Accelerating Bayesian Inference of expensive Likelihoods with Gaussian Processes

von

Jonas Elias El Gammal

Masterarbeit in Physik

vorgelegt der

Fakultät für Mathematik, Informatik und Naturwissenschaften

der

RWTH Aachen

im Dezember 2020

angefertigt im

Institut für Theoretische Teilchenphysik und Kosmologie (TTK)

bei

Prof. Dr. Julien Lesgourgues

Abstract

Numerically approximating multidimensional posterior distributions can be very expensive when evaluating the likelihood function involves expensive numerical computation. At the same time many likelihoods in physics show a "speed hierarchy" between the different dimensions of the parameter space which means that recomputing the likelihood function is much more expensive when changing some parameters than others. This naturally arises when some of these parameters come from theoretical models while others are associated to the data. Recently some attempts have been made at fast Bayesian inference using Bayesian quadrature [1, 2, 3] to reduce the number of samples required for mapping the posterior distributions drastically. While this approach works well in low dimensions it becomes prohibitively expensive if the number of dimensions exceeds $d \gtrsim 10$. Additionally these approaches cannot take advantage of the aforementioned speed hierarchy in the likelihood. In this thesis we develop an algorithm which mitigates these problems and improves on the current state of the art by (i) introducing a novel acquisition function which is well suited to performing Bayesian quadrature of log-probability distributions (ii) accelerating the Kriging believer [4] batch acquisition algorithm with blockwise matrix inversion [5] and (iii) Proposing an algorithm which can take advantages of speed hierarchies by marginalizing nuisance parameters with the `PolyChord` nested sampling algorithm [6, 7]. We test these algorithms on gaussian toy likelihoods and real cosmological likelihoods and report a decrease in wall clock time of up to several orders of magnitude for mapping the posterior space.

Erstgutachter und Betreuer

Prof. Dr. Julien Lesgourgues
Institut für Theoretische Teilchenphysik
und Kosmologie (TTK)
RWTH Aachen University

Zweitgutachter

Prof. Dr. Felix Kahlhoefer
Institut für Theoretische Teilchenphysik
und Kosmologie (TTK)
RWTH Aachen University

Contents

1 Introduction

Bayesian inference is one of the main tools which is used in science for comparing theories to data and for quantitatively analysing the parameters which govern these theories. Doing these analyses does however involve integrating the posterior distribution along one or multiple axes which usually has to be done numerically.

In the last decades the Metropolis Hastings MCMC algorithm has become one of the main methods for performing these numerical integrations as it is easy to implement, robust and scales well with the number of dimensions. It does however need a large number of posterior evaluations $\gtrsim 10^3$ to correctly recover the shape of the posterior. At the same time calculating likelihood and posterior distributions for comparing theoretical models to data can be computationally very expensive as it often involves heavy numerical calculations such as integrals, differential equations or simulations. This coupled with the high number of evaluations needed for MCMC poses a challenge to scientists to the point where it makes some analyses outright impossible due to the computational overhead involved.

This problem has recently gained a lot of attention by the machine learning community where numerous proposals for more efficient algorithms have been made. One of these algorithms which is particularly promising is Bayesian quadrature (BQ) which relies on Gaussian Process (GP) regression which is a Bayesian approach to non-parametric interpolation that has been applied to numerous machine learning problems in the past. This powerful approach needs less samples from the posterior to converge to the correct distribution but this unfortunately comes at the expense of added computational overhead and bad scaling with the number of dimensions. Furthermore these algorithms cannot exploit speed hierarchies (i.e. the difference in computation times when changing different parameters) that are inherent to many likelihoods in physics and that arise when an expensive to compute theoretical model is compared to data. These practical disadvantages have so far limited BQ to a small number of select problems and MCMC is still considered the gold standard in Bayesian inference.

The goal of this thesis is to develop a robust framework for BQ which extends the useful range of this approach to moderately high dimensions while limiting the computational overhead. Furthermore we will try to take advantage of the aforementioned speed hierarchies to increase performance further.

Chapter 2 will be a brief review of Bayesian inference where most of the relevant terminology and notation is introduced along with a brief discussion of the numerical considerations that have to be made when performing Bayesian inference. This is followed by the introduction of two of the most commonly used, state of the art algorithms for Bayesian inference, Metropolis Hastings MCMC and nested sampling. We discuss the numerical implications of these algorithms as well as the number of posterior evaluations required for convergence.

In chapter 3 we explain the concept of GPs with a focus on providing intuitive explanations in addition to establishing the relevant notation. This is followed by a detailed discussion of the role of the kernel function and the description of the GP regression algorithm. Next the idea of BQ is introduced together with active sampling.

Since our goal in this thesis is to perform BQ on probability distributions, chapter 4 will focus on discussing the considerations that have to be made for efficiently sampling this class of functions. First we motivate the use of BQ for probability distributions and highlight the specific advantages that this algorithm has compared to MCMC. Afterwards we introduce a power reduction operation followed by the derivation of a novel acquisition function which is appropriate for the efficient characterization of log-probability distributions. Next we discuss the preprocessing of the data followed by a proposal for an efficient implementation of the Kriging believer algorithm which allows for batch acquisition and hence for parallel evaluation of the posterior. Lastly the algorithm is completed by introducing a convergence criterion. This is followed by a discussion of two of the main problems that this algorithms brings with it. We then perform some experiments on artificial as well as real likelihoods and evaluate the performance against MCMC.

In chapter 5 we present a novel algorithm that takes advantage of the speed hierarchies that are present in many likelihoods by using both nested sampling and BQ in the Bayesian inference procedure. This is first motivated, then the idea of the algorithm is presented in detail followed by some considerations regarding the projected speed-up compared to MCMC. After this we test this algorithm on artificial likelihoods and report its performance.

2 An introduction to Bayesian inference

In this chapter we will give a brief introduction into the field of Bayesian inference. Although most of the relevant properties and mathematical definitions will be provided we will assume that the reader is familiar with the basic concepts of statistics and probability theory on the level of an introductory lecture. As Bayesian inference relies heavily on numerical methods this chapter will also provide some insight into the most commonly used numerical algorithms.

Section 2.1 provides a brief overview of the origin of Bayesian inference in probability theory and establishes the terminology used throughout this thesis. This is followed by a discussion of the role of priors and we will touch on some of the numerical difficulties that appear when performing Bayesian inference. T

he Markov Chain Monte Carlo algorithm, a widely used and generally considered as state of the art algorithm is introduced in section 2.2. Nested sampling, which is another state of the art algorithm which takes a different approach at numerical Bayesian inference is introduces in section 2.3.

Although this can hardly be considered a complete review we will try to address the specific advantages and shortcomings of these two different approaches as they will play an important role later in this thesis.

2.1 Bayesian inference

2.1.1 Bayes theorem

At the foundation of Bayesian inference lies Bayes theorem which states that for two events A and B with probabilities $P(A)$ and $P(B)$ the *conditional* probability of A given B is given by [8]

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \ . \tag{2.1}$$

The conditional probability $P(A|B)$ describes the probability of event A happening given that event B has happened. Bayes' theorem is one of the foundations of statistics and can very simply be derived from the definition of conditional probability [9].
The goal in Bayesian inference is to use this theorem to to update the probability of some *hypothesis* given some *data*. In this context Bayes' theorem states that

$$P(\text{hypothesis}|\text{data}) = \frac{P(\text{data}|\text{hypothesis})P(\text{hypothesis})}{P(\text{data})} \tag{2.2}$$

In Bayesian inference these quantities have special names [8]:

- $P(\text{hypothesis})$ is called the *prior* since it encodes the prior belief about the likelihood of the hypothesis
- $P(\text{hypothesis}|\text{data})$ is called the *posterior*.
- $P(\text{data}|\text{hypothesis})$ is called the *likelihood*.

- $P(\text{data})$ is called *marginal likelihood* or *model evidence.* This quantity usually does not need to be computed as it is the same for all hypotheses.
  Furthermore probability theory demands that for an exhaustive set of n hypotheses h_n and some data d
  $$\sum_{i=1}^{n} P(h_n|d) \equiv 1 \ . \tag{2.3}$$
  This means that the value of the evidence is fixed by the other quantities appearing in Bayes' theorem and that it can be computed by evaluating
  $$P(d) = \sum_{i=1}^{n} P(\text{data}|\text{hypothesis})P(\text{hypothesis}) \ . \tag{2.4}$$

2.1.2 Bayes theorem for probability distributions

In most cases Bayesian inference is not used for discrete probabilities but for continuous probability distributions. In this case one is usually interested in inferring some underlying parameters θ where $x \sim p(x|\theta)$ given some data $\boldsymbol{X} = \{x_1, \dots, x_n\}$. The parameters θ themselves again have some underlying distribution $\theta \sim p(\theta|\alpha)$ with some *hyperparameters* α.
In this case $p(\theta|\alpha)$ takes the role of the prior, and $p(\boldsymbol{X}|\theta)$ that of the likelihood. These take the form of continuous probability distributions[1] which means that Bayes' theorem is modified accordingly: The posterior distribution is

$$p(\theta|\boldsymbol{X},\alpha) = \frac{p(\boldsymbol{X}|\theta,\alpha)p(\theta|\alpha)}{p(\boldsymbol{X}|\alpha)} = \frac{p(\boldsymbol{X}|\theta,\alpha)p(\theta|\alpha)}{\int p(\boldsymbol{X}|\theta,\alpha)p(\theta|\alpha)\,\mathrm{d}\theta} \propto p(\boldsymbol{X}|\theta,\alpha)p(\theta|\alpha) \ .$$

As the prior does not depend on the data $\boldsymbol{X}$ the focus when performing Bayesian inference is usually placed on computing the correct Likelihood which is often written as $L_{\boldsymbol{X}}(\theta,\alpha) = L(\boldsymbol{X}|\theta,\alpha) = p(\boldsymbol{X}|\theta,\alpha)$. This emphasizes that the free parameters of the likelihood are θ and α.

Furthermore it is popular to take the logarithm of the Likelihood function which gives rise to the *log-likelihood*:

$$\mathcal{L} = \log(L) \tag{2.5}$$

The main motivation for this redefinition is the fact that the logarithm is a monotonic function which implies that any maximum of the Likelihood function is also a maximum of the log-Likelihood function. This is useful since we are usually interested in maximizing the posterior distribution which is a function of the likelihood. The reason that the log-Likelihood function is used instead is that it is often easier to compute than the Likelihood. For instance for i.i.d. data with common density the log-Likelihood can be

[1]While the prior needs to be a true (normalized) probability distribution the likelihood does not necessarily need to meet this criterion. In fact since the likelihood is a probability defined on the data space it is almost never normalized in the parameter space.

written as a sum

$$\mathcal{L}_{\boldsymbol{X}}(\theta) = \sum_{x\in\boldsymbol{X}} \mathcal{L}_x(\theta)$$

which is generally easier to process than the product appearing in the full likelihood. This can be especially beneficial in cases where derivatives of the Likelihood are required. An additional advantage of using the log-likelihood function is the fact that the likelihood function can only take positive values which are often either very small or very large while the log-likelihood function maps to $\mathbb{R}$ which is numerically beneficial for computation.

Similarly the marginal distribution can be calculated by taking the continuity limit of the sum in eq. 2.4 which gives rise to the integral:

$$p(\boldsymbol{X}|\alpha) = \int p(\boldsymbol{X}|\theta,\alpha)p(\theta|\alpha)\,\mathrm{d}\theta \tag{2.6}$$

Of course θ does not necessarily have to be a single parameter but can also be a set of n parameters $\theta = \{\theta_1,\dots,\theta_n\}$. In this case the integral over one or multiple of those parameters is usually still called the marginal distribution:

$$p(\boldsymbol{X}|\tilde{\theta},\alpha) = \int p(\boldsymbol{X}|\theta,\alpha)p(\theta|\alpha)\,\mathrm{d}\overline{\theta} \tag{2.7}$$

where $\tilde{\theta} \subset \theta$ and $\overline{\theta} = \theta\backslash\tilde{\theta}$. $\overline{\theta}$ are then also referred to as *nuisance parameters*. Since ambiguity in the terminology can sometimes be confusing one usually speaks of "marginalizing over $\overline{\theta}$".

An important example of a likelihood is that of n i.i.d. random variables $\boldsymbol{X} = (X_1,\dots,X_n)$ with some unknown mean $\mu \in \mathbb{R}$ and variance $\sigma^2 \in \mathbb{R}^+$:

$$L_{\boldsymbol{X}}(\mu,\sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(X-\mu)^2}{2\sigma^2}\right) \ . \tag{2.8}$$

This is a prime example of the practical advantages of taking the log-Likelihood since this is much easier to compute:

$$\mathcal{L}_{\boldsymbol{X}}(\mu,\sigma^2) = \sum_{i=1}^{n} -\log(2\pi\sigma^2) - \frac{(X-\mu)^2}{2\sigma^2} \tag{2.9}$$

2.1.3 Priors

Since the prior distribution assigns a probability to a hypothesis before looking at any data it should reflect the knowledge about the problem at hand (or the lack thereof) that can be drawn from initial considerations. As such a prior can e.g. restrict the possible domain of the parameters θ or encode some belief about the nature of the distribution.

Furthermore the posterior distribution of one combination of hypothesis and data can often be the prior for a different one. This is often the case if several experiments give rise to likelihoods for the same set of parameters.

In many cases we do not have very accurate information about the nature of the distribution on which we want to perform Bayesian inference in which case it is often advisable to construct a prior distribution which translates the information provided by the likelihood into the posterior while adding minimal bias. Many attempts at this have been made and this is still an area of active research. A full review of this would be beyond the scope of this thesis however it is worth mentioning that the most commonly used priors in practice are the flat or uniform prior and the gaussian prior. These take the shape of a uniform distribution and a gaussian distribution respectively. For additional information on this this topic please refer to [8, 10].

2.1.4 Numerical considerations

In Bayesian inference the main focus usually lies on computing the posterior distribution for a given theory and marginalizing over different combinations of its parameters θ, usually for visualization of the distribution and for the inference of marginalized quantities such as confidence intervals.

While computing the marginal distribution can be performed analytically for a select group of simple posterior distributions, this integral has to be calculated numerically for most applications in physics as the Likelihood function either has an analytically intractable integral or is not given as an analytic function in the first place but has to be calculated numerically.

This coupled with the fact that θ is often high dimensional (e.g. the Planck 2018 Likelihood has 27 free parameters [11]) requires careful consideration of the integration method used. In particular this calls for an efficient algorithm which allows a robust characterization of the full parameter space θ while also enabling easy marginalization over any set of dimensions of this space.

Some of these algorithms will be presented in the following.

2.2 Markov Chain Monte Carlo

2.2.1 Markov Chains

To understand the concept of Markov Chain Monte Carlo (MCMC) we first have to establish the definition of a Markov Chain [10]

Definition 2.1. A Markov Chain (MC) is a discrete stochastic process in which every next step in the process depends only on the current step. This property is also called *Markov property.* Formally it can be defined as a sequence of random variables $X_1, X_2, \ldots, X_n$ such that

$$p(X_{n+1}|X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = p(X_{n+1}|X_n = x_n) \ . \tag{2.10}$$

This means that at any point the chain only depends on the location of the previous sample which has the computational advantage that only two locations X_n, X_{n-1} need to be kept in the main memory at any time while the rest of the chain can be saved to the hard drive. In addition this property also implies that any sub-chain of an MC containing $X_j, X_{j+1}, \dots X_k$ where $1 < j < k < n$ is again an MC.

2.2.2 Markov Chain Monte Carlo

Markov Chain Monte Carlo is the most commonly used and most thoroughly studied algorithm which fulfills the requirements for an efficient integration algorithm which were established earlier. The idea of this algorithm is to construct a process which approaches the sampled probability density as its stationary distribution. In other words the goal is to construct a chain which follows a path in the parameter space such that the normalized histogram of values approaches the probability density function (PDF) [9].

This is done by constructing a Markov Chain which is defined as a sequence of possible events where the probability of each event only depends on the state attained in the last step. This requires some rule which describes the probability to jump from one value in the likelihood to the next. One of the simplest and most widely used algorithms for this is the Metropolis-Hastings algorithm.

For a probability distribution $p(x)$ that we want to map, this algorithms works by introducing an acceptance rate α which gives the probability that the chain will jump to a drawn value. This acceptance rate is calculated by sampling a point from some arbitrary probability density $g(x'|x_t)$ that suggests a candidate point x' based on the the previous candidate sample x_t. A common choice for g is a Gaussian distribution which is centered on x_t.

Having drawn x' from $g(x', x_t)$ the *acceptance ratio* α is calculated[2]:

$$\alpha = \frac{p(x')}{p(x_t)} \tag{2.11}$$

This acceptance ratio is compared to a uniformly drawn number $u \in [0, 1]$:

- If $\alpha \geq u$ the candidate is accepted and $x_{t+1} = x'$.
- If $\alpha < u$ the candidate is rejected and $x_{t+1} = x_t$

This way the chain always jumps towards higher values of p while the probability for jumping to points which have a value which is lower than the current one is smaller than 1. Furthermore it is noteworthy that g can only depend on the current value of the chain x_t to conserve the Markov property (see Def. 2.1).

The integration of the Monte Carlo approximation of the PDF can be done by summing the histogram of samples along one or multiple axes. It is one of the strengths of MCMC that this summation is computationally very cheap and that this histogram is a natural by-product of the Markov Chain. Furthermore the computational overhead of this method is independent of the number of samples in the chain and only consists of the computation of the proposal distribution, the evaluation of the likelihood and posterior function and the acceptance rate of the Metropolis Hastings proposal.

[2]It is important to note that we are in fact not mapping $p(x)$ itself but a target distribution $f(x)$ which is proportional to $p(x)$. This rarely matters in practical applications though since we can easily normalize the histogram.

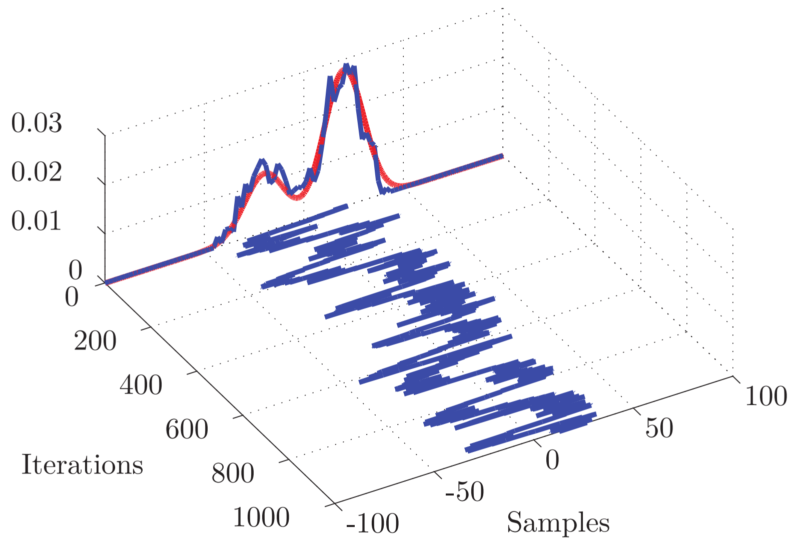

Figure 2.1: Example of the Metropolis Hastings MCMC algorithm on a mixture of two gaussian distributions with a uniform prior in $[-100, 100]$. The chain consists of 1000 steps with a gaussian proposal distribution $\mathcal{N}(X_n, 8)$ that is centered on X_n and has a standard deviation of 8. One can see how the normalized histogram of samples converges towards the real posterior distribution in the back. The image has been taken from [9].

Disadvantages to this method are that a lot of information about the posterior distribution is simply ignored. As such the chain is oblivious to rejected samples and the values of the posterior at the sampling locations. In addition it is somewhat difficult to calculate the total evidence since the values of the likelihood are discarded during sampling. Another issue is that MCMC requires a lot of fine tuning to correctly map multimodal distributions. An illustration of Metropolis-Hastings-MCMC of a one-dimensional distribution is shown in Fig. 2.1.

As mentioned above the Metropolis-Hastings algorithm is one of the most commonly used acceptance rules for MCMC. However there are other methods, most notably Hamilton Monte Carlo[3] (HMC) [12] and a adjusted version of the Metropolis-Hastings algorithm called Gibbs sampling for multidimensional distributions where instead of sampling from the joint distribution and integrating over it, one sequentially samples from the conditional distribution in each dimension [10].

2.3 Nested Sampling

An alternative approach to MCMC which has become increasingly popular recently due to its robustness against multimodality and the fact that it can calculate the total evidence is the nested sampling algorithm which has been pioneered by John Skilling [6] and this section closely follows this paper. This algorithm uses a principle which is

[3]In the literature this is also commonly referred to as Hybrid Monte Carlo

similar to that of Lebesgue integration to find an estimate for the evidence

$$Z = \int L(\theta)\pi(\theta)\,\mathrm{d}\theta \tag{2.12}$$

where $L(\theta)$ is the likelihood function and $\pi(\theta)$ the prior. While this integral is straightforward to compute numerically (e.g. by using the trapezoidal rule) if θ is low-dimensional and $L(\theta)\pi(\theta)$ is cheap to evaluate, one needs a more efficient algorithm if these two conditions are not met.
The nested sampling algorithm is such an algorithm which is obtained by observing that we can rewrite Eq. 2.12 by substituting

$$X(\lambda) = \int_{L(\theta)>\lambda} \pi(\theta)\,\mathrm{d}X\ . \tag{2.13}$$

For increasing λ the enclosed mass X hence decreases from 1 to 0 (due to the fact that $\pi(\theta)$ is a probability distribution). By renaming $L(X)$ to the inverse function i.e. $L(X(\lambda)) \equiv \lambda$ the evidence reduces to a one dimensional integral

$$Z = \int_0^1 L(X)\,\mathrm{d}X\ . \tag{2.14}$$

This transformation is advantageous since now the integrand $L(X)$ is positive and decreasing.
While this integral may look very easy at first the problem which remains is that inverting L is not trivial and usually cannot be done analytically. Intuitively this would be done by dividing the θ-space into a fine grid, evaluating the likelihood for each of these bins and sorting them by value

$$0 < X_m < \cdots < X_2 < X_1 < 1$$

where we assume that we have divided the space into m parts. The weighted sum

$$\sum_{i=1}^{m} w_i L_i \to Z \tag{2.15}$$

then converges to the value of the integral where $w_i = \Delta X$ is the distance between X_i and X_{i+1}. This corresponds to approximating the integral by summing up the histogram of the inverted likelihood. As such it is an approximate form of Lebesgue integration. An illustration of this approach is shown in Fig. 2.2.

In practice one wants the sampling to be linear in $\log(X)$ instead of X because in most real world cases the bulk of the posterior mass only occupies a small fraction e^{-H} of the prior volume where

$$H = \int \log(\mathrm{d}P/\mathrm{d}X)\,\mathrm{d}P \tag{2.16}$$

which is achieved by introducing a variable t such that

$$X_1 = t_1,\ X_2 = t_1t_2, \ldots, X_i = t_1t_2\ldots t_i, \ldots X_m = t_1t_2\ldots t_m \tag{2.17}$$

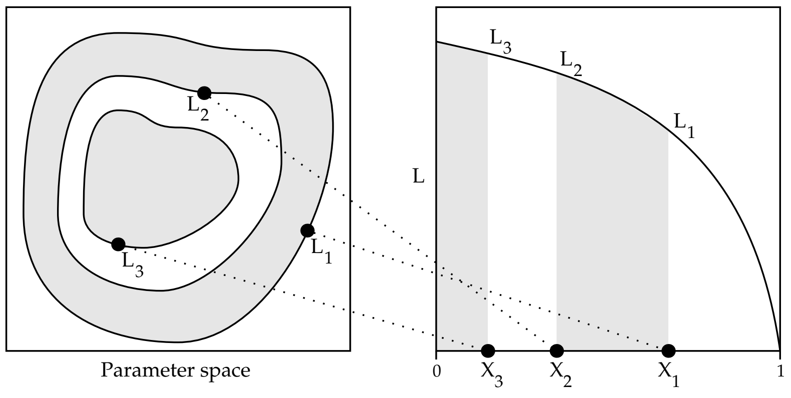

Figure 2.2: Illustration of the nested sampling approach. Each iso-likelihood contour L_i (left) in the parameter space corresponds to a value X_i (right) in the inverted likelihood space. The evidence can be computed by numerically approximating $Z = \int_0^1 L(X)\,\mathrm{d}X$ with the weighted sum in Eq. (2.15). Image taken from [6].

where each t_i is a value between 0 and 1. We can express this explicitly by writing

$$\sum_{i=1}^{m} w_i(\boldsymbol{t})L_i \to Z(\boldsymbol{t}) \ . \tag{2.18}$$

What remains is to set precise values for $\boldsymbol{t}$ and while this is difficult to set exactly it can be done statistically. The reason for this is that for any point X_i which is randomly drawn from the prior with the constraint that $X_i < X_{i-1}$ with $X_0 = 1$, we know that $X_i = t_i X_{i-1}$ where t_i is drawn from the uniform distribution covering $[0, 1)$. Since the constraint $X_i < X_{i-1}$ is equivalent to $L_i > L_{i-1}$ with $L_0 = 0$ we can use this to sample directly from the likelihood instead of having to invert it.

The idea of nested sampling is then to draw a convenient number N of points from the prior and at each iteration select the worst point (the one with the lowest L and highest Z) as the i'th point. The recurrence for X is then given by

$$X_0 = 0, X_i = t_i X_{i-1} \tag{2.19}$$

and since t_i is the largest of N numbers drawn from $\texttt{uniform}[0, 1)$ the probability for obtaining a value t_i is given by

$$p(t_i) \sim N t_i^{N-1}$$

If we treat this like a distribution in $\log(t)$ we can easily calculate the expectation value and variance analytically which yields

$$\mathbb{E}(\log(t)) = -1/N \qquad \mathrm{var}(\log(t)) = \frac{1}{N^2} \ . \tag{2.20}$$

The expectation value can then be used as an approximation for t_i. The point which has been previously selected is then deleted and replaced by another point which is drawn from the prior with the constraint that $L > L_i$. Finding an independent point that follows the prior distribution and fulfils $L > L_i$ can be a bit tricky if the bulk of the posterior volume is contained in a small region of the prior.

We can make our lives easier though by not sampling directly from the prior but rather evolving one of our N points which by definition have higher likelihood values than L_i with some algorithm which makes the new sample approximately independent from L_i, for example a sufficiently decorrelated MCMC chain. This procedure is done iteratively until the required precision is reached.

