FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Scalable Gaussian processes for surrogate modelling in Bayesian optimization
Student:	Bc. Iveta Šárfyová
Supervisor:	Ing. Jiří Vošmik
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

Bayesian optimization is a global black-box optimization method suitable for expensive-to-evaluate objective functions. Gaussian processes (GP) are commonly used as a surrogate model in a Bayesian optimization framework, where their probabilistic nature and flexibility are highly desirable, but their cubic training time complexity limits their deployment to small datasets.

The goal of this thesis is to research scalable Gaussian process architectures and evaluate their usability in Bayesian optimization tasks with large datasets.
1) Conduct a survey of the state-of-the-art scalable GPs for regression.
2) Experimentally evaluate the predictive performance and computational complexity of selected scalable GP architectures trained on several publicly available datasets.
3) Experimentally evaluate the performance of selected GP architectures in the context of simulated Bayesian optimization.

Electronically approved by Ing. Karel Klouda, Ph.D. on 3 December 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Scalable Gaussian Processes for surrogate modelling in Bayesian optimization

Bc. Iveta Šárfyová

Department of Applied Mathematics
Supervisor: Ing. Jiří Vošmik

December 21, 2022

Acknowledgements

My sincere appreciation goes out to Ing. Jiří Vošmik for being an incredible teacher, mentor and friend. Your knowledge, enthusiasm and patience have been a great source of motivation throughout this journey. I'm grateful for the guidance and advice you gave me.

Also, I want to thank Tomáš for his unwavering support and encouragement every step of the way. I appreciate you believing in me, even when I struggled. I'm thankful that you pushed me, made me smile and helped me get through challenges when I needed it.

Furthermore, my heartfelt thanks to my family, friends and colleagues. You bring so much joy and laughter to my life. Your unique personalities and senses of humour never fail to bring a smile to my face. More importantly, you make my life brighter and more colourful, even during cloudy days. I cannot thank you enough.

Special thanks go to all the lecturers and friends participating in Gaussian Process and Uncertainty Quantification Summer School. I value the effort put into preparing the presentations and all the insightful discussions that helped me to further grasp the topic.

Lastly, I would like to thank all the people in my life who have shared their wisdom and ignited my passion for learning. I'm grateful for all the invaluable lessons and knowledge you have passed on to me. Thanks for helping me grow!

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.
I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on December 21, 2022 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Šárfyová, Iveta. *Scalable Gaussian Processes for surrogate modelling in Bayesian optimization.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstrakt

Bayesovská optimalizace je globální optimalizační metoda vhodná pro hledání extrémů black-box účelových funkcí drahých na vyhodnocení. Jako modely pro aproximaci takových funkcí se často používají Gaussovské procesy. Jejich kubická časová složitost však omezuje jejich nasazení na aplikace v režimech s malým počtem dat. Tato práce poskytuje přehled moderních škálovatelných Gaussových procesů pro regresi. Experimenty provedené v rámci této práce se zabývají úlohami regrese a bayesovské optimalizace, přičemž v obou případech se využívá několik vybraných modelů založených na Gaussových procesech. Vyhodnocení se provádí pomocí více metrik, z nichž některé jsou zvláště vhodné pro pravděpodobnostní modely. Naše výsledky naznačují, že některé z modelů konzistentně překonávají ostatní v obou úkolech.

Klíčová slova gaussovské procesy, black-box optimalizace, bayesovská optimalizace, regrese, velké datasety

Abstract

Bayesian optimisation is a global optimisation method suitable for finding extrema of expensive-to-evaluate black-box objective functions. Gaussian Processes are frequently used as models for approximating such functions. However, their cubic time complexity limits their deployment to applications in small-data regimes. This thesis provides an overview of state-of-the-art scalable Gaussian Processes for regression. The experiments performed within this work deal with tasks of regression and Bayesian optimisation, both utilising several selected Gaussian Process models. Evaluation is done using multiple metrics, some of which are particularly appropriate for probabilistic models. Our results suggest that the same few models consistently outperform the others in both tasks.

Keywords gaussian processes, black-box optimisation, bayesian optimisation, regression, large datasets

Contents

List of Figures

List of Tables

Introduction

Optimisation of nontrivial functions has been vigorously studied for decades and is still a matter of interest and active research. In real-life scenarios, we often deal with problems including an almost or entirely unknown objective function. The sought optimum often represents a perfect design or parameter choice that will be later applied in some procedure, system or machine. Since the perfect setup is difficult to reach or come close to due to the nature of the function, we typically need to test out numerous possible input values. However, evaluating the function that typically corresponds to or describes some procedure can be expensive or even destructive.

For instance, tuning the software used daily by thousands of users inappropriately can lead to a loss of money and customers. Randomly selecting values for testing is particularly unwise because it might cause severe issues, such as hindering its functionality or leading to an unpleasant user experience. These outcomes might be very frustrating to users, which could discourage them from using the product in the future. In the worst case scenario, suffering from particular software malfunctions may lead to catastrophic consequences. In critical areas such as healthcare, such a defect or failure of medical devices could lead to losing users due to literally fatal outcomes.

Optimisation directly affecting living organisms requires special attention and diligent care due to moral and ethical questions. Many products from the pharmaceutical industry are tested on animals or people before being released to the general population and thus demand attentive oversight. Especially drugs or vaccines ought to be monitored and developed enough not to cause any side effects at that point. Conducting unnecessary experiments with pharmaceuticals composed of randomly chosen ingredients is considered unacceptable. Moreover, the number of possible combinations in the search space grows exponentially with the number of ingredients, which generally leads to an intractable exaggeration of the problem.

One of the renowned techniques for solving such complex tasks is Bayesian optimisation, which provides tools for estimating the optimised system or procedure and approximately simulating its evaluation. There are several models that are suitable to be used as an estimate of the function. One of the models that has gained popularity in recent years is the Gaussian Process model. Gaussian Processes are generally very powerful in terms of providing predictions and expressing uncertainty in these predictions but come short when working with large data. This limitation has led to extensive research aimed at developing scalable Gaussian Processes.

This work focuses on research and application of scalable Gaussian Processes in the tasks of regression and Bayesian optimisation with large datasets. The thesis is organised into five chapters. First, Chapter 1 describes the core concepts of Bayesian optimisation. Next, Chapter 2 discusses the fundamentals of Gaussian Process regression. A survey of the state-of-the-art scalable Gaussian Processes can be found in Chapter 3. The experimental part of the thesis is divided into two parts, both utilising several selected Gaussian Process models. The first part is dedicated to the regression task, which aims to evaluate and compare the models on multiple datasets based on their performance and complexity. The second part is devoted to evaluating the performance of the models in simulated Bayesian optimisation. The design of the experiments, evaluation, obtained results and the discussion of the results are presented in Chapter 4 for both of the tasks. Finally, the last chapter is devoted to the contribution of this thesis and possible future work directions.

Chapter 1

Bayesian Optimisation

Optimisation covers a mathematical field with a broad spectrum of applications ranging from academic to industrial domains. In general, optimisation is a process directed towards minimising or maximising an objective function f. The procedure of finding the location $\mathbf{x}^*$ of the function's optimum depends largely on the function's characteristics. There is an immense number of optimisation techniques that can be categorised according to various criteria, and the function's properties are one of them. One of the crucial distinctions is whether the function has a single or multiple optima. Another possibility of classifying the optimisation problems is based on the function's differentiability.

In the simplest case, we have the closed-form expression of the objective function, or we at least know it is convex or cheap to evaluate [1]. If the objective function is convex, finding any local maximum is equivalent to finding a global maximum. The same applies to searching for the minimum of concave functions. Convex and concave functions thus belong to a group of optimisation problems with a single optimum. For this scenario, we can employ gradient-based algorithms such as hill climbing algorithm, which is an iterative method well-suited for finding extrema in such functions. An example run of a gradient-based algorithm searching for global maximum is illustrated in Figure 1.1.

However, we might want to utilise different approaches for optimising over functions with more than one extrema since the hill climbing algorithm is generally known for being prone to getting stuck in local optima. Which one of the multiple optima is found depends on the part of the search space from which the algorithm starts. One possibility to alleviate this problem is to use hill climbing with restarts, where the algorithm repeatedly starts searching from various initial locations.

