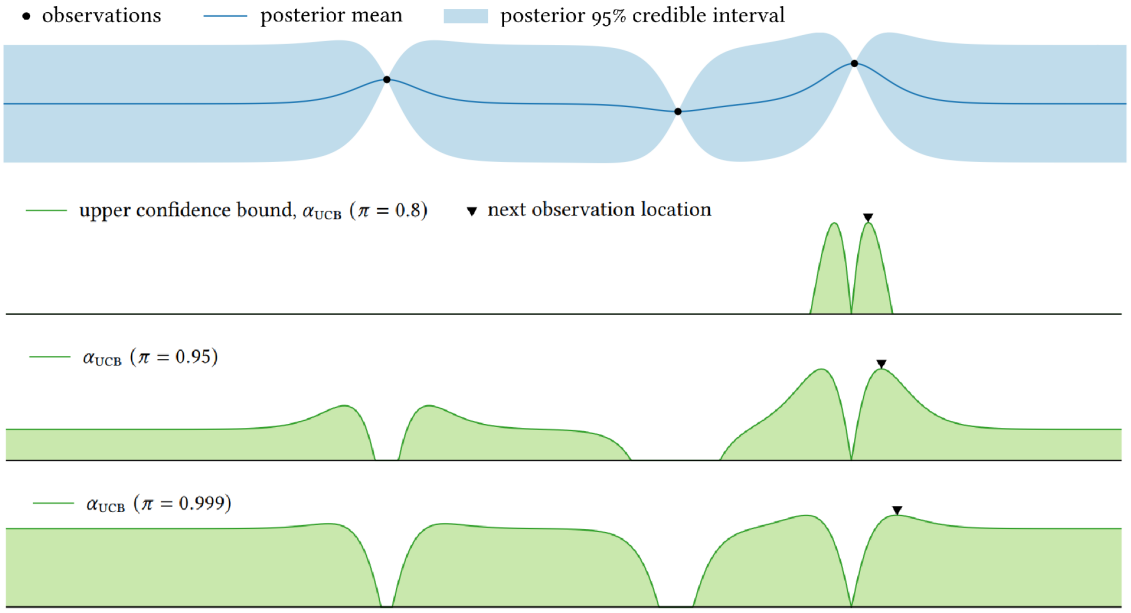


Figure 2.16: This shows the PoI acquisition function for 3 different margins (Bottom) based on a centered GP prior with the Matérn 5/2 covariance function with 3 data points (Top). The acquisition function has been translated such that it's zero at the found maximum. The confidence parameter π strongly influences how favorable points away from observations are, with larger values of π encouraging behavior. Still, all recommendations are at virtually the same point, close to the found maximum.

To end on, there is an interesting relation between PoI and UCB. Suppose $x = \arg\max_{x\in\mathcal{X}} \alpha_{\text{UCB}}(x; \mathcal{D}, \pi)$, then also $x = \arg\max_{x\in\mathcal{X}} \alpha_{\text{PI}}(x; \mathcal{D}, \tau)$ with $\tau = \max_{x\in\mathcal{X}} \mu(x) + \Phi^{-1}(\pi)\sigma(x)$. Thus for specific π and τ, these give the same recommendation every time.

2.13 Initial data

For an acquisition function to give a powerful suggestion, we need to have a powerful hypothesis. Nearing the end of the algorithm, many points have been incorporated and the hypothesis is quite strong, but at the beginning, few or none have been incorporated and the acquisition function is not so powerful. Because of this, we want to start the BO algorithm with a somewhat informed GP that can act as a jumping-off point, and this is done by creating initial points and sampling them to get an initial data set.

The initial data can be constructed in many ways, but ideally should cover the search space somewhat evenly so the initial hypothesis is not biased towards one region, and be somewhat random so the initial conditions can be varied easily and averaged out. With larger dimensions, a uniform random distribution of points will be disproportionately concentrated around the center, which is not ideal. A grid of points would give an even spread but is not random. A balanced solution to this is to use Latin Hypercube Sampling (LHS). A Latin hypercube is a grid of hypercubes, where in each axis-aligned plane there is exactly one point. In 2 dimensions, this would mean each row and column has exactly one point, see Figure 2.17. Then LHS comes down to randomly selecting points that adhere to this requirement [13]. There are multiple ways to achieve this, we have used the `scipy.stats.qmc.LatinHypercube` command from the `scipy` Python package, which uses Quasi-Monte Carlo methods [14].

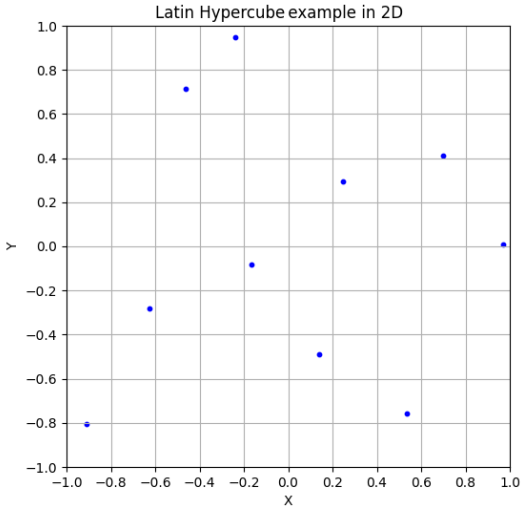


Figure 2.17: An example of a 2D Latin Hypercube, with 10 points spread out over $[-1, 1] \times [-1, 1]$. The points were generated using `scipy.stats.qmc.LatinHypercube`.

Chapter 3

Theoretical Analysis

3.1 Regret

Suppose at step n we have a data set $\mathcal{D}_n = \{(x_1, f(x_1)), \ldots, (x_n, f(x_n))\}$. To quantify the gap between our obtained maximum $\max\{f(x_1), \ldots, f(x_n)\}$ and the theoretical maximum f^*, we define *simple regret* r_n as follows:

$$r_n := f^* - \max_{i \leq n} f(x_i)$$

One of the things we want to show is that our algorithm does what it is supposed to, i.e. find a value as close as possible to the optimum. If $r_n \to 0$ as $n \to \infty$, we know the difference can get arbitrarily close, given enough iterations. This says nothing yet about how quickly this convergence is and is not much of a quality to compare algorithms with, as any algorithm that is not guaranteed to converge in some way is not desirable.

To consider a similar quantity but restricted to any single step, we define the *instantaneous regret* as follows:

$$\rho_n = f^* - f(x_n)$$

We don't necessarily need the instantaneous regret to decrease: it might be useful to do some exploration and thus sample some point that might have high instantaneous regret but which benefits future decisions. More interesting is to see what happens to the sum of instantaneous regret, for example, to see if high regret at one point is offset by low regret at another. To keep track like this, we define *cumulative regret* as follows:

$$R_n = \sum_{i=1}^{n} \rho_i = n\, f^* - \sum_{i=1}^{n} f(x_i)$$

This obviously always increases as we add more data points, so if we would want to minimize this or give an upper bound, we might have to stop adding data points, which is clearly not right. To solve this, we consider the *average regret*, defined as follows:

$$\frac{R_n}{n} = f^* - \frac{1}{n} \sum_{i=1}^{n} f(x_i)$$

If $R_n/n \to 0$ as $n \to \infty$, i.e. the average regret vanishes with more data, we say the algorithm has the *no regret property*.

Note that

$$n\, r_n = n\, f^* - n \max_{i \leq n} f(x_i) = n\, f^* - \sum_{i=1}^{n} \max_{j \leq n} f(x_j) \leq n\, f^* - \sum_{i=1}^{n} f(x_i) = R_n$$

$$\implies r_n \leq \frac{R_n}{n}$$

As $r_n \geq 0$, using the Squeeze Lemma [Elementary Analysis, Ross] we find

$$\lim_{n\to\infty} \frac{R_n}{n} \to 0 \implies 0 \leq \lim_{n\to\infty} r_n \leq \lim_{n\to\infty} \frac{R_n}{n} \to 0.$$

Thus, convergence in average regret is stronger than convergence in simple regret. Conceptually, to converge in simple regret corresponds to 'after some time *at least one* observation must be close to the optimum f^*', but converging in average regret corresponds to 'after some time *all subsequent* observations must be close to the optimum', which is stronger.

3.2 Function space

For a strong convergence result, we must make some assumptions about the type of function our objective function is. The space of functions that adhere to those assumptions is called the *function space*, and when we use such a space, we will assume our objective function can be found in it. We introduce the most relevant function spaces below.

3.2.1 Continuous function space

A continuous function can take so many forms that it can hide its optimum arbitrarily in small regions, meaning it could be arbitrarily hard to optimize. The following theorem from [4] shows how what's necessary for a convergence result:

Theorem Let $\mathcal{X}$ be a compact metric space. An optimization policy converges in simple regret on all continuous functions $f : \mathcal{X} \to \mathbb{R}$ if and only if the set of eventually observed points is always dense.

For $\mathcal{X} = [0, 1]$ (can be continuously translated to any closed simply connected subset of $\mathbb{R}$) convergence in simple regret is shown for PoI [Kushner 1964] and EI [Locatelli, 1997] with the objective function modeled by a Wiener process. For PoI, the same was done for once-differentiable functions like the once-integrated Wiener process. This function space is 'not strict enough' for useful convergence results and stronger assumptions are almost always reasonable, so we will not investigate this space further.

3.2.2 GP Sample space

As we use a GP to model the objective function, it is natural to assume the true objective function could be sampled from the GP used. Assuming the sample paths are continuous, which depends on the GP used as the model but most often is the case, the covariance function induces correlations among function values making sample paths much better behaved.

3.2.3 Reproducing kernel Hilbert space

One such space we can consider is the Reproducing kernel Hilbert space (RKHS). For any covariance function (kernel) $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, there exists an RKHS associated with it, denoted $\mathcal{H}_K$. Given K, the RKHS $\mathcal{H}_K$ is constructed as follows. Let

$$\mathcal{L} = \left\{ \sum_{i=1}^{n} \alpha_i K(\cdot, x_i) \,\middle|\, n \in \mathbb{N}, \{\alpha_i\} \subset \mathbb{R},\ \{x_i\} \subset \mathcal{X} \right\}.$$

Note the connection between GPs: If we consider $\mathcal{GP}(f; \mu, K)$ with $\mu \equiv 0$, then the functions in $\mathcal{L}$ are precisely the posterior mean functions arising from exact Bayesian inference as in (2.3)

$$\mu_{\mathcal{D}}(x) = K(x, \mathbf{x})\boldsymbol{\Sigma}^{-1}\big(f(\mathbf{x}) - \boldsymbol{\mu}\big)$$

We can endow this space with the following inner product:

$$\left\langle \sum_{i=1}^{n} \alpha_i K(\cdot, x_i), \sum_{j=1}^{m} \beta_j K(\cdot, x'_j) \right\rangle = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j K(x_i, x'_j).$$

With this inner product, $\mathcal{H}_K$ is an inner product space, and this inner product $\langle f, g \rangle$ induces a norm $\|f\| := \sqrt{\langle f, f \rangle}$ which has a metric associated with it $d(f, g) := \|f - g\|$. A metric space (i.e. a space with some metric) is *complete* if every Cauchy sequence in the space is convergent in the metric. An inner product that is complete with respect to the associated metric is called a *Hilbert space*. Let $\mathcal{H}_K$ be the completion of $\mathcal{L}$ with respect to this metric, then $\mathcal{H}_K$ is a RKHS [15].