Conveniently this algorithm also allows us to estimate the uncertainty of Z. Since the values for $\boldsymbol{t}$ in the weighted sum $\sum_i L_i w_i(\boldsymbol{t})$ are set statistically instead of actually inserting the correct value for $\boldsymbol{t}$ this induces an uncertainty in the estimate of Z. Luckily we can estimate the uncertainty in $\boldsymbol{t}$ by observing that the distribution for t_i gives rise to a "sequence probability"

$$\Pr(\boldsymbol{t})\,\mathrm{d}\boldsymbol{t} = \prod_i N t_i^{N-1} \tag{2.21}$$

which in turn induces a probability for the estimate of Z:

$$\Pr(Z) = \int \delta\left(Z - \sum_{i-1}^{m} L_i w_i(\boldsymbol{t})\right) \Pr(\boldsymbol{t})\,\mathrm{d}\boldsymbol{t} \tag{2.22}$$

The moments of this distribution can then be determined by any integration algorithm that is convenient. In practice MCMC is commonly used for this. In addition the integration is often done in log space as it is numerically more convenient and usually the the evidence itself is also calculated in log space.
Pseudocode of the full algorithm is shown in algorithm 1.

```
     Input: $N$ initial points $\theta = \{\theta_1, \dots, \theta_N\}$ from prior
[1]  $Z := 0$
[2]  $X_0 := 1$
[3]  for $i = 1, 2, \dots, j$ do
[4]      $L_i = \min(L(\theta))$
[5]      $X_i := \exp(-i/N)$ (crude) or sample $X_i$ to get uncertainty     eq. (2.20,2.21)
[6]      $w_i := X_{i-1} - X_i$ (simple) or $(X_{i-1} - X_{i+1})/2$ (trapezoidal)
[7]      $Z = Z + L_i w_i$
[8]      replace $L_i$ with point drawn from $\pi(\theta)$ with $L(\theta) > L_i$
[9]  end
[10] $Z = Z + \frac{1}{N} L(\theta_1 + \dots + L(\theta_N))X_j$
[11] return $Z$
```

Algorithm 1: Pseudocode of the nested sampling algorithm. The last step (line 10) takes advantage of the points left after j steps. If the algorithm has converged this should only add a small correction and could very well be omitted. The pseudocode is taken from [6].

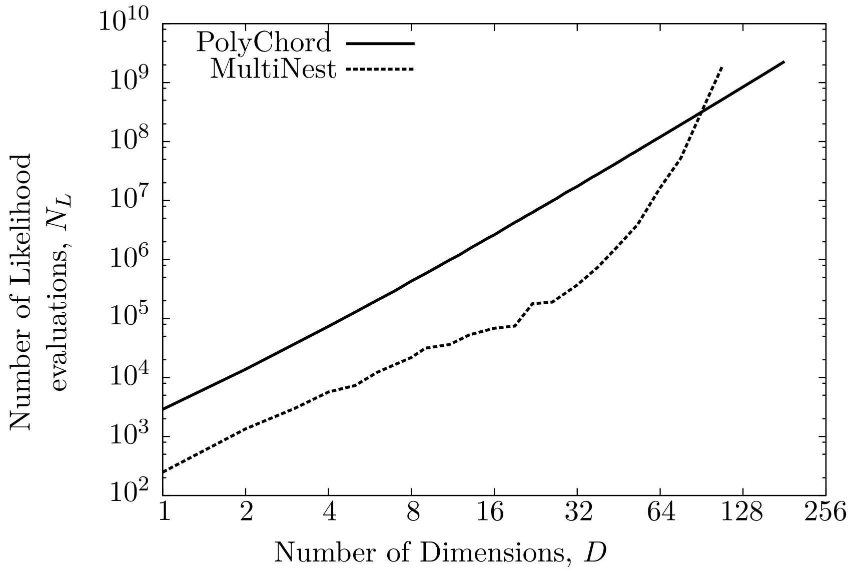

Figure 2.3: Number of evaluations needed to achieve convergence on a unimodal gaussian likelihood with nested sampling. The two programs, `PolyChord` [7, 13] and `MultiNest` [14, 15, 16] are different implementations of the basic nested sampling algorithm. One can see the exponential scaling of `MultiNest` emerging in high dimensions while `PolyChord` scales polynomially which comes from the differences in the ways both algorithms select their samples. The image has been taken from [7].

2.4 Typical computational complexity

The speed of convergence for both MCMC and nested sampling algorithms depends on a number of factors. These are the desired accuracy of the sampling (usually estimated by a convergence criterion), the number of modes of the posterior distribution (where a large number of modes typically means slower convergence) as well as the shape of the posterior distribution. For instance "banana"-shaped posterior distributions are challenging for these algorithms and thus need a higher number of evaluations.

Another important factor is the dimensionality of the posterior distribution. Algorithms typically either scale exponentially with the number of dimensions or polynomially with $N_L \propto \mathrm{const} \cdot D^{\rho}$. This scaling for two different nested sampling algorithms and simple, unimodal gaussian distributions is shown in Fig. 2.3. Similar data for an MCMC algorithm which is described in [17, 18] is shown in Fig. 2.4. Again random unimodal gaussians were used as posterior distributions. One can see that the number of evaluations for both algorithms is comparable.

The number of samples shown here can be seen as a lower bound on the number of samples which need to be evaluated to achieve convergence since unimodal gaussian distributions are especially simple to sample.

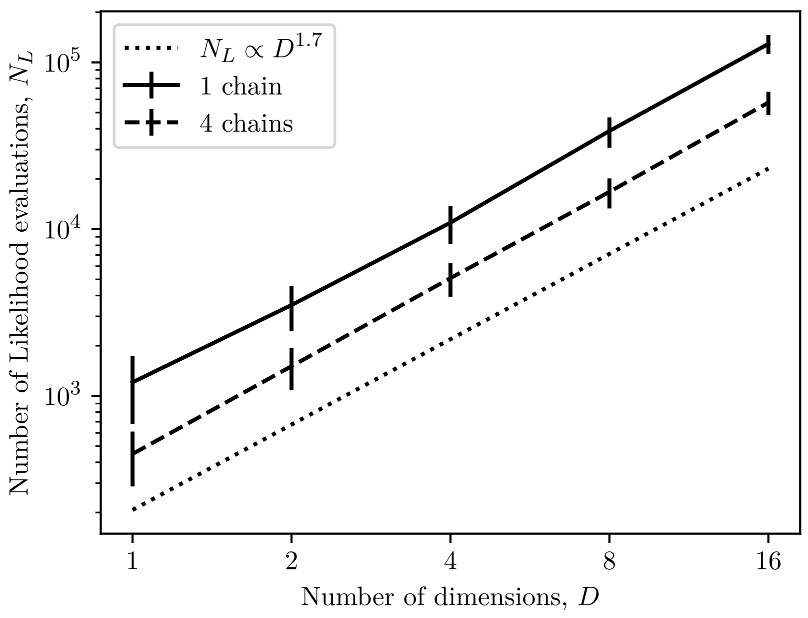

Figure 2.4: Number of evaluations to achieve convergence on a unimodal gaussian likelihood with MCMC. The algorithm used is a modified version of the Metropolis Hastings algorithm that is described in [17, 18]. 50 (10 in 16 dimensions) random, mildly correlated gaussian distributions were drawn and sampled using adaptive covariance matrix learning. The empirical mean and standard deviation for the number of evaluations per chain are shown for one and four chains. Furthermore the theoretical scaling which is $\propto D^{1.7}$ is shown. The number of chains corresponds to the number of parallel processes which can be run.

3 Gaussian Processes

This chapter gives an overview on Gaussian Processes (GPs), as these will be used later to present an alternative approach to using MCMC or nested sampling methods to construct a model of the posterior.

First, the concept of GPs and their mathematical foundations are introduced in section 3.1 followed by an intuitive description of the practical considerations that need to be taken into account when using GPs. The kernel function is discussed in detail in section 3.2, followed by examples of the most commonly used kernel functions.

Next, GP regression is introduced in section 3.3. This part of the chapter closely follows chapter 1 and 2 of [19] and can be viewed as a brief introduction to the main concepts of GP regression.

Section 3.4 introduces Bayesian quadrature together with a review of Bayesian optimization and active sampling in 3.4.1. As these topics are still areas of ongoing investigation we will try to give an overview of the current state of research.

3.1 Concept

While GPs are mathematically simple and well defined, getting an intuitive understanding of how they work requires us to introduce the concept of a stochastic process. Omitting the mathematical details it can be thought of as a function

$$\{Y(t) : t \in T\}$$

where Y is a random variable drawn from some probability measure P. T is often referred to as *index set.* Important examples of stochastic processes apart from GPs are Markov- and Wiener processes [20].
Armed with this idea of stochastic processes we can formally define GPs:

Definition 3.1. A Gaussian Process is a collection of random variables, any finite number of which have a joint Gaussian distribution. [19]

This means, that a GP is a stochastic process defined on any set $T = \{t_1, ..., t_n\}$ where the n values $\{y_1, ..., y_n\}$ are drawn from a joint Gaussian distribution

$$\mathcal{N}(t|\boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^n \det(\Sigma)}} \exp\left(-\frac{1}{2}(\boldsymbol{t}-\boldsymbol{\mu})^\top \Sigma^{-1} (\boldsymbol{t}-\boldsymbol{\mu})\right) \tag{3.1}$$

with

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \vdots \\ \mu_n \end{pmatrix}, \ \Sigma = \begin{pmatrix} \Sigma_{11} & \dots & \Sigma_{1n} \\ \vdots & \ddots & \vdots \\ \Sigma_{n1} & \dots & \Sigma_{nn} \end{pmatrix} .$$

$\boldsymbol{\mu}$ is called the mean vector (of length n) and Σ the covariance matrix (of size $n \times n$). While in principle any set T is allowed for a valid GP,[4] in practice only the case where

[4]In fact any multivariate Gaussian distribution between variables in the same space is a GP.

T is continuously defined on $\mathbb{R}$ is of particular interest[5]. In the following the index set of the continuous GP will be denoted as X.

Nevertheless it is useful to look at the finite integer set $\{1, ..., n\}$ in order to gain an intuitive understanding of how a GP defined on a continuous domain works.Fig. 3.1 shows how such a discrete GP looks like for $n = 5$ and $n = 30$. One observes that the GP is fully described if for every two points $t, t' \in T$ there exist (i) two mean values μ, μ' and (ii) a covariance matrix Σ. These can be expressed as two functions (here in the continuous case with the index set X):

(i) the mean function $m(x)$ and (ii) the covariance function (often called *kernel*) $k(x, x')$ with $x, x' \in X$. As such a continuous GP is a distribution over functions $f(x)$ which can be expressed as

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')) \tag{3.2}$$

where

$$\begin{aligned} m(x) &= \mathbb{E}[f(x)] && (3.3)\\ k(x, x') &= \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))] \ . && (3.4)\end{aligned}$$

Note however that the definition of a stochastic process automatically implies a consistency requirement (also called Kolmogrov's extension theorem [21]) which demands that any GP that specifies $(y_1, \ldots, y_n) \sim \mathcal{N}(x|\boldsymbol{\mu}, \Sigma)$ on any set X must equally specify $(y'_1, \ldots, y'_n) \sim \mathcal{N}(x'|\boldsymbol{\mu}', \Sigma')$ for any subset $X' \subset X$ by taking the relevant parts of $\boldsymbol{\mu}$ and Σ. In other words this means, that for $X = \mathbb{R}$ one does not need to know the (infinite-dimensional) full distribution between all indices to make predictions about a finite subset of indices.

3.1.1 Conditioning

In practice one does not only want to draw values from the prior distribution but wants to incorporate knowledge about a set of training points $\{(x_i, y_i = f_i)|i = 1, \ldots, n\}$. In the GP framework the joint distribution of these training points $\boldsymbol{f}$ and an arbitrary set of test points $\boldsymbol{f}_*(\boldsymbol{x}_*)$ is

$$\begin{bmatrix}\boldsymbol{f}\\ \boldsymbol{f}_*\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{m}(\boldsymbol{x})\\ \boldsymbol{m}(\boldsymbol{x}_*)\end{bmatrix}, \begin{bmatrix}K(\boldsymbol{x}, \boldsymbol{x}) & K(\boldsymbol{x}, \boldsymbol{x}_*)\\ K(\boldsymbol{x}_*, \boldsymbol{x}) & K(\boldsymbol{x}_*, \boldsymbol{x}_*)\end{bmatrix}\right) \tag{3.5}$$

such that if there are u training and v test points $\boldsymbol{m}(\boldsymbol{x})$ is the vector of length u with $\boldsymbol{m}(\boldsymbol{x})_i = m(x_i)$ and $K(\boldsymbol{x}, \boldsymbol{x})_{ij} = k(x_i, x_j)$ with dimension $u \times u$. Similarly $K(\boldsymbol{x}_*, \boldsymbol{x})$ has dimension $v \times u$ and $K(\boldsymbol{x}_*, \boldsymbol{x}_*)$ dimension $v \times v$. The matrix

$$K(\boldsymbol{x}, \boldsymbol{x}')_{ij} = k(x_i, x'_j) \tag{3.6}$$

for any kernel k is called the *Gram matrix* between $\boldsymbol{x}$ and $\boldsymbol{x}'$.
As the training points are known observations, it is necessary to *condition* the joint

[5]Actually the case where $X \subset \mathbb{R}^D$ is even more interesting for our purposes. How the formalism can be extended to this case is described in section 3.2.2

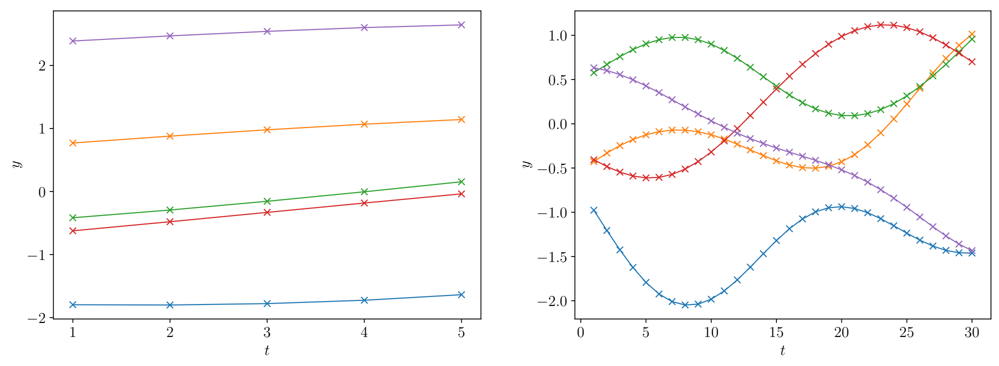

Figure 3.1: Left: Samples drawn from a multivariate gaussian distribution with $m(t) = 0$ and $k(t,t') = \exp\left(-\frac{(t-t')^2}{200}\right)$. The samples are indexed by the set $\{1,2,3,4,5\}$. Right: Samples drawn from the same multivariate gaussian distribution defined on the index set $\{t \in \mathbb{N}, 1 \leq t \leq 30\}$. Intuitively one can see how a the samples describe smooth functions. For the continuous index set X, $m(x)$ and $k(x,x')$ can be chosen as smooth functions $\mathbb{R} \to \mathbb{R}$ which makes the transition from the discrete index set T to the continuous index set X obvious.

probability on the observed values[6]. Fortunately conditioning for multivariate Gaussians is very simple:

$$\boldsymbol{f}_* | \boldsymbol{x}_*, \boldsymbol{x}, \boldsymbol{f} \sim \mathrm{N}(\overline{\boldsymbol{f}}_*, \Sigma_{\boldsymbol{f}_*}) \tag{3.7}$$

with

$$\overline{\boldsymbol{f}}_* = \boldsymbol{m}(\boldsymbol{x}_*) + K(\boldsymbol{x}_*, \boldsymbol{x}) K(\boldsymbol{x}, \boldsymbol{x})^{-1} (\boldsymbol{f} - \boldsymbol{m}(\boldsymbol{x})) \tag{3.8}$$

$$\mathrm{cov}(\boldsymbol{f}_*) = \Sigma_{\boldsymbol{f}_*} = K(\boldsymbol{x}_*, \boldsymbol{x}_*) - K(\boldsymbol{x}_*, \boldsymbol{x}) K(\boldsymbol{x}, \boldsymbol{x})^{-1} K(\boldsymbol{x}, \boldsymbol{x}_*) \tag{3.9}$$

This conditioned GP is then called the *posterior GP*.

One thing to note about these equations is, that even for a zero-mean prior function $m(x) = 0 \; \forall x$ it is possible to obtain a non-zero posterior mean value for the conditioned distribution. This is motivates the choice $m(x) = 0$ as mean function which reduces the problem of constructing a GP solely to the choice of an appropriate kernel function (see section 3.2 for more details). Unless mentioned otherwise all further calculations in this thesis will assume a zero mean function.

A graphical example of how this conditioning works can be seen in Fig. 3.2 where on the left are sample functions drawn from a prior GP with $m(x) = 0$ and $k(x,x') = \exp\left(-\frac{(x-x')^2}{2}\right)$ and on the right sample functions from a GP that is conditioned on a set of training points (observations). The value of their prior and posterior mean functions as well as their standard deviations are also shown. The standard deviations are the

[6]This is equivalent to taking the set of the (infinitely) many possible functions $f(x)$ and rejecting all functions which do not pass through the points $\{x_i, y_i | i = 1, \ldots, n\}$

square-root of the diagonal entries of the covariance matrix:

$$\sigma(x_i) = \sqrt{\Sigma_{f(x_i),ii}} \tag{3.10}$$

Predictions with noisy observations
In practice the training data often has some associated statistical noise $y = f(x) + \varepsilon$. Where ε has an associated variance σ_n^2 which in most cases can be assumed to be i.i.d. gaussian. It is very simple to include this into the GP framework by simply adding this noise term to the kernel function:

$$\tilde{k}(x, x') = k(x, x') + \sigma_n^2 \delta_{x,x'} \tag{3.11}$$

where $\delta_{x,x'}$ is the Kronecker delta. This also changes the equations for conditioned GPs as now the common distribution between $\boldsymbol{y}$ and $\boldsymbol{f}_*$ is given by

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{m}(\boldsymbol{x}) \\ \boldsymbol{m}(\boldsymbol{x}_*) \end{bmatrix}, \begin{bmatrix} K(\boldsymbol{x}, \boldsymbol{x}) + \sigma_n^2 I & K(\boldsymbol{x}, \boldsymbol{x}_*) \\ K(\boldsymbol{x}_*, \boldsymbol{x}) & K(\boldsymbol{x}_*, \boldsymbol{x}_*) \end{bmatrix} \right) . \tag{3.12}$$

This means that the conditional distribution changes to

$$\boldsymbol{f}_* | \boldsymbol{x}_*, \boldsymbol{x}, \boldsymbol{y} \sim \mathrm{N}(\overline{\boldsymbol{f}}_*, \Sigma_{\boldsymbol{f}_*}) \tag{3.13}$$

with

$$\overline{\boldsymbol{f}}_* = \boldsymbol{m}(\boldsymbol{x}_*) + K(\boldsymbol{x}_*, \boldsymbol{x})[K(\boldsymbol{x}, \boldsymbol{x}) + \sigma_n^2 I]^{-1}(\boldsymbol{y} - \boldsymbol{m}(\boldsymbol{x})) \tag{3.14}$$

$$\mathrm{cov}(\boldsymbol{f}_*) = \Sigma_{\boldsymbol{f}_*} = K(\boldsymbol{x}_*, \boldsymbol{x}_*) - K(\boldsymbol{x}_*, \boldsymbol{x})[K(\boldsymbol{x}, \boldsymbol{x}) + \sigma_n^2 I]^{-1} K(\boldsymbol{x}, \boldsymbol{x}_*) . \tag{3.15}$$

3.2 The kernel function

As discussed in the last section, with the mean function usually assumed to be zero, the GP is fully characterized by its kernel or covariance function. This raises the need to closely examine the properties of a kernel function and inspect which kernels are suitable to accurately predict a functional dependence from measurements.

As kernel functions cannot be any arbitrary function, we have to first establish the requirements which have to be fulfilled by a kernel to be valid. As the kernel is a function of two (potentially vector-like) variables $k(x, x')$, that describes the covariance between two points, it trivially needs to fulfil three conditions:

1. The kernel needs to map $k : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$
2. It needs to be symmetric: $k(x, x') = k(x', x)$
3. The covariance matrix obtained from the kernel needs to be positive definite: $\boldsymbol{z}^T K(\boldsymbol{x}, \boldsymbol{x}')\boldsymbol{z} \geq 0$ for all $z \in \mathbb{R}^D \backslash 0$ and with $K(\boldsymbol{x}, \boldsymbol{x}')_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}'_j)$ being the Gram matrix of $\boldsymbol{x}$ and $\boldsymbol{x}'$. This condition is fulfilled if and only if $k(x, x') \geq 0$ for all $x, x' \in X$ [22].

Apart from these mathematical requirements a kernel function should also represent all knowledge that is available on the underlying functional dependence of the training set.

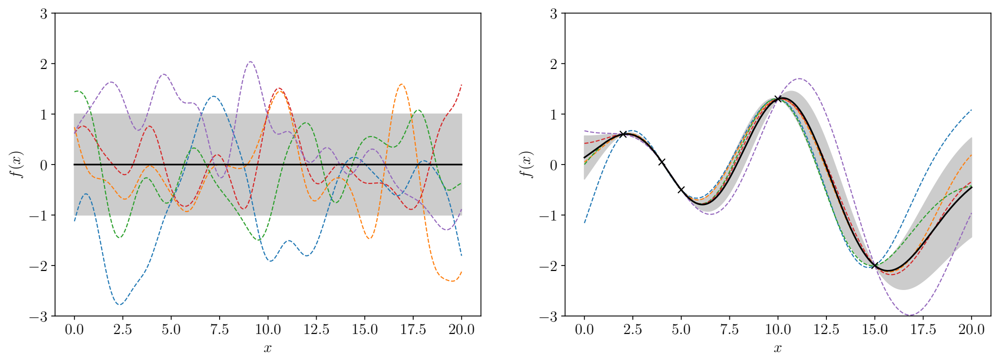

Figure 3.2: Left: Sample functions drawn from a GP with $\mu(x) = 0$ and $k(x,x') = \exp\left(-\frac{(x-x')^2}{2}\right)$ (dashed lines) as well as the value of the prior mean function and the standard deviation $\sqrt{k(x,x)}$ (solid black line and grey band). Note how in principle any function $f(x)$ is allowed but the probability that any function is drawn is dictated by the mean function and kernel. Right: Sample functions drawn from the same GP after conditioning (dashed lines) on five observations (black crosses). Again mean function and standard deviation are shown, this time for the posterior GP. Note how even with a zero prior mean function $m(x) = 0$ one obtains a non-zero posterior mean. Furthermore after conditioning, only those functions which pass through the training points are allowed.

A kernel function can encode information such as differentiability, periodicity and variability in the data. As GPs are non-parametric models unfortunately it is quite hard to infer a suitable kernel function from prior knowledge (unlike for parametric models where the functional dependence is chosen beforehand). This makes the choice of kernels for specific problems complicated.

Nevertheless some general statements can be made about some kernels and how their characteristics translate to the GP. The most important ones are listed below:

1. The first important characteristic is *stationarity*. This simply means, that a kernel function is invariant to translations, i.e.

$$k(x+z, x'+z) = k(x,x) \quad \forall\ z\ .$$

   Any stationary kernel can hence be rewritten as a function of the distance $d = |x - x'|$ The stationarity of a kernel will directly translate to a GP. In most cases stationarity is desirable.

2. The second important characteristic of kernels is *differentiability*. Again this translates directly to the GP. If a kernel function is n times differentiable so is the resulting GP.

3. Lastly a kernel can be periodic, which is defined as

$$k(x,x') = k(x, x' + n \cdot z), \quad n \in \mathbb{Z}\ .$$

   Again periodicity of the kernel also imposes periodicity on the GP.

In the following the most commonly used kernel functions are presented. This list is by no means complete. In principle there are infinitely many viable kernel functions.

The Constant kernel

This kernel is perhaps the easiest kernel imaginable as it is just defined as

$$k(x, x') = C, \qquad C \in \mathbb{R}^+ \tag{3.16}$$

While this kernel is mathematically very easy, in practice it is rarely useful on its own because it results in an infinite correlation length if $C \neq 0$. This means that all points have the same value.

Usually the constant kernel is only used as a building block for more complex composite kernels (see section 3.2.1). An example of a GP with a constant kernel before and after conditioning is shown in Fig. 3.3.

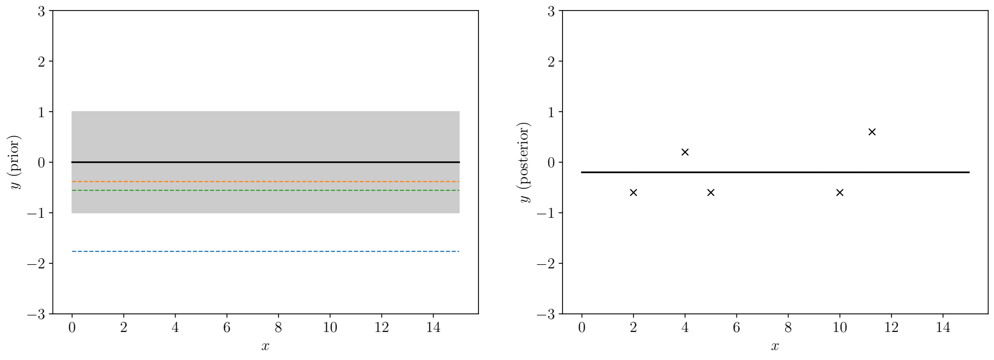

Figure 3.3: Left: Sample functions drawn from a GP with a constant kernel ($C = 1$) (dashed lines) as well as the value of the prior mean function and the standard deviation (solid black line and grey band). Right: Same GP after conditioning on five observations (black crosses). The standard deviation for the posterior GP is not visible.

The White (noise) kernel

The white kernel is almost as simple as the constant kernel. It imposes a variance on points but no correlation between them:

$$k(x, x') = \sigma_n^2 \delta_{x,x'} \tag{3.17}$$

where $\delta_{x,x'}$ is the Kronecker delta, which is one if $x = x'$ and zero otherwise. As this kernel does not establish any relation between any pair of x-values, a GP with only a white kernel is equivalent to a random noise generator. Mathematically speaking this is due to the fact that the Gram matrix of $k(x, x')$ only has non-zero values on its diagonal and thus all covariances between points vanish.