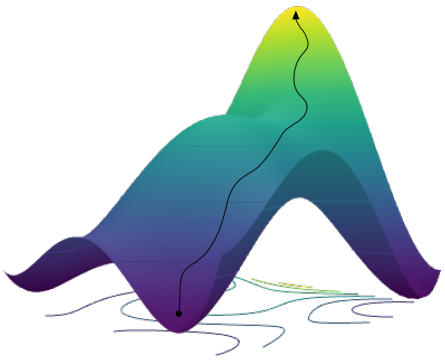


Figure 1.1: Illustration of gradient-based optimisation.

Another option for optimising a function is to use its differentiability. This approach works if we have access to the derivatives of the function or if we can estimate them. One way to find the optimal solution using the derivatives is by analytically solving for the maximum, minimum, or saddle points of the function. This involves finding the roots of the first derivative, also known as the critical points, and then determining their nature using the second derivative. This approach is different from Newton's method, which involves an iterative process of updating an initial guess of the optimal solution based on the gradient of the function at that point. The first approach requires the function to be twice differentiable, while the second approach only requires it to be once differentiable in its basic variant.

In many real-world cases, the objective function's properties are unclear, or there is only a little information available about the function's nature. We attempt to find the absolute optimum or come to its immediate proximity while being aware of possibly having multiple local optima and being restricted to an input domain. Generally, such a task is called global optimisation. This category encompasses extremely difficult optimisation tasks. Moreover, the objective function can be completely unknown, which makes the task more challenging in practice. When the exact mathematical form is unavailable, it is often considered to be a black-box optimisation problem.

In cases with a cheap objective function evaluation, there is no need for complex optimisation strategies and the usage of methods that rely almost solely on random sampling would be sufficient, e.g. Monte Carlo methods. However, in many real-world situations, any unnecessary sampling is highly undesirable due to its cost. Therefore, we try to minimise the number of objective function evaluations. There are several possible ways to interpret the evaluation cost and why it concerns us when it is expensive in a particular context.

One of the common problems that can be encountered is a large computational demand, which is expensive in regard to both money and time. Wasting money or resources can also be an issue in cases when each evaluation might have a destructive effect on the subject of interest due to reasons related to external factors or exposure to various environmental conditions.

Last but not least, there is a moral cost of repeated evaluations. When a human or an animal interaction is needed in order to obtain the results, performing as few evaluations as possible can be desired from a moral point of view with the goal of mitigating possible harmful effects of the evaluations.

To sum up, considering that the evaluations might often be costly but are needed to learn the objective function, a necessity for an efficient optimisation strategy emerges. This can be carried out by guided and automated sampling in the input space.

The appropriate choice of an optimisation strategy for a black-box optimisation is a problem that has been addressed for several decades and has led to the development of numerous methods. We will distinguish between three main black-box optimisation types [2]:

- metaheuristics,
- direct search,
- model-based methods.

Each group represents a different strategy for guiding the search for the optimum. Since there is no consensus in classifying the optimisation methods in categories, any finer division of the methods will be omitted.

Metaheuristics encompass mainly randomised and evolutionary algorithms, such as simulated annealing or genetic algorithms. It can be demonstrated that they are often unable to converge near the global optimum and usually need a significant amount of iterations to find a solution of acceptable quality [2, 3]. This can be costly or impractical, particularly when evaluations take a long time. For instance, simulations can be computationally intractable, especially when a single evaluation takes days or even more. As a consequence of randomness, there is a chance of repeatedly sampling from the same locations, which is undesirable given the possibility of being stuck in a suboptimal part of the search space. On the other hand, metaheuristic methods do not need to rely on assumptions about the function's properties and are usually parallelisable.

The second black-box optimisation type comprises deterministic algorithms. Unlike the previously mentioned category, direct search methods implement a sequential evaluation approach according to a specific deterministic strategy. In each step, the next candidate's selection is conditioned on all earlier observations. Consequently, in each iteration, the expected improvement is likely to be more significant compared to using metaheuristics. As a result of often finding a better solution in fewer iterations, possibly also meaning a shorter amount of time [2, 4], the deterministic algorithms typically outperform metaheuristics. Nevertheless, each iteration can be quite expensive since we still evaluate the original objective function f directly.

Model-based methods also perform sampling of the evaluation points. The selection of points is based on the estimated objective function's values obtained by building an approximation model that indicates what values to expect. In scientific literature, this model is typically referred to as a surrogate since it substitutes the unknown objective function. In comparison with the two other types, model-based optimisation typically requires fewer evaluations to converge [2], and computations are not performed using the objective function f. This property allows us to use them in the case of expensive evaluations, and thus they can be applied to a broader spectrum of real-life scenarios.

Bayesian optimisation belongs among the most popular model-based approaches [5, 6, 7]. By its nature, Bayesian methods depend on a prior distribution, which means that the way new information is incorporated into a model is influenced by its initial prior [1]. In other words, a prior represents the basis on which the method builds as it progresses.

In the Bayesian optimisation framework, the prior over the possible objective functions is defined by a surrogate model. The surrogate model is sequentially updated in the light of new data. The more data we observe, the more adrift we can get from our prior beliefs. As a result of combining prior beliefs with the data, the surrogate model describes the posterior over possible objective functions f. At the same time, the uncertainty in function's values at locations close to the observed data is often reduced, so the approximation of the objective function is likely to be more accurate.

Since having a suitable prior is essential, the surrogate model should correspond to our initial beliefs about the function and its plausible characteristics. Even though the closed-form expression of the objective function is unknown, we typically have certain expectations, and we can make some assumptions, e.g. based on domain knowledge. Nevertheless, the selection of the surrogate model and its subsequent design, including adjustments of hyperparameters, should be conducted very thoroughly.

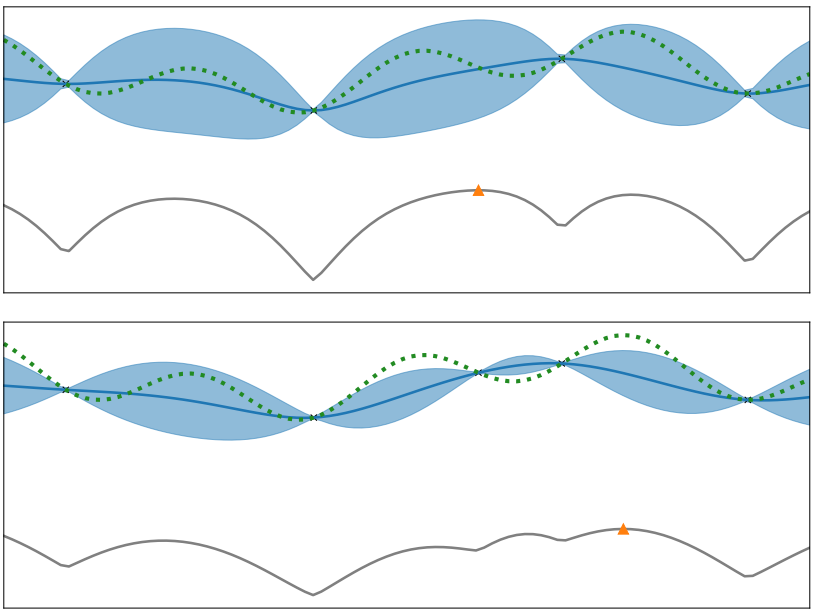


Figure 1.2: Illustration of Bayesian optimisation. The figure shows a Gaussian Process mean prediction along with its uncertainty (blue) of a true objective function (dotted green) over two iterations of Bayesian optimisation. Each iteration is supplied with a visualised acquisition function (grey) and its maximum value (orange).

There are several ways of adjusting a model with new observations. Typically, a surrogate model employed in Bayesian optimisation is probabilistic and gets adjusted via Bayesian posterior updating [5, 8]. If the model is non-probabilistic, e.g. a random forest, the model can be retrained with the new data points added to data obtained from previous iterations. Specific surrogate models will be further discussed in Section 1.1.

The acquisition function is a key component of the Bayesian optimisation framework. Its role is to choose the next point for evaluation by selecting its maximum value. Figure 1.2 shows two iterations of Bayesian optimisation on a 1D dataset.