Assuming the objective function lies in the RKHS $\mathcal{H}_K$ is a stronger assumption than assuming it is a sample path. Except for finite-dimensional RHKSs, which we won't discuss, sample paths from a GP lie in the RKHS with probability 0: $\mathbb{P}[f \in \mathcal{H}_K] = 0$. On the other hand, the posterior mean of the GP always lies in the RKHS. Looking at figure 3.1, we see the functions from the RKHS are almost everywhere smooth whereas the sample paths are nowhere smooth. Taking linear combinations of sample paths smooths out the undesirable behavior, and by taking linear combinations of posterior mean functions, functions in the RKHS are also smoother.

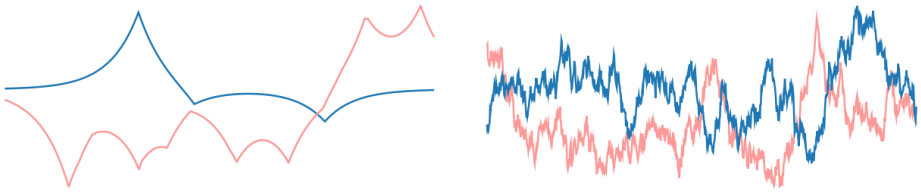


Figure 3.1: Left: Two functions from the RKHS. Right: Two sample paths from a centered GP. Both correspond to the Matérn kernel with $\nu = 1/2$. [4]

The RKHS norm can be seen as a measure of the complexity of the function: If the norm is larger, the underlying function is allowed to have bigger deviations on smaller scales whereas functions

with smaller norm are more well-behaved. By restricting the size of the norms, we can construct a function space of arbitrarily well-behaved functions, depending on restriction. We can take a RKHS ball of radius B, which gives us the space

$$\mathcal{H}_K[B] = \{f \in \mathcal{H}_K \,:\, \|f\| \leq B\}$$

If we make B large we consider a function space with more complex functions and if we make B small, we consider a function space with less complex and more well-behaved functions. We will see B come back as a parameter of convergence bounds, for example, $\overline{r}_n[B]$ and $\overline{R}_n[B]$ denotes the worst-case simple and cumulative regret with the assumption that the objective function lies in the space $\mathcal{H}_K[B]$.

3.3 Bayesian Analysis

Assuming a GP sample space as function space, the objective function f is assumed to be a sample from the GP. Then, for each observation location x, we have a Gaussian distribution in terms of $\mu(x)$ and $K(x,x)$ describing the uncertainty in $f^*(x)$. We define *Bayesian simple regret* and *Bayesian cumulative regret* as the expected value of these quantities with respect to this uncertainty:

$$\mathbb{E}[r_n] \qquad \mathbb{E}[R_n]$$

This can be used to analyze the performance of the average case.

3.4 Frequentist Analysis

The Bayesian approach leads to an analysis of the *average-case*, but one might also want to consider the worst-case. An alternative to the sample path assumption is to assume the objective function lies in some space $\mathcal{H}$ of nice functions, and then considering the *worst-case* expected simple regret or cumulative regret within that space:

$$\overline{r}_n[\mathcal{H}] = \sup_{f \in \mathcal{H}} \mathbb{E}[r_n], \qquad \overline{R}_n[\mathcal{H}] = \sup_{f \in \mathcal{H}} \mathbb{E}[R_n]$$

The algorithms we analyze like this are based on GPs but the objective function is not assumed to be generated by one. In the rest of this work, this nice space of functions $\mathcal{H}$ will be some RKHS ball $\mathcal{H}_K[B]$.

3.5 Bounds

There are two types of bounds we can consider: an upper bound in a quantity like simple regret or average regret can give us a guarantee on the performance of the algorithm as it is proven it cannot perform worse. A lower bound on the other hand can be used to study the best-case scenario and is independent of the algorithm.

For the following bounds, we need to assume $\mathcal{X} \subset \mathbb{R}^d$ is compact, and $K(x,x) \leq 1$. The first assumption is met in our experiments, the second assumption depends on the output scale.

Most of these bounds are achieved by bounding the width of the posterior confidence interval, i.e. with observed locations $\mathbf{x} \subset \mathcal{X}$, we wish to restrict the maximum posterior standard deviation of the GP:

$$\overline{\sigma}_{\mathbf{x}} = \max_{x \in \mathcal{X}} \sigma = \max_{x \in \mathcal{X}} \sqrt{K_{\mathcal{D}}(x, x)}$$

Here $K_{\mathcal{D}}$ is the posterior covariance function after performing inference with $\mathcal{D} = \big(\mathbf{x}, f^*(\mathbf{x})\big)$. For a Matérn covariance function, independent of the algorithm used, we have [4]

$$\overline{\sigma}_{\mathbf{x}} = \mathcal{O}(n^{-\nu/d}) \tag{3.1}$$

Here, n denotes the number of steps and thus the number of acquired points.

For a Matérn covariance function, for the worst-case simple regret, we have [16]

$$\overline{r}_n[B] = \mathcal{O}(n^{-\nu/d}).$$

This results from combining the bound on the standard deviation (3.1)with a simple grid algorithm and recognizing that any other algorithm is as good or better. If we use the Expected Improvement BO algorithm, we get [17]

$$\overline{r}_n[B] = \mathcal{O}^*(n^{-\min(\nu,1)/d}).$$

For a Matérn covariance function with $\nu > 2$ using the UCB algorithm, the following exponential bound for Bayesian instantaneous regret was found [18]:

$$\mathbb{E}[\rho_n] = \mathcal{O}\left(\exp\left(-\frac{cn}{(\log n)^{d/4}}\right)\right).$$

Here $c > 0$ is a constant depending on the GP, not on n. As $r_n \leq \rho_n$, we have the same bound for the simple Bayesian regret. Also, $R_n = \sum_{i=1}^{n} \rho_i$, so the cumulative regret is bounded. Exponential bounds are very strong and this is a really powerful result, making UCB under these conditions very promising.

Chapter 4

Toy Problem Experiments

4.1 Goal of experiments

The goal of our experiments is not to find an optimal geometric configuration but to find the optimal conditions for our algorithms to perform well. For example, we are interested in how the performance of the algorithm is affected by different acquisition functions, different exploration parameters the acquisition functions, time of termination, initial data set, the GP model chosen, and much more. To conduct these experiments, we need a reduced version of the NP array heating problem, which we call the toy problem. For the toy problem, we have constructed a toy model.

4.2 The Toy Model

The COMSOL model used for our experiments, 'the Toy model', is meant to be both accurate and computationally inexpensive, with a focus on the latter, as the goal is to find the optimal BO algorithm rather than the optimal geometry.

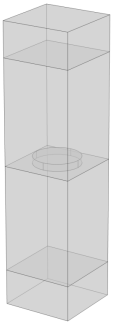


Figure 4.1: This figure shows the unit cell of the model as viewed in COMSOL.

The model consists of a rectangular *unit cell*, with a cylindrical silver NP at the center. The glass substrate is modeled with a glass box directly underneath the NP, and the surrounding air with an air box around and above the NP. All material characteristics were taken from the COMSOL library. At the top of the air box, the illuminating laser is modeled by an EM field port moving perpendicular to the NP. This laser has a wavelength of 532 nm. The fields should diminish to 0 outside the relevant part of the model. We use a Perfectly Matched Layer (PML) above the air box and below the glass substrate box so that any excitation leaving these boxes is gradually removed. We chose for the COMSOL model to model an infinite array, so we can use periodic boundaries in the x and y axes, significantly reducing computational costs. The boundary conditions are constructed by identifying the solution on opposing sides

(i.e. copying it over). We also impose periodic conditions on the port,
so the laser illuminates the entire infinite array. The meshing is done automatically by COMSOL using the extremely fine auto meshing setting.

We chose to let the simulation only work using the Electromagnetic Wave Frequency Domain (ewfd) module, instead of also using the Heat Transfer in Solids (ht) module, which would also require the multiphysics Electromagnetic Heating 1 (emh1) module to couple them, all of which increases simulation time. The ewfd module only works with electro-magnetic properties, so the model cannot return a temperature directly, but it can return the absorption cross-section σ_{abs}. We then use formula 1.1 to compute the increase in temperature instead. Solving only in the ewfd domain further decreases time per simulation.

For the code to generate the model, see [19]

4.3 General Methods

Here we describe how the optimization problem in the 3 dimensional search space was set up in general terms and restricted to what is relevant to the thesis. In this problem, the bounds are $p \in [100\text{ nm}, 1000\text{ nm}]$, $r \in [15\text{ nm}, 150\text{ nm}]$ & $h \in [15\text{ nm}, 60\text{ nm}]$. The height, pitch, and radius all have a step size of 1 nm so the suggested points are rounded to the nearest nm. This means there are $(60-15)\cdot(1000-100)\cdot(150-15) = 5,467,500$ possibilities. We require the NPs to have a minimal distance of 50 nm as this is our assumed manufacturing limit. If the combination of p, r & h of a point doesn't adhere to this, we return $\Delta T_0^{\text{ext}} = 0$. If the location is valid, we give it as input to the COMSOL model, which returns the corresponding ΔT_0^{ext}.

The initial data set is created using Latin Hypercube Sampling (LHS), as described in 2.13. The LHS is implemented with constant random seed (`seed=0`), so every run has the same initial data, thus making sure the initial data is not a factor in algorithm performance. After the initial data set is acquired and the GP is updated, the initial hypothesis GP is constructed. This GP acts as a jumping-off point for the BO algorithm to suggest all subsequent points, which get rounded to the nearest nm. The BO algorithm is based on an initial GP that is centered and uses the Matérn 5/2 covariance function. This means we implicitly assume the underlying function is 2 times continuously differentiable. This assumption is motivated by empirical results like those of Martijn [5], as well as the fact that most responses in physics are pretty smooth. In the end, all probed points and corresponding values are stored in a `.json` log file. For the files and the code used, see [19].

4.4 Termination

After looking at some of the graphs produced early on, we wanted to investigate what a good number of iterations would be. My predecessor went with 250 total iterations [5], but before continuing with this number, we wanted to experiment with it first. Thus we ran the algorithm once with $n = 1000$, using MEI with $\varepsilon = 10^{-4}$ and the BO package, the results of which are plotted in figure 4.2. One run took about 8 hours, and due to server time constraints, we ran this experiment only once. We see that the algorithm continues to make improvements for increasing n, but the further along it is, the smaller the improvement gets.

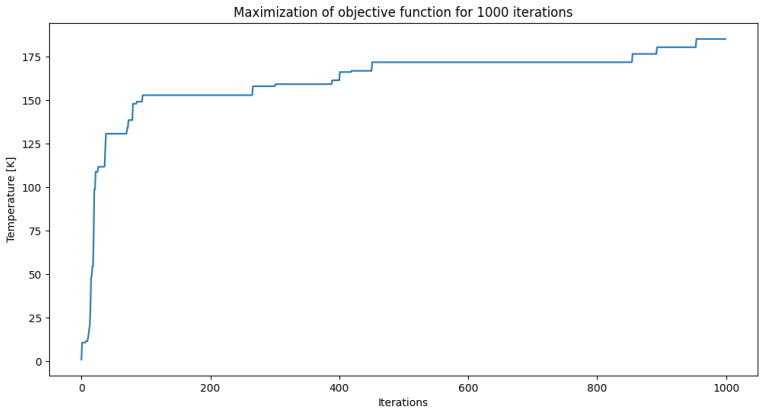


Figure 4.2: Maximum target value observed plotted against the number of iterations.