Again this kernel is almost exclusively used to build composite kernels. In fact including i.i.d. statistical noise into a GP is equivalent to adding a white kernel to an existing kernel as shown in Eq. 3.11. Fig. 3.4 shows an example of a GP with a white kernel before and after conditioning on five points. In this example σ_n^2 is chosen to be one so that the GP corresponds to white noise with a standard deviation of one.

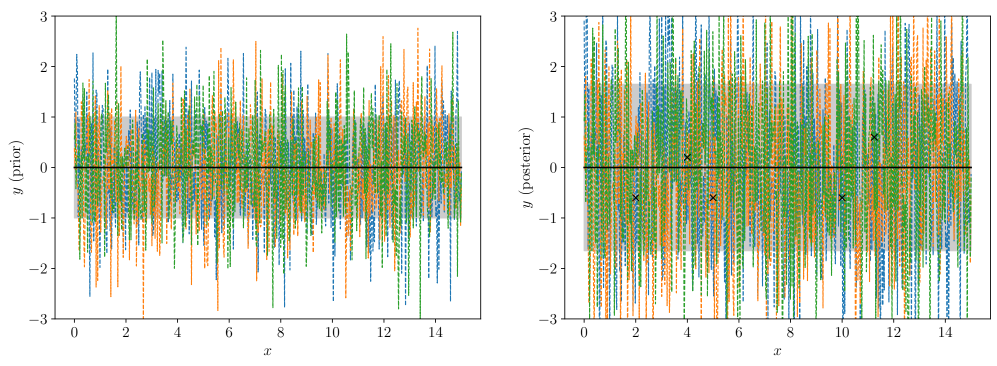

Figure 3.4: Left: Sample functions drawn from a GP with a white kernel ($\sigma_n^2 = 1$) as well as the value of the prior mean function and the standard deviation (solid black line and grey band). Right: Same GP after conditioning on five observations (black crosses). This GP corresponds to white noise with a standard deviation of $\sigma_n = 1$

The RBF kernel

Perhaps most commonly used kernel function is the *Radial Basis Bunction* (RBF) kernel[7]. This kernel is usually defined as

$$k(d) = \exp\left(-\frac{d^2}{2 \cdot l^2}\right), \quad l \in \mathbb{R} \; . \tag{3.18}$$

It is infinitely many times differentiable and stationary. A drawback of this kernel is that is produces very "smooth" GPs which in case of real world data rarely reflects the truth. Therefore it has been argued by many authors that the RBF kernel should not be used in applications where real world data is involved [19].
Fig. 3.5 shows a GP with an RBF kernel with $l = 1$. Note how the sample functions drawn from the GP are very smooth.

The Matérn kernel The Matérn kernel can be seen as a generalization of the RBF kernel with the introduction of an additional parameter ν which controls the differentiability. Like the RBF kernel, it is stationary. It is defined as [9]

$$k_\nu(d) = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}d}{l}\right)^\nu \cdot K_\nu\left(\frac{\sqrt{2\nu}d}{l}\right), \qquad l \in \mathbb{R}^+ \; . \tag{3.19}$$

where Γ is the gamma function and K_ν the modified Bessel function of the second kind A value of $\nu = n + \frac{1}{2}$ means that the kernel is n times differentiable. For $n = 0$ it is

[7]In the literature this kernel has many different names among which are *Squared Exponential* kernel and *Gaussian* kernel. Throughout this thesis we will stick to RBF kernel though.

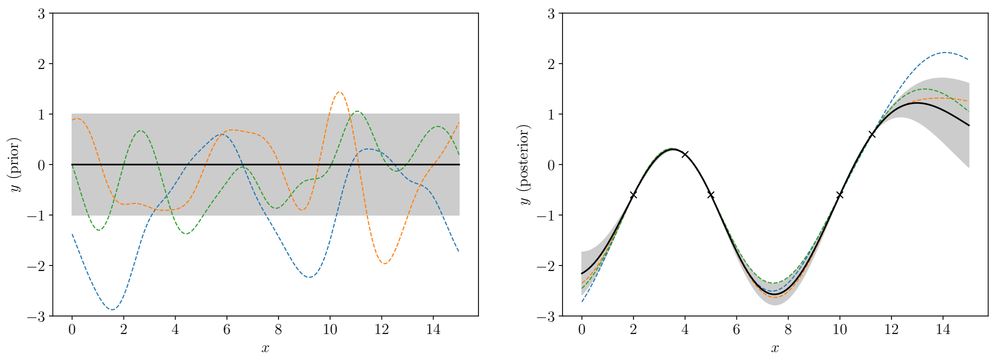

Figure 3.5: Left: Sample functions drawn from a GP with an RBF kernel ($l = 1$) as well as the value of the prior mean function and the standard deviation (solid black line and grey band). Right: Same GP after conditioning on five observations (black crosses).

simply given as

$$k_{1/2}(d) = \exp\left(-\frac{d}{l}\right)$$

and is often referred to as Ornstein-Uhlenbeck kernel, as when it is used to define a GP it describes the (one-dimensional) velocity of a particle undergoing brownian motion [9]. Furthermore for $\nu \to \infty$ the Matérn kernel approaches the RBF kernel. Fig. 3.6 shows a GP with a Matérn kernel with $\nu = \frac{3}{2}, l = 1$. The sample functions drawn from the GP are therefore once differentiable which is clearly visible when compared to samples drawn from the RBF kernel.

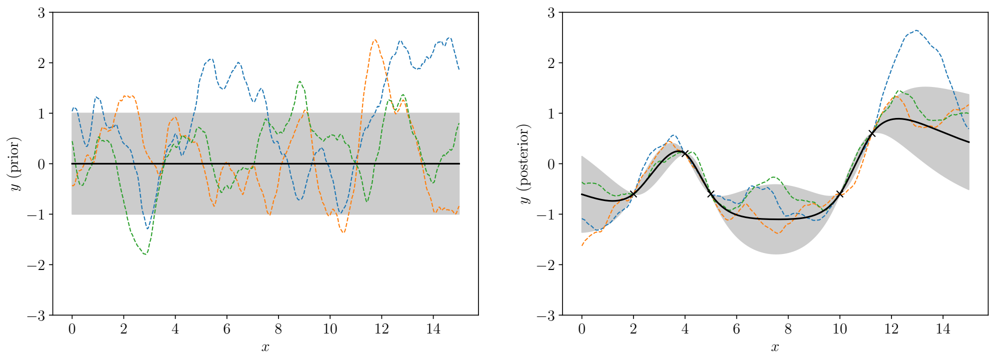

Figure 3.6: Left: Sample functions drawn from a GP with a Matérn kernel ($\nu = \frac{3}{2}, l = 1$) as well as the value of the prior mean function and the standard deviation (solid black line and grey band). Right: Same GP after conditioning on five observations (black crosses). Note how the sample functions drawn from the GP are much less smooth than those drawn from the GP with RBF kernel.

The exponential-sine squared kernel The exponential sine squared (ESS) kernel is a prime example of a periodic kernel.[8] It is stationary and defined as [23]

$$k(d) = \exp\left(\frac{2\sin^2\left(\frac{\pi d}{p}\right)}{l^2}\right) \tag{3.20}$$

where p controls the distance between different periods of the function and l plays the same role as in the RBF kernel. Fig 3.7 shows an example of a GP with ESS kernel. The periodicity is clearly visible.

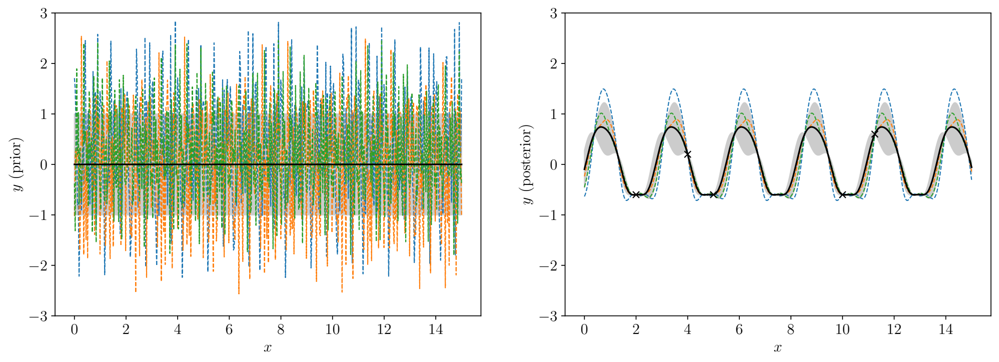

Figure 3.7: Left: Sample functions drawn from a GP with a ESS kernel ($l = 10^{-5}, p = 1$) as well as the value of the prior mean function and the standard deviation (solid black line and grey band). Right: Same GP after conditioning on five observations (black crosses). The interpretation of the data differs a lot from the one by the RBF and Matérn kernels due to the imposed periodicity.

3.2.1 Composite kernels

From the three conditions for kernel functions specified in section 3.2 one can combine known kernel functions into new ones through simple operations. Most importantly two operations always conserve these conditions and can hence be used to construct composite kernels:

1. The sum of two or more kernels is a valid kernel.
2. The product of two or more kernels is also a valid kernel.

In practice these rules can be used to construct composite kernels to better incorporate prior knowledge about the functional dependence of the data one tries to represent. Furthermore it is possible to assign different covariance functions to different domains as explained in [25].

[8] In the literature, this kernel function is often simply referred to as periodic kernel [23, 19, 24]. Since this is somewhat misleading, as this is just one example of possible periodic kernels we will stick to "ESS kernel".

3.2.2 Higher dimensionality

So far we have only looked at the case where a GP is a function $\mathcal{GP} : \mathbb{R} \to \mathbb{R}$. However we often encounter the case where the function to be approximated by the GP is $\mathbb{R}^D \to \mathbb{R}$. In this case the formalism can be trivially modified by changing the kernel function such that $k(\boldsymbol{x}, \boldsymbol{x}') : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$. An example of a GP mapping $\mathbb{R}^2 \to \mathbb{R}$ can be seen in Fig. 3.8 (top) where the the kernel function is the RBF kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{|\boldsymbol{x} - \boldsymbol{x}'|^2}{2}\right)$$

Additionally a way to construct a kernel is as a composition of kernels where each kernel only acts on a single direction. In this case a valid two-dimensional example using the RBF kernel could be

$$k\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}\right) = \exp\left(-\frac{|x_1 - x_1'|^2}{2l_1}\right) \cdot \exp\left(-\frac{|x_2 - x_2'|^2}{2l_2}\right)$$

This GP with $l_1 = 10^{-3}, l_2 = 1$ is shown in Fig. 3.8 (bottom). The different length scales cause the GP to behave differently along the two dimensions. In principle it would also be possible to use different types of kernels e.g. an RBF kernel along one axis and a periodic kernel along the other if this reflects the prior knowledge about the hypothesis.

From a computational standpoint it is important to point out that for anisotropic kernels the number of hyperparameters increases proportionally to the number of dimensions of the data. This in turn means that obtaining the MAP estimate for these hyperparameters as explained in section 3.2.3 becomes computationally more expensive.

3.2.3 Tuning the kernel's hyperparameters

All the kernels that have been presented in the previous section have one or more free parameters, which are in principle dependent on the prior knowledge of the data. In the context of GPs these are usually referred to as *hyperparameters*[9] θ. In many cases however the knowledge about e.g. the characteristic length scale l or the variance σ_n^2 is missing or not immediately obvious.

Fortunately the GP itself can be used to obtain a best estimate of these parameters using Bayesian inference. This is done by maximizing the marginal likelihood (or evidence) of the training data under the GP which is given as the integral over the prior times the likelihood [19]:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \int p(\boldsymbol{y}|\boldsymbol{f}, \boldsymbol{x}) p(\boldsymbol{f}|\boldsymbol{x}) \, \mathrm{d}f \tag{3.21}$$

where $p(\boldsymbol{f}|\boldsymbol{x})$ is the prior which in our case is gaussian $\boldsymbol{f}|\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{0}, K)$. Furthermore the

[9]There exists an ambiguity here since we also introduced hyperparameters and the letter θ in chapter 2.1.

This is no coincidence since the Gaussian Process prior and the training set also induce a likelihood. To avoid confusion the hyperparameters of the GP will be explicitly called *GP hyperparameters* if not clear from the contest.

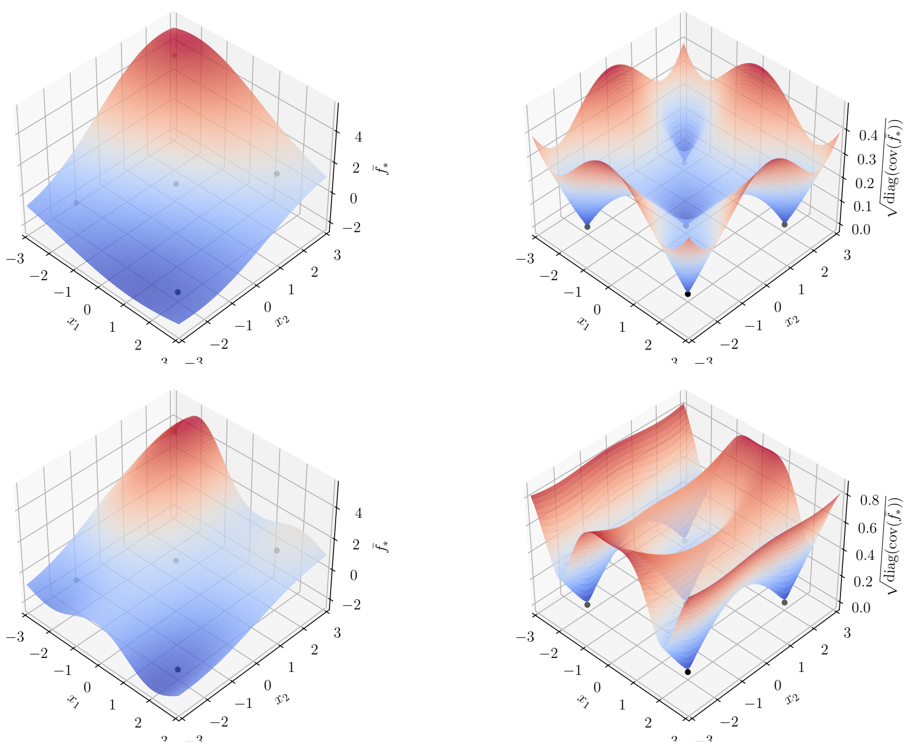

Figure 3.8: Top: Mean (left) and standard deviation (right) for Posterior GP with an isotropic RBF kernel with $l = 1$ conditioned on five training points (black dots).
Bottom: GP conditioned on the same training points with an anisotropic kernel with $l_1 = 10^{-3}, l_2 = 1$. The interpretation of the data changes considerably from the isotropic case to the anisotropic one.

likelihood is just the function (or a factorized gaussian if the training data has associated noise) with $\boldsymbol{y}|\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{f}, \sigma_n^2 I)$ which means that the integral can be analytically calculated which gives:

$$\log p(\boldsymbol{y}|X) = -\frac{1}{2}\boldsymbol{y}^T(K + \sigma_n^2 I)^{-1}\boldsymbol{y} - \frac{1}{2}\log|K + \sigma_n^2 I| - \frac{n}{2}\log 2\pi \tag{3.22}$$

For a full proof please refer to 6. This induces a posterior distribution for the hyperparameters θ which can be treated in two ways:

- In the full Bayesian treatment the posterior distribution will be fully taken into account. This is numerically problematic as the distribution of θ modifies the variance of the conditioned GP. Taking this into account is somewhat tricky and usually involves Monte Carlo methods which are numerically expensive.
- One can obtain a *maximum a posteriori* (MAP) or *Maximum Likelihood type II* (ML II) estimator of the optimal hyperparameters like in frequentist statistics. This is numerically less challenging since we are only required to maximize Eq. 3.22 with a suitable algorithm however this essentially approximates the posterior distribution to a δ-distribution which is no proper Bayesian treatment. To save on the computational complexity while still taking this into account some approximation techniques have been proposed in [1] but we will ignore this detail for the moment and discuss the implications later.

3.3 GP Regression

Having discussed the ingredients of GPs it is now time to look at the full algorithm which is used to do GP regression. GP regression (sometimes also referred to as *Kriging*) describes the construction of a GP with the help of some kernel $k(\boldsymbol{x}, \boldsymbol{x}'|\theta)$ with some unknown hyperparameters θ which are determined by Bayesian inference using the MAP estimate of θ. The advantage of this method lies in the fact that the only choice which is left to the user is that of a suitable prior kernel which usually only incorporates some broad knowledge about the characteristics of the function which describes the data (differentiablity, periodicity, etc.). Furthermore the MAP estimators of the hyperparameters have an easy interpretation as shown in section 3.2.

The algorithm to approximate some unknown function $f(x)$ given some data pairs $\boldsymbol{x}, \boldsymbol{y}$ (training inputs) with associated noise $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{\sigma}_n$ can be divided into two steps:

1. In the *training* step the hyperparameters of the model are optimized by maximizing Eq. 3.22 and $(K + \sigma_n^2 I)$ is precomputed
2. In the *prediction* step the GP is conditioned according to Eq. 3.14.

Of course this can in principle be done in one step but it is specifically divided into two steps here to stress the fact that there is a distinct part which trains the GPs hyperparameters and another part which conditions this trained GP on the observations. In practice the prediction (conditioning) step is often called multiple times while the training only has to be performed once for a given training set $\boldsymbol{x}, \boldsymbol{y}$.

Pseudocode of the training step is shown in Algorithm 2.

```
Input: $\boldsymbol{x}, \boldsymbol{y}$ (training inputs & targets), $k(\boldsymbol{x}, \boldsymbol{x}'|\theta)$ (covariance function), $\sigma_n$ (noise level), $\theta_0$ (initial hyperparameter(s))
[1] $\theta := \theta_0$
[2] repeat
[3]     $K := K(\boldsymbol{x}, \boldsymbol{x}|\theta)$                                   eq. (3.6)
[4]     $L := \text{cholesky}(K + \sigma_n^2 I)$                          eq. (.1)
[5]     $\alpha := L^T \backslash (L \backslash \boldsymbol{y})$                                 eq. (.3)
[6]     $\log(p(\boldsymbol{y}|\boldsymbol{x}, \theta)) := -\frac{1}{2}\boldsymbol{y}^T\alpha - \sum_i \log(L_{ii}) - \frac{n}{2}\log 2\pi$   eq. (3.22)
[7]     vary $\theta$ according to some global optimizer
[8] until $\max[\log(p(\boldsymbol{y}|\boldsymbol{x}))]$ reached
[9] return $\alpha$, $L$, $\theta_{\text{opt}}$
```

Algorithm 2: Training step of the GP regression algorithm. The Inversion of the Gram matrix of $\boldsymbol{x}, \boldsymbol{x}'$ is not done directly but through Cholesky decomposition since it is numerically more stable. The computational complexity is dominated by the Cholesky decomposition in line 4 ($\mathcal{O}(n^3)$). The optimum θ_{opt} is $\theta_{\text{opt}} = \text{argmax}[\log(p(\boldsymbol{y}|\theta))]$. The algorithm has been taken from [19].

The computation of $(K + \sigma_n^2 I)^{-1}$ is usually not done directly but rather by first performing a Cholesky decomposition and then calculating $(K + \sigma_n^2 I)^{-1}\boldsymbol{y}$ directly (see 6). This is done since it is numerically more stable than direct inversion and because it allows for easy calculation of $|K + \sigma_n^2 I|$. In practice some small σ_n is added for numerical stability of the Cholesky decomposition even if the training data has no associated noise [19].

The prediction step for some test inputs $\boldsymbol{x}_*$ evaluates the conditioned mean and covariance at $\boldsymbol{x}_*$. In practice this step is often called for multiple different test inputs $\boldsymbol{x}_*$. Pseudocode of this step is shown in algorithm 3.

```
Input: $L, \alpha, \theta_{\text{opt}}, \boldsymbol{x}, \boldsymbol{y}, k(\boldsymbol{x}, \boldsymbol{x}'|\theta)$ (from training step), $\boldsymbol{x}_*$ (test input)
[1] $K_* := K(\boldsymbol{x}, \boldsymbol{x}_*|\theta_{\text{opt}})$
[2] $K_{**} := K(\boldsymbol{x}_*, \boldsymbol{x}_*|\theta_{\text{opt}})$
[3] $\boldsymbol{v} := L \backslash K_*$                                                  eq. (.4)
[4] $\overline{\boldsymbol{f}}_* := K_*^T \alpha$                               Predictive mean eq. (3.14)
[5] $\text{cov}(\boldsymbol{f}_*) = K_{**} - \boldsymbol{v}^T\boldsymbol{v}$                Predictive covariance eq. (3.14)
[6] return $\overline{\boldsymbol{f}}_*, \text{cov}(\boldsymbol{f}_*)$
```

Algorithm 3: Prediction step of the GP regression algorithm. Returns the predictive mean and covariance at arbitrary locations $\boldsymbol{x}_*$. Since the computational complexity of this algorithm is $\mathcal{O}(n^2 m)$ for n training and m test points it is usually much faster than algorithm 2. The algorithm has been taken from [19].

3.4 Bayesian quadrature

Let

$$I = \int_{\boldsymbol{x}_0}^{\boldsymbol{x}_1} f(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} \tag{3.23}$$

be the integral of some arbitrary function $f(\boldsymbol{x})$ over possibly multiple dimensions such that $\boldsymbol{x} \in \mathbb{R}^d$. Bayesian quadrature (BQ) takes a statistical approach to calculating this integral by using a GP as an interpolator for $f(x)$ [1]. This has two advantages compared to other commonly used numerical integration algorithms such as Monte Carlo or the trapezoid rule (see 2.1):

1. By staying in the Bayesian framework we can treat I like a random variable for which we can then calculate a prior and Likelihood. In practice however it is easier to treat f itself as a random function for which we can set a GP as prior:

$$p(f((\boldsymbol{x})) = \mathcal{GP}(f, m, k) \tag{3.24}$$

   By being able to place a prior on m and k this allows us to inject information we have about f like differentiability, periodicity etc. (see section 3.2) into our approximation. This information is available in many cases where numerical integration is needed and can often be derived from the nature of the problem.

2. The reason that a GP is such a good choice for placing a prior onto the function is that it is closed under any affine transformation L:

$$p(L(f)) = \mathcal{GP}(L(f), L(\mu), L^2(k)) \tag{3.25}$$

   which is handy because integration is a linear operation which means that

$$p\left(\int_{\boldsymbol{x}_0}^{\boldsymbol{x}_1} f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}\right) = \mathcal{N}\left(Z; \int_{\boldsymbol{x}_0}^{\boldsymbol{x}_1} \mu(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}, \int_{\boldsymbol{x}_0}^{\boldsymbol{x}_1}\int_{\boldsymbol{x}_0}^{\boldsymbol{x}_1} k(\boldsymbol{x}, \boldsymbol{x}')\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{x}'\right) . \tag{3.26}$$

   We can then proceed to condition our GP according to Eq. 3.7 on some training data D which is drawn from f which gives us a posterior for I with the expectation value

$$\mathbb{E}[I|D] = \int_{\boldsymbol{x}_0}^{\boldsymbol{x}_1} \mu_{f|D}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \tag{3.27}$$

   which means that the mean of our estimation of the integral is simply the integral of the mean of the GP. Furthermore we get a variance that is given by

$$\mathrm{var}[I|D] = \int_{\boldsymbol{x}_0}^{\boldsymbol{x}_1}\int_{\boldsymbol{x}_0}^{\boldsymbol{x}_1} k_{f|D}(\boldsymbol{x}, \boldsymbol{x}')\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{x}' . \tag{3.28}$$

   This variance (or its square root) naturally lends itself to be used as a natural precision criterion for I.

If one takes a close look at Eqs. 3.27 and 3.28 one might observe that they still contain integrals, which means that we have replaced one integration by another. This does not seem to be particularly useful at first. However if the integrations in Eq. 3.26 can be performed quicker than the integration of I directly or, more importantly, if it is outright impossible to compute I analytically, we do gain performance. An illustration of BQ with the function $f(x) = \sin(2 \cdot x) \cdot \frac{x}{2}$ is shown in Fig 3.9.

Another important observation which can be extracted from Eq. 3.7 is that the conditioned kernel $k_{f|D}$ and subsequently $\mathrm{var}[I|D]$ do not not directly depend on the training data. Despite this there is an implicit dependence on y_{train} if the MAP estimate is used

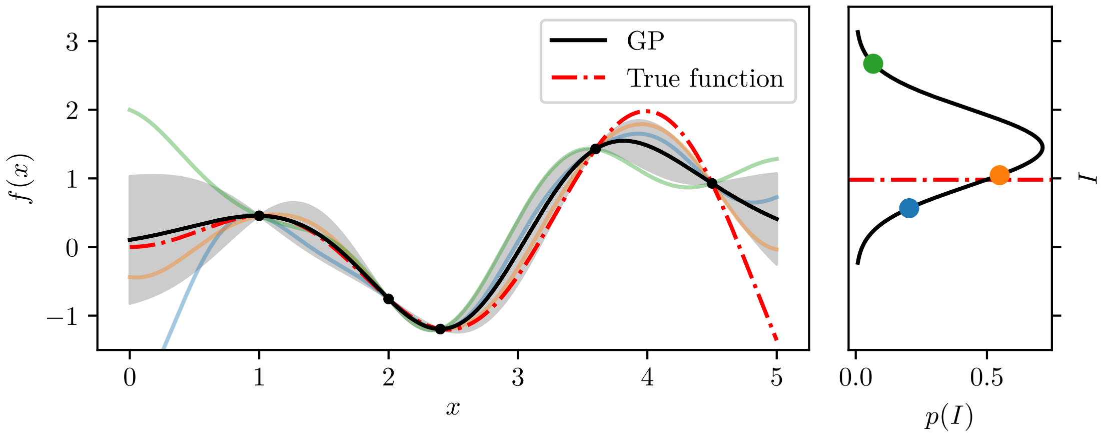

Figure 3.9: Illustration of the BQ procedure. Left: GP fit to the function $f(x) = \sin(2 \cdot x) \cdot \frac{x}{2}$ with three sample functions (green, blue, orange) with $x \in [0, 5]$. Right: Normal distribution induced by BQ for the integral $I = \int_0^5 f(x)\,\mathrm{d}x$. The three coloured dots correspond to the integrals of the sample functions. The true value for I is shown in red.

to assign values to the hyperparameters of the kernel as described in 3.3. This property of GPs can be used to derive optimal quadrature rules for different classes of functions purely through conditioning the kernel function [26]. This procedure runs under the name of *Bayes-Hermite quadrature* [8]

In this thesis we will take a different approach which aims at finding the next best point to evaluate sequentially by optimization. This procedure is known as *active sampling* and will be explained in the following.