An acquisition function aims to balance exploration and exploitation. It seeks to sample from locations where the surrogate model has high uncertainty (exploration) and where it predicts values that are likely to improve over the current best solution (exploitation) [1]. This trade-off between exploration and exploitation allows for an efficient search for the global optimum rather than conducting a greedy search.

The utilisation of the acquisition function pays off only if its cheaper to evaluate than the original objective function f. Another reason to opt for a cheap acquisition function is that there is often a need to evaluate a vast number of candidate points [5]. The examples of acquisition functions used within Bayesian optimisation will be presented in Section 1.2.

In conclusion, the Bayesian optimisation framework is a robust mechanism with respect to collecting data and modelling functions effectively, though it comes with many challenges. There are numerous decisions that need to be made to construct a framework appropriate for a particular problem. As a first step, it is required to select the two key components, an acquisition function and a surrogate model. Additionally, both components need to be tuned, which might often require extensive search or a certain amount of expertise [6]. In the acquisition function, the trade-off between exploration and exploitation should be adjusted so that we do not end up sampling only from local optima or keep exploring the input domain without improving for many iterations [1]. As implied before, the importance of the prior in Bayesian optimisation cannot be understated, making the selection of a suitable surrogate model paramount. A significant focus should also be put on designing and tuning its hyperparameters.

1.1 Surrogate Models

In this section, we discuss the use of surrogate models within Bayesian optimisation. In cases when the underlying objective function is unknown, its approximation is necessary.

The surrogate model represents the prior beliefs about the form, properties, and possible function values of the objective function. Observed values are then used to refine the surrogate model sequentially, which should lead to more accurate estimates of the objective function values. To find the next candidate points for evaluation, the surrogate model is coupled with an acquisition function. The model is updated during the optimisation process, and the points sampled for the evaluation should become more informative and converge to locations near the optimum.

As mentioned above, we can only estimate the objective function's values or make educated guesses rather than determine their true values. Modelling the exact objective function might be an impossible task in general, considering the uncertainty in the data itself. In reality, it is common for the observed data to be noisy due to several reasons, such as deviations in measurements or varying conditions during the data collection process. Being aware of pos-

sible flaws present in the data, it is desirable to express the uncertainty in the obtained predictions.

Several frameworks offer the possibility to yield the expected values along with the uncertainty in these estimates. According to [5, 7, 9], typical choices for surrogate models include probabilistic regression models such as Gaussian Processes (GP) [10], Tree-Parzen Estimators (TPE) [9], and non-probabilistic Random Forests (RF) [11]. Although it is generally more common to utilise models with probabilistic nature due to their ability to handle uncertainty, properly adjusted non-probabilistic models are also a viable option.

In recent years, there has been a trend in employing GPs to model the objective function [1, 6, 12]. GPs are considered to have superior predictive performance in the sense of estimated predictions, but especially in the modelled uncertainty measure around these estimates [10]. On the other hand, capturing the predictive distribution when working with a large amount of data is associated with significant computational requirements. The method calculates a matrix inversion, which requires $\mathcal{O}(n^3)$ time for n data points.

Existing state-of-the-art algorithms address this limitation and modify the standard GP model in order to reduce the computational burden. Such scalable models make approximate calculations with the intention of maintaining the predictive performance of the standard GP [13]. A detailed description of model approximation types is provided in Chapter 3.

The Tree-Parzen estimator [9] is a probabilistic model that, thanks to its tree structure, can handle both discrete and categorical variables. In contrast to the traditional approach employed in Bayesian optimisation, which models the probability $p(y|\mathbf{x})$ of the observed value y given the data point $\mathbf{x}$, TPE models $p(\mathbf{x}|y)$ instead. In other words, rather than being interested in capturing the posterior, we model the probability of the data point given the observation.

To model $p(\mathbf{x}|y)$, TPE uses two probability distributions that differ based on the quality of observations. The observations are split based on a percentile α (typically set to 15% [12]), resulting in the modelling of the best observations $p(\mathbf{x}|y{<}\alpha)$ and the rest $p(\mathbf{x}|y{\geq}\alpha)$ separately. This allows the TPE algorithm to focus on the most promising observations, leading to a more efficient optimisation. Scalability is not an issue as the algorithm scales linearly with the number of data points.

Another alternative to avoid scalability issues is to use an RF model. RF is an ensemble of decision trees belonging to bagging methods, which means that the models are trained independently without influencing each other. Each decision tree is trained on a set where each data point is a random sample with

a replacement from the training data. Thanks to this property, the training process can also be easily parallelised. Fitting the weak learners concurrently would provide us with faster training but the same performance as training them one by one, given that trees do not affect each other. The predictions are obtained by averaging the individual predictions made by weak learners. However, information about the model's confidence in these predictions is not provided. Since the ability of the surrogate model to express the uncertainty is needed, we must estimate it. The usual approach used to obtain the uncertainty is using the variance of its individual trees predictions [12].

Due to the tree structure of ensemble models, RF can also handle discrete variables, while GPs are more suitable to work with continuous variables [7]. Another consequence of building the models based on multiple consecutive hierarchical splits is the ability to handle categorical variables [7]. Every node in a tree represents a decision rule based on a specific feature selected from random subsets of the features. The data point that ends up in a node of the deeper level is already known to be within some range in several attribute dimensions. Random sampling subsets of data for individual trees and using only random subsets of features in constructing the decision rule for every node helps prevent overfitting and handle high-dimensional data, respectively [5]. Another advantage of RFs is that their time complexity grows linearly with data points, while GPs scale cubically.

Nevertheless, when it comes to predicting the data points far from the previously obtained observations, we may observe an undesirable behaviour [5]. Individual decision trees can often output similar predictions, resulting in small both variance and uncertainty. Although it may seem that the model is confident in the predicted values, the small uncertainty might be caused by a lack of ability to handle parts of the search space absent in training data.

1.2 Acquisition Functions

The acquisition function is the mechanism for iteratively selecting the points to be used by the surrogate model. There are several strategies for utilising the surrogate model to sample effectively. Generally, acquisition functions are designed to trade-off areas of search space where the predictions are uncertain and more promising areas, increasing the possibility to improve over the current best solution. Considering the maximisation task, the mentioned areas correspond to locations with large surrogate variance and places with high surrogate mean, respectively [1]. With regard to the needs of balancing exploration and exploitation, several functions have been invented and employed for this purpose.

The upper confidence bound (UCB) [14] chooses the following candidate $\mathbf{x}$ for the evaluation according to the current surrogate model. The selected candidate of this function is picked based on the maximum of the upper confidence bound composed of mean and standard deviations

$$\mathrm{UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}),$$

where κ is a non-negative value representing the number of standard deviations that will be used. Mean $\mu(\mathbf{x})$ and standard deviation $\sigma(\mathbf{x})$ at point $\mathbf{x}$ are obtained from the surrogate model's prediction and its uncertainty. Analogously, in the case of the minimisation task, the addition in the equation is replaced with subtraction, and the acquisition function poses in the role of a lower confidence bound.

The probability of improvement (PI) [15] takes the best observed value τ and measures the likelihood that a new point evaluation will be higher.

$$\mathrm{PI}(\mathbf{x}) = \Phi\left(\frac{\mu(\mathbf{x}) - \tau - \xi}{\sigma(\mathbf{x})}\right),$$

where Φ is the standard normal cumulative distribution function and $\xi \geq 0$ is a trade-off parameter controlling the degree of exploration. The limitation of the PI function is that it neglects the magnitude of improvement, so all possible improvements are, in this regard, equally good [5].

Since the amount by which we can improve over the current best solution τ can be very different, we want to prefer more considerable improvements. This problem can be addressed using the expected improvement (EI) [16] acquisition function.

$$\mathrm{EI}(\mathbf{x}) = (\mu(\mathbf{x}) - \tau - \xi)\Phi\left(\frac{\mu(\mathbf{x}) - \tau - \xi}{\sigma(\mathbf{x})}\right) + \sigma(\mathbf{x})\phi\left(\frac{\mu(\mathbf{x}) - \tau - \xi}{\sigma(\mathbf{x})}\right),$$

where ϕ is the standard normal probability density function. The formula is used as defined except for the case when $\sigma(\mathbf{x}) = 0$ for which we set $\mathrm{EI}(\mathbf{x}) = 0$. Both PI and EI determine the next candidate point $\mathbf{x}$ according to their maximum values. Visual comparison of UCB, PI and EI is shown in Figure 1.3.