Deciding the number of total iterations comes down to a trade-off between computational cost and final reward. The more iterations we do, the more likely we are to find a good maximum, and in the limit, we check every possible location and are 100% sure we find the global optimum. Yet due to limitations in computing resources, we must restrict ourselves to a computational budget, and looking at Figure 4.2, a computational budget of 250 points (about 2 hours per run) seems like a well-balanced decision. All subsequent runs of the algorithm on the toy problem were done with 250 points. Based on this number, an initial data set size of 10 points was chosen, again inherited from Martijn.

4.5 Comparing EI with different exploration parameters

My predecessor worked only with the Expected Improvement (EI) acquisition function, which is one of the most common acquisition functions in Bayesian Optimization. Martijn actually chose to work with an MEI acquisition function with $\varepsilon = 10^{-4}$. He remarked it was set this way to have a fair balance between exploration and exploitation, without any further detail or proof [5]. So an open question was: Within the EI class of acquisition functions, which performs best? To answer this question, we wanted to investigate the performance of EI and MEI with different exploration parameters, and also try out the dynamic acquisition function DMEI, all of which are described in 2.11.1. In the BO package, the parameter ε is referred to as `xi`, so these two will be used interchangeably. The BO package used suggests in the documentation that `xi` be between 0 and 0.1, with 0 being the most exploitative and 0.1 being the most exploratory [6]. The acquisition functions we chose to compare were: EI (so MEI with $\varepsilon = 0$), MEI with $\varepsilon = 10^{-4}$, MEI with $\varepsilon = 10^{-2}$, MEI with $\varepsilon = 0.1$ and DMEI with $\varepsilon(n) = 0.1 \cdot \frac{250-n}{250}$, so $\varepsilon(0) = 0.1$ and $\varepsilon(250) = 0$.

To make a fair comparison, all algorithms were run using the exact same LHS initial data set. Every run of a BO algorithm is random, even when the initial data is the same, and so we chose to run every algorithm 10 times to account for this randomness. This means we are doing an approximation of Bayesian analysis as discussed in section 3.3. We then took these 10 runs and

found the average value at each iteration to create the average run. Figure 4.3 shows for each iteration the maximum target value in the average run for each type of acquisition function. We also sampled 250 using LHS as a baseline to compare the other algorithms to.

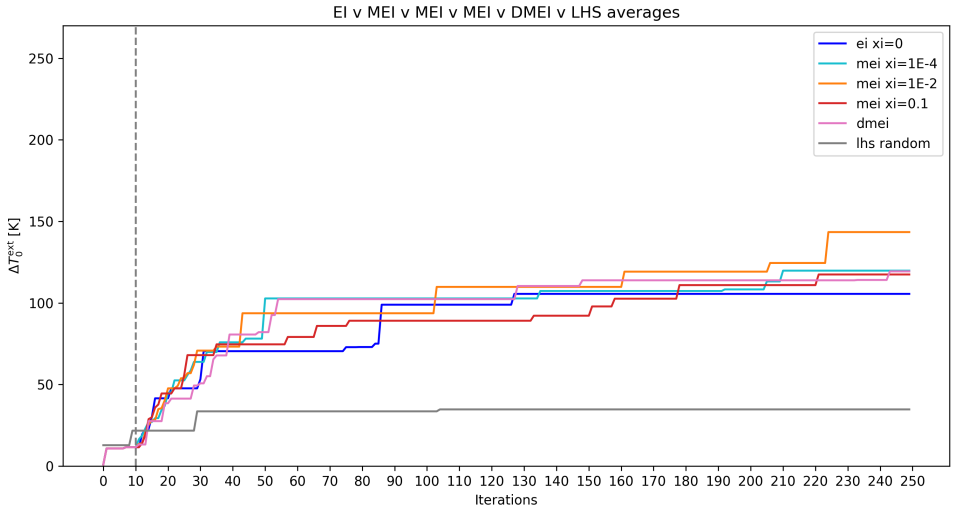


Figure 4.3: At each iteration, the maximum achieved target is plotted for each type of EI. The grey vertical dotted line indicates the initial data set of the BO algorithms.

In Figure 4.3, it is clear that the LHS search algorithm performs far more poorly, which was expected. Looking at the final recommendation of the others, the results are pretty close. The pure EI scored lowest, which would mean that setting the exploration parameter for maximum exploitation does not perform well. However, the MEI with $\varepsilon = 0.1$ did not do much better, which shows that setting up for maximum exploration does also not perform well. Note that MEI with $\varepsilon = 0.1$ ended up approximately together with MEI with $\varepsilon = 10^{-4}$ and DMEI, which all performed reasonably well not exceptionally well. The best performing seems to be MEI with $\varepsilon = 10^{-2}$, by quite a margin, which shows that indeed a moderate exploration parameter is best, between $\varepsilon = 10^{-4}$ and $\varepsilon = 0.1$ and probably around $\varepsilon = 10^{-2}$. The question if better values for ε exist and what they are, as well as if different dynamic exploration parameters might be effective, are still open questions. However, it is a worthwhile discovery that pure EI performs relatively badly, as it seems to be the most widely used and praised.

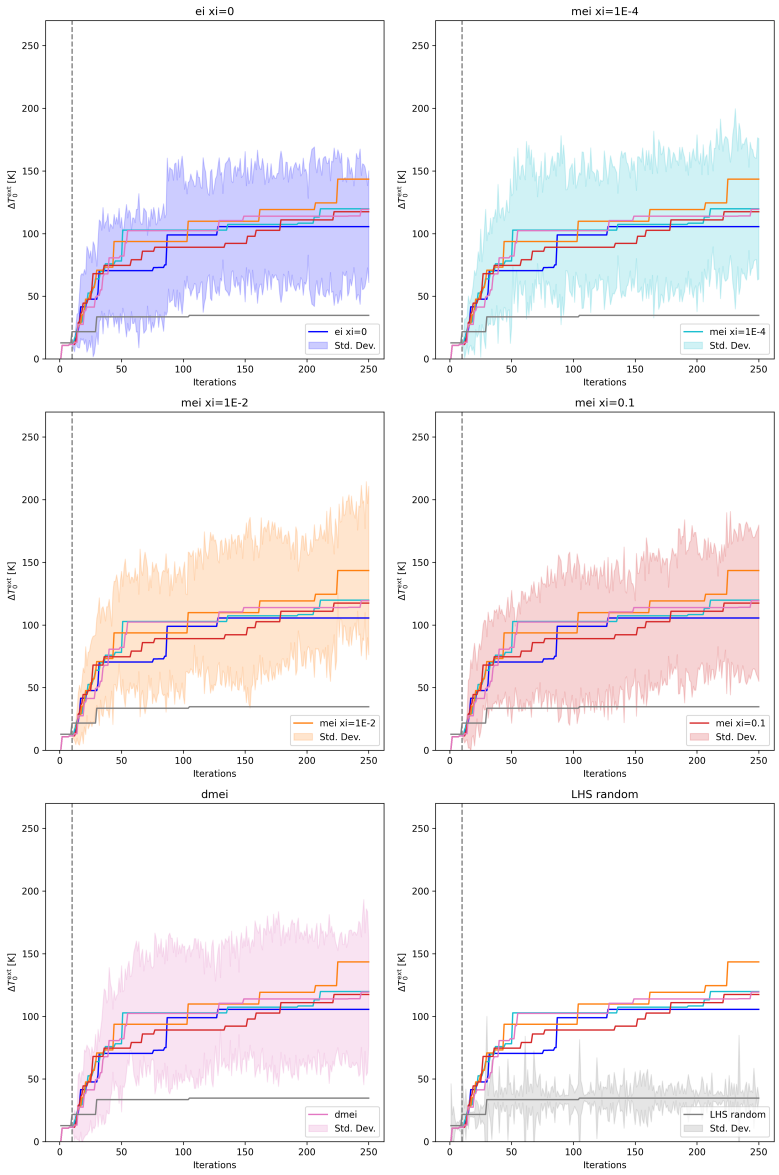


Figure 4.4: The six lines in these six plots are identical and coincide with the averages plot in figure 4.3, with the same colors corresponding to the same labels. Each plot highlights one of the labels by adding the sample standard deviation shown with a shaded area of the same color.

We need to take into account though that the individual runs have a large variance. Looking at figure 4.4, we can see all graphs are within 1 sample standard deviation of any other graph, except for the LHS graph. This means that any analysis of the averages graph should taken with a grain of salt, as it is not that unlikely that repeating the experiments would give different outcomes. To be

more certain, we would need to run each algorithm much more often to really remove probability as a factor. Due to time constraints and limited server accessibility, we had to keep it to 10 runs per algorithm. For plots of each individual run, see appendix B.

4.6 Comparing EI with other acquisition functions

The BO package has two other acquisition functions built-in: the Probability of Improvement (PoI) and Upper Confidence Bound (UCB) acquisition functions. An obvious question is how these algorithms compare to EI and each other. Thus we ran another 10 runs for PoI and 10 runs for UCB, whose averages are plotted in figure 4.5, along with EI (because of its popularity), MEI with $\varepsilon = 10^{-2}$ (because it performed best in the previous experiment), DMEI and LHS. Note that for PoI we used $\varepsilon = 10^{-2}$ and $\beta = 1$, which we've chosen semi-arbitrarily based on the documentation [6].

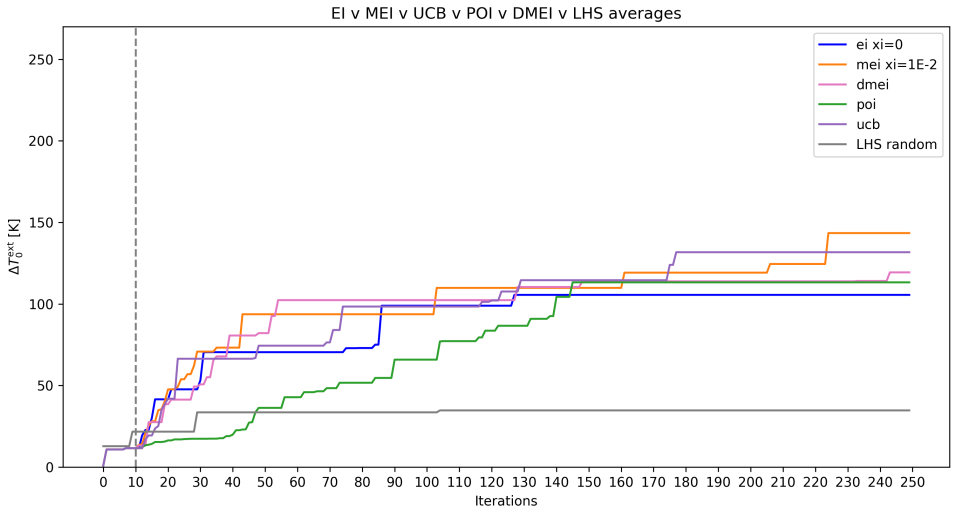


Figure 4.5: At each iteration, the maximum achieved target is plotted for some of the types of EI, as well as PoI and UCB. The grey vertical dotted line indicates the initial data set of the BO algorithms.

We can see that PoI performs quite badly, with the final recommendation being just above that of EI. Also, note that even for the average run, for a smaller number of iterations, PoI performs very poorly, on average only overtaking LHS after about 50 iterations. Compare that with the other algorithms, which overtake LHS after about 15 iterations, and it is clear that PoI performs especially poorly for a smaller number of iterations. UCB, however, performs quite well, with the final target value coming in second, only below MEI with $\varepsilon = 10^{-2}$. This indicates that PoI is not competitive but UCB is, for these specific parameters of course. An open question remains how the performances of both PoI and UCB change with different exploration parameters ε and β.