3.4.1 Active sampling

Optimization of non-analytic functions has long been a main concern in mathematics and science and has seen extensive research throughout the last decades. This is partly fueled by fast advancements in computational possibilities which enable evermore complex numerical simulations but also often include the numerical optimization of complicated and often high-dimensional functions. It is therefore no surprise that the idea of active sampling has its origins in optimization.

In search of efficient optimization algorithms the science community has often looked towards statistics and probability theory in the same way that probabilistic Monte Carlo methods provide efficient estimations for quadrature of high dimensional integrals (see section 2.2).

A different approach within the Bayesian framework relies on interpolating a function with a GP and using the mean and standard deviation of the interpolator to obtain a proposal for the next location to sample. The advantage of this is that with a good choice of this rule sampling will always be performed at the locations which add the most information to the GP model.

Furthermore initial knowledge about the shape of the function can be incorporated into the mean function and kernel of the GP. This procedure is often referred to as *active sampling* or *Bayesian optimization* (BO) although the latter is only used when the objective is to find the global optimum of a function. Since the concept of active sampling is best explained in the optimization case we will use the maximization of an exemplary function f to introduce the concept:

In active sampling the approach to finding a new point to sample is inferring it by optimizing an *acquisition function* (AF) which tries to maximize the information gained in each step. This AF usually only depends on the conditional mean and variance (or standard deviation) of the GP (see eq. 3.14).

Finding an appropriate AF depends very much on the task at hand which in BO is global optimization where we need to find a trade-off between exploration of the sampling space and exploitation of known maxima.

One of the most commonly used AF for this is the *expected improvement* (EI) AF [27]. This is the expectation value of the improvement over the current best sample $f^+(\boldsymbol{x}) = \max(\boldsymbol{y})$ of the training set $\boldsymbol{x}, \boldsymbol{y}$ of our GP:

$$a_{\mathrm{EI}}(\boldsymbol{x}) = \mathbb{E}(\max(f(\boldsymbol{x}) - f^+(\boldsymbol{x})), 0)$$

By using the GP as estimator for $f(\boldsymbol{x})$ this yields

$$a_{\mathrm{EI}}(\boldsymbol{x}) = \begin{cases} (\mu(\boldsymbol{x}) - f^+(\boldsymbol{x}) - \xi)\Phi\left(\frac{\boldsymbol{x}) - f^+(\boldsymbol{x}) - \xi}{\sigma(\boldsymbol{x})}\right) + \sigma(\boldsymbol{x})\phi\left(\frac{\boldsymbol{x}) - f^+(\boldsymbol{x}) - \xi}{\sigma(\boldsymbol{x})}\right) & \text{if } \sigma(\boldsymbol{x}) > 0 \\ 0 & \text{else} \end{cases} \tag{3.29}$$

where Φ and ϕ are the PDF and CDF of the normal distribution, respectively. While this acquisition function is optimal for BO there are better choices if the goal is not optimization. An example of BO with the EI AF can be seen in Fig 3.10. In this example four points are acquired sequentially. One can clearly see that the algorithm converges to the maximum of the function in a few steps and does not get stuck in local maxima. Pseudocode of the active sampling algorithm with N steps is shown in algorithm 4

While the concept of active sampling is best explained in the optimization case this is just as straightforward to apply to BQ. In this case there are usually more efficient AFs than EI. Since the choice of acquisition function depends on the nature of the integrand and the goal of this thesis is to integrate a special class of integrands (probability distributions) the discussion of suitable acquisition functions for BQ will be done in section 4.3.

Unfortunately the computational complexity of sequential active sampling scales proportional to n^4 where n is the number of samples in the GP regressor. This is due to the fact that refitting the hyperparameters of the GP scales proportional to n^3 and thus the sequential algorithm with n^4.

This effectively limits the number of samples which can be acquired to $\mathcal{O}(10^3 - 10^4)$ for most modern computers. This has led to many different approaches aimed at reducing this computational complexity which can be summarized under the term *approximate*

```
Input: $\mathcal{GP}(0, k(\boldsymbol{x}, \boldsymbol{x}'|\theta))$ (GP regressor with kernel), $\boldsymbol{x}_0, \boldsymbol{y}_0$ (initial
       training set), $a(\mu(x), \sigma(x))$ (acquisition function), $f(x)$
       (function to be sampled)
[1]  $x_{\text{train}} := x_0$
[2]  $y_{\text{train}} := y_0$
[3]  for $N$ times do
[4]      Fit $\theta_{\text{GP}}$                                        alg. 2
[5]      Draw $x$
[6]      repeat
[7]          $\mu(x), \sigma(x)$ :=GP prediction                        alg. 3
[8]          $a_x := a(\mu(x), \sigma(x))$
[9]          vary $x$ according to some global optimizer
[10]     until $\max[a(\mu(x), \sigma(x))]$ reached
[11]     $x_{\max} := \text{argmax}(a(x))$
[12]     $y_{\max} := f(x_{\max})$
[13]     $x_{\text{train}} = \{x_{\text{train}}, x_{\max}\}$
[14]     $y_{\text{train}} = \{y_{\text{train}}, y_{\max}\}$
[15] end
[16] return GP regressor
```

Algorithm 4: Illustration of the active sampling algorithm for N acquisition steps. The algorithm first optimizes the hyperparameters of the GP surrogate model according to algorithm 2 and then optimizes the acquisition function using algorithm 3 to evaluate μ and σ of the position x. When the optimization is done the x-coordinate of the maximum of the acquisition function is recorded and the true function is interrogated at this point. The computational complexity is roughly $\mathcal{O}(N^4)$ if one assumes that the time it takes to optimize the acquisition function is much smaller than the time it takes to optimize the hyperparameters of the GP. For large N this is usually a good assumption since the fitting of the hyperparameters scales with $O(N^3)$ while predictions with the GP scale with $O(N^2)$.

GPs[10] [28, 29, 30, 31].

Approximate GPs have gained a lot of attention in recent years as combining the predictive power of GPs with a low computational cost is a very tempting idea even if it sacrifices the analytical tractability of the equations. A full review of the literature would be beyond the scope of this thesis but it is useful to keep in mind that some of these methods could in principle be applied to the algorithm presented in chapter 4.

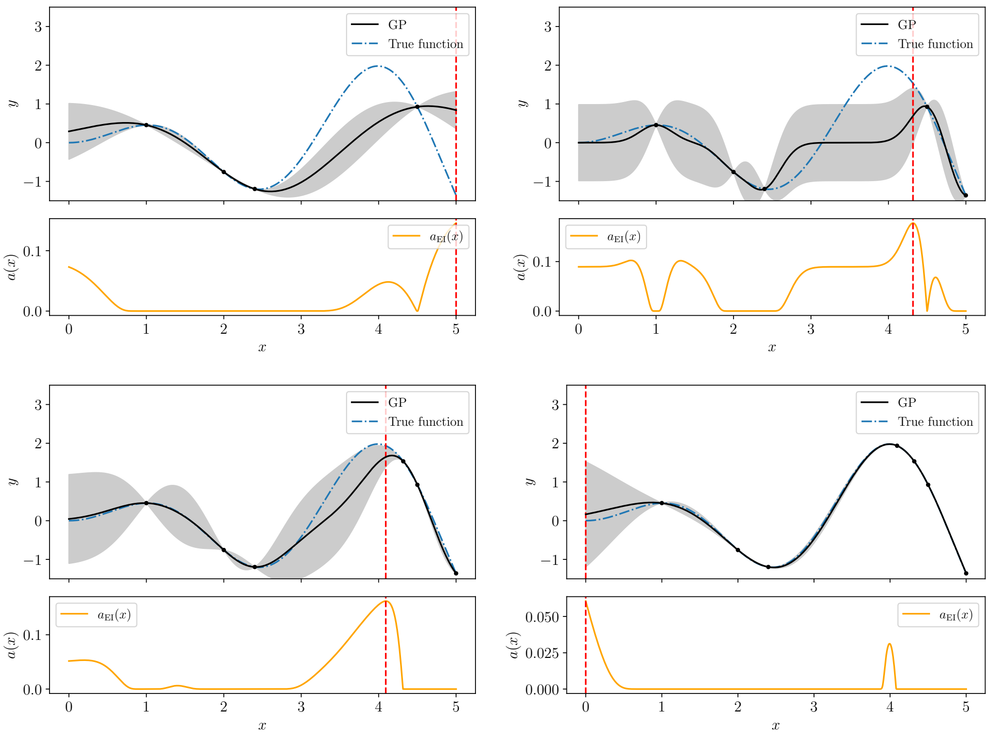

Figure 3.10: Four iterations of BO of the function $f(x) = \sin(2 \cdot x) \cdot \frac{x}{2}$ with EI in the interval $[0, 5]$. The real function (blue) and the GP approximation (black) with its standard deviation (grey shaded area) are shown on the top and the values of the acquisition function (orange) on the bottom of each plot. The red dashed line indicates the location of the maximum of $a_{\mathrm{EI}}(x)$ where the function will be sampled next. Note how the algorithm finds a trade off between exploration of the full function space and exploitation of regions with high values.

[10]These approaches can broadly be divided into two categories: *Sparse* GPs aim at introducing some sparsity in the covariance matrix [28, 29] and *variational* GPs where the goal is approximation of the marginal likelihood for optimizing the hyperparameters of the model [30, 31].

4 Bayesian quadrature for probability distributions

As our goal is to do Bayesian inference with GPs we want to effectively perform Bayesian quadrature with the constraint that the function to integrate is an (unnormalized) probability distribution. The advantages of BQ for this task compared to other algorithms is motivated in section 4.1 followed by a discussion of the use of performing a power reduction operation of the posterior in section 4.2. Afterwards a suitable acquisition function for this task is derived in section 4.3. Preprocessing of the training data is explained in section 4.4. Afterwards a novel way to parallelize the algorithm is introduced in section 4.5.

A convergence criterion is then introduced in section 4.6 followed by a discussion of the drawbacks of this approach which are explained in section 4.7 and 4.8.

Lastly we perform some tests on artificial and real examples in section 4.9 where we compare the performance to other state of the art methods.

4.1 Why Bayesian quadrature?

Solving integrals of the form

$$Z = \int L(\boldsymbol{\theta})\pi(\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{\theta} \tag{4.1}$$

where $L(\boldsymbol{\theta})$ is a likelihood function and $\pi(\boldsymbol{\theta})$ a prior distribution is by no means a novel problem. In fact it might be one of the most extensively studied integration problems as integrals of this type appear in numerous scientific fields[11]. Crucially this problem also appears in many machine learning algorithms, which has led the machine learning community to conduct extensive studies into possibilities for solving this type of integral using Bayesian quadrature [3, 32, 1].

The reason for choosing this approach instead of the popular alternatives, the most important of which is MCMC, is that it solves some of MCMCs inherent problems[12]. These fundamental flaws have been neatly summarized in a very influential paper in 1992 [33] where the author points out two main objections to using MCMC.

First, the estimator for Z does not only depend on the values of the posterior $L(\boldsymbol{\theta})\pi(\boldsymbol{\theta})$ but also on the sampling distribution $g(\boldsymbol{\theta}'|\boldsymbol{\theta})$ (see section 2.2) which is arbitrary. This means that for different choices of sampling distributions different sets of sampling points are obtained even though the integral is the same. This dependence violates the Likelihood principle.

The second objection is that classical Monte Carlo methods such as MCMC entirely

[11]In this case integration does not mean performing the full integral of $L(\boldsymbol{\theta})\pi(\boldsymbol{\theta})$ along all dimensions but instead only integrating along some subset of dimensions to get confidence intervals and marginal distributions.

[12]The term MCMC is a very broadly defined term and encompasses a large number of very different algorithms which have different sampling schemes. The problems with these which are pointed out in the following do not necessarily apply to every MCMC algorithm in existence. For instance HMC does not us a proposal distribution for jumps of the chain. The criticism here is mainly directed towards the Metropolis-Hastings algorithm and variants thereof. Nevertheless these shortcomings can be applied to all MCMC algorithms even if the arguments have to be slightly adjusted.

ignore the values of the posterior during inference. Instead they are only used for determining where the chain should jump next. This throws away valuable information. To illustrate this imagine that the chain reaches some point $\boldsymbol{\theta}$ in the parameter space which is visited again after a number of iterations. Even though the value of the Likelihood at that point has been evaluated and the point contains no additional information, the algorithm is oblivious to this as it discards the values of the likelihood.

While these two objections cannot directly be translated to nested sampling, where no arbitrary additional distribution has to be introduced and the value of the the integrand enters into the estimate for the integral, there is a third important flaw which is inherent to both MCMC and nested sampling. This is the fact that both algorithms do not sample the space in a deterministic way but rather rely on Monte Carlo techniques to map the posterior. While this brings with it the benefit of fast computation, as the computational complexity of sampling new points is independent of the number of points which have previously been sampled, it does have the downside that statistical sampling by definition cannot be the most efficient way to sample a space.

All of these problems are solved by sequential Bayesian quadrature. The active sampling procedure (or alternatively Bayes-Hermite quadrature) makes the addition of new samples a purely deterministic optimization and the values of the posterior are not wasted as they are incorporated into the GP.

4.2 Power reduction operation

PDFs are always positive. This is an important consideration that we need to take into account when integrating them with Bayesian quadrature and which we can use to our advantage. This knowledge is usually accounted for by performing a power reduction operation $P\left(L(\boldsymbol{\theta})\right)$ on the likelihood and approximating this function by a GP. Several different proposals have been made for suitable P in the literature [3, 1].

In this work we will use the log transformation as power reduction operation following [1] since many Likelihoods naturally "live" in the log space[13]. Particularly in physics it is very common that likelihoods are directly generated in the log space due to the reasons given in section 2.1.2. Therefore no additional transformation is required when working with these likelihoods which makes this especially convenient.

If we want to model Z with BQ and our goal is to achieve this by placing a GP prior on the likelihood function $L(\boldsymbol{\theta})$ this means that our Bayesian estimate for $\overline{Z} \approx \mathbb{E}(Z)$ becomes

$$\overline{Z} = \int m_{\boldsymbol{L}|\boldsymbol{D}}(\boldsymbol{\theta})\pi(\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{\theta} \tag{4.2}$$

where D is some training data (samples) from L. This result can simply be obtained by modifying Eq. 3.27 to our specific case. If we now perform the log transformation on

[13]More precisely many likelihoods belong to the *exponential family* of probability distributions [34].

$L(\boldsymbol{\theta})$ and set a GP prior on this the expectation value for Z becomes

$$\overline{Z}(\log(L(\boldsymbol{\theta}_D))) = \int (\exp(\log(L(\boldsymbol{\theta}))\pi(\boldsymbol{\theta})\,\mathrm{d}\theta))\,\mathcal{N}(\log(L); \overline{\log L}_D, \mathrm{cov}_D(\log L))\,\mathrm{d}\log L\ . \tag{4.3}$$

Unfortunately this integral is intractable. A possibility for restoring tractability that has been proposed in [1] is to do a Taylor expansion of $\exp(\log(L))$ around some point L_0

$$\exp(\log(L)(\boldsymbol{\theta})) \approx \exp(\log L_0(\boldsymbol{\theta})) + \exp(\log L_0(\boldsymbol{\theta}))\exp(\log L(\boldsymbol{\theta}) - \log L_0(\boldsymbol{\theta})) \tag{4.4}$$

This approximation restores tractability, however we will use a different approach in this work which relies on integrating the log-GP numerically with MCMC. This approach is taken since we want to be able to marginalize any number of dimensions without recomputing the whole integral, for which MCMC is a good choice.

Furthermore we want to avoid having to actually do the (very tedious) analytical integration of Eq. 4.3 along any subset of dimensions of $\boldsymbol{\theta}$. Nested sampling would be a similarly good choice. In addition both MCMC and nested sampling naturally operate in the log space which again makes transformations unnecessary.

Another advantage of sampling in the log-space that has been pointed out in [1] is that the characteristic length scale l of the kernel when using a kernel of the form

$$k(d) = c \cdot \tilde{k}_l(d) \tag{4.5}$$

where $\tilde{k}_l$ is an isotropic kernel like an RBF or Matérn kernel, is larger when sampling the log-likelihood as opposed to the likelihood as illustrated in Fig 4.1. This allows the GP approximation to generalize better to distant parts of the function.

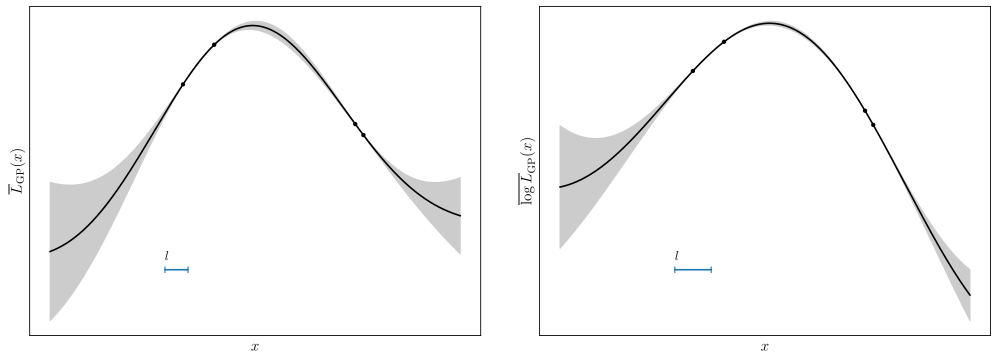

Figure 4.1: Illustration of how fitting a GP to the likelihood $L(x)$ (left) causes the correlation length l to be shorter than when fitting the GP to $\log L(x)$ (right). This Longer correlation length usually means that the GP will generalize better to function values which are far away.

Now one might ask why we have replaced MCMC by BQ just to perform the integration step with MCMC again which, at first glance, does not seem to be an advantageous thing to do. It is however if we keep computational speed in mind. For both cases the number of samples which need to be evaluated by the MCMC chain are roughly the same if the GP is a good approximation to the Likelihood function.

However the samples which actually have to be drawn from the likelihood function differ greatly as placing a prior on the functional shape of the likelihood allows us to use the advantages of active sampling to deterministically find an optimal set of inducing locations. This is a considerable advantage if $t_{\log L} \gg t_{\mathrm{GP}}$ where $t_{\log L}$ is the time it takes to calculate the likelihood function for a single sample and t_{GP} the time it takes our GP model to make a prediction.

4.3 Appropriate acquisition functions

Now that we have established that the likelihood function (or the posterior distribution) shall be sampled in the log space, the next question which arises is which acquisition function is suitable to sampling (log) probability distributions. To derive this consider the one-dimensional gaussian likelihood $L(x) = \mathcal{N}(x; 0, 1)$ with a uniform prior in $[-3, 3]$ as shown in Fig. 4.2.

Obviously the contribution of every small element $\mathrm{d}x$ to the integral is proportional to its posterior value since $L(x)\pi(x)$ is by definition positive. This means that a natural choice for an acquisition function is one which is designed such that the standard deviation of the GP is inversely proportional to the likelihood value at this point.[14] The easiest example which achieves this is

$$a_L(x) = L(x) \cdot \sqrt{\mathrm{var}_{L|D}(x)} \ . \tag{4.6}$$

Unfortunately we do not know the value of the likelihood at every point (otherwise our sampling would be pointless) but luckily our GP also gives an estimate for the value of $L(x)$ which means that the best estimate for acquisition function is

$$a_L(x) = \overline{L}_{L|D}(x) \cdot \sigma_{L|D}(x) \ . \tag{4.7}$$

Of course we do not want to sample $L(x)$ but its logarithm which means that we need to transform our acquisition function into log space too. This is very straight forward since we can just perform gaussian error propagation:

$$a_{\log L}(x) = \exp(\tilde{\mu}(x)) \cdot \exp(\tilde{\mu}(x))\sigma_{\tilde{\mu}}(x) = \exp(2 \cdot \tilde{\mu}(x))\sigma_{\tilde{\mu}}(x) \tag{4.8}$$

with $\mu = \overline{L}, \tilde{\mu} = \overline{\log L}$.

There is a catch though. The assumption which led to this conclusion is that $\overline{L}_{\mathrm{GP}}(x) \approx L(x)$ but is this is not necessarily a good assumption, especially when the number of acquired points is low. In fact it is an inherent mechanism of the algorithm to explore regions in which the model is very uncertain to achieve fast convergence. The longer the active sampling is running the better the space will be explored and our assumption will

[14]Strictly speaking it should be proportional to the posterior value at this point. We will omit this detail since we assume uniform priors in all examples which means that the uniform prior is essentially only a normalization constant. The formalism can trivially be extended to non-uniform priors by replacing $L(x)$ with $L(x)\pi(x)$ in all subsequent calculations.

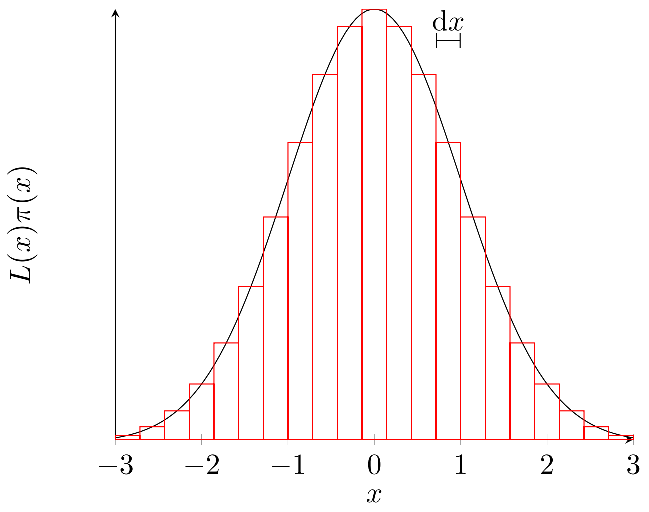

Figure 4.2: Illustration of the reasoning behind our choice of acquisition function. Every small interval $\mathrm{d}x$ contributes proportionally to its posterior value to the integral so the acquisition function should be designed to aim for high precision in regions of high posterior.

eventually hold true though.

In light of this one can argue that the factor 2 appearing in the first exponent should be relaxed in the early phases of active sampling to encourage exploration and slowly converge towards its final value. One way to achieve this is by introducing a monotonically increasing term $\zeta \in (0, 1]$ into the exponent:

$$a_{\log L}(x) = \exp(2\zeta \cdot \tilde{\mu}(x))\sigma_{\tilde{\mu}}(x) \tag{4.9}$$

A possibility inspired by simulated annealing [9] could be $\zeta = \exp(-N_0/N)$ where N are the numbers of samples on which the GP is conditioned and N_0 a characteristic "decay constant" or temperature. This term will will suppress the first term $\exp(2\zeta \cdot \tilde{\mu}(x))$ for $N \ll N_0$ and monotonically approach one for $N \gg N_0$.

In addition to the aforementioned correction factor we also need to account for statistical noise that the likelihood (or the log-likelihood) may have. While this is not the case for the examples we will be discussing in this chapter we will come back to this in chapter 5. For the moment let us assume that this statistical noise is i.i.d. gaussian noise σ_n in which case the lower bound on σ_{GP} is effectively the amplitude of the statistical noise. This means that we need to adjust our acquisition function to take this into account by replacing $\sigma_{\tilde{\mu}}(x)$ by $(\sigma_{\tilde{\mu}}(x) - \sigma_n)$ such that the acquisition function is given by

$$a_{\log L}(x) = \exp(2\zeta \cdot \tilde{\mu}(x))(\sigma_{\tilde{\mu}}(x) - \sigma_n) \tag{4.10}$$

A visualization of this is shown in Fig. 4.3. Without this addition the maximum of the AF will eventually converge towards one value which the algorithm will visit multiple times.

This behaviour should be avoided at all costs since GP regressors are numerically unstable to training samples which are very close together. In addition we would be wasting evaluations when sampling the same location multiple times.

Lastly one usually takes the logarithm of Eq. 4.10 for numerical reasons:

$$\log(a_{\log L})(x) = 2\zeta \cdot \tilde{\mu}(x) + \log(\sigma_{\tilde{\mu}}(x) - \sigma_n) \tag{4.11}$$

This does not impact the optimization itself since the logarithm is monotonic. The logarithmic version of this has the disadvantage that it falls off towards $-\infty$ for $\sigma_{\tilde{\mu}}(x) \to \sigma_n$ but the advantage that it does not become flat far away from its maxima, which can be challenging to navigate for numerical optimizers. For both versions gradients are simple to calculate if there are gradients for the GP (the mean and variance) available which can aid in accelerating the numerical optimization of the acquisition function.

To the best of my knowledge, this acquisition function has so far not been used in the literature.

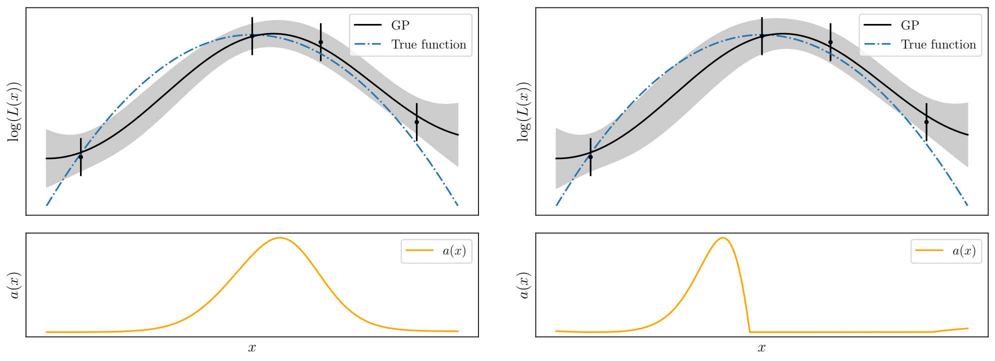

Figure 4.3: Illustration of the difference between including the statistical noise σ_n into the acquisition function $a(x)$ (right) versus ignoring it (left). It is clear that the maximum of the acquisition function will converge towards the maximum of $\log L$ if the noise is ignored even though there is no information to be gained in this point. In addition to this, GPs cannot properly deal with samples that are very close together meaning that the algorithm will numerically break down if the effect of σ_n is not subtracted.

4.4 Preprocessing

Preprocessing of data is a standard procedure when applying machine learning algorithms. This is done in an attempt to limit the input data to values which are not numerically problematic to compute. These requirements are different for the input locations x_{train} than for the target values y_{train}.