Instead of maximising the amount of improvement over the current best solution, we can aim to maximise the information about the maximum $\mathbf{x}^*$ of the function. This approach is presented by the entropy search (ES) [17] acquisition function, which attempts to improve the information about the maximum by minimising the uncertainty in its position. Given n observed data points $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ and some arbitrary point $\mathbf{x}$, we measure the reduction in entropy after considering this point as a new measurement. The point selected

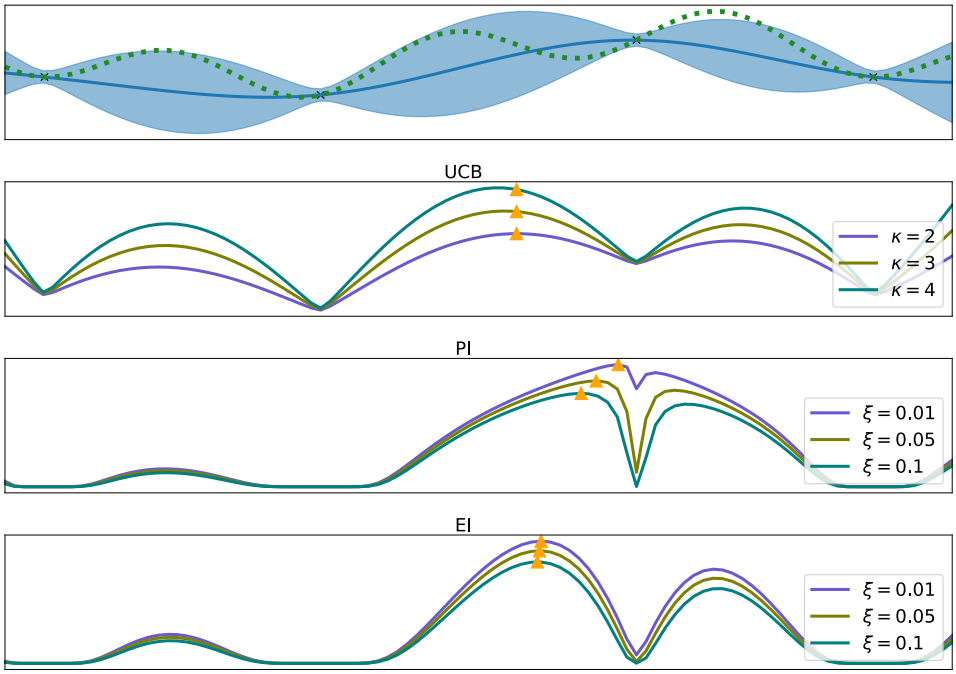


Figure 1.3: Figure shows a Gaussian Process posterior (blue) of an objective function (dotted green) and three types of acquisition function with multiple parameter values. The maximum value of each acquisition function, which corresponds to the next point to be sampled during Bayesian optimisation, is visualised with the orange triangle.

for the next evaluation indicates the point with the largest information gain computed as:

$$\mathrm{ES}(\mathbf{x}, D) = H(\mathbf{x}^\star|D) - \mathbb{E}_{y|D,\mathbf{x}} H(\mathbf{x}^\star|D \cup \{(\mathbf{x}, y)\}),$$

where the left term stands for the entropy given the available data D and the right term represents the expected entropy with the arbitrary point $\mathbf{x}$ in addition to D. This expectation is included with regards to the entire set of values $f(\mathbf{x})$, which could be possibly obtained after evaluating the point $\mathbf{x}$. Since we evaluate numerous points $\mathbf{x}$ and for each consider its every feasible value $f(\mathbf{x})$, ES acquisition function can be relatively costly.

There are also randomised acquisition functions such as Thompson sampling [18]. This acquisition strategy samples a function from the posterior distribution after the surrogate model fits all the observations available so far. The function's maximum is used as the next evaluation point.

The acquisition functions mentioned above are commonly used and represent only a fraction of possible acquisition strategies. The choice of appropriate acquisition function might be challenging and hard to determine even after adapting and comparing several acquisition functions [5]. Therefore, instead of using a single acquisition function during the whole optimisation process, the acquisition function can be varied, e.g. changed every fixed number of iterations.

There is also a possibility of using multiple acquisition functions simultaneously. One option is adopting a multi-armed bandit strategy and selecting from a pool of acquisition functions at each iteration [8, 19]. Another alternative is to use numerous acquisition functions to obtain candidate points and randomly sample the next point for the evaluation [20]. Multiple acquisition function strategies have proven to be on par with or even better performing than the standard approach, but tend to have higher computational time requirements [5].

Chapter 2

Gaussian Process Regression

Gaussian Process (GP) models are suitable for supervised learning tasks, one of which is the regression problem [21, 22]. A GP can be interpreted as a probability distribution over all possible functions that fit a set of data points. More formally, GP is defined as a type of a stochastic process - a collection of random variables, such that any finite number of which have a joint (multivariate) Gaussian distribution [22]. Respecting the notation used in [10], a GP model is given as:

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

where the parameters are to be interpreted as:

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

Hence, a GP is fully determined by its mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$, where $\mathbf{x}$ is a vector of real numbers from a D dimensional space $\mathbb{R}^D$. It is important to note that even though GP is comparable to a function, the output for any input $\mathbf{x}$ is not a single scalar but rather a mean and variance instead. The two values specify the Gaussian distribution over the possible function values at input point $\mathbf{x}$.

The mean function $m(\mathbf{x})$ represents the expected value of the distribution over functions at point $\mathbf{x}$. The mean of the prior is typically set to zero to conduct the computations related to the inference at a lower cost by using only the covariance function [22]. To centre the observed function values around zero, we can subtract their prior mean from all observations before fitting the model and add it back to the predicted values. Setting the prior to zero at the initial stage of modelling is not a limitation but rather a matter of convenience, as the mean of the posterior process is not constrained to any value.

The choice of the covariance function reflects the assumptions about the points' influence level on each other, given the distance between the points and the likely shape of the modelled function. The prior beliefs encoded within the specific selection of the covariance function limit the distribution over possible functions and thus imply the family of functions [10]. Since the covariance function $k(\mathbf{x}, \mathbf{x}')$ models the dependency of two input points, it is crucial in GP regression and will be further discussed in the following Section 2.1.

In summary, the entire assumptions about the functions that would likely fit the data are incorporated into the GP model via a prior distribution over functions determined by chosen mean and, more importantly, covariance function. Initially, the prior knowledge about the modelled function is provided by the set of n available observations, also viewed as the training data $\mathbf{X}$. Covariances between all training points are written in the covariance matrix $K(\mathbf{X}, \mathbf{X})$ as follows:

$$K(\mathbf{X}, \mathbf{X}) = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}.$$

This is a valid formulation when the observations are true objective function values without any noise. Given the noise-free observed data X, zero mean, chosen covariance function and n_* test points X_* we want to predict, the joint distribution is

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(\mathbf{X}, \mathbf{X}) & K(\mathbf{X}, \mathbf{X}_*) \\ K(\mathbf{X}_*, \mathbf{X}) & K(\mathbf{X}_*, \mathbf{X}_*) \end{bmatrix}\right),$$

where $K(\mathbf{X}_*, \mathbf{X}_*)$ is the covariance matrix between all pairs of test points, the matrix $K(\mathbf{X}, \mathbf{X}_*)$ composes of covariances between training and test data set and the same applies to $K(\mathbf{X}_*, \mathbf{X})$. The training outputs $\mathbf{f}$ are function values $[f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_n)]$ at training points locations $\mathbf{X}$, and analogously $\mathbf{f}_* = [f_*(\mathbf{x}_1), f_*(\mathbf{x}_2), \dots, f_*(\mathbf{x}_{n_*})]$ are function values of the test set $\mathbf{X}_*$.