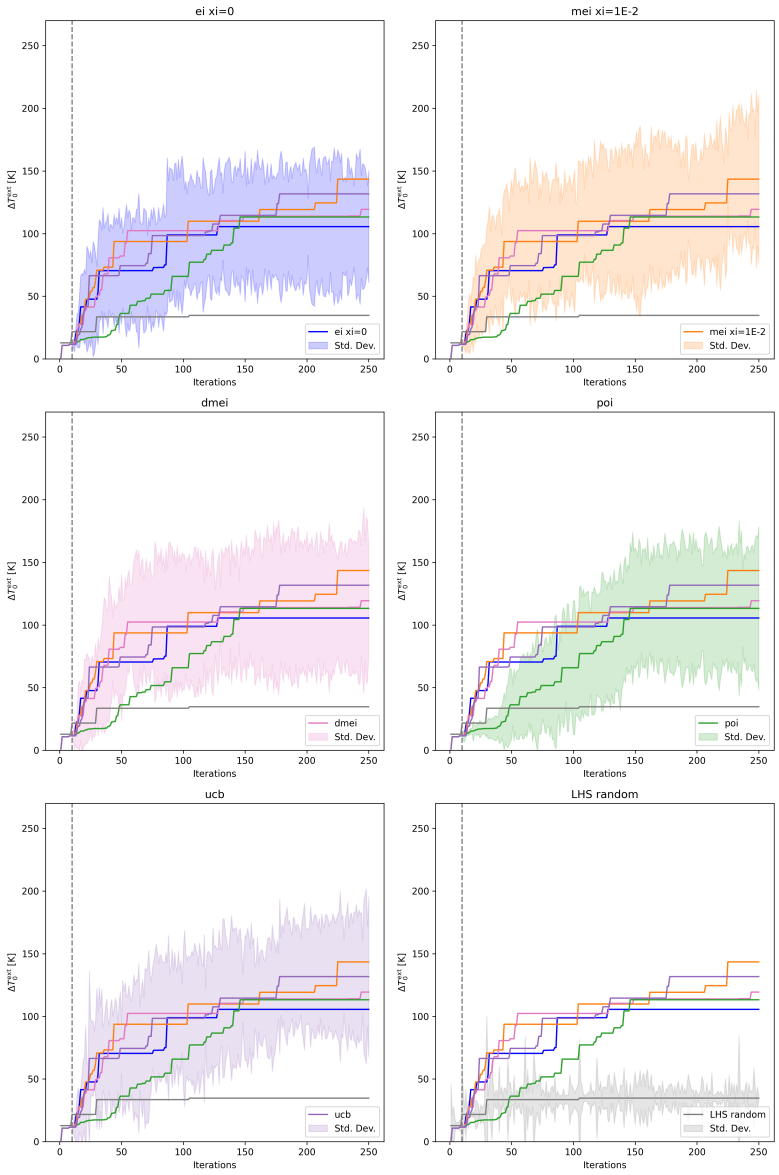


Figure 4.6: The six lines in these six plots are identical and coincide with the averages plot in figure 4.5, with the same colors corresponding to the same labels. Each plot highlights one of the labels by adding the sample standard deviation shown with a shaded area of the same color.

Again, there is an issue with the variance between individual runs, hence we plotted the sample standard deviations again in figure 4.6. Again, we see the final targets mostly lie within 1 sample standard deviation of any other algorithm, except LHS. We can see however in the PoI plot that the graphs of the competitive algorithms lie outside 1 sample standard deviation of PoI in the first

50 iterations, showing this difference is less likely to be due to the randomness of the algorithm and more likely to be due to the algorithm itself.

4.7 1D plots

To gain further insight into the BO algorithm and the BO python package, we thought it would be interesting to see it work on a 1D version of our problem. The 1D version is a 1D slice in one of the dimensions, so keeping two of the variables constant and only varying the third. Making it 1D means we can actually plot the entire objective function by sampling the few hundred points needed, and we can visualize how the mean function and standard deviation interval change as points get added. Below is the implementation using the Python package we have thus far used, the BO package [6].

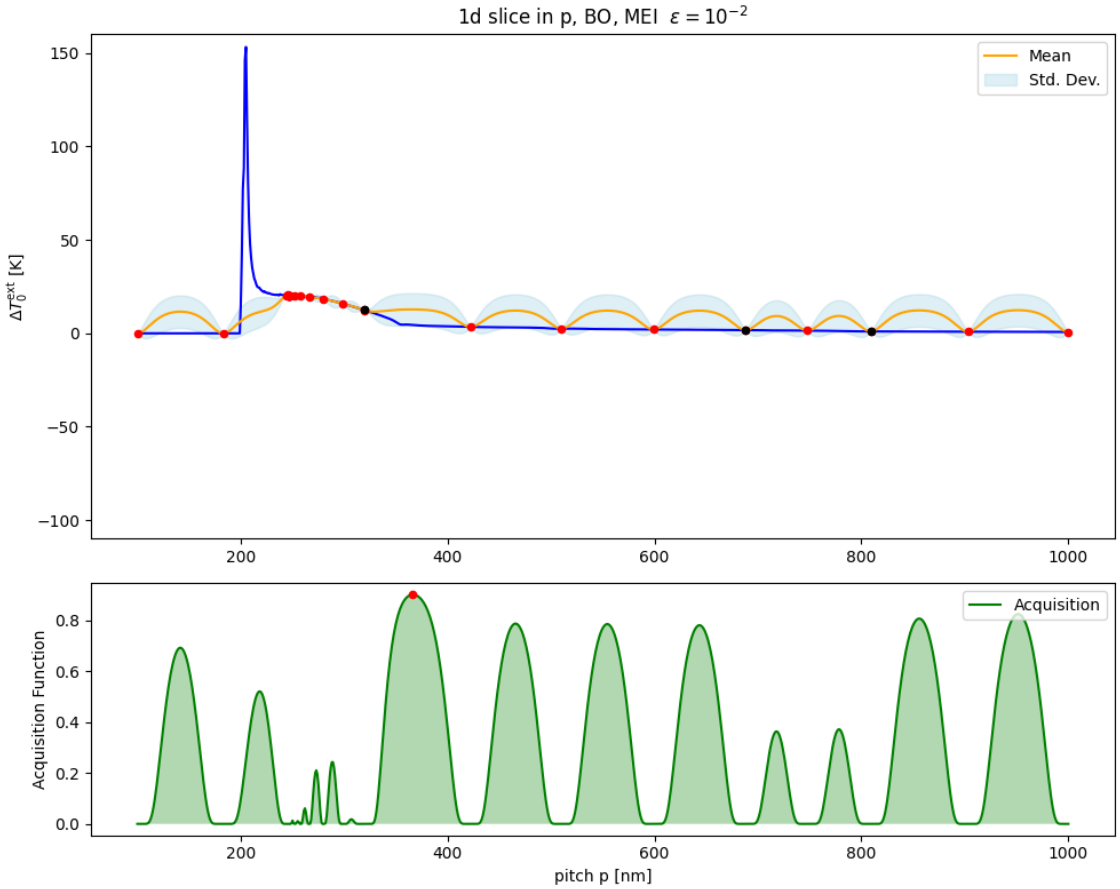


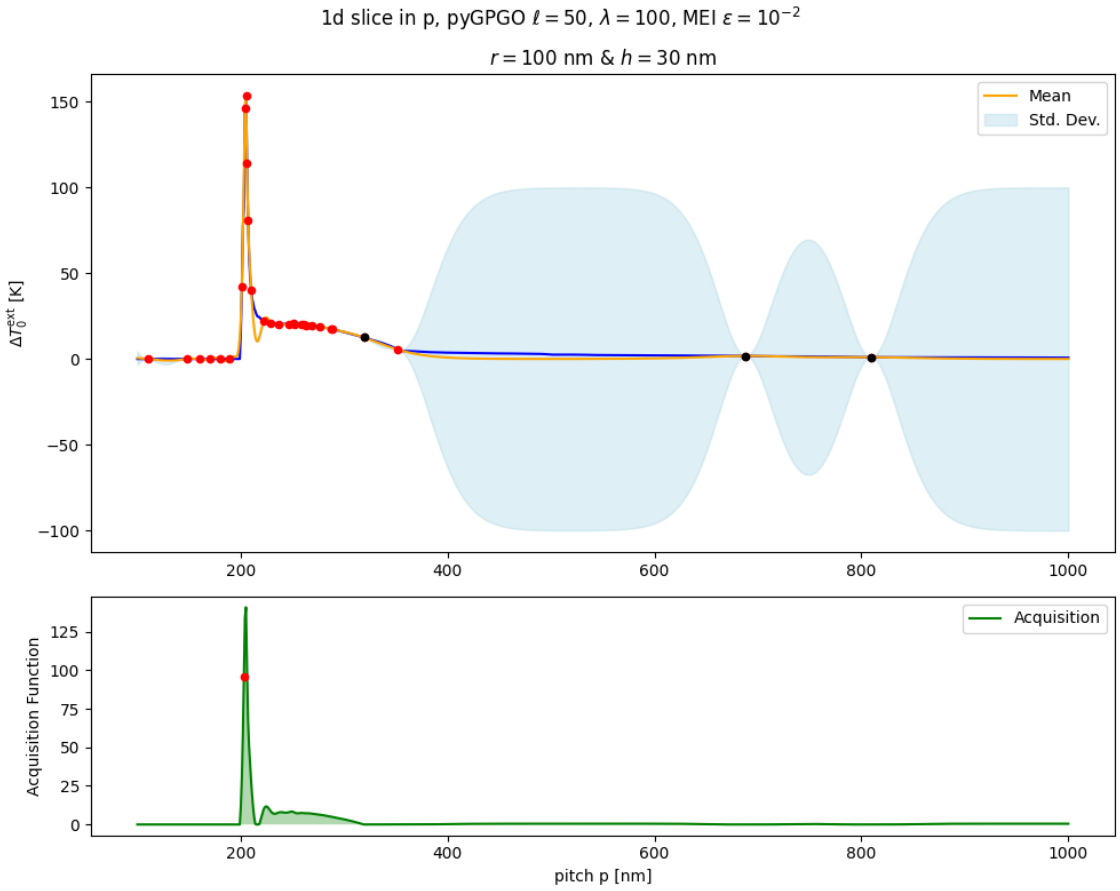
Figure 4.7: [Click for animation] This shows the BO algorithm implemented using BO, applied on a 1D slice in p, with r and h held constant. The top shows the objective function with GP, bottom shows the acquisition function MEI with $\varepsilon = 10^{-2}$. The 3 black dots indicate the initial data set (LHS), and the 27 red dots show the probed points suggested by the BO algorithm. The red point at the bottom shows where the algorithm decides to sample next.