Input locations

The input locations (points in θ-space) need to fulfil two conditions for the active sampling algorithm to be effective. First the scales along each dimension need to be roughly similar. This is necessary for the optimization algorithm for the AF to be efficient.

Intuitively this becomes clear if one imagines a posterior distribution which is very stretched out along one axis while being very broad along another one. In this circumstance it will be very hard for a numerical optimizer to navigate along the narrow strip of optimal solutions, at least if this difference in length scales is not accounted for by the optimizer itself. This is the case for most standard numerical optimizers since they operate with a single characteristic length scale that the algorithm jumps at each iteration. Since we only need the length scales to be approximately similar i.e. a factor of two or three will not have a significant impact, it is enough to just take the bounds of the prior distribution (or some bounds which encompass the majority of the mass for unbounded priors) and normalize them to cover the unit hypercube.

Another important observation is that for an isotropic kernel function the distribution should not show any correlation along the different axes and have the same characteristic length scale along every dimension. This is not necessarily given but can be enforced if we assume that our function is a unimodal, multivariate gaussian distribution, which is the approach that is used in [2]. In this scenario we can empirically estimate the mean vector $\boldsymbol{\mu}$ and covariance matrix Σ along each dimension[15]. For a model containing N sampling locations $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_N$ with $\boldsymbol{\theta} = (\theta^1, \dots, \theta^d)^T$ the mean and covariance are given by

$$\mu^i = \frac{1}{\sum_{k=1}^N w_k} \sum_{k=1}^N \theta_k^i w_k \qquad \Sigma^{ij} = \frac{1}{\sum_{k=1}^N w_k} \sum_{k=1}^N w_k(\theta_k^i - \mu^i)(\theta_k^j - \mu^j) \tag{4.12}$$

where w_k is the value of the posterior distribution at location $\boldsymbol{\theta}_k$. The transformation that is applied to $\boldsymbol{\theta}_k$ is then

$$\theta_k^i \to \frac{\mathbf{R}^{ij}(\theta_k^j - m^j)}{\sigma^i} \tag{4.13}$$

where $\mathbf{R}$ is the matrix which solves $\Sigma = \mathbf{R}\boldsymbol{\Lambda}\mathbf{R}^T$ where $\boldsymbol{\Lambda}$ is a diagonal matrix and $\sigma^i = \sqrt{\Sigma^{ii}}$. This transformation is often called *whitening transformation* [35] since it transforms the data into a decorrelated gaussian with unit variance.

Unfortunately there are three major problems attached to this approach:

1. Correctly estimating the mean vector and the covariance matrix along each dimension is a challenge due to the high variability of the likelihood function compared to the log-likelihood function. One way to prevent this is to draw a large number of samples from the GP model and estimate the covariance from this however this introduces considerable computational overhead and also one needs an additional sampling algorithm like MCMC to correctly estimate the covariance matrix and mean vector.
2. The assumption that the posterior distribution is a multivariate gaussian distribution is very restrictive and generally not a very good assumption. For any

[15]Note that this is not the same as the mean vector and kernel of the GP. While the covariance matrix from the kernel for a GP model in d dimensions with N inputs has shape $N \times N$ while the covariance matrix in this method is of dimension $d \times d$.

posterior shape which does not closely match a multivariate gaussian distribution this transformation will inevitably fail.

3. The third issue which this transformation brings with it is the fact that we are empirically fitting the covariance matrix to the data. This is in essence MLII which means that it is technically not fully bayesian.

It is due to the aforementioned reasons that this transformation was not used in the final algorithm presented in this thesis but instead an anisotropic kernel with a different characteristic length scale is used along every dimension. This proved to be more versatile and less prone to failure than the whitening transformation.
This means that our choice for a kernel is a composite kernel consisting of a constant kernel multiplied with an anisotropic RBF or Matérn kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = C \cdot \prod_{i=1}^{d} \mathrm{RBF}(x_i, x_i') \quad \text{or} \quad k(\boldsymbol{x}, \boldsymbol{x}') = C \cdot \prod_{i=1}^{d} \text{Matérn}(x_i, x_i'|\nu) \tag{4.14}$$

with $\boldsymbol{x} = (x_1, \ldots, x_d)^T, \boldsymbol{x}' = (x_1', \ldots, x_d') \in \mathbb{R}^d$. Whether the RBF or Matérn kernel is the correct choice depends on the expected "smoothness" of the posterior distribution.

Target values
The target values (values of the likelihood or posterior distribution) are transformed to be centered around zero with unit variance. This has two reasons:

- Scaling the target values in this way is a standard procedure with GP regression as it limits the scale of the constant kernel component in the composite kernel.
- The transformation of the data such that it has zero mean effectively acts as a constant mean function of the GP where the value is the empirical mean of the data. This means that the GP will approach this value far away from any sampling locations. This approach is chosen to encourage exploration when the majority of samples are close to the maximum of the posterior distribution and exploitation when the majority of samples is situated at lower values. An illustration of this is shown in Fig. 4.4.

4.5 Parallelizing the algorithm

When doing Bayesian inference on expensive likelihoods there is usually an advantage to having some parallelization available in order to sample from the likelihood at different points at once. Many likelihoods have some internal mechanism for parallelization however we want to present a different approach which allows to acquire a batch of points for which the likelihood can then be interrogated. There have been a myriad of different proposals for batch acquisition for GPs in the past [36, 37, 38, 4] which all have their advantages and disadvantages.

Generally these algorithms can be divided into two broad categories. Algorithms like the q-EI algorithm [36, 37, 4, 39] construct an acquisition function which can be jointly optimized for several points at once while the second category [38, 4, 39] works by sequentially acquiring multiple points without having to sample from the function.

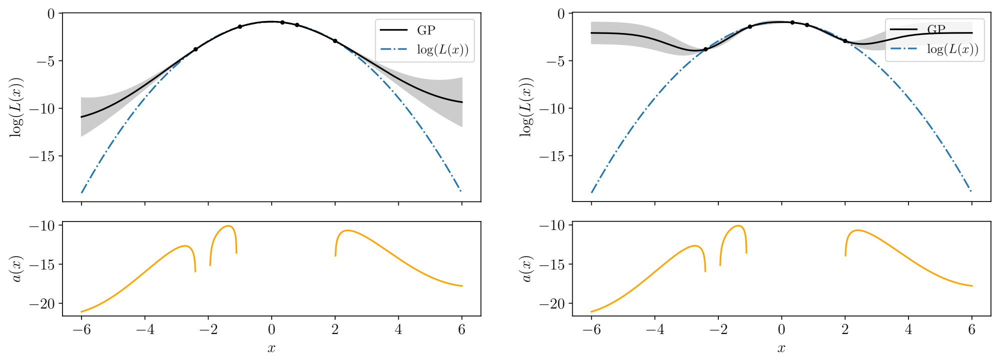

Figure 4.4: Illustration of the behaviour of a GP fitted to a normal distribution without preprocessing the target values (left) and with preprocessing (right). One can clearly see that the GP model with preprocessed data encourages exploration since it tends towards the mean of its samples in far away regions.

The advantage of the former method is that it will accurately estimate the impact that the acquisition of q points and their location has on sampling and thus find a more efficient set of samples that the latter method. This however comes at the expense that the computational overhead is much bigger. The reason for this is that for a GP which has d dimensions acquiring q points at once involves global optimization in $d \cdot q$ dimensions. One can easily imagine that this will quickly reach a point where performing this optimization becomes unfeasible.

Since our goal in this thesis is to eventually be able to tackle problems in higher dimensions we will focus on the latter, sequential method for batch acquisition. One of these methods is the *Kriging believer* method [39].

Kriging believer

The fundamental assumption of the Kriging believer method is that instead of optimizing a joint acquisition function for q points they can be acquired sequentially assuming that, at the location of the i-th point, the value of the likelihood function equals the predictive mean of the GP. While this method does not take into account the implications of sampling a batch of points given the current surrogate model, it has the advantage that the optimization problem is still a d-dimensional one.

In addition this method will be increasingly accurate as points are added to the GP since the underlying assumption of this method is, similar to the assumption for constructing the acquisition function, that $\overline{L}_{\text{GP}}(x) \approx L(x)$. Another advantage of this method is that it can be used for any acquisition function without additional computation in contrast to the joint approach where the acquisition function has to be derived from the single point AF by hand.

This approach furthermore assumes that sequentially acquiring points is much faster than sampling from the likelihood, otherwise the algorithm will be inefficient. An illustration of the Kriging believer algorithm sampling on the log of a normal distribution is shown in Fig. 4.5.

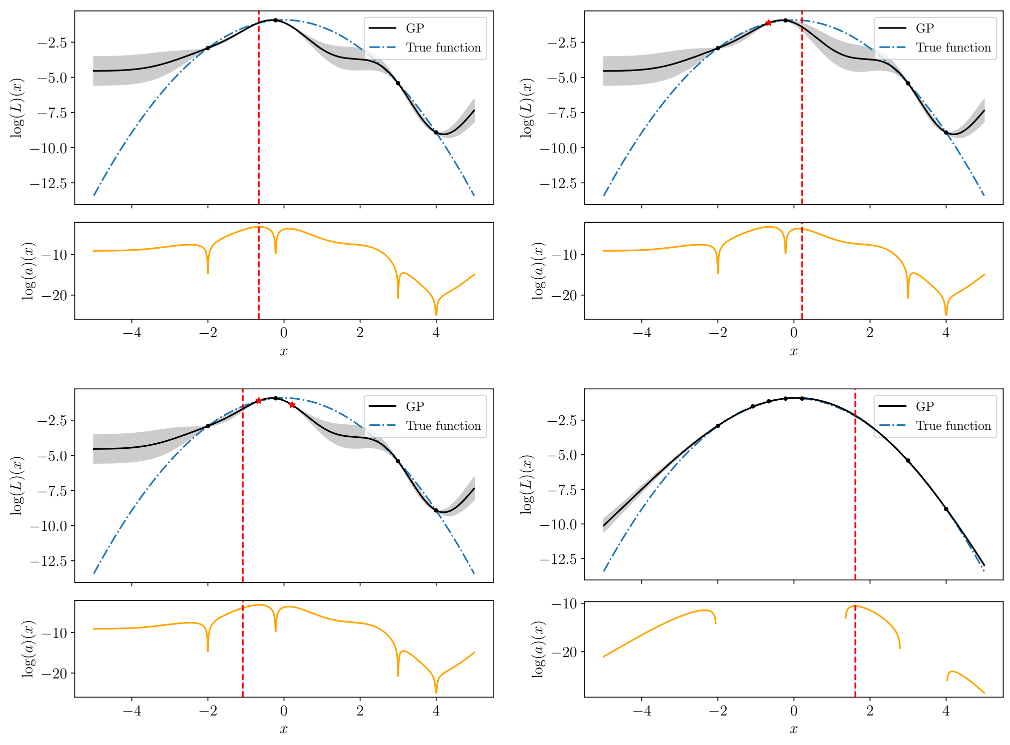

Figure 4.5: Illustration of the Kriging believer method. Three points are acquired sequentially (top left, top right, bottom left) by using the prediction from the GP instead of sampling the likelihood at each iteration. After the three samples have been acquired the likelihood function can be interrogated for the true values at these points (bottom right). The hyperparameters of the GP regressor only need to be refit at the last step. Obviously using this approach comes at the expense of requiring more points to converge towards the right value of Z. This can however be compensated by the computation time that is saved by evaluating the posterior in parallel.

The update equation

One convenient property of GP regression that has been pointed out by multiple authors [29, 40, 5] is the fact that data can be added to GPs without having to perform the whole inversion of the Covariance matrix in Eq. 3.8 when we do not want to perform regression (hence cannot change the hyperparameters of the kernel) but only want to condition the GP on additional data. In this case we can use the blockwise matrix inversion formula: Let $\mathbf{A}^{n\times n}, \mathbf{B}^{n\times m}, \mathbf{C}^{m\times n}, \mathbf{D}^{m\times m}$ be four matrices and let $\mathbf{A}^{-1}$ and $(\mathbf{CA}^{-1}\mathbf{B})^{-1}$ be invertible. Then the inverse of the blockmatrix $\mathbf{M}$ is given by

$$\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{BS}^{-1}\mathbf{CA}^{-1} & -\mathbf{A}^{-1}\mathbf{BS}^{-1} \\ -\mathbf{S}^{-1}\mathbf{CA}^{-1} & \mathbf{S}^{-1} \end{bmatrix}. \tag{4.15}$$

where $\mathbf{S} = \left(\mathbf{D} - \mathbf{B}^T\mathbf{A}^{-1}\mathbf{B}^T\right)$ is called the *Schur complement* of $\mathbf{A}$. In our case the assumption that all inverses exist always holds true if $\mathbf{M}$ is a valid covariance matrix. In addition the formula can be simplified for the covariance matrix since $C = B^T$ which yields

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}\mathbf{S}^{-1}\mathbf{B}^T\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}\mathbf{S}^{-1} \\ -\mathbf{S}^{-1}\mathbf{B}^T\mathbf{A}^{-1} & \mathbf{S}^{-1} \, . \end{bmatrix} \tag{4.16}$$

A similar formula exists for the Cholesky decomposition which is handy since this is mostly used in practice when performing GP regression. This formula is especially useful if $\mathbf{A}^{-1}$ is already known from previous computations and $n \gg m$ since the algorithm scales with $\mathcal{O}(n^2 m)$.

Combining the two

With the help of the blockwise matrix inversion formula it can be shown that the Kriging believer algorithm does not affect the mean prediction of the GP and thus we do not need to change the hyperparameters of our surrogate model between acquisitions. For a full proof see 6. Intuitively this makes sense since we are not adding any new information to the model. This in turn means that we can simply use the blockwise matrix inversion formula which saves time on the inversion of the $K(x, x)$ matrix. In particular the computational overhead when adding a single point to the GP containing n points is only $\mathcal{O}(n^2)$ as opposed to $\mathcal{O}((n+1)^3)$ for the full inversion. To my knowledge this has not been pointed out in the past and is a good way to save time on the acquisition of points.

With that in mind we can not only take advantage of the Kriging believer method to evaluate the likelihood in parallel but in addition also speed up the algorithm itself. The final algorithm taking this into account with m points which are acquired by the Kriging believer algorithm and a total of $m \cdot n$ acquired points is presented in algorithm 5 which is one of the main results of this thesis.

4.6 Convergence criterion

To efficiently be able to perform Bayesian inference we need to have some convergence criterion in place which we can use to assess to what degree our algorithm has converged. A natural candidate would be the variance of the integral over Z. For a d-dimensional likelihood this is unfortunately a $2 \cdot d$ dimensional integral which is numerically challenging to handle. Furthermore our approach to optimizing the hyperparameters of the GP violates bayesianity and hence cannot be trusted, which is discussed in detail in section 4.8. Therefore we follow the approach from [2] and use the *Kullback Leibler* (KL) divergence between consecutive acquisition steps to estimate the progress of the Bayesian inference. For two continuous probability distributions $P(\boldsymbol{x})$ and $Q(\boldsymbol{x})$ this is defined as the integral

$$D_{\mathrm{KL}}(P|Q) = \int P(\boldsymbol{x}) \log\left(\frac{P(\boldsymbol{x})}{Q(\boldsymbol{x})}\right) \mathrm{d}\boldsymbol{x} \tag{4.17}$$

Since this is again computationally quite expensive we approximate the posterior distribution to be a multivariate gaussian distribution for which the KL divergence is analyt-

```
Input: $\mathcal{GP}(0, k(x, x'|\theta))$ (GP regressor with kernel), $n_{\text{init}}$ (Number of
       initial samples), $a(\mu(x), \sigma(x))$ (acquisition function), $f(x)$
       (function to be sampled)
[1]  Randomly draw $n_{\text{init}}$ initial samples $x_0$
[2]  $y_0 := f(x_0)$
[3]  $x_{\text{train}} := x_0$
[4]  $y_{\text{train}} := y_0$
[5]  for $n$ times do
[6]      Fit $\theta_{\text{GP}}$ with $x_{\text{train}}, y_{\text{train}}$                      alg. (2)
[7]      $x_{\text{kb}} := \text{copy}(x_{\text{trian}})$
[8]      $y_{\text{kb}} := \text{copy}(y_{\text{trian}})$
[9]      for $m$ times do
[10]         Draw $x$
[11]         repeat
[12]             $\mu(x), \sigma(x)$ :=GP prediction                  alg. (3)
[13]             $a_x := a(\mu(x), \sigma(x))$
[14]             vary $x$ according to some global optimizer
[15]         until $\max[a(\mu(x), \sigma(x))]$ reached
[16]         $x_{\text{kb}} = \{x_{\text{kb}}, \text{argmax}(a(x))\}$
[17]         $y_{\text{kb}} = \{y_{\text{kb}}, \mu(\text{argmax}(a(x))\}$         Kriging believer
[18]         Update $K(x, x)^{-1}$ with Eq. (4.16)
[19]     end
[20]     $x_{\text{train}} = x_{\text{kb}}$
[21]     $y_{\text{train}} = f(x_{\text{kb}})$                        Can be parallelized
[22] end
[23] return GP regressor
```

Algorithm 5: Illustration of the active sampling algorithm with m points being acquired at once with Kriging believer. n Kriging believer steps are performed so a total of $n \cdot m$ points are added to the GP by active sampling. The advantage to the simple active sampling shown in algorithm 4 is the faster computation time and the possibility for the evaluation of the likelihood (line 19) to be performed in parallel. In practice the values for y_{train} are saved at every step which means that for each iteration of the outer loop only m points have to be sampled from the likelihood.

ical and given by the formula

$$D_{\mathrm{KL}}\left(\mathcal{N}_0|\mathcal{N}_1\right) = \frac{1}{2}\left(\mathrm{tr}\left(\Sigma_1^{-1}\Sigma_0\right) + \left(\mu_1 - \mu_0\right)^{\mathsf{T}}\Sigma_1^{-1}\left(\mu_1 - \mu_0\right) - k + \ln\left(\frac{\det\Sigma_1}{\det\Sigma_0}\right)\right) \quad (4.18)$$

where μ_0, Σ_0 and μ_1, Σ_1 are the mean and covariance of $\mathcal{N}_0$ and $\mathcal{N}_1$ and k is the number of dimensions. $\boldsymbol{mu}$ and Σ can be empirically estimated from the training data with Eq. 4.12. This has the same problems that have been explained for preprocessing in section 4.4 but since this is only a stopping criterion it is not crucial that this gives the correct result right away.

The KL divergence can be used as a convergence criterion by stopping the algorithm when $D_{\mathrm{KL}} < \varepsilon$ between a number of acquisition steps. [2] sets $\varepsilon = 0.01$ which we will also do. Note that this convergence criterion assumes that the posterior distribution is a multivariate gaussian distribution which is by no means given. Therefore finding a better criterion is likely possible and advisable. However for our purposes this will suffice. The performance of this criterion is assessed in section 4.9 and illustrated in Fig. 4.11.

4.7 The problem with infinity

An issue that has to be addressed when sampling the log-posterior distribution is that it falls off towards $-\infty$ towards the edges. This is a problem since GPs cannot deal with infinite values as Eq. 3.7 (or Eq. 3.14 in the noisy case) become ill defined. This means that there needs to be a solid mechanism in place to deal with these points. In the context of BO some research has been conducted towards solving this problem which relies on defining a region in which it is safe to explore which is based on the observations that have been made up to this point [41, 42]. This approach has the advantage that it is keeps the GP stable but it comes at the expense of added computational overhead. Additionally finding a suitable region in high dimensions is a very challenging task.

A different approach is to replace the $-\infty$ value by some large, negative finite value. While this is a computationally cheap and easy to implement method it unfortunately does not work very well in practice. This is because our initial assumption about the GP rests on the idea that the posterior distribution is a continuous, differentiable function which is violated by introducing discontinuity. Including these points will either distort the correlation length if the fixed points are added during the regression step or lead to unwanted oscillations if they are added with the blockwise matrix inversion formula. This behaviour is shown in Fig. 4.6 (left).

The third idea for dealing with these points is to use the Kriging believer framework to make a prediction for these points. This preserves the continuity of the function but it drastically overestimates the posterior distribution in this point. This behaviour is shown in Fig. 4.6 (right).

Lastly there is the possibility to simply ignore samples which yield $-\infty$. This has the advantage that the model stays exact but the disadvantage that the algorithm will repeatedly sample this location since the information about it is discarded. It can be somewhat alleviated when using batch acquisition with the Kriging believer algorithm since there is a good chance that this point will not be in the next batch. Eventually the algorithm will start exploring though, which inevitably will lead it into regions where the log-posterior distribution is $-\infty$. This topic will therefore require more research and a rigorous solution for a robust and versatile framework but for the sake of simplicity we will simply use the last option and ignore sampling locations where the posterior distribution vanishes.

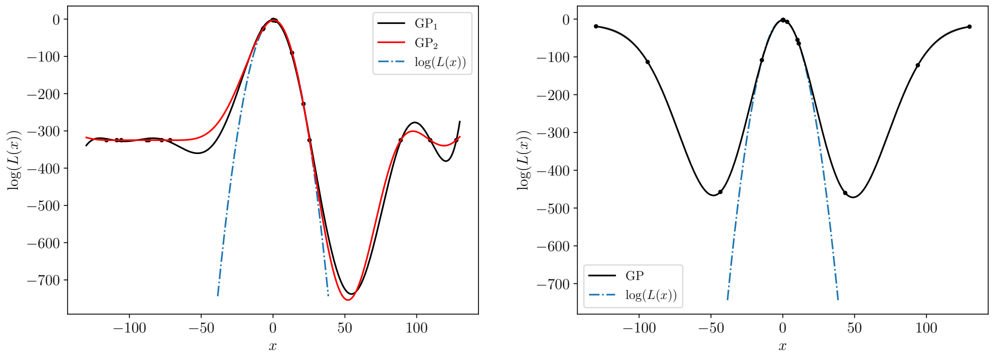

Figure 4.6: Illustration of different computationally cheap methods to deal with $-\infty$ values in the log likelihood with a gaussian posterior. The left plot shows the behaviour when samples which yield $-\infty$ are replaced by a constant value which in this case is the minimum of all finite values. GP$_1$ (black) shows the behaviour when adding the points with the blockwise matrix inversion lemma and GP$_2$ (red) when fitting the hyperparameters of the GP with the artificial samples included. The oscillations which this approach introduces are clearly visible. When the artificial values are set even lower the oscillations become more pronounced. The right plot shows the behaviour when setting all infinite samples to the mean of the GP at this point (essentially this is like the Kriging believer algorithm). This does not introduce any oscillations but gives a wrong shape for the posterior distribution.

4.8 Preserving bayesianity

A lot of effort so far has been directed into the conservation of bayesianity, i.e. using only Bayesian statistics to construct our full model. The final goal of this thesis is however to create a practical, fast and robust framework which unfortunately means that the bayesian approach had to be sacrificed when fitting the hyperparameters of the GP regressor since this is done with MLII. This could have been avoided but would have added computational overhead. This translates to a general underestimation of the variance of predictions by the GP.

The second point where bayesianity is omitted is when performing the integration of the GP with MCMC. In principle it would be possible to include the variance of the predictions during this step but it again requires some computational overhead. Both of these points are connected since they both relate to the variance of the predictions by the GP. It is therefore only a good assumption to omit these when this variance is very small. It is therefore in our best interest to sample until the variance of the GP is a negligible contribution to the evidence. This is achieved by using the KL divergence as a convergence criterion and sampling until a good precision is reached.

A more rigorous treatment of the two aforementioned issues would certainly be desirable but is beyond the scope of this thesis.

4.9 Experiments

Now that we have established the full algorithm it is time to test it and compare its performance to the state of the art. Here we will use toy examples where we restrict ourselves to simple, unimodal gaussian posteriors since much of our algorithm assumes approximate gaussianity. Furthermore we will investigate how the algorithm performs on two real likelihoods. In these tests we are interested in three metrics:

1. Scaling of the required number of samples with dimensionality
2. The computational overhead that our algorithm introduces
3. The impact of the hyperparameters of the algorithm on the performance

These hyperparameters are:

- The choice of kernel (RBF or Matérn)
- The number of initial samples before acquisition n_{init}
- The number of times $n_{\mathrm{restart,GP}}, n_{\mathrm{restart},a}$ that the optimizers for the GP hyperparameters and for the acquisition function are restarted from random initial points.
- The number of acquired points with Kriging believer m which is done at every iteration before interrogating the likelihood and refitting the hyperparameters of the GP. This will henceforth be called the number of Kriging believer steps
- The correction factor of the acquisition function ζ

However only the last two of these parameters are really variable. The choice of kernel depends on the prior knowledge of the shape of the likelihood and should thus be set according to the same criteria as the prior distribution. T

he number of initial samples before acquisition is barely relevant as the algorithm is designed to explore the parameter space efficiently. However an initial set of samples which roughly follows the posterior distribution can accelerate convergence considerably.

How often the optimizers for the acquisition function and GP regressor should be restarted at random locations depends on the optimizer used and on the computational budget available. A good optimizer global should in principle be able to find the global maximum from any starting position. In practice this value will be set so that it ensures that the optimizer reliably finds the global maximum.

This means that we are essentially only left with two free parameters to choose which are m and ζ. These will be investigated in the following.

Toy examples

As the Kriging believer framework allows us to evaluate the posterior in parallel we should take this into account when benchmarking the algorithm. In addition the computational overhead of running the algorithm, which is performed sequentially and hence does not decrease with the number of cores available should be taken into consideration.

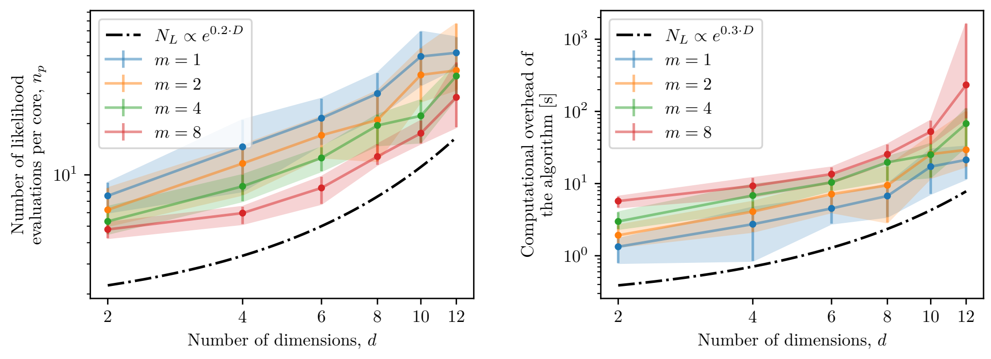

Figure 4.7: Scaling of the number of posterior evaluations (left) and computational overhead of our algorithm measured in wall clock time (right). The approximately exponential scaling of the number of points becomes apparent towards higher values of d. The computational overhead that our algorithm produces is purely sequential and hence increases with the number of Kriging believer steps. Nevertheless, for slow enough likelihoods, this overhead is compensated for by the low number of total evaluations required for the algorithm to converge which is shown in Fig. 4.9.