The posterior predictive distribution is acquired by constraining the prior distribution, which is obtained by conditioning the joint prior distribution on the observed data as follows

$$\mathbf{f}_* | \mathbf{f}, \mathbf{X}, \mathbf{X}_* \sim \mathcal{N}(\mu(\mathbf{f}_*), \sigma^2(\mathbf{f}_*)),$$

where

$$\begin{aligned} \mu(\mathbf{f}_*) &= K(\mathbf{X}_*, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} \mathbf{f}, \\ \sigma^2(\mathbf{f}_*) &= K(\mathbf{X}_*, \mathbf{X}_*) - K(\mathbf{X}_*, \mathbf{X}) K(\mathbf{X}, \mathbf{X})^{-1} K(\mathbf{X}, \mathbf{X}_*). \end{aligned}$$

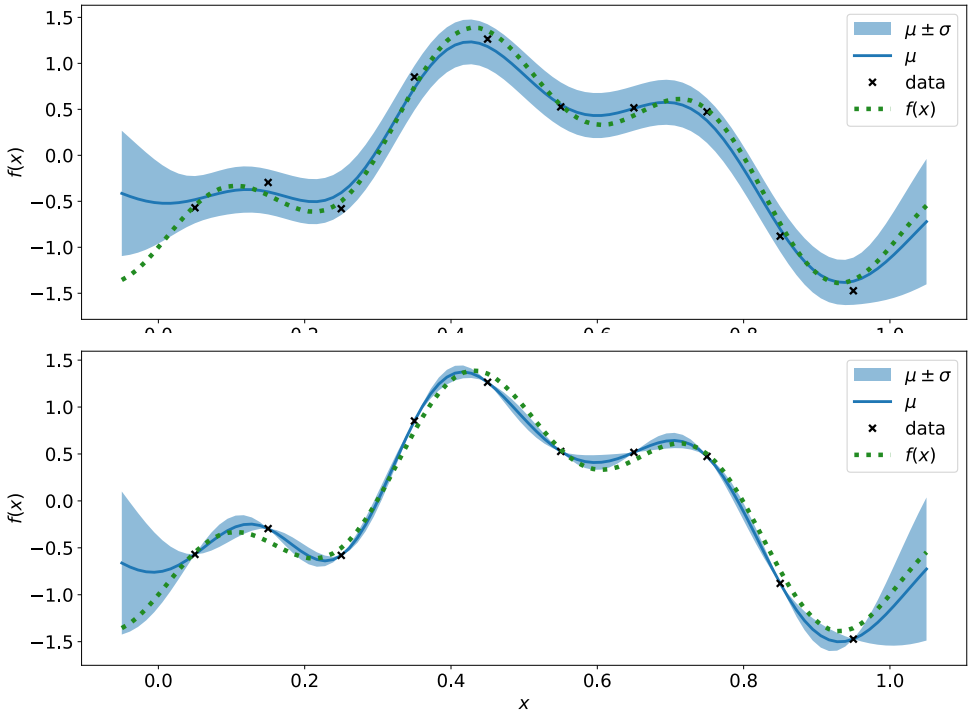


Figure 2.1: Gaussian Process fit on noisy data with additive Gaussian noise term (top) and without it (bottom).

The function values $\mathbf{f}_*$ at test locations $\mathbf{X}_*$ are obtained by sampling from the $\mathcal{N}(\mu(\mathbf{f}_*), \sigma^2(\mathbf{f}_*))$ posterior distribution. Both expressions $\mu(\mathbf{f}_*), \sigma^2(\mathbf{f}_*)$ arise from the rules for deriving conditional Gaussian distribution from joint Gaussian distribution [10].

Prediction of function values at new locations is commonly preceded by tuning the covariance function's hyperparameters $\boldsymbol{\theta}$ since its initial values might not be the best choice as they often represent only a rough estimate based on domain knowledge or information inferred from the training data [23].

For a more compact notation, let us denote the probability density $p(.)$ of the observed training outputs given the hyperparameters and training input values $p(\mathbf{f}|\mathbf{X}, \boldsymbol{\theta})$ as $\mathcal{L}(\boldsymbol{\theta})$, and rewrite $K(\mathbf{X}, \mathbf{X})$ as $\mathbf{K}$. The determinant of the matrix $\mathbf{K}$ is denoted as $|\mathbf{K}|$. A typical approach [10, 23] is to estimate the hyperparameters $\boldsymbol{\theta}$ from the training data by maximising log marginal likelihood

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \log \mathcal{L}(\boldsymbol{\theta}),$$

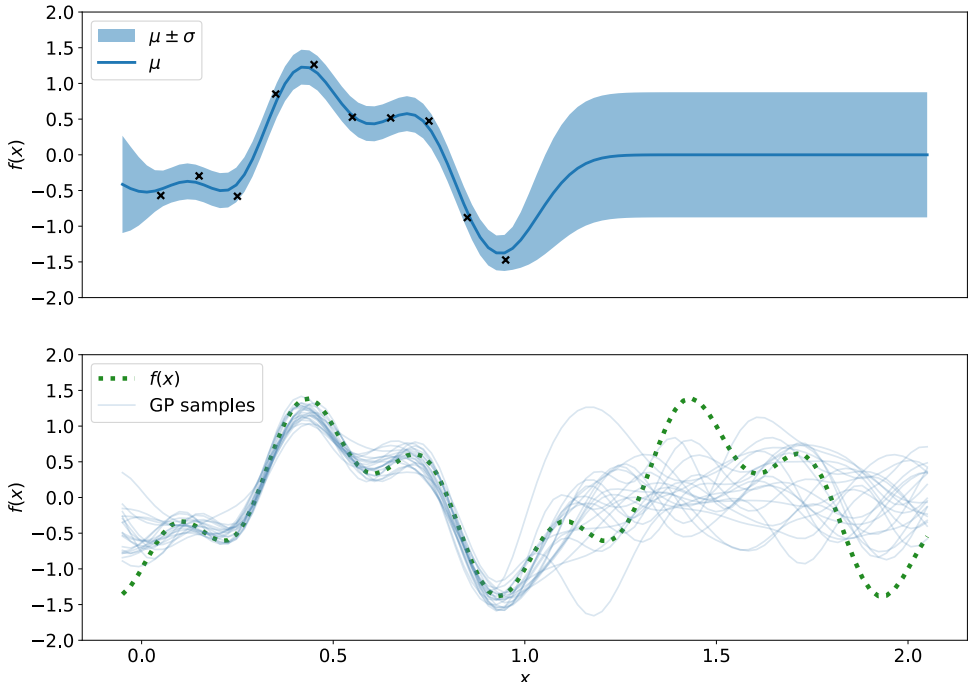


Figure 2.2: Gaussian Process fit and samples from its posterior.

where

$$\log \mathcal{L}(\boldsymbol{\theta}) = \log p(\mathbf{f}|\mathbf{X}, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{f}^\top \mathbf{K}^{-1}\mathbf{f} - \frac{1}{2}\log|\mathbf{K}| - \frac{n}{2}\log 2\pi.$$

The equation is adapted from the formula for the log likelihood for multivariate Gaussian distribution since $\mathbf{f} \sim \mathcal{N}(\mathbf{0}, \mathbf{K})$. Computing derivatives of the log marginal likelihood enables maximising it using gradient-based optimisation algorithms [10].

The formulas introduced above are applicable only when the observations are without noise, which can be viewed as a simplification of real-world situations. Indeed, it is unlikely that the true objective function values would be at our disposal. Unless the observed values are produced by computer simulation or some artificial process, they are more likely to be imprecise due to measurements deviations or the collection process itself.

The values we can usually access are noisy observations y of the objective function instead. Based on implication of the central limit theorem that random noise tends to be normally distributed, we consider additive identically distributed Gaussian noise $\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ and model the function values as $y = f(\mathbf{x}) + \varepsilon$. With the introduction of the noise term and utilisation of the identity matrix $\mathbf{I}$, the covariance matrix is extended to $K(\mathbf{X}, \mathbf{X}) + \sigma_\varepsilon^2\mathbf{I}$.

Gaussian Process fit with the additional $\sigma_\varepsilon^2\mathbf{I}$ term compared to previous formulation can be seen in Figure 2.1. It is apparent that a GP model not taking the possibility of having noisy data into account is unnecessarily overconfident in its prediction.