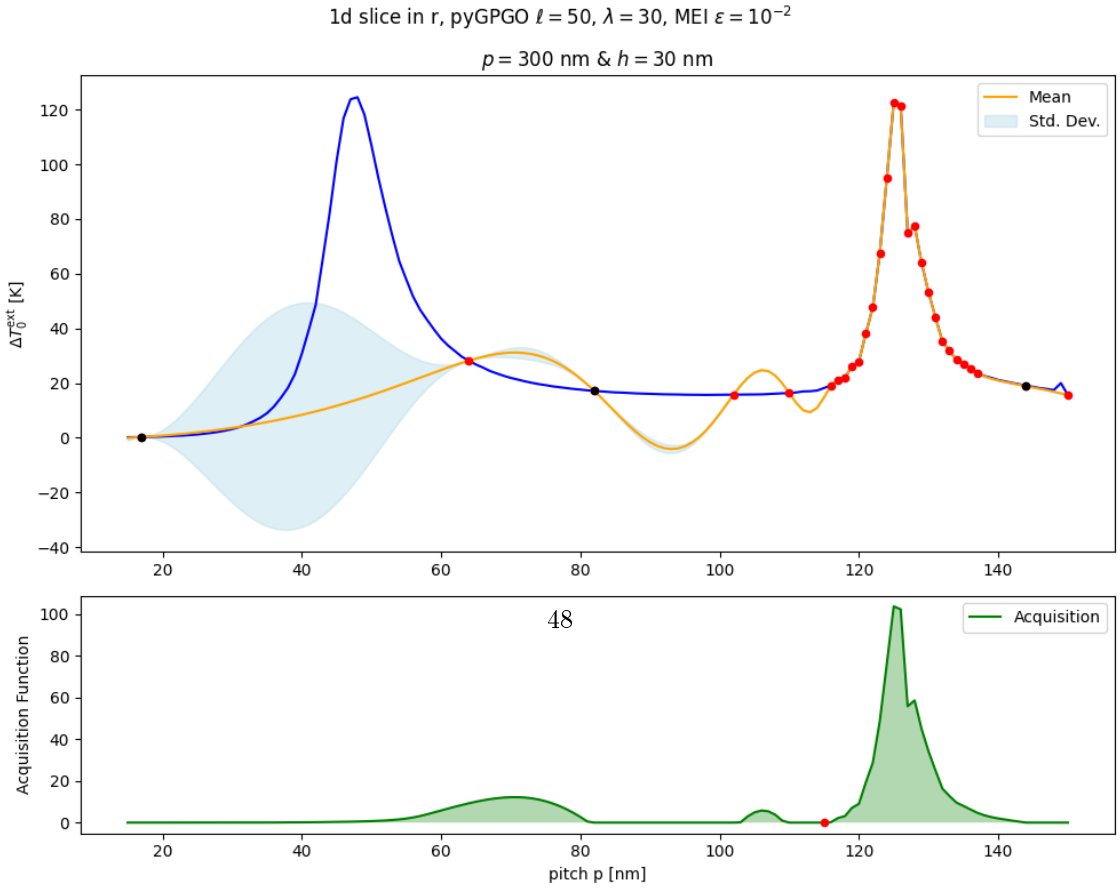
We can see that there are a few issues here. First of all, the optimum is not explored, and not for a lack of points. Second, the final mean function behaves unexpectedly, moving up between points where it really should not. Also, the standard deviation band is pretty small, being only about 10K in height at the most, whilst the function reaches a height of about 150K. This means the objective function has a very small likelihood of being a sample path of the GP and means the BO algorithm is not adapted to optimize well. A solution would be to alter the output scale, as described in 2.6. However, this is not possible using the BO package. This is why we remade the algorithm using the pyGPGO package [7].

The BO package only allows you to work with centered GPs with the Matérn 5/2 covariance function and does not allow for changing the characteristic length scale ℓ and output scale λ. How ℓ and λ

are determined must be hidden in the code but after careful inspection by myself, I was not able to retrieve this information. The pyGPGO package, however, allows you to select from a range of covariance functions and any prior mean function. Also, when choosing the Matérn 5/2 covariance function, both ℓ and λ can be set to any desired value. The question of what are appropriate values does however remain an open one. Also, in my opinion, the way the programmer interacts with the package more closely resembles how it is described in the literature, making it much more intuitive for those who know the literature. One drawback however is that the acquisition function definitions do not align with the literature used in this work. This was remedied by redefining the corresponding Python classes. Another thing is dealing with duplicate points. Due to our round the suggested point to the nearest nm, the acquisition function can suggest the same point twice. The BO package allows for an option `allow_duplicate_points`, which we have set to `True`. How this affects the algorithm is not clear, as even after careful inspection I could not make up how the package deals with this variable. The pyGPGO package does not have this option, and instead, I made a script that, in case of a suggested point having already been probed, we find a point nearest the suggested point in Euclidean distance. Both the old and new algorithms can be found on [19].

To compare BO and pyGPGO, we used the same example as in figure4.7, but using pyGPGO, resulting in the top plot of Figure 4.8. Note that ℓ and λ were chosen based on what seemed right on the plots and are in no way perfect. However, any unexpected behavior has vanished and the algorithm now finds the optimum, with the same initial data set and amount of points. Now that we have a satisfactory 1D algorithm, we also took the time to give the same example in slices in the remaining variables, namely r and h, shown below in Figure 4.8.


1d slice in p, pyGPGO $\ell = 50$, $\lambda = 100$, MEI $\varepsilon = 10^{-2}$
$r = 100$ nm & $h = 30$ nm
Mean
Std. Dev.
ΔT_0^{ext} [K]
Acquisition
Acquisition Function
pitch p [nm]


1d slice in r, pyGPGO $\ell = 50$, $\lambda = 30$, MEI $\varepsilon = 10^{-2}$
$p = 300$ nm & $h = 30$ nm
Mean
Std. Dev.
ΔT_0^{ext} [K]
Acquisition
Acquisition Function
pitch p [nm]

4.8 Toy Problem Winner

Looking at the raw data of the acquisition function experiments, there is one single run notable for achieving the highest maximum target value amongst all single runs, namely run #7 of MEI with $\varepsilon = 10^{-4}$. Understanding how and why this run was successful can help increase our understanding both of the algorithm's performance and the underlying physics, which is why we want to study it further.

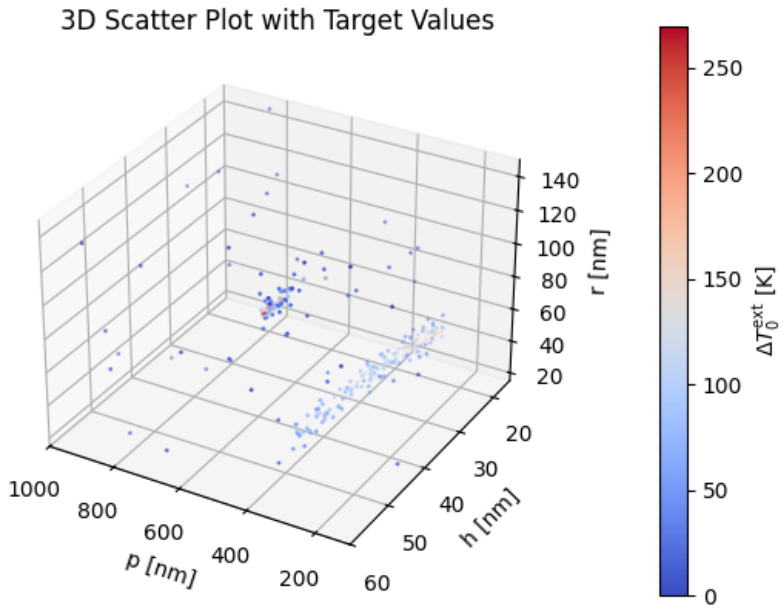


Figure 4.9: This plot shows the order in which the BO algorithm acquires points and their location in the 3D search space, with the target value shown in color.

In Figures 4.9, we can see one smaller and one elongated 'cloud' of points forming, regions where many points were sampled. Both clouds contain relatively high targets. The formation of clouds of points is interesting but predicted, as these are the regions where the acquisition function takes on high values and where the hypothesis GP's mean function is most optimistic.

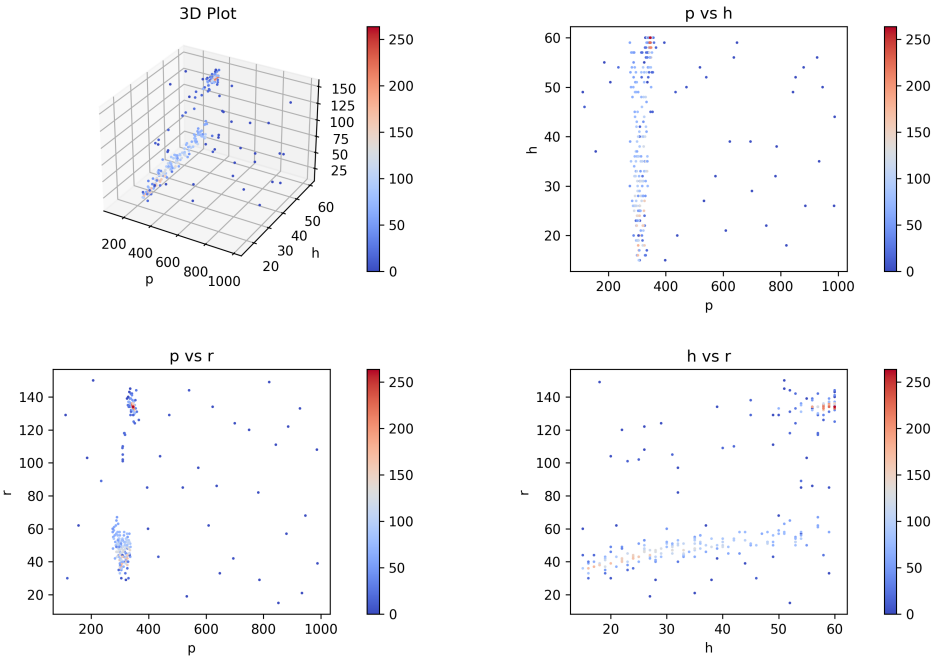


Figure 4.10: (Top left) This plot shows the all acquired points in the 3D search space. The corresponding observed target value is shown with color. The remaining plots show 2D projections, projecting in the r dimension (Top right), the h dimension (Bottom left), and the p dimension (Bottom right).

Looking at Figure 4.10, two clouds can now be better identified, both at $p \approx 300$ nm, the smaller one at $r \approx 130$ nm and $h \approx 60$ nm and the elongated one at $r \approx 50$ nm and all h values. At $p = 346$ nm, $r = 134$ nm, and $h = 60$ nm, a value of $\Delta T_0^{\text{ext}} = 263.73$ K was observed, the highest value in any simulation we've done. In the immediate area, points vary greatly, as there are both high and much lower target values observed very closely to each other. This means the objective function here varies a lot in a small area. This could be because of the underlying physics, and/or because the COMSOL simulation is inaccurate.

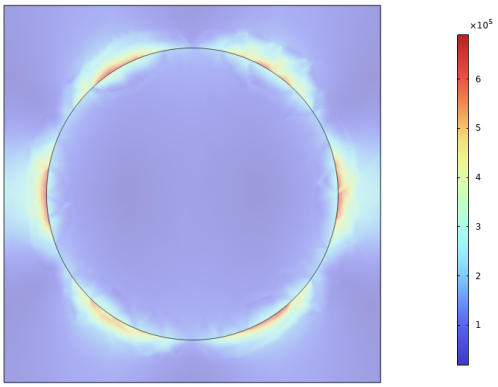


Figure 4.11: Slice of electric field [V/m] through the NP for $p = 346$ nm, $r = 134$ nm and $h = 60$ nm.

Looking at Figure 4.11, there don't seem to be any serious issues as the amount of artifacts is minimal. The image shows a strong symmetry, which could be connected to the strong response and high temperature increase.