Both of these metrics are shown against the number of dimensions with 1, 2, 4 and 8 acquired points in parallel. To be able to compare against MCMC the tests were performed the same as the tests done for Fig. 2.4 meaning that we used mildly correlated[16] multivariate gaussian distributions were used. ζ was set to 1 and the stopping criterion for reaching convergence was $D_{\mathrm{KL}} = 0.01$.

During this test and all consecutive tests we set $n_{\mathrm{restart},GP} = 10$ and $n_{\mathrm{restart},a} = 5$ to make the optimizations robust. The results are shown in Fig. 4.7. One can see that both the number of points required as well as the computational overhead increase exponentially with the number of dimensions. Furthermore it is clear from the plot on the left that the Kriging believer algorithm decreases the number of points required per core although the algorithm does not parallelize perfectly. It is furthermore obvious from the right plot that the computational overhead of the algorithm increases with the number of Kriging believer steps m. This makes sense since overall more points are required until convergence is reached and the computational overhead of our algorithm does not decrease with the number of cores.

This effect is however mitigated by the fact that the computational overhead for acquiring a number of training points n_{train} decreases when raising m because of the use of the blockwise inversion formula. This is visible in Fig 4.8 (left). On the right of this figure the computational overhead of our algorithm is dissected into its main contributing components which are the acquisition of new samples and the GP regression step. One can see that the regression step gives the dominant contribution which was to be expected due to the $\mathcal{O}(n^3)$ scaling of the Cholesky decomposition.

Having investigated both the computational overhead and the number of evaluations

[16]Mildly correlated means here that we transformed the randomly drawn covariance matrix such that the correlation coefficient between dimensions does not exceed 0.1.

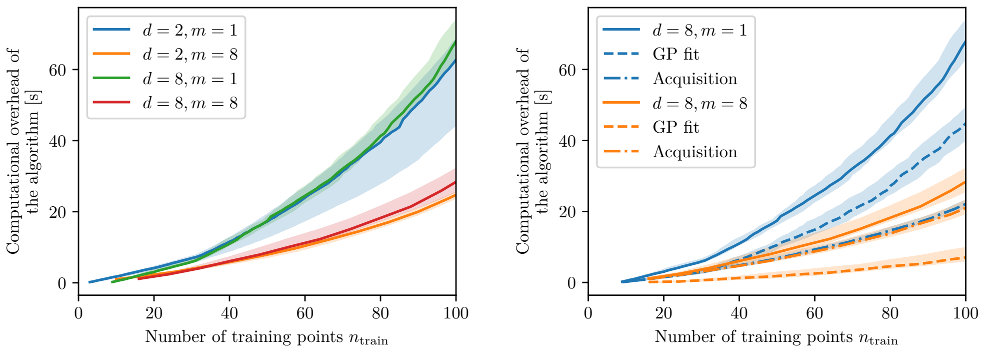

Figure 4.8: Cumulative computational overhead in seconds against the number of training samples n_{train} for $d = 2, 8$ (right). It is clear that the computational overhead per added training sample decreases for increased m. This is because the batch acquisition algorithm increases performance. Furthermore the computational overhead increases for higher dimensions d but this is only a small effect. On the left side the computational overhead is dissected into its main contributions, the acquisition step and the regression step for $d = 8$ and $m = 1, 8$. When using batch acquisition the computational overhead decreases both for the acquisition step and the regression step.

needed to achieve convergence for the toy cases we can now proceed to make some predictions of the performance of our algorithm compared to MCMC. We are particularly interested in whether our algorithm can outperform MCMC in terms of speed (in wall clock time). Therefore we need to take into consideration the time it takes to calculate the posterior for a single sample (this will be assumed to be constant in the following), the number of evaluations for both MCMC and our algorithm which are required to achieve convergence as well as the computational overhead for our algorithm. We conservatively neglect the computational overhead of MCMC.

The results of this are shown for four different exemplary combinations of m and d in Fig. 4.9. From this comparison it is clear that our algorithm will always win in terms of efficiency for slow likelihoods/posteriors since our algorithm only requires a small fraction of the samples that MCMC requires to converge. This comes at the cost of added computational overhead though which means that for fast likelihoods (in our examples $10^{-4} - 10^{-3}$ evaluations per second) MCMC will converge faster. In addition to this MCMC is more robust and does not make the strong assumptions on smoothness that our algorithm makes. Furthermore it is important to keep in mind that the exponential scaling of the computational overhead and required numbers of samples with dimensionality effectively limit the number of dimensions this algorithm can efficiently used in to $d \lesssim 10$ for most practical examples.

Real world data

Now that we have a robust understanding of how our algorithm scales with dimensionality and where its strengths and weaknesses lie it is time to try it on some real world data. For this two examples which fit our criteria (approximate gaussianity and $d < 10$) were chosen:

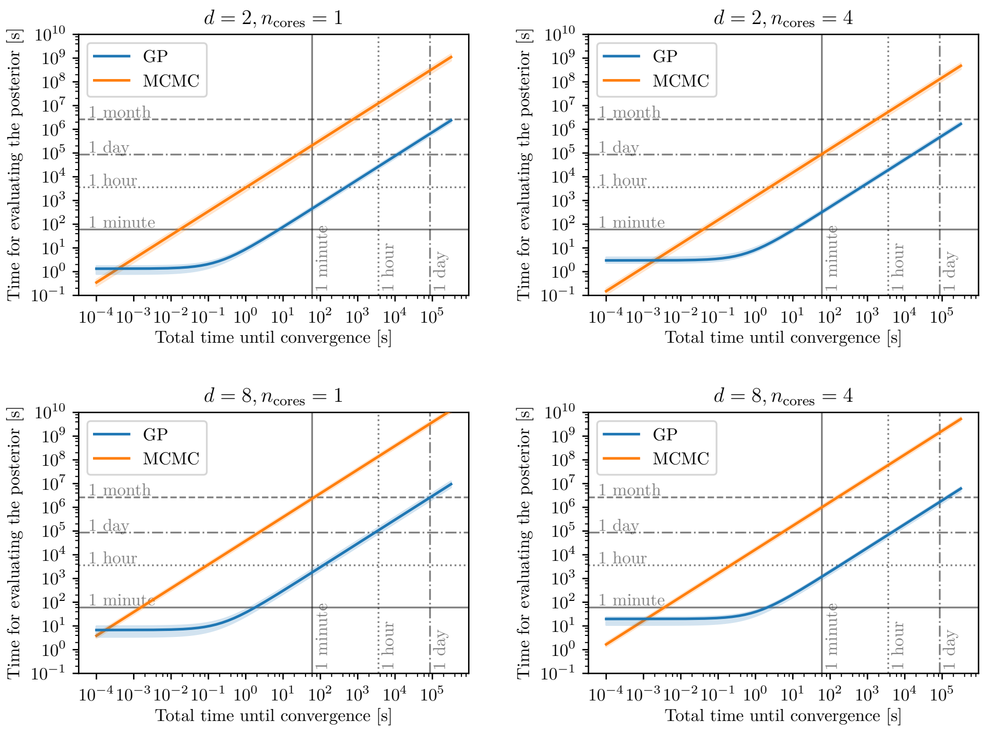

Figure 4.9: Comparison of the performance of our algorithm vs. MCMC in the toy setting for different combinations of m and d and assuming different times for evaluating the posterior distribution. One can see that our algorithm becomes more efficient than MCMC if the posterior evaluation time passes a certain limit. The reason for this is the small number of posterior samples required for our algorithm to converge. In addition the benefit of using batch acquisition if multiple cores are available is visible. This however moves the threshold after which our algorithm is faster than MCMC towards higher posterior evaluation times.

- The two dimensional big bang nucleosynthesis (BBN) likelihood used in [43].
- The six dimensional Planck lite likelihood [44, 45] which is the 27 dimensional Planck likelihood marginalized over the 21 dimensions which do not correspond to cosmological observables. The theoretical predictions for the cosmological observables have been computed using the `CLASS` cosmological Boltzmann code [46].

We start by testing our algorithm on the Planck lite likelihood with uniform priors which have been set to $\sim 3\sigma$ of the marginalized distribution which was obtained from the official Planck chains [47]. We then ran our algorithm with $m = 2$ Kriging believer steps, $\zeta = 1$ and $n_{\mathrm{init}} = 10$ initial points until a KL divergence of 0.01 was reached.

An MCMC chain was then run on the GP regressor and the resulting MCMC chain was marginalized and plotted with `GetDist` [48]. This was compared to the official Planck chains. The result is shown in Fig 4.10. In total the posterior was evaluated 94 times with an average time of $\approx 5\,\mathrm{s}$ per posterior evaluation. The total computational overhead of our algorithm was $\approx 45\,\mathrm{s}$. This gives a total runtime of $\approx 280\,\mathrm{s}$. If we assume that the MCMC algorithm needs $\sim 10^4$ evaluations to reach convergence this amounts to a speed-up of $\mathcal{O}(10^2)$ which effectively allows the computation of this on a laptop instead of a cluster.

Additionally the Planck likelihood can be used to test the influence of m, ζ and n_{init} on the convergence. The KL divergence against the number of posterior evaluations if either m, ζ or n_{init} is varied is shown in Fig. 4.11. One can see that increasing m leads to an increase in posterior evaluations to achieve convergence which is consistent with the results shown in Fig. 4.7.

Furthermore increasing N_0 where $\zeta = \exp(-N_0/N)$ where N is the number of training points for the GP leads to slower convergence although the KL divergence shows less variability. Again this was to be expected due to the reasons given in section 4.3. Changing the number of randomly drawn initial samples essentially has no impact on convergence which underlines the fact that our algorithm can efficiently explore the parameter space and find the maximum of the distribution. This also highlights the numerical issues with using the KL divergence though as it is numerically not very stable.

Next the algorithm was tested against the BBN likelihood with $m = 2, n_{\mathrm{init}} = 3$ and $N_0 = 0$. The result of this is shown in Fig. 4.12. The posterior was evaluated 29 times with a total computational overhead of $\approx 9\,\mathrm{s}$. The BBN likelihood is quite fast however as a single posterior evaluation takes $\approx 10^{-4}$ seconds. This means that if we conservatively assume that MCMC needs $\sim 10^3$ evaluations for convergence our algorithm is $\mathcal{O}(10^2)$ times slower than MCMC. This illustrates that the posterior evaluation time is the crucial component when assessing whether using our algorithm is advantageous to using MCMC.

The same BBN likelihood also lends itself to experimentation regarding the robustness of the algorithm for non-gaussian likelihoods. We can get such likelihoods if we only include the measurement of the primordial helium abundance Y_P by [49] or the primordial deuterium abundance $y_{\mathrm{DP}} = 10^5 n_{\mathrm{D}}/n_{\mathrm{H}}$ instead of both. This gives an elongated, non-gaussian shape in both cases which we can give to our algorithm. The result is shown in Fig 4.13 with $m = 2$, $n_{init} = 1$ and $\zeta = 1$ the KL divergence for terminating the

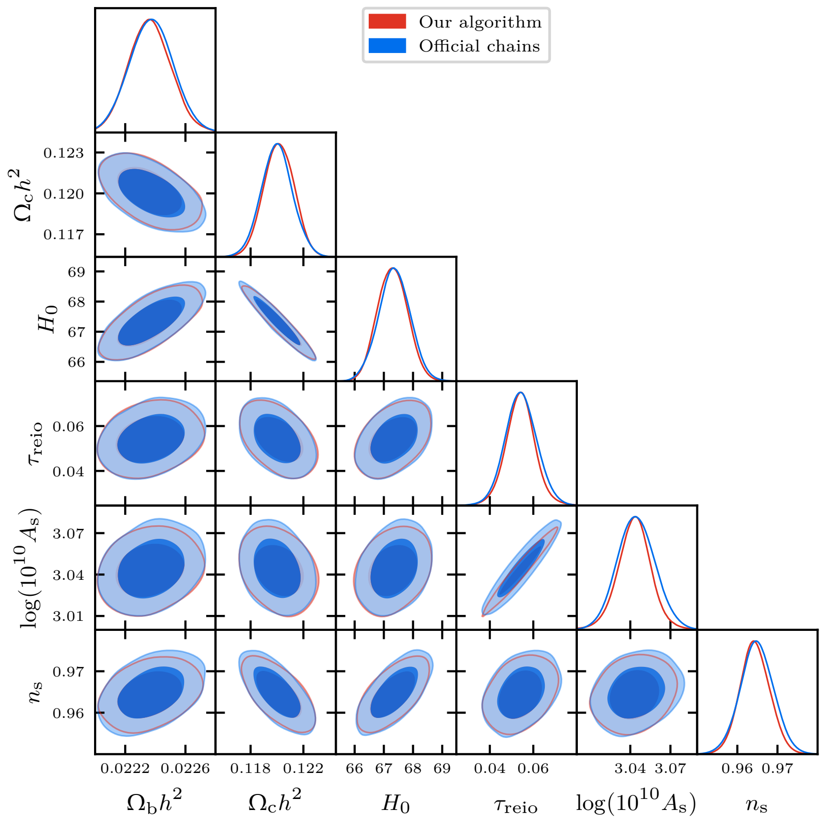

Figure 4.10: Marginalized 2d 1σ and 2σ contours as well as marginalized 1d distributions of our algorithm compared with the official Planck chains. The contours, correlations and marginal distributions are well recovered by our algorithm which offers a computational speed-up of $\mathcal{O}(10^2)$ (in wall clock time) compared to MCMC.

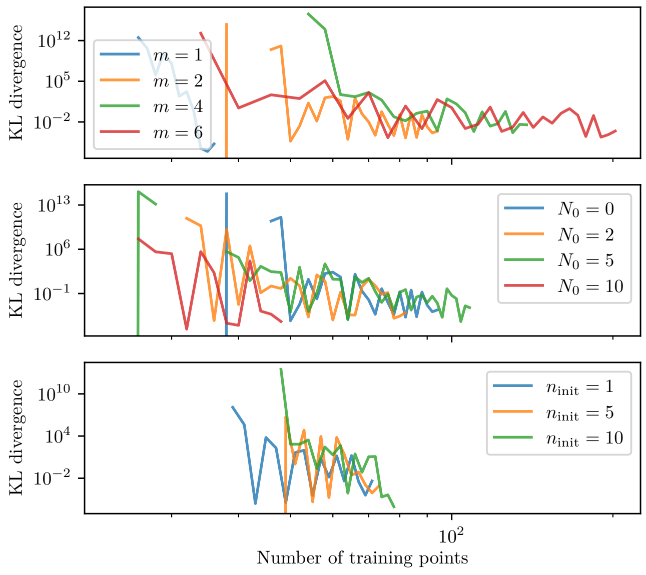

Figure 4.11: KL divergence against the number of training samples of the GP (posterior evaluations) when varying m, N_0 where $\zeta = \exp(-N_0/N)$ and n_{init}. Only one parameter is varied for each plot while the other ones are kept at $m = 2$, $N_0 = 0$ and $n_{\text{init}} = 10$. One can see that increasing m and N_0 causes the algorithm to converge slower although increasing m allows for the parallel evaluation of the posterior and $N_0 > 0$ stabilizes convergence. Changing n_{init} essentially has no impact.

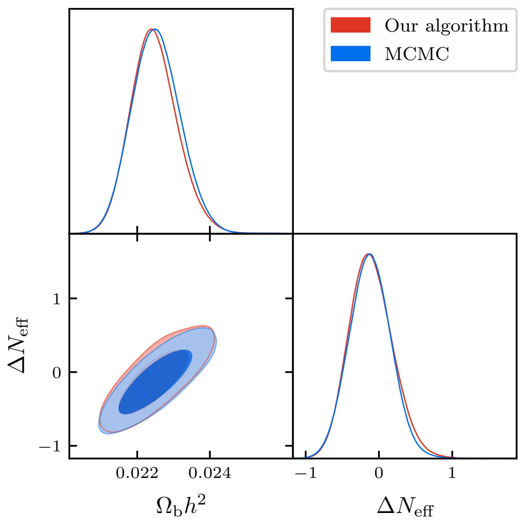

Figure 4.12: Comparison of our algorithm with MCMC for the two-dimensional BBN likelihood from [43]. Our algorithm correctly captures the 1 and 2 σ contours. It is however less efficient than MCMC since the computational overhead dominates the evaluation time.

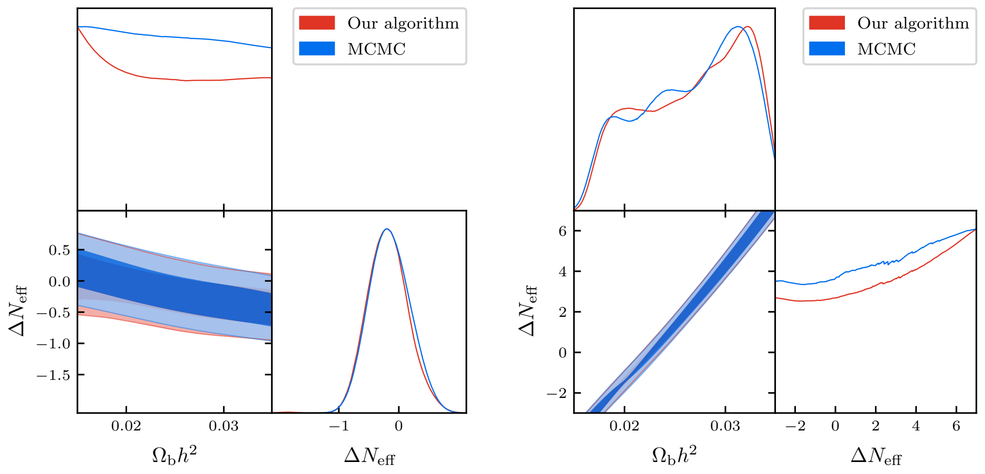

Figure 4.13: The same BBN likelihood as in Fig 4.12 when only including the measurement of the primordial helium abundance Y_P (left) or primordial deuterium abundance y_DP (right). Even though both posterior distributions are non-gaussian the 1 and 2 σ contours are recovered correctly. The algorithm struggles to correctly recover the 1d marginal distributions though.

algorithm was set to 0.005.

When only including the measurement of Y_P convergence convergence was reached after 37 posterior evaluations, when only including y_DP after 55 evaluations. Although the 2d 1σ and 2σ contours are recovered well the marginalized distributions do not match fully. Nevertheless this shows that the algorithm is robust enough to deal with non gaussian posterior shapes.

All tests shown here were done on an Intel Core i5-6300U CPU with 4×2.4 GHZ clock speed. The posterior of the BBN likelihood as well as the numerical integration of the GP regressor has been performed with the MCMC algorithm that has been developed for `CosmoMC` [17, 18]. The `Cobaya` package has been used for translating between our algorithm and the MCMC sampler and between our algorithm and the Planck likelihood. Figures 4.10, 4.12 and 4.13 were generated using `GetDist` [48]. Our algorithm which has the preliminary name `GPry` uses the packages `numpy` [50], `Scikit-learn` [51] and modified parts of `scikit-optimize` [52]. The GP hyperparameters and the Acquisition function are optimized using the `scipy` [53] implementation of the Large-Scale Bound Constrained Optimization algorithm (L-BFGS-B) [54, 55]. `GPry` will be released for public use in the near future.

5 A hybrid Nested Sampling/GP approach

This chapter will present the methodology used to develop an algorithm that uses a combination of GPs and nested sampling to build a framework which can exploit the advantages of both methods.

5.1 Taking advantage of speed hierarchies

As Bayesian inference requires the partial or full marginalization of the Likelihood function as explained in 2.1 this necessitates the accurate numerical modelling of the latter such that the value of the integral convergences to the true value. This can be done with Nested Sampling methods as explained in 2.2.

For simple (analytic) p the Likelihood function can be computed analytically too which usually means that its calculation is not particularly computationally demanding. However for many examples in science this Likelihood can be quite tricky to compute and may involve numerical methods like integration, n-body simulations or approximate solvers for differential equations [56, 57]. This means that an evaluation for a single set of parameters θ demands a lot of computational resources which raises the need for more efficient algorithms than MCMC which discards many of the evaluations of the Likelihood.

On the other hand Bayesian quadrature which is very efficient at sampling a parameter space have been brought forward as possible solutions to this problem [2]. However this method suffers heavily under the curse of dimensionality. This has three reasons:

1. With a higher dimensional parameter space one inevitably needs to get a higher number of samples from the likelihood to achieve convergence as the hyper volume of the parameter space increases exponentially. Furthermore the number of corners also rises exponentially which means that the volume where the bulk of the mass is concentrated decreases exponentially with respect to the hypervolume of the prior.
2. The computational complexity of fitting the hyperparameters of a GP with MAP increases proportional to n^3. This means that sequentially acquiring points with active sampling increases the computational complexity to $\sim n^4$. It is therefore in our interest to keep the number of samples in the GP as low as possible which conflicts with the first point.
3. The acquisition function has as many dimensions as the parameter space which needs to be sampled. Since this needs to be optimized that means that higher dimensionality also requires higher dimensional global optimization which is generally harder to achieve than in lower dimensions. This problem is not quite as significant as the last two ones as even if the global maximum is not reached during the optimization of the acquisition function getting stuck in a local maximum does not break or slow down the algorithm significantly. Furthermore with enough tries and a good choice of acquisition function all global optima will be explored eventually.
   Nevertheless it is good to keep this issue in mind. This overall exponential scaling of the number of evaluations as well as the computational overhead required to reach convergence means that the GP is only useful for likelihoods which are below

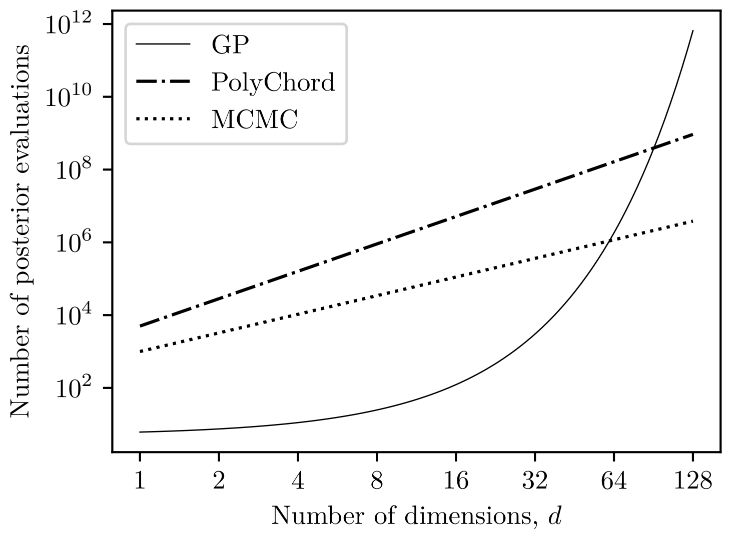

Figure 5.1: Illustration of the approximate number of evaluations required for convergence against the number of dimensions. We empirically assume that MCMC and `PolyChord` scale polynomially $\propto d^{1.7}$ and $\propto d^{2.5}$ respectively and that BQ scales exponentially $\propto \exp(0.2d)$. It is clear that MCMC is preferable over BQ when $n_{\mathrm{GP}} > n_{\mathrm{MCMC}}$ which is at $d \approx 60$. If one also takes into consideration the computational overhead that BQ introduces however which also scales approximately exponentially it is clear that BQ becomes prohibitively expensive for most cases if $d \gtrsim 10$

a certain number of dimensions which is illustrated in Fig. 5.1. Here we assumed that the number of evaluations for the GP increases $\propto \exp(0.2 \cdot d)$ while it increases for MCMC and `PolyChord` polynomially with $\propto d^{1.7}$ and $\propto d^{2.5}$ respectively. One can see that at $d \approx 60$ BQ and MCMC need approximately the same number of evaluations for convergence which makes MCMC the method of choice since it does not require any additional computational overhead. Furthermore one has to keep in mind that the computational overhead of BQ also approximately increases exponentially which makes this prohibitively expensive for most likelihoods if the number of points reaches $\mathcal{O}(10^3)$. This means that for most practical applications BQ can only efficiently be applied if $d \lesssim 10$.

In addition to the aforementioned issues associated with MCMC and GP algorithms for Bayesian inference, likelihoods in physics often have some common properties which can be exploited in an algorithm:

1. In most cases one is only interested in performing Bayesian inference[17] on a small subset of parameters $\boldsymbol{\theta}_{\mathrm{p}} = (\theta_{\mathrm{p},1}, \theta_{\mathrm{p},2}, \dots)^T \subseteq \boldsymbol{\theta}$ which we will henceforth call *physical* parameters since they are usually associated with underlying physics in contrast to the rest $\boldsymbol{\theta}_{\mathrm{n}} = (\theta_{\mathrm{n},1}, \theta_{\mathrm{n},2}, \dots)^T = \boldsymbol{\theta} \backslash \boldsymbol{\theta}_{\mathrm{p}}$ which we will summarize under the name of *nuisance* parameters. These are typically parameters which are asso-

[17]In this context Bayesian inference means that we want a full characterization of a parameter space $\overline{\boldsymbol{\theta}} \subseteq \boldsymbol{\theta}$ such that we can obtain marginal quantities etc.

ciated to the experiment like cuts, filters and instrument parameters. The number of nuisance parameters often exceeds that of physical parameters.

2. Many likelihoods in physics have an inherent speed hierarchy meaning that changing some parameters requires more computation than changing others. As such parameters can usually be grouped into a "slow" and a "fast" category where the difference in time it takes for recomputing the likelihood when changing a slow parameter can often exceed the time it takes when changing a fast parameter by a factor of 10^3 or more.
   Luckily for us this speed hierarchy very often translates to physical and nuisance parameters where $\boldsymbol{\theta}_\mathrm{p}$ are slow and $\boldsymbol{\theta}_\mathrm{n}$ fast. The reason for this is that recalculating the physical model often involves some heavy numerical computation of integrals and non-linear effects while changing $\boldsymbol{\theta}_\mathrm{n}$ essentially just reweighs the likelihood function.