The joint distribution of the training data function values $\mathbf{y}$ and the function values of test inputs $\mathbf{f}_*$ then becomes

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K(\mathbf{X},\mathbf{X}) + \sigma_\varepsilon^2\mathbf{I} & K(\mathbf{X},\mathbf{X}_*) \\ K(\mathbf{X}_*,\mathbf{X}) & K(\mathbf{X}_*,\mathbf{X}_*) \end{bmatrix}\right).$$

The posterior predictive distribution is

$$\mathbf{f}_*|\mathbf{y},\mathbf{X},\mathbf{X}_* \sim \mathcal{N}(\mu(\mathbf{f}_*), \sigma^2(\mathbf{f}_*)),$$

where the parameters are redefined as

$$\begin{aligned} \mu(\mathbf{f}_*) &= K(\mathbf{X}_*,\mathbf{X})\left[K(\mathbf{X},\mathbf{X}) + \sigma_\varepsilon^2 I\right]^{-1}\mathbf{y}, \\ \sigma^2(\mathbf{f}_*) &= K(\mathbf{X}_*,\mathbf{X}_*) - K(\mathbf{X}_*,\mathbf{X})\left[K(\mathbf{X},\mathbf{X}) + \sigma_\varepsilon^2 I\right]^{-1}K(\mathbf{X},\mathbf{X}_*). \end{aligned}$$

The extended expressions may look rather complicated as opposed to the previous formulations, but the sole difference lies in adding the $\sigma_\varepsilon^2\mathbf{I}$ term to the equations for modelling noise-free data. Since the distribution of the training outputs $\mathbf{y}$ is now given as $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{K} + \sigma_\varepsilon^2\mathbf{I})$, the hyperparameters $\boldsymbol{\theta}$ of the covariance function are optimised by maximising the log marginal likelihood given by

$$\log \mathcal{L}(\boldsymbol{\theta}) = \log p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^\top(\mathbf{K} + \sigma_\varepsilon^2\mathbf{I})^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K} + \sigma_\varepsilon^2\mathbf{I}| - \frac{n}{2}\log 2\pi.$$

An example of an optimised GP model and samples from its posterior are illustrated in Figure 2.2.

2.1 Kernels

As implied before, the covariance function specifies the family of possible functions. The selection of the covariance function should reflect our assumptions about the data, such as its smoothness properties or periodic patterns. It is natural to assume that nearby points have similar function values and distant points less similar function values. In the sense of similarity, the covariance function defines how much information of training points close to the test point we can exploit in its prediction.

In the context of the Gaussian Process, the covariance function is generally called a kernel. One of the kernel's required properties is that it needs to be symmetric $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$ [10]. This requirement follows from the definition of the covariance function computed as

$$\operatorname{cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}X)(Y - \mathbb{E}Y)],$$

where X and Y are random variables. The covariance reflects how the random variables are related, and switching their positions in the equation does not affect the result.

The second fundamental requirement for every kernel is positive semidefiniteness. A positive semidefinite kernel is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, such that for $\forall n \in \mathbb{N}$, $\forall \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathcal{X}$ a matrix $\mathbf{G} = (G_{i,j})$, where $G_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, is symmetric positive semidefinite. A symmetric positive semidefinite matrix is a symmetric matrix with non-negative eigenvalues.

Positive semidefiniteness is a desirable property in many areas, including optimisation, since it ensures the correctness of several operations regarding computation with matrices. One of the reasons why positive semidefiniteness of matrices is particularly useful is that numerous algorithms working with matrices perform Cholesky decomposition, which would fail without this property. Since the Cholesky decomposition provides a fast and numerically stable way of performing matrix inversion, it is one of the strategies used instead of direct inversion when implementing the GP algorithm [10].

Other than the requirement for positive semidefiniteness, there is flexibility in designing a kernel that allows modelling processes of any shape. One of the modelled properties is the smoothness of a process that can be expressed by the degree of its mean square (MS) differentiability.

A stochastic process $g(t)$, where $t \in T \subseteq \mathbb{R}$, is MS differentiable if

$$\lim_{\Delta t \to 0} \mathbb{E}\left[\left(\frac{g(t + \Delta t) - g(t)}{\Delta t} - \frac{dg}{dt}\right)^2\right] = 0.$$

A subset of stationary covariance functions is a broadly used category of kernel types [10, 22]. A stationary function can be expressed as a function of $\mathbf{x} - \mathbf{x}'$ and is, therefore, invariant to translations in the input space. Moreover, if the kernel is the function of $|\mathbf{x} - \mathbf{x}'|$, then it is invariant to any transformations (e.g. rotations). The kernels with this property are called isotropic and given $r = |\mathbf{x} - \mathbf{x}'|$, they can be written as a function of a single argument. There are several isotropic kernels of interest, some of which have become a common

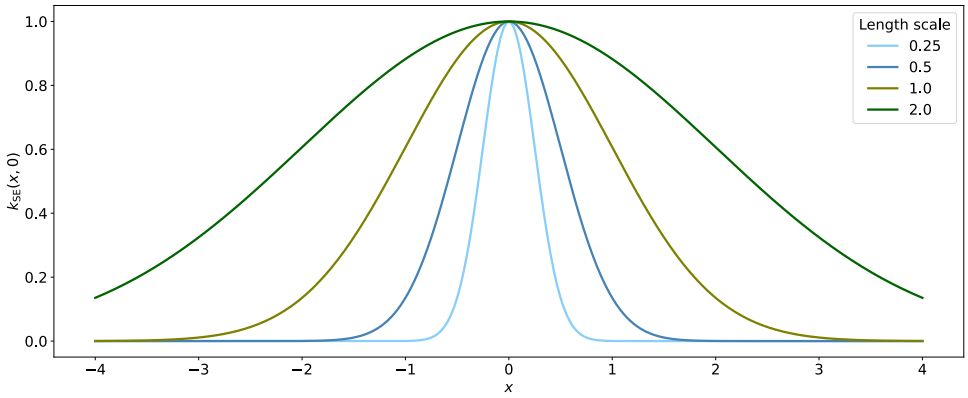


Figure 2.3: Effects of different length scales demonstrated using SE kernel.

choice within the application in Gaussian Processes, e.g., squared exponential, Matérn and rational quadratic kernel.

One of the frequently employed kernels [1, 7] is the squared exponential (SE) kernel

$$k_{\mathrm{SE}}(r) = \sigma^2 \exp\left(-\frac{r^2}{2\ell^2}\right),$$

where ℓ denotes the characteristic length scale and σ^2 represents variance, which acts as a scaling factor. The kernel is infinitely MS differentiable and therefore the corresponding process $f(\mathbf{x})$ is generally relatively smooth. The smoothness is adjusted with the ℓ parameter. A small length scale value results in more wiggly functions, as we consider only the closest points to be correlated. The further the point is from the observed data, the more uncertain the predictions. On the contrary, large values of ℓ give smoother functions, as more distant points are also taken into the account.

An example using SE kernel (also known as Gaussian or RBF) with varying length scale values is shown in a 1D scenario in Figure 2.3. The effect of using different length scale values on GPs is shown in Figure 2.4, depicting samples from GPs and its covariance matrices. Regarding the variance, increasing its value allows more variation, thus modelling the data further away from the mean.