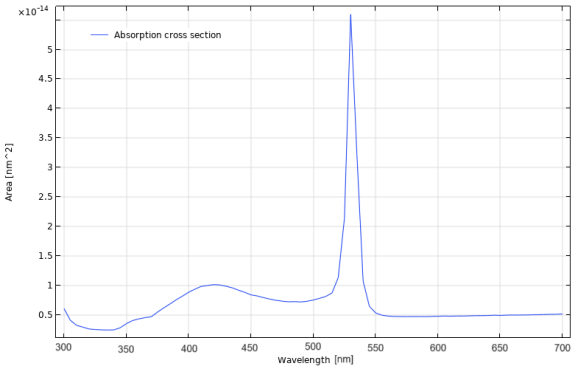


Figure 4.12: The absorption cross section σ_{abs} [m^2] for wavelengths ranging from 300 nm to 700 nm for $p = 346$ nm, $r = 134$ nm and $h = 60$ nm.

In Figure 4.12, we can see there is a clear peak in the absorption cross-section σ_{abs} around $\lambda = 532$ nm. This means the high target value is highly dependent on the wavelength for the configuration with $p = 346$ nm, $r = 134$ nm, and $h = 60$ nm.

4.9 BO vs pyGPGO package

As mentioned before, in this project we have constructed and worked with two different Python implementations of the BO algorithm, one relying on the BO package, covering the majority of the results in Chapter 4, and the other relying on the pyGPGO package, which in section 4.7 we concluded was probably more accurate. We have run the pyGPGO package for our two most successful acquisition functions: the MEI with $\varepsilon = 10^{-2}$ and UCB. Again, 10 runs for each were condensed to a more representative average run.

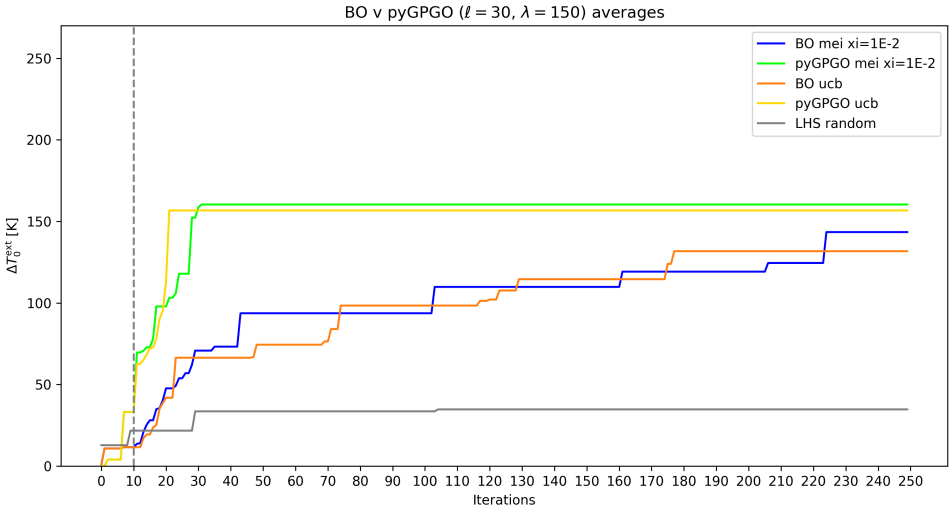


Figure 4.13: At each iteration, the maximum achieved target is plotted for MEI with $\varepsilon = 10^{-2}$ and UCB, both in the BO and pyGPGO implementation. The grey vertical dotted line indicates the initial data set of the BO algorithms.

In Figure 4.13, we can see that the pyGPGO versions of MEI and UCB far outperform the BO versions, as they both find a good target value early on. However, they both don't improve for the last 200+ iterations, meaning the difference between the BO versions narrows as the number of iterations grows. A direct comparison cannot be drawn because the initial data sets are different. For a fair comparison, the experiment should be run with the same initial data across both versions, but due to time constraints, we won't do that here.

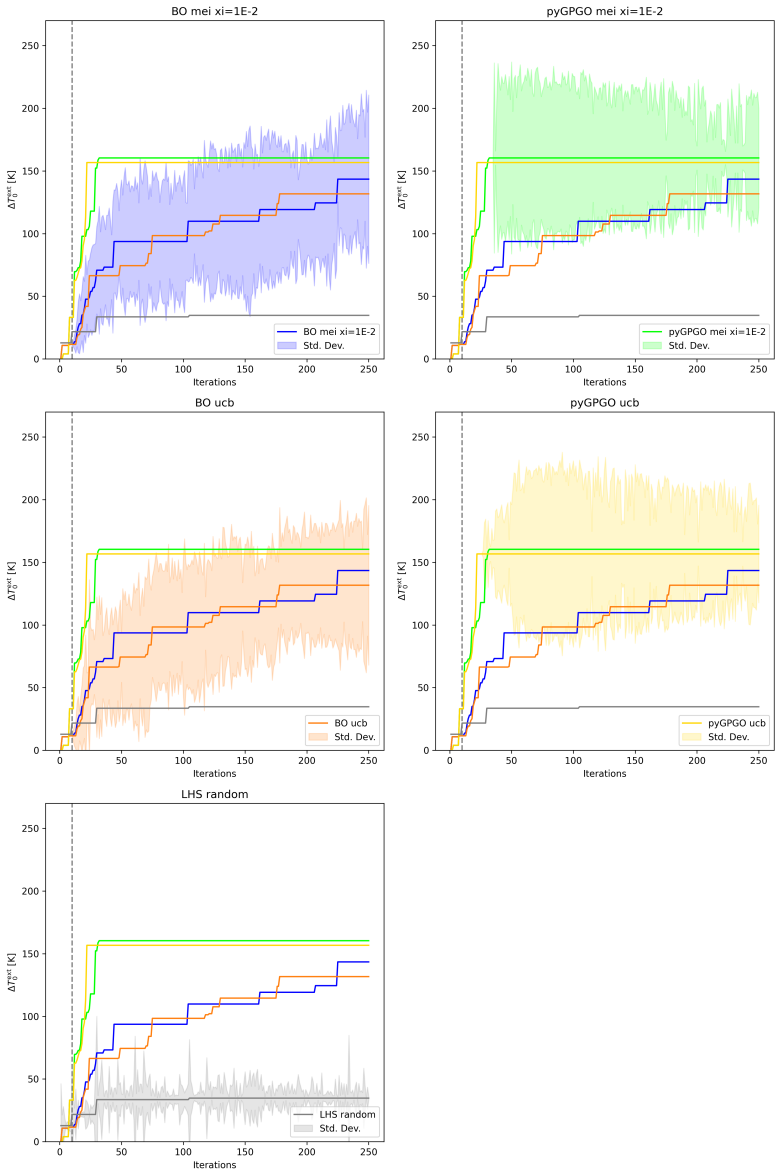


Figure 4.14: The five lines in these five plots are identical and coincide with the plot of the average in figure 4.13, with the same colors corresponding to the same labels. Each plot highlights one of the labels by adding the sample standard deviation shown with a shaded area of the same color.

Really the only new plots in Figure 4.14 are the pyGPGO MEI (green) and pyGPGO UCB (yellow). Curious about these is the 0 variance early on. For pyGPGO UCB, the variance between runs only sets in after about 30 iterations, and for pyGPGO MEI this is only after about 40 iterations. Then the variance instantly goes from 0 to a wider variance margin.

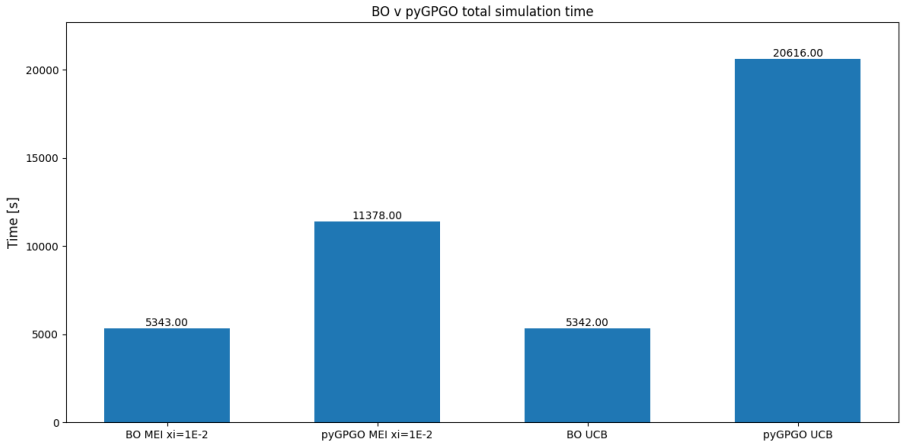


Figure 4.15: The total simulation time in seconds, meaning all 250 points, for each of the considered algorithms.

There is another factor to consider: The time it takes to run the algorithm. From Figure 4.15, we can see that pyGPGO is slower, by a considerable amount. This difference is especially large when employing the UCB acquisition function, as BO and pyGPGO differ by a factor of almost 4. The difference is less pronounced when using MEI with $\varepsilon = 10^{-2}$, as they differ by a factor of about 2.

Chapter 5

Discussions

5.1 Conclusion

In this work, we started by looking at how the COMSOL NP array heating problem works and how Bayesian Optimization works. We described how these can be combined to form a good-performing optimization algorithm, and we described what choices can be made within the BO framework, including the covariance function, the acquisition function, and the exploration parameter. We looked at different choices for exploration parameters within the Expected Improvement class of acquisition functions and found that a stationary exploration parameter chosen for moderate exploration performs best under the chosen conditions. More interestingly perhaps, we found the pure Expected Improvement acquisition function, which has exploration parameter 0 and skews heavily to the exploitation side, to perform worst in this class. We also compared this class with different acquisition functions, where Probability of Improvement proved not so competitive contrary to Upper Confidence Bound which proved quite competitive, again under the chosen conditions. Either way, due to the small number of runs and high variance between runs, all results are not statistically significant and should be taken as such until more runs have been done. When applying the algorithm in 1D to gain insight visually, we found the BO package to be behaving strangely, which is why the algorithm was written again using a new package. This research can be used to build even better BO algorithms as it has uncovered many questions and ideas related to performance. That being said, some of our implementations have proven quite competitive already and can be used on similar problems as are. With optimization algorithms like this, not only can optimal NP array geometries be discovered, but they can also be analyzed and compared to understand why they happen. Hopefully, our research can act as a building stone for more research, eventually creating and improving nano-technologies.

5.2 Future research

The research done in this work has given us a comprehension of how to use a Bayesian Optimization algorithm on these kinds of problems. Yet, there is much more in the theory that might be applicable. Also, many of the choices we have made are motivated but not investigated properly,

leaving their effect unknown. Lastly, within the theory we have laid out, there are still many different options to investigate that we have not touched on yet. To address this, we have a few recommendations as to what might be worth looking into for future research.

5.2.1 Theory

There is a lot of theory on Bayesian Optimization and Gaussian Processes. A few interesting things I found but did not have the time to fully investigate and/or process in this work are listed below.

Bayesian Optimization is built on modeling the objective function using a Gaussian Process, which can work well if the objective function is continuous. Since our objective function is not continuous, it might make sense to model the objective function not as a random field (multivariate continuous time stochastic process) like a Gaussian Process, but instead, use a discrete variant of this (multivariate discrete-time stochastic process). If the modeling takes into account the discrete nature of the objective function, it could be that the whole optimization algorithm becomes more appropriate.