An example of such a hierarchy is the likelihood of the Planck experiment which 27 parameters assuming ΛCDM. Only six of those are parameters which depend on the underlying cosmology which require numerically solving the Boltzmann equation while the 21 other parameters are instrument parameters and astrophysical effects which are $\mathcal{O}(10^3)$ faster to compute.

Due to the reasons explained above it is clear that MCMC, nested sampling and Bayesian quadrature each have their strengths and weaknesses: MCMC and nested sampling scale better with dimensions but need a higher number of posterior evaluations in low dimensions while Bayesian quadrature is very efficient in low dimensions but does not scale very well. In addition the computational overhead of GP regression makes it prohibitively expensive for $d \gtrsim 10$. This naturally raises the question whether we can exploit these characteristics by using nested sampling for nuisance parameters, which are typically fast parameters and more numerous than the physical parameters and using Bayesian quadrature for the slow, physical parameters.

It turns out that this is indeed possible when a number of criteria are met which will be detailed in the following.

5.2 The method

Our goal in Bayesian inference is to solve the integral

$$I = \int L(D|\boldsymbol{\theta})\pi(\boldsymbol{\theta})\,\mathrm{d}\tilde{\boldsymbol{\theta}}$$

where D is some data, L the likelihood and π the prior. $\tilde{\boldsymbol{\theta}}$ is any subset of $\boldsymbol{\theta}$. As discussed above $\tilde{\boldsymbol{\theta}}$ usually includes all nuisance parameters $\boldsymbol{\theta}_\mathrm{n}$ and usually all physical parameters apart from one or two to display the marginal distribution of each pair of parameters graphically. This means that we can perform the integration in two steps by noticing that

$$I = \iint L(D|\boldsymbol{\theta})\pi(\boldsymbol{\theta})\,\mathrm{d}\tilde{\boldsymbol{\theta}}_\mathrm{p}\,\mathrm{d}\boldsymbol{\theta}_\mathrm{n} \tag{5.1}$$

where $\tilde{\boldsymbol{\theta}}_\mathrm{p} \subseteq \boldsymbol{\theta}_\mathrm{p}$. If the prior is separable (i.e. $\pi(\boldsymbol{\theta}) = \pi(\boldsymbol{\theta}_\mathrm{p}) \cdot \pi(\boldsymbol{\theta}_\mathrm{n})$), which is given in most cases, we can simplify this integral further:

$$I = \iint L(D|\boldsymbol{\theta})\pi(\boldsymbol{\theta}_\mathrm{p})\,\mathrm{d}\tilde{\boldsymbol{\theta}}_\mathrm{p}\pi(\boldsymbol{\theta}_\mathrm{n})\,\mathrm{d}\boldsymbol{\theta}_\mathrm{n} \tag{5.2}$$

The outer integral can be computed using nested sampling and the inner integral with Bayesian quadrature. This way we can exploit the advantages of both methods. This method is illustrated in two dimensions in Fig. 5.2.

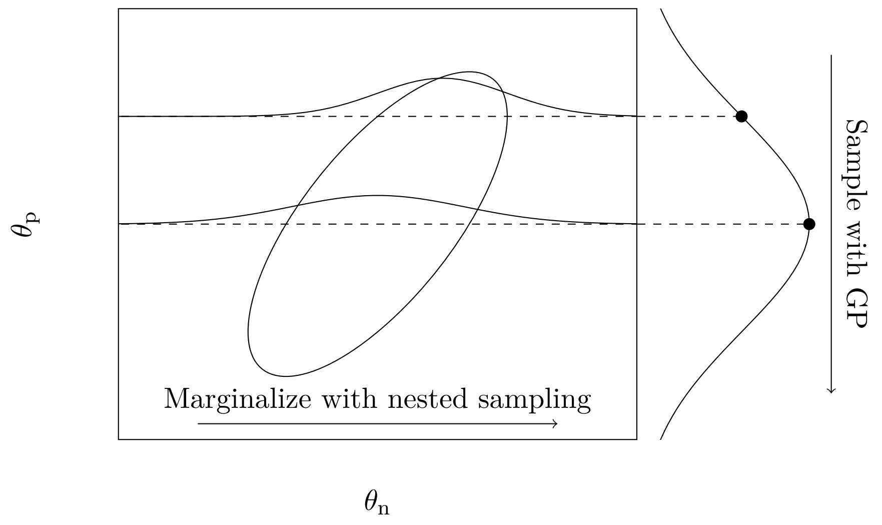

Figure 5.2: Graphical illustration of the principle of the hybrid GP/nested sampling method with one physical and one nuisance parameter. The ellipse represents the one σ contour of the posterior distribution. For every θ_p that is sampled by the active sampling algorithm, the posterior distribution is marginalized along θ_n with nested sampling.

In particular we can fully utilize the algorithm that was developed in chapter 4 for performing the bayesian quadrature part of this while the nested sampling can be performed with `PolyChord` which conveniently also gives us an estimate for the variance of $\log(Z)$ where Z is posterior distribution marginalized over the nuisance parameters:

$$Z = \int L(D|\boldsymbol{\theta})\pi(\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{\theta}_\mathrm{n} \tag{5.3}$$

This variance can naturally be accommodated in our algorithm since the GP regressor can account for statistical noise. This means that the GP needs more training points and thus more evaluations of the posterior distribution are needed for convergence. It however does not generally limit the ability of our algorithm to correctly map the posterior if the estimate for the statistical noise that we get for $\log(Z)$ is correct. One issue which has to be addressed though is that subtracting the numerical noise in the acquisition function assumes i.i.d. gaussian noise which means that incorporating different noise levels at different locations in the parameter space is not supported. There are two possibilities to work around this:

1. We could interpolate the noise levels between different regions in the parameter space. This lets us correctly deal with the statistical noise in $\log(Z)$. A natural

choice for an interpolator here would be a GP however this almost doubles the amount of computational overhead of our algorithm. In addition to this the noise level estimation also has some statistical noise attached to it which makes the problem even harder. Therefore this is, while being the rigorous solution, not an efficient one.

2. The noise level on $\log(Z)$ that `PolyChord` returns essentially only depends on the number of live points (N in algorithm 1). If we keep this fixed for all evaluations we will receive very similar estimates for this noise level and treat it like a constant number by setting the constant variance σ_n^2 to the mean value of the variances[18].

Pseudocode for this method is shown in algorithm 6. The integration of the nuisance parameters $\boldsymbol{\theta}_\mathrm{n}$ with nested sampling (algorithm 1) is done every time the physical parameters $\boldsymbol{theta}_\mathrm{p}$ are changed when evaluating the posterior for a newly acquired sample. Furthermore while the Kriging believer algorithm can be used in this case for batch acquisition it generally parallelizes worse than `PolyChord` meaning that it is likely most efficient to acquire a single point per step and use parallel computation for integrating over the nuisance parameters.

```
Input: $\mathcal{GP}(0, k(\boldsymbol{\theta}_\mathrm{p}, \boldsymbol{\theta}'_\mathrm{p}|\xi))$ (GP regressor with kernel), $n_\mathrm{init}$
       (Number of initial samples), $a(\mu(\boldsymbol{\theta}_\mathrm{p}), \sigma(\boldsymbol{\theta}_\mathrm{p}))$ (acquisition
       function), $P(\boldsymbol{\theta})$ (log-posterior)
[1]  Randomly draw $n_\mathrm{init}$ initial phys. samples $\boldsymbol{\theta}_\mathrm{p,0}$
[2]  $\log(Z_0), \sigma_{\log(Z_0)} := \int P(\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{\theta}_n$          alg. 1
[3]  $\boldsymbol{\theta}_\mathrm{p,train} := \boldsymbol{\theta}_\mathrm{p,0}$
[4]  $y_\mathrm{train} := \log(Z_0)$
[5]  $\sigma_n^2 := \sigma^2_{\log(Z_0)}$
[6]  $\Sigma_n^2 = \mathrm{diag}(\overline{\sigma_n^2})$ (crude) or $\mathrm{diag}(\sigma^2_{n,i})$
[7]  for $N$ times do
[8]      Fit $\xi$ with $\boldsymbol{\theta}_\mathrm{p,train}, y_\mathrm{train}, \Sigma_n^2$          alg. 2
[9]      Acquire $m$ points $\boldsymbol{\theta}_\mathrm{p}$ with Kriging believer          alg. 5
[10]     $\log(Z), \sigma_{\log(Z)} = \int P(\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{\theta}_\mathrm{n}$          alg. 1
[11]     $y_\mathrm{train} = \{y_\mathrm{train}, \log(Z)\}$
[12]     $\sigma_n^2 = \{\sigma_n^2, \sigma^2_{\log(Z)}\}$
[13] end
[14] return GP regressor
```

Algorithm 6: Pseudocode of the method presented in section 5.2. The integration over the nuisance parameters $\boldsymbol{\theta}_\mathrm{n}$ is done using `PolyChord` at every location $\boldsymbol{\theta}_\mathrm{p}$ in the physical parameter space, that our algorithm proposes. While the Kriging believer can be used for batch acquisition it generally parallelizes worse than `PolyChord` which means that in practice it is usually best to not use batch acquisition but rather parallelize the integration over $\boldsymbol{\theta}_\mathrm{n}$.

[18] One could also argue that we should take the conservative approach and take the maximum of all noise levels or something like the 95% quantile. This could easily be done but if $\sigma_{\log(Z_0)}$ is approximately the same for all points this should only have a minor impact.

Again we can use MCMC or any other convenient algorithm to perform the integration in

5.3 Efficiency

One unfortunate drawback of this approach is that the nested sampling procedure needs to be performed for every posterior evaluation of the GP algorithm. This means that the total evaluation time for the algorithm is given by

$$t_{\text{tot}} \simeq n_{\text{GP}} \cdot n_{\text{NS}} \cdot t_{\text{fast}} + n_{\text{GP}} \cdot t_{\text{slow}} + t_{\text{overhead}} \tag{5.4}$$

where n_{GP} and n_{NS} are the number of evaluations required for the GP algorithm or nested sampling to converge respectively and t_{fast} and t_{slow} are the times it takes for evaluating the posterior when only a fast parameter is changed or when slow parameters are changed too. t_{overhead} is the computational overhead that the GP algorithm introduces[19]. We will be using `PolyChord` which is implemented in `Cobaya` to perform the nested sampling marginalization for which we know the scaling of n_{NS} with the number of dimensions from Fig. 2.3 which can be empirically estimated to be $n_{\text{NS}} = 5000 \cdot d^{2.5}$.

Additionally we can empirically estimate the number of evaluations that the MCMC algorithm needs which is $n_{\text{MCMC}} = 1000 \cdot 10^{1.7}$. However the MCMC algorithm that we use can also take advantage of a speed hierarchy by *dragging* the fast parameters [58]. This means that the slow parameters are not changed at every iteration. We account for this effect by conservatively assuming that the speed-up that this method provides scales proportionally to the ratio of the number fast dimensions d_{fast} to the total number of dimensions $d_{\text{slow}} + d_{\text{fast}}$:

$$n_{\text{MCMC,fast}} = n_{\text{MCMC}} \cdot \frac{d_{\text{fast}}}{d_{\text{slow}} + d_{\text{fast}}}\ , \qquad n_{\text{MCMC,slow}} = n_{\text{MCMC}} \cdot \frac{d_{\text{slow}}}{d_{\text{slow}} + d_{\text{fast}}} \tag{5.5}$$

The statistical noise, that the estimation of Z with `PolyChord` introduces means that our algorithm will need more evaluations to converge which we account for by conservatively estimating that this doubles the total number of training points in the GP required for convergence.

These estimates now allow us to evaluate the efficiency of our hybrid approach as a function of t_{slow} and the speed hierarchy $t_{\text{fast}}/t_{\text{slow}}$ for different combinations of d_{fast} and d_{slow} which is shown in Fig. 5.3. We are especially interested in cases where $d_{\text{slow}} \lesssim 10$ since the exponential scaling of the number of evaluations and of the computational overhead means that this is the maximum amount that is practical except for very slow likelihoods as illustrated in Fig. 5.1.

Furthermore the number of nuisance parameters is often greater than the number of physical dimensions hence we will investigate how the algorithm behaves when we increase the number of fast dimensions drastically. As visible in Fig. 5.3 there is a distinctive area in where our algorithm is faster than MCMC. This depends mostly on the speed hierarchy $t_{\text{fast}}/t_{\text{slow}}$ with a less pronounced dependence on t_{slow}. The former comes mostly from the big difference in the number of evaluations required for MCMC to converge

[19]We will henceforth neglect the computational overhead that that the nested sampling algorithm introduces.

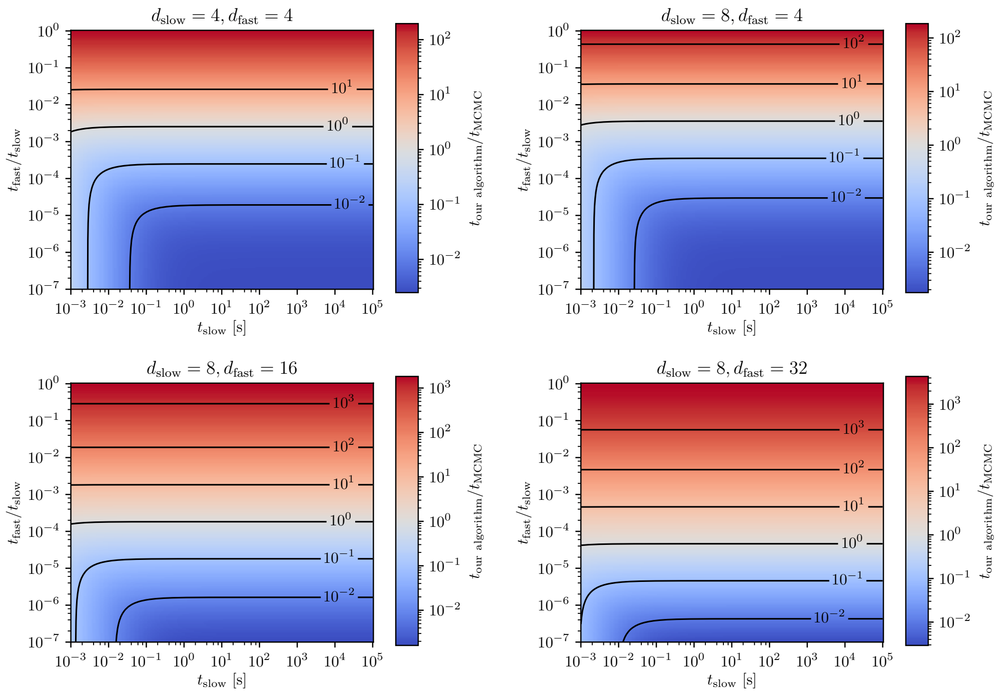

Figure 5.3: Ratio of (wall clock) evaluation times for our algorithm and MCMC for different combinations of d_{slow} and d_{fast} assuming that the algorithm is not parallelized as function of the slow evaluation time t_{slow} and the speed hierarchy $t_{\text{fast}}/t_{\text{slow}}$. Increasing the number of slow dimensions shifts the region where our algorithm is preferred over MCMC towards higher values of $t_{\text{fast}}/t_{\text{slow}}$ while increasing the number of fast dimensions pushes this region towards lower $t_{\text{fast}}/t_{\text{slow}}$. Increasing d_{slow} a value which is significantly higher than 10 is not feasible since the number of evaluations required for convergence will drastically increase the computational overhead. We can however observe that there clearly is a region where our algorithm can achieve much faster convergence than MCMC.

versus the GP approach while the latter comes from the computational overhead that our algorithm introduces.

The region where our algorithm is faster than MCMC does not change very much with the number of slow dimensions d_{slow} as visible in the first two plots of Fig. 5.3 however changing the number of fast dimensions has a large impact on the area where our algorithm wins, the reason for which is because for large d_{fast} the fist term in Eq. 5.4 dominates and n_{NS} scales worse than n_{MCMC} with the number of dimensions ($\propto d^{2.5}$ vs $\propto d^{1.7}$).

This shows that there is a range where this hybrid approach can extend the range of problems for which using `GPry` can be advantageous to using MCMC. Another problem which is solved by this approach is that the fraction of the prior hypervolume that is occupied by a non-infinite log-likelihood shrinks with e^{-d}. This is a problem for the GP algorithm which has been discussed in detail in section 4.7. `PolyChord` however can deal with $-\infty$ in the log-likelihood allowing us to keep the number of dimensions of the GP low enough so that this is not an issue.

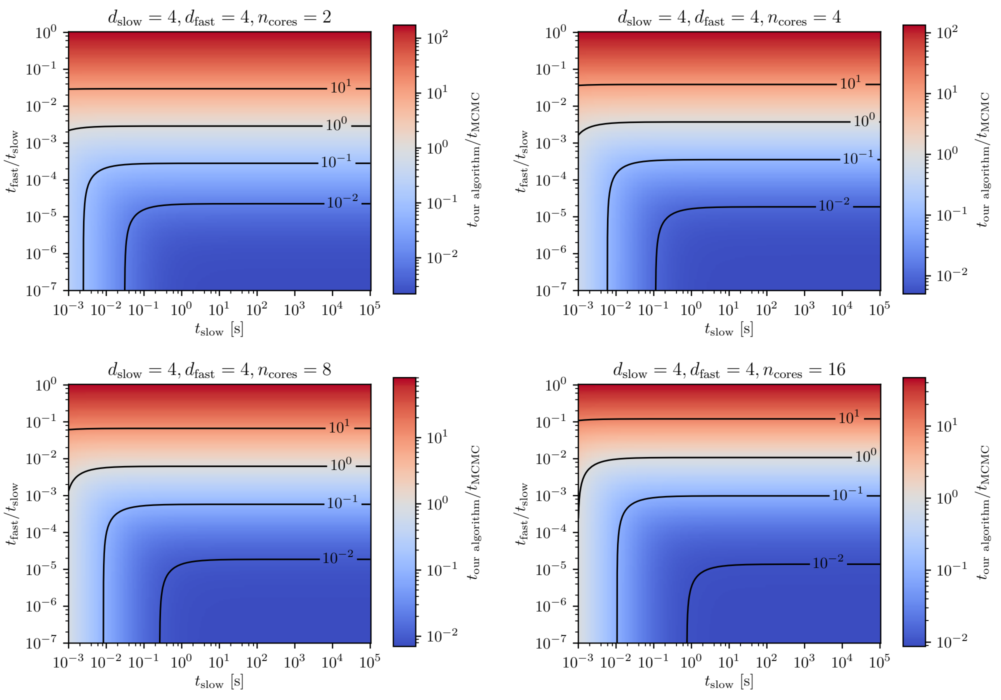

Figure 5.4: Ratio of evaluation times for our algorithm and MCMC for $d_{\text{slow}} = d_{\text{fast}} = 4$ with different numbers of parallel processes. Running more parallel processes shifts the area where our algorithm is preferred to higher values of $t$slow$/t_{\text{fast}}$ but at the same time the computational overhead of BQ, which does not decrease by parallelizing, becomes more pronounced.

These test assume however, that there is only one single core available for computation which does not take advantage of the very good parallel performance of `PolyChord` which parallelizes almost perfectly (actually it parallelizes proportional to $n_{\text{cores}} - 1$ because one core stays idle when running `PolyChord` in parallel) while the parallel performance of MCMC is much worse. This changes the equations in our favour.

An illustration of this effect is given in Fig. 5.4 for $d_{\text{slow}} = d_{\text{fast}} = 4$. Having more cores also makes the MCMC chains converge in less iterations per core which does not happen for the GP algorithm. This makes our algorithm less efficient than MCMC towards low t_{slow} however the better parallelization of `PolyChord` also pushes the equilibrium (10^0) line towards higher values for $t_{\text{fast}}/t_{\text{slow}}$. This means that parallelization works in favour of our algorithm if t_{slow} is sufficiently high which is reflected in the plot.

5.4 Experiments

What is left to do for us is to actually try our approach on some posterior distribution. For this we are using an artificial multivariate gaussian distribution with a randomly drawn mean vector and covariance matrix in 8 dimensions of which we will treat 4 as "nuisance dimensions" and marginalize over them (so in total we have $d_{\text{fast}} = d_{\text{slow}} = 4$).

The resulting 1σ and 2σ contours as well as the marginalized $1d$ distributions are shown

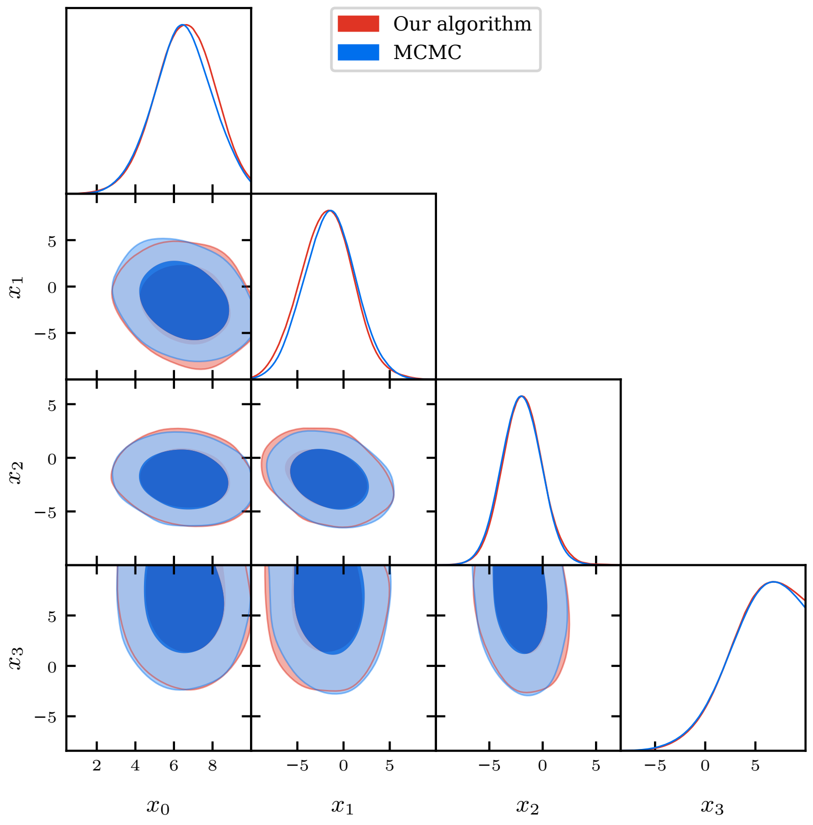

Figure 5.5: 8 dimensional toy examples ($d_{\text{slow}} = d_{\text{fast}} = 4$) where 4 dimensions are marginalized over with our algorithm and MCMC. Our algorithm correctly recovers the posterior shapes and converges with only 35 evaluations in the slow parameter space as opposed to $\mathcal{O}(10^4)$ slow evaluations for MCMC. This comes at the expense of having to evaluate the posterior at more locations in the fast dimensions which can be viable if the evaluation in the fast dimensions is computationally much cheaper than evaluations in the slow dimensions.

in in Fig. 5.5 where we compare the the results we would get when running an MCMC chain on the same posterior distribution. The MCMC algorithm needed $\approx 6 \cdot 10^4$ evaluations of the posterior which amounts to $\approx 3 \cdot 10^4$ slow evaluations. In contrast our algorithm needs a total of $\approx 10^6$ posterior evaluations but only 35 slow evaluations with an added computational overhead of ≈ 18 s. It is easy to see that if the slow evaluation time is for example 10 s and the fast one 10^{-4} s our algorithm will be far superior to MCMC since it converges in ~ 500 s ≈ 10 min compared to MCMC which would need $\sim 6 \cdot 10^5$ s ≈ 1 week to converge.

In addition one can see in Fig. 5.5 that the posterior shape is correctly recovered even though some of the mode is partly cut off by the prior bounds which shows that our algorithm has no problem handling these kinds of posteriors. Additionally the number of posterior evaluations in the slow dimensions seems to be consistent with our approximation which assumed that the added statistical noise roughly doubles the required number of posterior evaluations for convergence (see Fig. 4.7 for comparison).

Issues

While the algorithm developed in this section shows some great potential for making BQ a viable solution for performing Bayesian inference on a wide variety of different problems there are some known caveats that need to be addressed before this algorithm can be considered robust. These issues have to do with the posterior shape in the nuisance dimensions. For this imagine being close to the priors in the physical dimension while having a degeneracy between physical parameters and nuisance parameters.

In this scenario the posterior shape in the nuisance dimensions is pushed towards the edge of the prior box and only a tiny proportion of the prior volume in the nuisance dimensions is occupied by a non-vanishing posterior distribution. This effect is illustrated in Fig. 5.6 where one can see that the posterior contour shifts in the nuisance dimensions for different sampling locations in the physical dimensions. The effect in this example is very mild so `PolyChord` can easily deal with this, however with a higher number of nuisance dimensions and more pronounced degeneracies between the physical and nuisance dimensions this becomes problematic.

The problem is even worse when the sample in the physical dimensions is located at a point where the posterior in the nuisance dimensions vanishes everywhere. This is equivalent to having a $-\infty$ value in the log-posterior in the physical dimensions. Since `PolyChord` is not designed to deal with such cases it gets stuck in an infinite loop.

Solving these problems should however be a manageable task and with this our algorithm would be a robust and very efficient alternative to MCMC for a wide range of problems.