The Matérn kernel allows controlling the level of smoothness in the sense of MS differentiability. While the SE kernel was fixed to be infinitely differentiable, the Matérn kernel introduces parameter ν representing the degree of smoothness. Typical values of the parameter ν are in the form of $\nu = i + 0.5$ where $i \in \mathbb{N}$. Popular choices are $\nu = 3/2$ and $\nu = 5/2$ [7], leading to the

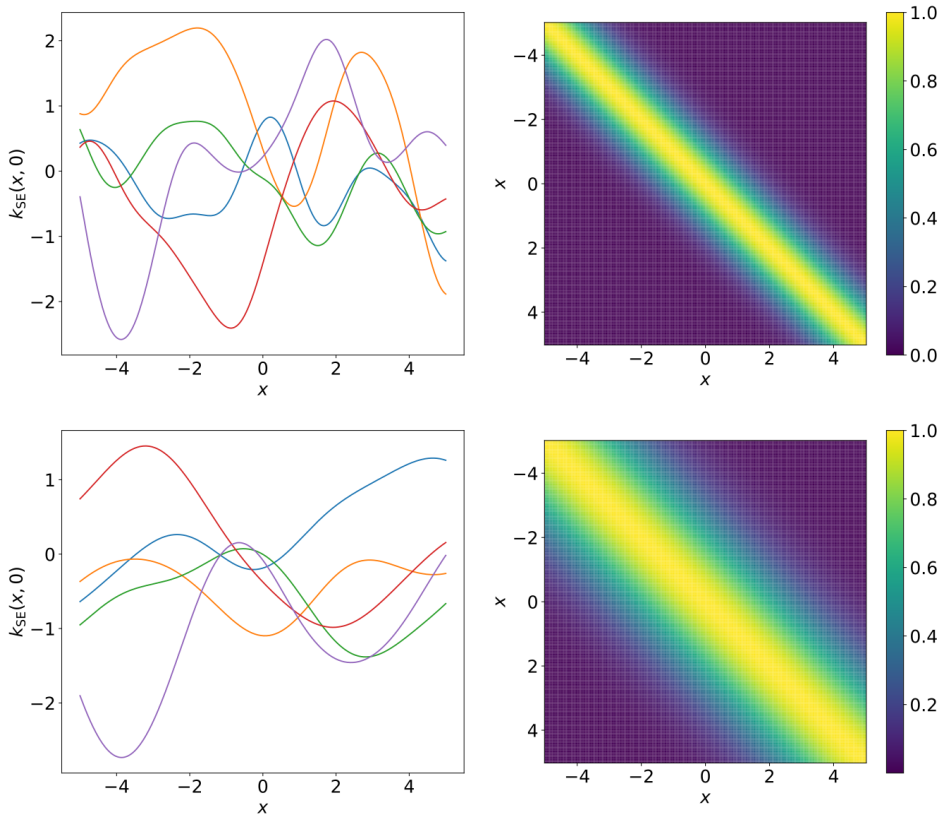
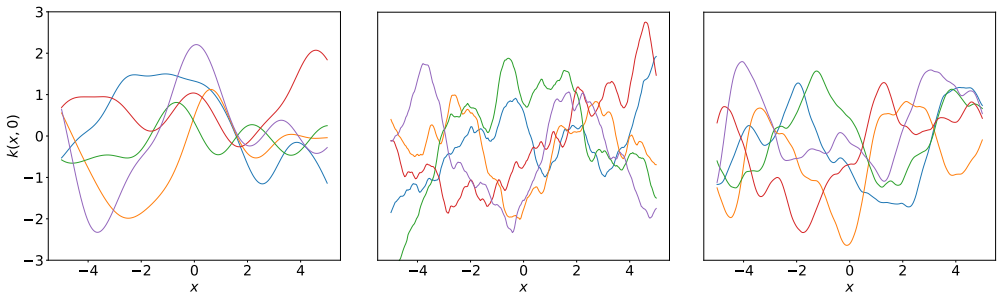


Figure 2.4: Effects of different length scales on the smoothness of samples from Gaussian Process using SE kernel. Samples from the GP posterior are shown on the left and their kernel's covariance matrix is shown on the right. Smaller length scale is used in the upper part of the image and larger length scale on the lower part.



(a) SE (b) Matérn32 (c) Matérn52

Figure 2.5: Visual comparison of SE (a), Matérn32 (b) and Matérn52 (c) kernels. Image was generated using unit length scale for every kernel.

following kernel formulations

$$k_{\nu=3/2}(r) = \sigma^2 \left(1 + \frac{\sqrt{3}r}{\ell}\right) \exp\left(-\frac{\sqrt{3}r}{\ell}\right),$$
$$k_{\nu=5/2}(r) = \sigma^2 \left(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}r}{\ell}\right).$$

Such kernels result in process $f(\mathbf{x})$ being MS differentiable once and twice, respectively. A family of Matérn kernels can also be used when it is assumed that the objective function is very rough. In that case, the parameter p is set to zero, resulting in a particular case of Matérn kernel with the corresponding process not being MS differentiable [22]. As a consequence, the points are informative only in their immediate proximity. Another special case arises with setting $\nu \to \infty$, which results in the already specified SE kernel.

The rational quadratic (RQ) kernel encompasses the SE kernel as well. Indeed, the RQ kernel is formed by the sum of SE kernels

$$k_{\mathrm{RQ}}(r) = \sigma^2 \left(1 + \frac{r^2}{2\alpha\ell^2}\right)^{-\alpha},$$

where α, ℓ are non-negative parameters. The RQ kernel enables us to combine multiple SE kernels with different length scales.

There is a large number isotropic kernels, and the above-mentioned kernels represent only a fraction of them. Other representatives of this kernel class are, for instance, linear, periodic and polynomial kernels [10].

Another attractive group of kernels includes sparse kernels, often referred to as compact kernels. The kernel matrix has a sparse representation obtained by setting the $k(\mathbf{x}_i, \mathbf{x}_j)$ to zero when the distance between the $\mathbf{x}_i$ and $\mathbf{x}_j$ exceeds a certain threshold [10]. The sparsity of the matrix allows fast matrix-vector operations as only non-zero elements are used in the computation. On the other hand, constructing a valid sparse kernel is problematic as the matrix has to be positive semidefinite. One possibility of acquiring this property is using the function from the family of covariance functions with compact support, such as the Wendland polynomials [24]. Nevertheless, the constructed kernels are typically valid only for low dimensional data, and thus the computational advantages are feasible in limited use cases.

Even though the stationary kernels are more prevalent in scientific literature [5, 22], the non-stationary kernels can also be applied. In addition, the standard kernels may not be sufficient in modelling data with more complex patterns and structures. The existing kernels can be modified or combined to

create a new kernel with desired properties, as seen before when introducing the RQ kernel. In that case, the SE kernels are incorporated into a single one by summation. Other operations, such as kernels multiplication or a combination of both mentioned operations, can be used as well to create a valid kernel [10]. Even with the preferred characteristics in mind, we may lack the confidence to choose the kernel and set the values of its parameters. It is important to keep in mind that restricting the kernel to a single family of kernels may still leave numerous parameter combinations.

Chapter 3

State-of-the-art Scalable GPs

In many real-world applications, the employment of standard GPs is rather impractical considering that the memory and computational requirements grow with quadratic and cubic complexity, respectively. Such a high time complexity is a consequence of computing the matrix inversion, which requires $\mathcal{O}(n^3)$ operations, where n is the number of training points. This key limitation results in the model being unsuitable for large data. However, the prohibitive amount of calculations necessary to inverse the matrix does not make the application of GPs in big-data domains completely impossible.

Various approaches have been proposed that led to the development of numerous methods to circumvent this shortcoming. Their primary purpose is to avoid cubic time complexity and perform the calculations at lower computational costs, which is usually $\mathcal{O}(nm^2)$ time and $\mathcal{O}(nm)$ memory, where $m \ll n$. Moreover, the goal is not only to alleviate the computational demands but also to maintain the prediction quality of the standard GP model. The models that mitigate the issues hampering the practical application of a standard GP are generally called scalable GPs.

3.1 Taxonomy of Scalable GPs

The strategies used to address the high computational complexity of GP models can be divided into two main categories [13]. The first approach aims to tackle the limitation on a global level, while the second focuses on addressing the issue on a local level. It is unclear which approach is the best or even the most effective, and some methods combine both approaches in order to overcome the computational demands of GP models. The taxonomy of the approximation types and their strategies presented below follows the structure of the scalable GPs overview presented in [13] as shown in Figure 3.1, which was among the first to provide an extensive survey and detailed analysis.

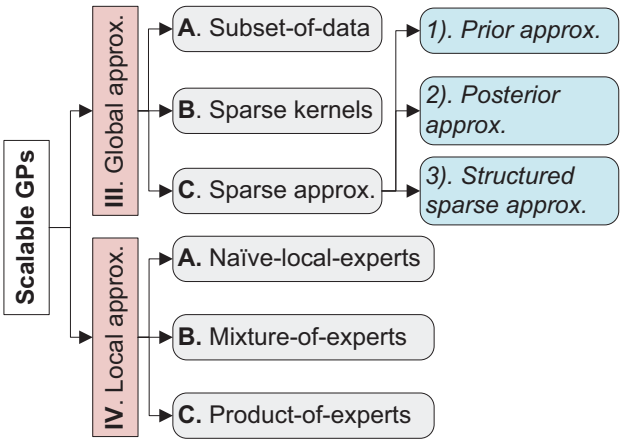


Figure 3.1: An overview of scalable GPs as presented in [13].