There is a section on model selection and averaging in [4]. I have not taken the time to understand it, but it seems it might hold promising insight into how to best select the GP model and its parameters. Before doing experiments on different models, I would suggest looking into this.

There seems to be a significance to the Mahalanobis distance, which I described in 2.3. It is mentioned in [4] and it seems it is connected both to the GP model and to the RKHS norm.

Due to the rounding, working with the BO algorithm can result in points being suggested that have already been probed, i.e. duplicate points. A literature study could be done on how to treat duplicate points or how to prevent them.

5.2.2 Covariance functions

The covariance function we used to model the objective function was the Matérn 5/2 covariance function, which has the parameters ℓ and λ. When using the pyGPGO implementation, the user has the freedom to select these, and it would be worthwhile to investigate how these parameters affect the performance.

Based on intuition, the Matérn family of covariance functions seems like a good choice, but what parameter for ν to select is still an open question. It would therefore be interesting to conduct experiments aimed at finding what effect ν has on algorithm performance.

Comparing Matérn covariance functions with other types would also make for an interesting series of experiments. For this and other objective functions underpinned by resonance phenomena, it might for example be that a periodic covariance function is more suitable.

Taking it even further, it is in no way ruled out that combinations of covariance functions are most suitable. How these experiments would be set up is getting more unclear but it remains an open question nonetheless.

Note that the choice of covariance function should depend on the underlying objective function. Therefore, these parameters/choices/combinations would probably need to be adapted to each specific objective function used and will likely not generalize to other objective functions.

5.2.3 Acquisition functions

The exploration parameter experiments we did for Expected Improvement (EI) could also be done with the Probability of Improvement (PoI) and Upper Confidence Bound (UCB) exploration parameters as well, as the exploration/exploitation trade-off in these was not explored. It might be that optimizing the exploration parameters for PoI and UCB and then comparing them to each other and with the optimal exploration parameter for EI gives unexpected or stronger results as we found without considering this.

In the literature, there are more acquisition functions. For example, starting with (EI) and changing its utility function (the maximum target value in the observed data set) to something similar, namely the maximum of the posterior mean, gives rise to the Knowledge Gradient (KG) acquisition function. Being so similar to EI, it might be worthwhile to look into this. Note that KG is not included with the BO package, nor the pyGPGO package.

What is included in the pyGPGO package (but not in the BO package) is the Entropy Search (ES) acquisition function. This is part of a larger class of information-based policies, which prioritize increasing the knowledge of the function. It would be very interesting to see how algorithms based on this principle compare to the ones we have seen in this work. There are many variations of ES, for example, it can be defined for a discrete space or a continuous space. The discrete version might be well suited for the discrete objective function we have looked at [4].

5.2.4 Initial data & termination

How the initial data set is constructed was motivated, but in this work, there is no conclusive argument or evidence on the topic. More research and experiments on the types of initial data could be done to come closer to an answer. Not only how the initial data set is constructed, but also how many initial data points should be included is an open question. In the article '*Sequential robust optimization of a V-bending process using numerical simulations*' by Wiebenga et el., 2012 [20], it is suggested to use 10 times the number of variables, but it is not clear how the total number of iterations is decided and how they compare.

Also, with the exact same construction for the initial data set, the random seed impacts the performance of the BO algorithm. For example, if the initial data set accidentally gets close to a really good optimum, the algorithm can go straight into exploiting that region, whereas if the initial data set only has low target values, this does not happen and the algorithm has a disadvantage. By how the random seed affects the performance was not addressed in this project, so research and experiments should be done. If not and this algorithm was put into use to find an optimum, it is suggested to run it with a multitude of random seeds to try to counter this issue.

We did address the total number of iterations in this work, however, more could be done here as well, as the numbers we arrived at were still somewhat arbitrary.

5.2.5 Duplicate points

As mentioned, the BO package has a setting to allow for duplicate points, but the pyGPGO package does not. In the case that the BO package remains in use, I would suggest looking into how duplicate points are handled by this package. When using pyGPGO, it could be necessary to construct a self-made solution to this as I have done. In case of a duplicate point, my solution was to try

points in the immediate area (in Euclidean distance) until an unprobed point was found and use that. Another idea may be to adapt the acquisition function to the discrete setting. What could be promising is to define the acquisition function only on the discrete inputs and 0 everywhere else, instead of defining it on the continuous domain. This would mean we would not have to round our suggested points and would do away with the duplicate points problem, as at the exact location of an observed point, the acquisition function is always zero. This and other solutions should be explored.

5.2.6 Domain

In our implementation, the objective function will return 0 under certain criteria. The problem is that this has an instant cut-off and no smooth transition, whereas the rest of the function is assumed to be smooth. This means that for the Matérn 5/2 covariance function with certain values for ℓ and λ set for the whole domain are not compatible with this region of instant cut-off. This problem can also be seen in **??** and 4.7, as the GP is very inaccurate and unstable in the region where the function is 0. A solution to this could be to redefine the domain making it smaller so the domain has a boundary where the cut-off would happen and the region where the function is 0 is removed. Another solution could be to set the acquisition function to 0 wherever the objective function is 0, as the function is not really unknown there. This would require a different initial data set as with LHS there is a high probability points are selected in the region where the function is 0, and then the GP hypothesis will still be incompatible.

5.2.7 pyGPGO and BO

A lot more could be figured out about how the BO and pyGPGO packages operate and why for example there is such a large variance between runs with identical setups and why for example with pyGPGO this variance seems to be delayed and with BO not. Also the experiment in section 4.9 should be run again but with the same initial data set across all versions to see if one actually achieves better results than the other.

Bibliography

[1] S. Lv, Y. Du, F. Wu, Y. Cai, T. Zhou, Review on lspr assisted photocatalysis: effects of physical fields and opportunities in multifield decoupling, *Nanoscale Adv.* (2022).

[2] P. TP, R. P. Botteon CEA, M. PD, R. M, Nanoparticle systems for cancer phototherapy: An overview, *Nanomaterials* (2021).

[3] COMSOL Inc., *Wave Optics Module Users Guide*.

[4] R. Garnett, *Bayesian Optimization* (Cambridge University Press, 2023).

[5] M. Kievit, Optimization of light-induced heat generation in nanoparticles, See `https://github.com/TijsWiegman/Bachelor-Project` (2023).

[6] F. N. et al., Bayesian optimization, `https://github.com/bayesian-optimization/BayesianOptimization` (2024).

[7] J. J.-L. et al., pygpgo: Bayesian optimization for python, See `https://pygpgo.readthedocs.io/en/latest/index.html` (2024).

[8] L. Zundel, K. Malone, L. Cerdán, R. Martínez-Herrero, A. Manjavacas, Lattice resonances for thermoplasmonics, *ACS Photonics* (2023).

[9] U. D. Zeitner, M. Banasch, M. Trost, The potential of e-beam lithography for micro- and nano-optics on large areas, *SPIE* (2023).

[10] G. Baffou, *et al.*, Photoinduced heating of nanoparticle arrays, *ACS Nano* (2013).

[11] M. M. Arat, Univariate/multivariate gaussian distribution and their properties, `https://mmuratarat.github.io/2019-10-05/univariate-multivariate_gaussian` (2019).

[12] Y. Ding, Control and optimization of soft exosuit to improve the efficiency of human walking, Ph.D. thesis, Harvard (2018).

[13] B. de Gooijer, Surrogate modelling and robust optimization of multi-stage metal forming processes, Ph.D. thesis, University of Twente (2022).

[14] scipy.stats.qmc.latinhypercube, `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.qmc.LatinHypercube.html`.

[15] B. P. Rynne, M. A. Youngson, *Linear Functional Analysis* (Springer, 2008), second edn.

[16] M. Kanagawa, P. Hennig, D. Sejdinovic, B. K. Sriperumbudur, Gaussian processes and kernel methods: A review on connections and equivalences, *Arxiv* (2018).

[17] A. D. Bull, Convergence rates of efficient global optimization algorithms, *Journal of Machine Learning Research* (2011).

[18] N. de Freitas, A. Smola, M. Zoghi, Exponential regret bounds for gaussian process bandits with deterministic observations, *Arxiv* (2012).

[19] T. Wiegman, Bachelor project github repository, `https://github.com/TijsWiegman/Bachelor-Project` (2024).

[20] J. Wiebenga, A. van den Boogaard, G. Klaseboer, Sequential robust optimization of a v-bending process using numerical simulations, *Struct Multidisc Opti* (2012).

Popular Summary

In my bachelor thesis, we worked with nanophysics and machine learning. We looked at nanoparticles (NPs), which are particles at the nanoscale where everything is expressed in nanometers, which is one billionth of a meter. The NPs we looked at were made up of silver and have a cylindrical shape. When NPs are put in a periodic array, they form a so-called NP array. We looked at an NP array with a square lattice formation where we can control the geometry, for example how far away the particles are away from each other or the radius of the cylinders. We then put a laser on the NP array, and depending on the parameters, the array's temperature increase is different. This is because, for certain geometries, resonance phenomena occur more strongly than with others. The problem posed by physicists is then: What is the optimal geometric configuration to maximize the heat produced? We did not seek to answer this question but rather looked into what makes a good optimization algorithm for this problem.

We did not experiment with the setup in a lab but rather used software to model the NP array and simulate what happens when the laser turns on. Using simulation software means we can easily experiment with many different arrays without having to manufacture them. The software was set up in such a way that we could input the array parameters, those being the distance between particles (p), the radius of the particles (r), and the height of the particles (h). After running the simulation, which takes a few minutes each time, a temperature increase ΔT_0^{ext} is returned. This denotes the temperature increase of the center NP in the NP array, but only the temperature increase contribution of the other particles. This inputting of parameters p, r, and h and receiving a value ΔT_0^{ext} can be viewed as a function with three inputs and one output, and it is this function we want to optimize.

We restricted ourselves in the number of geometric configurations we considered, but with the amount of time per simulation (≈ 1 minute) and the number of possible geometric configurations i.e. function inputs (in our case $\approx 5.500.000$), it would take nearly 10.5 years to know the function everywhere. This is completely infeasible, but we still want to find a good approximate optimum, and instead of years it should only take a few hours. This means we need to be very intelligent about which function inputs we do and do not choose to observe, as we can only check $\approx 0.005\%$ of the locations.

The idea is to construct a hypothesis about the function, informed by the points we have thus far observed. We treat the function as being random, modeling it as a Gaussian Process (GP). A GP is a continuous extension of the Gaussian distribution and is described by a mean function and how likely it is for the function to deviate from that mean. There are many possibilities for GPs, and we selected a GP that fits with the physical setting we're working with. Starting out,

the GP is basically an empty guess, but as we observe points, we can update the GP to reflect our observations. With this updated hypothesis, we can make a more educated guess as to where the optimum may be. The acquisition function is used to recommend new points based on the current hypothesis, taking the GP as input and returning a score for each point in the domain, with the highest-scored point being our next observation point. In the Bayesian Optimization (BO) algorithm, we do this repeatedly.

There are multiple types of acquisition functions, each backed by a slightly different philosophy, but all with the goal of finding an approximate optimum. We looked at the Expected Improvement (EI), Probability of Improvement (PoI), and Upper Confidence Bound (UCB) acquisition functions, as they are most used. Each acquisition function has a parameter called the exploration parameter. The acquisition function suggests the next point for observation, but within this, there is a trade-off. If we don't rely on the hypothesis at all, we are just randomly exploring which is not very effective. If we want to exploit our hypothesis as much as possible, we get stuck in the areas we have a pretty good hypothesis for but we completely miss other areas of the domain where an even greater optimum may lie. This is the exploration/exploitation trade-off, and each acquisition function has its own way of handling it, as well as an exploration parameter. The exploration parameter can be set to a small value if we want to favor exploitation and to a large value if we want to favor exploration.