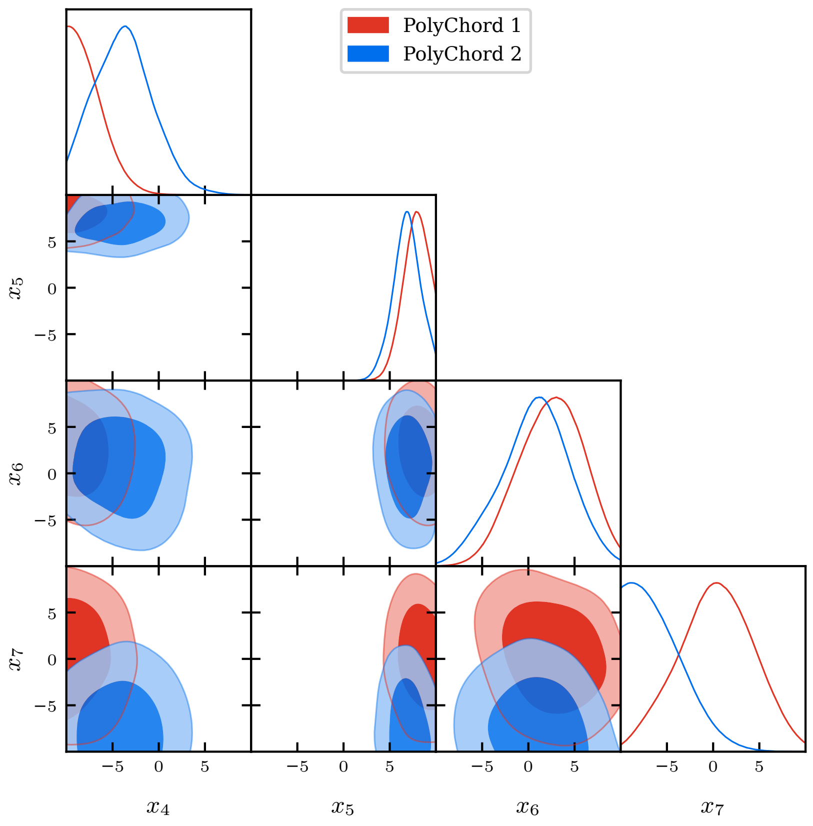

Figure 5.6: Illustration of how the posterior shape shifts in the nuisance dimensions for different sampling locations in the physical dimensions. In high dimensions and with large degeneracies between physical and nuisance parameters this can lead to the mode moving such that it only occupies a small strip along the edge of the prior. This makes it very hard for `PolyChord` to navigate such a space.

6 Conclusion & Outlook

In this thesis we developed a set of algorithms which can be used for performing Bayesian inference by interpolating the posterior distribution with a GP regressor where we improved on previous work in three ways.

First we derived an acquisition function for efficient sampling of the log-posterior distribution for Bayesian quadrature from a few basic assumptions. This was successfully used in our algorithm and proved to provide good performance.

Second we introduces an improved version of the Kriging believer algorithm for batch acquisition which saves computation time by using the blockwise matrix inversion formula to update the inverse gram matrix instead of fully recomputing it. We showed that this reduces the computational overhead per acquired point considerably and that it is numerically robust. As such it can be used to parallelize the evaluation of the posterior distribution.

Lastly we proposed a novel algorithm which takes advantage of the inherent speed hierarchies of many likelihoods in physics by marginalizing over the posterior's nuisance parameters with nested sampling while using a GP to interpolate the remaining dimensions. We show that this can extend the number of dimensions in which the GP can efficiently operate by a considerable amount and that it can speed up overall computation time.

We developed an algorithm that incorporates the first two of these improvements and tested its performance when performing Bayesian inference on unimodal gaussian likelihood toy models as well as two real likelihoods from cosmology. With the six dimensional Planck Lite likelihood we report a decrease of $\mathcal{O}(10^2)$ in wall clock time for convergence which effectively reduces the computation time required for characterizing the posterior from $\mathcal{O}$(hours) required on a cluster to $\mathcal{O}$(minutes) on a laptop. We also tested our model on some highly non-gaussian posterior shapes which it could recover correctly.

Next we tested an algorithm which incorporates all three of these novel ideas on gaussian toy likelihoods as a proof of concept. We were able to successfully apply this algorithm to gaussian toy likelihoods and were also able to assess for which posterior distributions it is superior to MCMC in terms of convergence speed. Nevertheless there are some issues with this approach that still need to be addressed in the future.

Despite these minor issues our proof of concept shows that this algorithm can outperform MCMC by several orders of magnitude in terms of wall clock time for a wide range of likelihoods which significantly reduces the time and computational power required to infer parameters from these models. This will provide exciting opportunities for testing theoretical models against data which have previously been out of reach due to the computation time which would be required to perform Bayesian inference of these models with classical approaches like MCMC.

Appendix

The Cholesky decomposition

Let $A \in M$ be matrix. If and only if A is positive semidefinite it can be decomposed into a product of a lower triangular matrix $L \in M$ with non-negative diagonal elements and its conjugate transpose:

$$A = LL^* \tag{.1}$$

If A is positive definite L is unique [59].

The Cholesky decomposition can be used to solve linear equations of the form

$$A\boldsymbol{x} = \boldsymbol{b} \tag{.2}$$

To solve for $\boldsymbol{x}$ one first solves the system $L\boldsymbol{z} = \boldsymbol{b}$ by forward substitution and then $L^T\boldsymbol{x} = \boldsymbol{z}$ by backward substitution [19]. This can be written as $x = L^T\backslash(L\backslash\boldsymbol{b})$

In algorithm2 $(K + \sigma_n^2 I)^{-1}\boldsymbol{y}$ is needed to solve eq. (3.14) and (3.22) which can be done by replacing $A = (K + \sigma_n^2 I)$, $\boldsymbol{x} = (K + \sigma_n^2 I)^{-1}\boldsymbol{y}$ and $\boldsymbol{b} = y$:

$$(K + \sigma_n^2 I)^{-1}\boldsymbol{y} = L^T\backslash(L\backslash\boldsymbol{y}) \tag{.3}$$

The computational complexity for the Cholesky decomposition is $\mathcal{O}(n^3)$ and for foreward and backwards substitution $\mathcal{O}(n^2)$. Furthermore this algorithm is numerically very robust [19].

For the second expression appearing in eq. (3.14) one needs to compute

$$\boldsymbol{v} := L^{-1}K(\boldsymbol{x}, \boldsymbol{x}_*) = L\backslash K(\boldsymbol{x}, \boldsymbol{x}_*) \tag{.4}$$

since

$$\begin{aligned} \boldsymbol{v}^T\boldsymbol{v} &= K(\boldsymbol{x}_*, \boldsymbol{x})(L^{-1})^T(L^{-1})K(\boldsymbol{x}, \boldsymbol{x}_*) && (.5)\\ &= K(\boldsymbol{x}_*, \boldsymbol{x})(K + \sigma_n^2 I)^{-1}K(\boldsymbol{x}, \boldsymbol{x}_*) && (.6) \end{aligned}$$

Lastly the determinant of A can be computed very easily if L is known:

$$|A| = \prod_{i=1}^{n} L_{ii}^2 \tag{.7}$$

Derivation of the marginalized GP

The goal is to derive the marginal distribution for a GP:

$$\log p(\boldsymbol{y}|X) = -\frac{1}{2}\boldsymbol{y}^T(K+\sigma_n^2 I)^{-1}\boldsymbol{y} - \frac{1}{2}\log|K+\sigma_n^2 I| - \frac{n}{2}\log 2\pi$$

We start by taking the general expression for the marginal distribution

$$p(\boldsymbol{y}|\boldsymbol{x}) = \int p(\boldsymbol{y}|\boldsymbol{f},\boldsymbol{x})p(\boldsymbol{f}|\boldsymbol{x})\,\mathrm{d}f$$

and inserting the expressions for the prior and posterior which are

$$p(\boldsymbol{f}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{0},K) = (2\pi)^{-n/2}|K|^{-1/2}\exp\left(-\boldsymbol{f}^T K^{-1}\boldsymbol{f}\right)$$

and

$$p(\boldsymbol{y}|\boldsymbol{f},\boldsymbol{x}) = \mathcal{N}(\boldsymbol{f},\sigma_n^2 I) = (2\pi)^{-n/2}|\sigma_n^2 I|^{-1/2}\exp\left(-(\boldsymbol{y}-\boldsymbol{f})^T(\sigma_n^2 I)^{-1}(\boldsymbol{y}-\boldsymbol{f})\right)\ .$$

Their product can then be calculated using [19]

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{a},A)\mathcal{N}(\boldsymbol{x}|\boldsymbol{b},B) = Z^{-1}\mathcal{N}(\boldsymbol{x}|\boldsymbol{c},C)\ .$$

Since the second term integrates out to one we only need to know Z^{-1} which is

$$Z^{-1} = (2\pi)^{d/2}|A+B|^{-1}\exp\left((\boldsymbol{a}-\boldsymbol{b})^T(A+B)^{-1}(\boldsymbol{a}-\boldsymbol{b})\right)\ .$$

and with $\boldsymbol{a}=\boldsymbol{0},\ A=K,\ \boldsymbol{b}=-\boldsymbol{y},\ B=\sigma_n^2 I$ we get

$$p(\boldsymbol{y}|X) = (2\pi)^{d/2}|K+\sigma_n^2 I|^{-1}\exp\left(\boldsymbol{y}^T(K+\sigma_n^2 I)^{-1}\boldsymbol{y}\right)\ .$$

By taking the logarithm we arrive at the desired result.

Proof that the Kriging believer algorithm conserves μ_{GP}

The idea is to prove that the expectation value of the GP does not change if we lie to the model by pretending that $y_0(x_1) = \mu_0(x_1)$.
First start by predicting $\mu_0(x_1)$ for given training points x_0 and inference points x_1. The formula for $\mu_0(x_1)$ (the prediction) is given by:

$$\mu_0(x_1) = K(x_1,x_0)K(x_0,x_0)^{-1}y_0$$

where $K(x,x')$ is the covariance matrix resulting from the covariance function $k(x,x')$. The next step is to assume $y_0(x_1) = \mu_0(x_1)$ and then make a new prediction at a new place x_2. The mean at this point will be given by:

$$\begin{aligned}\mu_1(x_2) &= (K_{20}, K_{21}) \begin{pmatrix} K_{00} & K_{01} \\ K_{10} & K_{11} \end{pmatrix}^{1} \begin{pmatrix} y_0 \\ \mu_0(x_1) \end{pmatrix} \\ &= (K_{20}, K_{21}) \begin{pmatrix} K_{00} & K_{01} \\ K_{10} & K_{11} \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ K_{10}K_{00}^{-1} \end{pmatrix} y_0\end{aligned}$$

where $K(x_i, x_j) := K_{ij}$. Now we can use the blockwise inversion lemma to get:

$$\begin{aligned}\mu_1(x_2) &= (K_{20}, K_{21}) \begin{pmatrix} K_{00}^{-1} + K_{00}^{-1}K_{01}SK_{10}K_{00}^{-1} & -K_{00}^{-1}K_{01}S \\ SK_{10}K_{00}^{-1} & S \end{pmatrix} \begin{pmatrix} 1 \\ K_{10}K_{00}^{-1} \end{pmatrix} y_0 \\ &= (K_{20}, K_{21}) K_{00}^{-1} + K_{00}^{-1}K_{01}SK_{10}K_{00}^{-1} - K_{00}^{-1}K_{01}SK_{10}K_{00}^{-1}K_{00}^{-1}K_{01}SK_{10}K_{00}^{-1} - K_{00}^{-1}K_{01}SK_{10}K_{00}^{-1}y_0 \\ &= K_{20}K_{00}^{-1}y_0 \\ &= \mu_0(x_2)\end{aligned}$$

where $S = (K_{11} - K_{01}K_{00}^{-1}K_{10})^{-1}$. This proves that indeed if we lie to the model by giving it its own mean we do not need change the hyperparameters of the GP. As such we can use the matrix inversion lemma to include this point into the GP without having to do the expensive recalculation of the full inverse.

References

[1] Michael Osborne et al. "Active Learning of Model Evidence Using Bayesian Quadrature". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012, pp. 46–54. URL: `https://proceedings.neurips.cc/paper/2012/file/6364d3f0f495b6ab9dcf8d3b5c6e0b01-Paper.pdf`.

[2] Marcos Pellejero-Ibañez et al. "Cosmological parameter estimation via iterative emulation of likelihoods". In: *Monthly Notices of the Royal Astronomical Society* 499.4 (Oct. 2020), pp. 5257–5268. ISSN: 0035-8711. DOI: `10.1093/mnras/staa3075`.

[3] T. Gunter et al. "Sampling for Inference in Probabilistic Models with Fast Bayesian Quadrature". In: *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., 2014, pp. 2789–2797. URL: `http://papers.nips.cc/paper/5483-sampling-for-inference-in-probabilistic-models-with-fast-bayesian-quadrature.pdf`.

[4] David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. *A Multi-points Criterion for Deterministic Parallel Global Optimization based on Gaussian Processes*. Tech. rep. Mar. 2008. URL: `https://hal.archives-ouvertes.fr/hal-00260579`.

[5] R. J. Barnes and A. G. Watson. "Efficient updating of kriging estimates and variances". In: *Mathematical Geology* 24.1 (Jan. 1992), pp. 129–133. ISSN: 1573-8868. DOI: `10.1007/BF00890091`.

[6] John Skilling. "Nested Sampling". In: *AIP Conference Proceedings* 735.1 (2004), pp. 395–405. DOI: `10.1063/1.1835238`.

[7] W. J. Handley, M. P. Hobson, and A. N. Lasenby. "polychord: nested sampling for cosmology". In: *Monthly Notices of the Royal Astronomical Society: Letters* 450.1 (Apr. 2015), pp. L61–L65. ISSN: 1745-3925. DOI: `10.1093/mnrasl/slv047`.

[8] Glen Cowan. *Statistical Data Analysis*. Oxford: Clarendon Press, 1998. ISBN: 978-0-198-50155-8.

[9] Kevin P. Murphy. *Machine Learning - A Probabilistic Perspective*. Cambridge: MIT Press, 2012. ISBN: 978-0-262-01802-9.

[10] Andrew Gelman et al. *Bayesian Data Analysis*. 3rd ed. Boca Raton, Fla: CRC Press, 2013. ISBN: 978-1-439-84095-5.

[11] N. Aghanim et al. "Planck 2018 results". In: *Astronomy & Astrophysics* 641 (Sept. 2020), A5. ISSN: 1432-0746. DOI: `10.1051/0004-6361/201936386`.

[12] Radford Neal. "MCMC Using Hamiltonian Dynamics". In: *Handbook of Markov Chain Monte Carlo*. Ed. by Steve Brooks et al. CRC Press, 2011. Chap. 5, pp. 113–162. ISBN: 978-1-420-07942-5. DOI: `10.1201/b10905`.

[13] W. J. Handley, M. P. Hobson, and A. N. Lasenby. "polychord: next-generation nested sampling". In: *Monthly Notices of the Royal Astronomical Society* 453.4 (Sept. 2015), pp. 4385–4399. ISSN: 1365-2966. DOI: `10.1093/mnras/stv1911`.

[14] F. Feroz and M. P. Hobson. "Multimodal nested sampling: an efficient and robust alternative to Markov Chain Monte Carlo methods for astronomical data analyses". In: *Monthly Notices of the Royal Astronomical Society* 384.2 (Jan. 2008), pp. 449–463. ISSN: 1365-2966. DOI: `10.1111/j.1365-2966.2007.12353.x`.

[15] F. Feroz, M. P. Hobson, and M. Bridges. “MultiNest: an efficient and robust Bayesian inference tool for cosmology and particle physics”. In: *Monthly Notices of the Royal Astronomical Society* 398.4 (Oct. 2009), pp. 1601–1614. ISSN: 1365-2966. DOI: `10.1111/j.1365-2966.2009.14548.x`.

[16] Farhan Feroz et al. “Importance Nested Sampling and the MultiNest Algorithm”. In: *The Open Journal of Astrophysics* 2.1 (Nov. 2019). ISSN: 2565-6120. DOI: `10.21105/astro.1306.2144`.

[17] Antony Lewis. “Efficient sampling of fast and slow cosmological parameters”. In: *Phys. Rev.* D87.10 (2013), p. 103529. DOI: `10.1103/PhysRevD.87.103529`. arXiv: `1304.4473 [astro-ph.CO]`.

[18] Antony Lewis and Sarah Bridle. “Cosmological parameters from CMB and other data: A Monte Carlo approach”. In: *Phys. Rev.* D66 (2002), p. 103511. DOI: `10.1103/PhysRevD.66.103511`. arXiv: `astro-ph/0205436 [astro-ph]`.

[19] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning.* eng. Adaptive computation and machine learning. Cambridge, Mass. [u.a.]: MIT Press, 2006, XVIII, 248 S. ISBN: 0-262-18253-X and 978-0-262-18253-9.

[20] J. Lamperti. *Stochastic Processes - A Survey of the Mathematical Theory.* Berlin Heidelberg: Springer Science & Business Media, 2012. ISBN: 978-1-468-49358-0.

[21] Bernt Øksendal. *Stochastic Differential Equations - An Introduction with Applications.* Berlin Heidelberg: Springer Science & Business Media, 2010. ISBN: 978-3-642-14394-6.

[22] Noel A. C. Cressie. “Geostatistics”. In: *Statistics for Spatial Data.* John Wiley & Sons, Ltd, 2015. Chap. 2, pp. 27–104. ISBN: 978-1-119-11515-1. DOI: `10.1002/9781119115151.ch2`.

[23] David Duvenaud. “Automatic Model Construction with Gaussian Processes”. PhD thesis. Cambridge, 2014.

[24] Andrew Gordon Wilson and Ryan Prescott Adams. *Gaussian Process Kernels for Pattern Discovery and Extrapolation.* 2013. arXiv: `1302.4245 [stat.ML]`.

[25] Christian Steinruecken et al. “The Automatic Statistician”. In: *Automated Machine Learning: Methods, Systems, Challenges.* Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Cham: Springer International Publishing, 2019, pp. 161–173. ISBN: 978-3-030-05318-5. DOI: `10.1007/978-3-030-05318-5_9`.

[26] Tom Minka. *Deriving Quadrature Rules from Gaussian Processes.* Tech. rep. Cambridge, UK: Microsoft Research, June 2000. URL: `https://www.microsoft.com/en-us/rminka2000esearch/publication/deriving-quadrature-rules-gaussian-processes/`.

[27] Donald R. Jones. “A Taxonomy of Global Optimization Methods Based on Response Surfaces”. In: *Journal of Global Optimization* 21.4 (Dec. 2001), pp. 345–383. ISSN: 1573-2916. DOI: `10.1023/A:1012771025575`.

[28] Lehel Csató. “Gaussian processes:iterative sparse approximations”. Aston University, Mar. 2002. URL: `http://publications.aston.ac.uk/id/eprint/1327/`.

[29] Alex J Smola and Peter L Bartlett. “Sparse greedy Gaussian process regression”. In: *Advances in neural information processing systems 14*. 2001, pp. 619–625. ISBN: 978-0-262-04208-6.

[30] Michalis Titsias. “Variational Learning of Inducing Variables in Sparse Gaussian Processes”. In: *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*. Ed. by David van Dyk and Max Welling. Vol. 5. Proceedings of Machine Learning Research. Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA: PMLR, 16–18 Apr 2009, pp. 567–574. URL: http://proceedings.mlr.press/v5/titsias09a.html.

[31] Dustin Tran, Rajesh Ranganath, and David M. Blei. *The Variational Gaussian Process*. 2016. arXiv: 1511.06499 [stat.ML].

[32] Carl Edward Rasmussen and Zoubin Ghahramani. “Bayesian Monte Carlo”. In: *Proceedings of the 15th International Conference on Neural Information Processing Systems*. NIPS’02. Cambridge, MA, USA: MIT Press, 2002, pp. 505–512.

[33] A. O’Hagan. “Monte Carlo is Fundamentally Unsound”. In: *Journal of the Royal Statistical Society: Series D (The Statistician)* 36.2-3 (1987), pp. 247–249. DOI: https://doi.org/10.2307/2348519.

[34] Morton Kupperman. “Probabilities of Hypotheses and Information-Statistics in Sampling from Exponential-Class Populations”. In: *Ann. Math. Statist.* 29.2 (June 1958), pp. 571–575. DOI: 10.1214/aoms/1177706633.

[35] Agnan Kessy, Alex Lewin, and Korbinian Strimmer. “Optimal Whitening and Decorrelation”. In: *The American Statistician* 72.4 (Jan. 2018), pp. 309–314. ISSN: 1537-2731. DOI: 10.1080/00031305.2016.1277159.

[36] Thomas Desautels, Andreas Krause, and Joel W. Burdick. “Parallelizing Exploration-Exploitation Tradeoffs in Gaussian Process Bandit Optimization”. In: *Journal of Machine Learning Research* 15.119 (2014), pp. 4053–4103. URL: http://jmlr.org/papers/v15/desautels14a.html.

[37] Clément Chevalier and David Ginsbourger. “Fast Computation of the Multi-Points Expected Improvement with Applications in Batch Selection”. In: *Learning and Intelligent Optimization*. Ed. by Giuseppe Nicosia and Panos Pardalos. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 59–69. ISBN: 978-3-642-44973-4.

[38] J. González et al. “Batch Bayesian Optimization via Local Penalization”. In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS)*. Vol. 51. JMLR Workshop and Conference Proceedings. May 2016, pp. 648–657. URL: http://jmlr.org/proceedings/papers/v51/gonzalez16a.pdf.

[39] David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. “Kriging Is Well-Suited to Parallelize Optimization”. In: vol. 2. Jan. 2010, pp. 131–162. DOI: 10.1007/978-3-642-10701-6_6.

[40] Mark Gibbs and David J.C. MacKay. *Efficient Implementation of Gaussian Processes*. Tech. rep. Cavendish Laboratory, Cambridge UK, 1997.

[41] Yanan Sui et al. “Stagewise Safe Bayesian Optimization with Gaussian Processes”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, Oct. 2018, pp. 4781–4789. URL: `http://proceedings.mlr.press/v80/sui18a.html`.

[42] Felix Berkenkamp, Andreas Krause, and Angela P. Schoellig. “Bayesian Optimization with Safety Constraints: Safe and Automatic Parameter Tuning in Robotics”. In: *CoRR* abs/1602.04450 (2016). arXiv: `1602.04450`.

[43] Nils Schöneberg, Julien Lesgourgues, and Deanna C. Hooper. “The BAO+BBN take on the Hubble tension”. In: *Journal of Cosmology and Astroparticle Physics* 2019.10 (Oct. 2019), pp. 029–029. ISSN: 1475-7516. DOI: `10.1088/1475-7516/2019/10/029`.

[44] N. Aghanim et al. “Planck 2018 results. V. CMB power spectra and likelihoods”. In: (2019). arXiv: `1907.12875 [astro-ph.CO]`.

[45] N. Aghanim et al. “Planck 2018 results. VIII. Gravitational lensing”. In: (2018). arXiv: `1807.06210 [astro-ph.CO]`.

[46] Diego Blas, Julien Lesgourgues, and Thomas Tram. “The Cosmic Linear Anisotropy Solving System (CLASS) II: Approximation schemes”. In: *JCAP* 1107 (2011), p. 034. DOI: `10.1088/1475-7516/2011/07/034`. arXiv: `1104.2933 [astro-ph.CO]`.

[47] Planck Collaboration et al. “Planck 2018 results - VI. Cosmological parameters”. In: *Astronomy & Astrophysics* 641 (2020), A6. DOI: `10.1051/0004-6361/201833910`.

[48] Antony Lewis. “GetDist: a Python package for analysing Monte Carlo samples”. In: (2019). arXiv: `1910.13970 [astro-ph.IM]`.

[49] Erik Aver, Keith A. Olive, and Evan D. Skillman. “The effects of He I λ10830 on helium abundance determinations”. In: *Journal of Cosmology and Astroparticle Physics* 2015.07 (July 2015), pp. 011–011. ISSN: 1475-7516. DOI: `10.1088/1475-7516/2015/07/011`.

[50] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: `10.1038/s41586-020-2649-2`.

[51] Fabian Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12.85 (2011), pp. 2825–2830. URL: `http://jmlr.org/papers/v12/pedregosa11a.html`.

[52] Tim Head et al. *scikit-optimize/scikit-optimize: v0.5.2*. Version v0.5.2. Mar. 2018. DOI: `10.5281/zenodo.1207017`.

[53] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: `10.1038/s41592-019-0686-2`.

[54] Richard H. Byrd et al. “A Limited Memory Algorithm for Bound Constrained Optimization”. In: *SIAM Journal on Scientific Computing* 16.5 (1995), pp. 1190–1208. DOI: `10.1137/0916069`.

[55] Ciyou Zhu et al. “Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization”. In: *ACM Trans. Math. Softw.* 23.4 (Dec. 1997), pp. 550–560. ISSN: 0098-3500. DOI: `10.1145/279232.279236`.

[56] Tim Sprenger et al. “Cosmology in the era of Euclid and the Square Kilometre Array”. In: *Journal of Cosmology and Astroparticle Physics* 2019.02 (Feb. 2019), pp. 047–047. ISSN: 1475-7516. DOI: `10.1088/1475-7516/2019/02/047`.

[57] Jan-Hendrik Prinz et al. “Markov models of molecular kinetics: Generation and validation”. In: *The Journal of Chemical Physics* 134.17 (2011), p. 174105. DOI: `10.1063/1.3565032`.

[58] R. M. Neal. “Taking Bigger Metropolis Steps by Dragging Fast Variables”. In: *ArXiv Mathematics e-prints* (Feb. 2005). eprint: `math/0502099`.

[59] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. 2nd ed. Cambridge: Cambridge University Press, 2013. ISBN: 978-1-139-78888-5.

Acknowledgements

First of all I would like to thank my main supervisor Jesús Torrado for countless hours of answering questions, discussing, correcting and helping me at any day and any time. It is hard to overstate the amount of dedication he has put towards this project and I am very grateful for this. I would also like to thank my official supervisor Julien Lesgourgues as well as my two other inofficial supervisors Nils Schöneberg and Christian Fidler for being very patient, helpful and interested. We had many insightful and enlightening discussions and this was a great year for me, both academically and personally.
I would also like to thank Felix Kahlhoefer for taking on the task of being my second corrector.
Despite the special circumstances which have unfortunately severely limited the time I could physically spend at the institute I want to thank the whole TTK for becoming friends to me. It was a great time which I will look back to in joy and I will sincerely miss you all. It has has been a great pleasure to share my office with my fellow master students and I would like to thank them for a lot of good conversations, laughs and advices. Last but not least I would like to thank my partner Anne who has helped me throughout the last, very stressful weeks of this year with a lot of emotional support.

Zentrales Prüfungsamt/Central Examination Office

RWTHAACHEN UNIVERSITY

Eidesstattliche Versicherung
Statutory Declaration in Lieu of an Oath

El Gammal, Jonas Elias	355338
Name, Vorname/Last Name, First Name	Matrikelnummer (freiwillige Angabe) Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende ~~Arbeit/Bachelorarbeit~~/ Masterarbeit* mit dem Titel
I hereby declare in lieu of an oath that I have completed the present ~~paper/Bachelor thesis~~/Master thesis* entitled

Accelerating Bayesian Inference of expensive Likelihoods with Gaussian Processes

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Aachen, den 16.12.2020	
Ort, Datum/City, Date	Unterschrift/Signature

*Nichtzutreffendes bitte streichen
*Please delete as appropriate

Belehrung:
Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.
Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.
§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.
Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Aachen, den 16.12.2020	
Ort, Datum/City, Date	Unterschrift/Signature