We did some experiments to compare the performance of the BO algorithm, first using different exploration parameters for EI, then using EI, PoI, and UCB. We found that the BO algorithm performs much better than a baseline random search-like algorithm. For EI, we found that a moderate exploration parameter works best. Between EI, PoI, and UCB, we found that PoI performs poorly but UCB performs quite well, however, we did look at only one exploration parameter for PoI and UCB, so for different exploration parameters results might differ. Also, the algorithm is inherently random, so we did the experiments a few times. We were limited by the amount of time, and there were pretty large variations in performance between runs, so if the experiments were to be repeated, there is a chance the results would be different.

Appendix

A Meshing experiments

Before running other experiments, we wanted to know what the optimal meshing size is for our model. Ideally, we would like to cut down on computational cost as much as possible without getting (significantly) different output. We first look at the differences arising from running the simulation once under different meshing conditions. Keeping all other parameters the same, with $h = 30$ nm, $p = 300$ nm, and $r = 100$ nm, only the meshing settings were changed.

Meshing Configuration	Domain/Boundary/Edge Elements	Max. Size	Min. Size
Extremely Coarse	3,282/1,024/220	100 nm	3 nm
Normal	3,604/1,078/224	100 nm	3 nm
Extremely Fine	10,964/1,796/284	100 nm	0.2 nm
Ultra Fine	33,319/3,006/332	10 nm	0.1 nm
Uber fine	245,933/11,620/512	5nm	0.05 nm

Table 1: Four meshing configurations were considered, with the properties described above: Number of Domain/Boundary/Edge Elements, maximum element size, and minimum element size.

The above table shows (some of) the defining qualities of each of the meshings. The first three, extremely coarse, normal, and extremely fine are auto meshing settings in COMSOL, so COMSOL analyses the size of the model (based on p, r & h) and decides how many elements go where. The last two are manually defined by me, by setting the maximum and minimum element size only, which COMSOL then uses to find a suitable meshing distribution. The last setting, uber fine, is especially fine, to ensure as much as possible convergence in the fineness of the meshing.

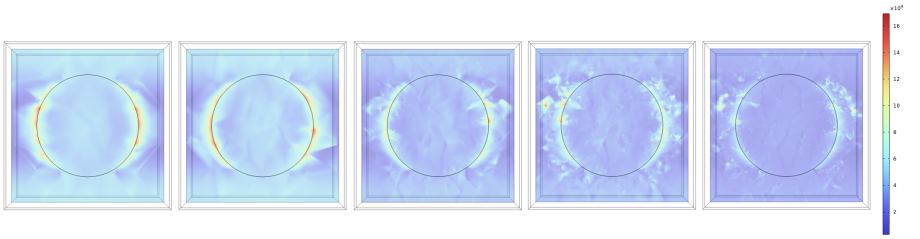


Figure 1: Slice of electric field [V/m] through the NP for five meshing configurations. From left to right: Extremely coarse, normal, extremely fine, ultra fine, uber fine.

Looking at figure 1, as expected, the figure shows a clear increase in detail from left to right, from coarser to finer meshings. Note that the amount of artifacts also increases, which is peculiar. It's especially bad for the finest two meshings and it might indicate some mistake(s) being made, either by COMSOL or how COMSOL is used (after all the meshings used we're defined by me, an inexperienced user). From here on we only use the extremely fine auto meshing, so we did not get to resolve this problem and it requires further investigation.

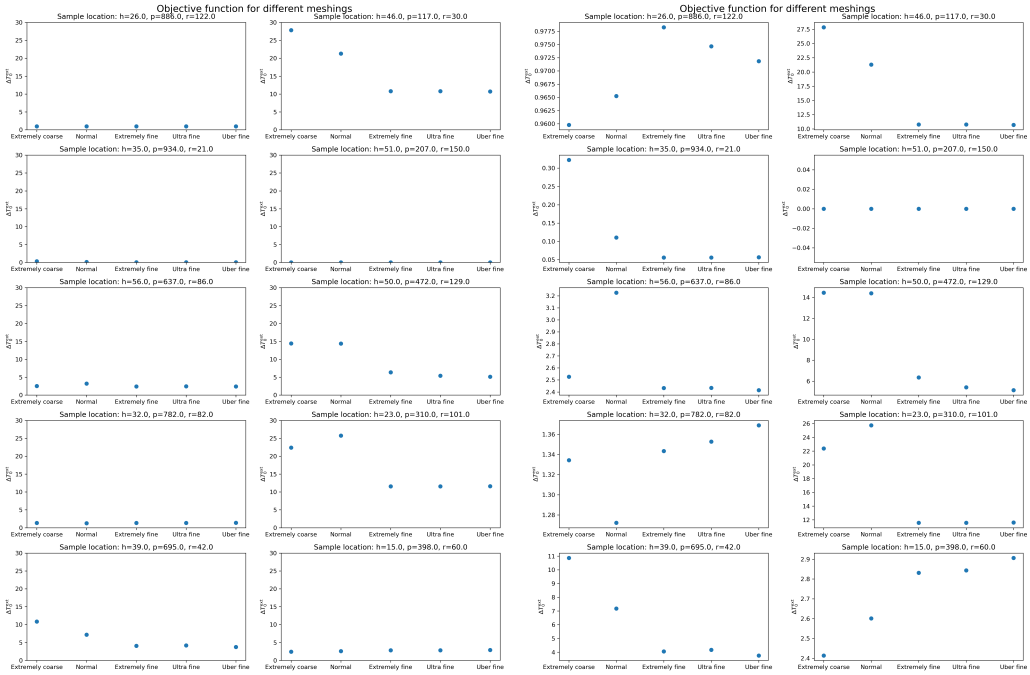


Figure 2: The left two columns show for 10 LHS sampled points the target value ΔT_0^{ext} for different meshing sizes, from extremely coarse to uber fine. We can see that the three finest meshings, extremely fine, ultra fine, and uber fine all have basically the same result. The right two columns show the same but the scale is flexible so that we can better see the small differences that are present.

Since the finest three meshings give very similar results see convergence in the fineness of the meshing. From this, it's safe to conclude that these meshings are approximately equivalent and very close to physical reality, especially because the last setting is so fine. As the extremely fine auto meshing setting is a feature of COMSOL and the others aren't, and since it is the fastest, we ran all other experiments using the extremely fine auto meshing setting.

B Acquisition function experiments data

Above are plots showing the raw data of the acquisition function experiments, with only the last two being the pyGPGO versions, the rest are BO versions of the algorithm. You can see the large variance between single runs even with the same acquisition function and initial data set, explaining the large sample standard deviations observed. It's also clear most single runs reach a maximum target value of at most about 200 K or less. There are 3 single runs that reach targets higher than this: For MEI $\varepsilon = 10^{-4}$, runs # 5 and #7, and for UCB run #2. Of these, MEI $\varepsilon = 10^{-4}$ run #7 reaches the highest target value and thus is investigated further in 4.8.

C Code and data sets

To ensure the reproducibility of my experiments, my code is available on my Bachelor Project Github page [19], where I will post video instructions on how the code can be used.

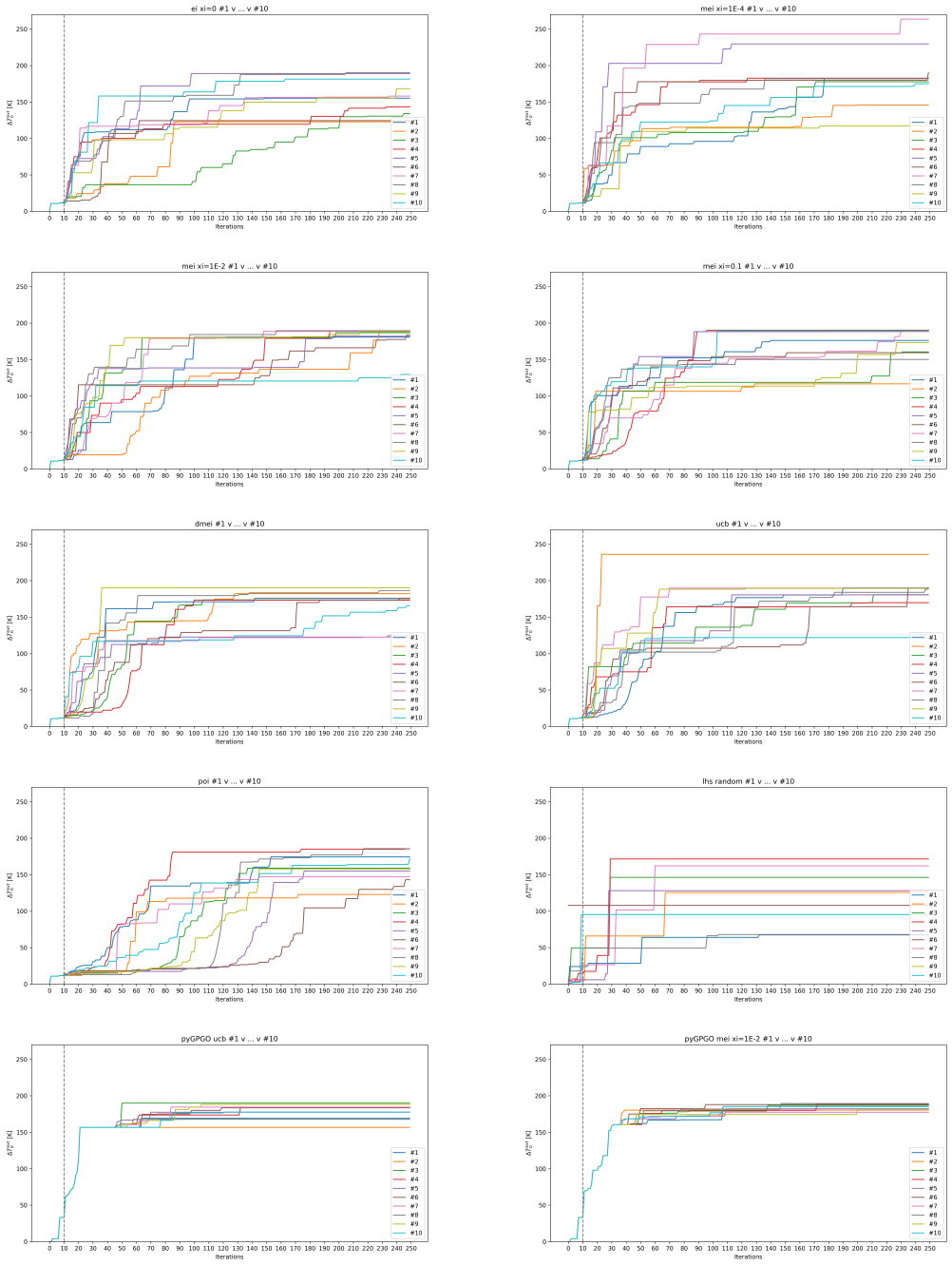


Figure 3: Each plot shows the maximum target value at that iteration for all 10 runs of the corresponding acquisition function. A large variability can be seen between runs, which explains the large sample standard deviations encountered in sections 4.5 and 4.6.