It is crucial to point out that this categorisation represents only one of many ways to perceive these approximations. The following sections describe the characteristics of several local and global approximation types.

3.1.1 Local Approximations

Local approximations tend to capture local patterns more effectively by dividing the data into smaller subsets and building individual models, known as experts, for each subset. Consequently, the experts focused on the partitioned data are more likely to achieve better results on data with some particular characteristics, e.g. non-stationarity, where the global approximation methods might fail [13]. On the other hand, there is a possibility of overfitting the individual models, missing the long-term spatial correlations and lacking continuity in obtained predictions. Within local approximations, we recognise three approaches.

Naive-local-experts [25] employ each expert independently, resulting in no interaction between the experts' predictions. The prediction of the test point is conducted only by a single expert responsible for its neighbourhood, which leads to discontinuous predictions. For this reason, the preferred strategy is to gather predictions from multiple experts. Such an approach allows for more continuous and accurate predictions and will be introduced in the following text.

The usage of several experts for a single point prediction is applied in the mixture-of-experts (MoE) [26] and product-of-experts (PoE) [27] methods. Both techniques belong to ensemble learning methods due to the possibility of using various models and composing their predictions with a gating function.

This function determines the amount of contribution of individual models and the aggregation type used to produce the final answer. The approach used in MoE to combine the experts can be seen as summation, while PoE uses a multiplication of probability distributions. Another factor that contributes to receiving smoother predictions is partitioning the data space into overlapped subsets instead of having non-intersecting subspaces for each expert.

3.1.2 Global Approximations

Another approach to approximating GPs is to use global approximations, which focus on capturing global rather than local patterns. The methods are of three categories, and unlike the previous approximation type categories, the individual groups substantially vary in the approach they employ. Nevertheless, the prevalent common aspect lies in the utilisation of inducing inputs that refer to a subset of some entity, e.g. points selected from the training data. The literature also refers to these data points as support points, pseudo-inputs, or active set [13, 28].

The first and most basic method is to use only a subset of the original training data. Therefore, the kernel matrix is of size $m \times m$ instead of $n \times n$, where $m < n$, which leads to a reduction in time and memory complexity to $\mathcal{O}(m^3)$ and $\mathcal{O}(m^2)$, respectively [13]. However, the active set can be chosen in many ways, and naturally, one of them is random selection. A better strategy might be to use clustering algorithms or learning criteria to ensure a sufficiently representative subset.

More sophisticated global approximation type includes usage of sparse kernels. In this case, the kernel matrix size remains the same, but its elements do not. The sparsity is achieved by setting many matrix elements to zero. A detailed explanation of how to construct such a matrix can be found in the Section 2.1.

The third type represents sparse approximations that focus on decreasing the complexity of the matrix inversion. This approach is gaining popularity in the GP community due to its potential to improve computational efficiency while retaining good prediction accuracy. Therefore, the following section will be devoted solely to the global sparse approximation, including a comprehensive characterisation of particular GP models of this type later employed in experiments.

3.2 Global Sparse Approximate Methods

Global sparse approximations attempt to construct a low-rank representation of the kernel matrix and thus reduce the computational burden of the matrix inversion. A common way to achieve this is to use eigenvalue decomposition

requiring $\mathcal{O}(n^3)$ operations. However, this is highly undesirable. Therefore, the eigenvalue decomposition is approximated using Nyström approximation

$$\mathbf{K}_{nn} \approx \mathbf{Q}_{nn} = \mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{K}_{nm}^{\top},$$

taking $\mathcal{O}(m^2 n)$ time when using m inducing points $\mathbf{u}$ at input locations $\mathbf{Z}$, where covariance matrices $\mathbf{K}_{nn} = K(\mathbf{X}, \mathbf{X})$, $\mathbf{K}_{nm} = K(\mathbf{X}, \mathbf{Z})$, $\mathbf{K}_{mm} = K(\mathbf{Z}, \mathbf{Z})$ and the inducing points $\mathbf{u} \sim \mathcal{N}(0, \mathbf{K}_{mm})$. The joint distribution of the training outputs $\mathbf{y}$ and inducing outputs $\mathbf{u}$ is given by

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{u} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K}_{nn} + \sigma^2\mathbf{I} & \mathbf{K}_{nm} \\ \mathbf{K}_{mn} & \mathbf{K}_{mm} \end{bmatrix}\right).$$

Given $\mathbf{u}$, assuming that the training outputs $\mathbf{y}$ and the test outputs $\mathbf{f}_*$ are conditionally independent leads to the dependency between $\mathbf{y}$ and $\mathbf{f}_*$ being induced via $\mathbf{u}$ [28]. Derived conditional Gaussian distributions have the following forms:

$$\begin{array}{lll} \text{training conditional} & p(\mathbf{y}|\mathbf{u}) & \sim \mathcal{N}(\mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{u}, \mathbf{K}_{nn} - \mathbf{Q}_{nn} + \sigma_n^2\mathbf{I}), \\ \text{test conditional} & p(\mathbf{f}_*|\mathbf{u}) & \sim \mathcal{N}(\mathbf{K}_{*m}\mathbf{K}_{mm}^{-1}\mathbf{u}, \mathbf{K}_{**} - \mathbf{Q}_{**}), \end{array}$$

where $\mathbf{K}_{*m} = K(\mathbf{X}_*, \mathbf{Z})$ and $\mathbf{K}_{**} = K(\mathbf{X}_*, \mathbf{X}_*)$. Both conditionals, also referred to as inducing conditionals, serve as a baseline for numerous global sparse approximations. For this reason, we introduce notation $p(.)$ for a probability density function defining a probability distribution and and $q(.)$ for a probability density function defining approximate probability distribution. Employing the exact inducing conditionals causes no speed up since there is still the need for a full matrix inversion. Various sparse approximation methods can be obtained by performing further approximations of inducing conditionals via modifying covariance matrices [28].

Global sparse approximations can be further subdivided into three categories:

- prior approximations,
- posterior approximations,
- structured sparse approximations.

The latter approximation type employs a strategy of overcoming the computational limitations by utilising matrix-vector multiplication. One of the well-known methods from this group is the Structured Kernel Interpolation (SKI) [29]. A detailed explanation of this model will be omitted since the number of inducing points grows exponentially with dimensions, and therefore SKI is rather suitable for lower-dimensional problems [30].

Prior approximations approximate the joint prior but perform the exact inference. The likely most influential representant from the prior approximation category is the Fully Independent Training Conditional (FITC) [31, 32]. On the other hand, the posterior approximations keep the prior exact, but approximate the posterior resulting in approximate inference. A particularly popular posterior approximation is the Variational Free Energy (VFE) [33, 32]. The following sections will provide an in-depth explanation of both approaches.

3.2.1 FITC

As opposed to the baseline conditionals introduced for global sparse approximations, the change proposed within the FITC model includes further approximating the training conditional while retaining the same test conditional [28]. The form of the covariance matrix is restricted to a diagonal matrix, which results in

$$q(\mathbf{y}|\mathbf{u}) = \mathcal{N}(\mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{u}, \operatorname{diag}\left[\mathbf{K}_{nn} - \mathbf{Q}_{nn}\right] + \sigma_n^2\mathbf{I}).$$

To highlight the difference between the baseline and the FITC approximation stated above, let us derive the corresponding joint distribution using the common rules for deriving joint Gaussian distribution from conditional Gaussian distribution [10]. The joint distribution in the FITC model is given as

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{u} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{G} + \sigma^2\mathbf{I} & \mathbf{K}_{nm} \\ \mathbf{K}_{mn} & \mathbf{K}_{mm} \end{bmatrix}\right),$$

where $\mathbf{G} = \mathbf{Q}_{nn} + \operatorname{diag}\left[\mathbf{K}_{nn} - \mathbf{Q}_{nn}\right]$ and $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{G} + \sigma_n^2\mathbf{I})$. Respecting the notation introduced in Chapter 2, we denote log marginal likelihood maximised to train the model as $\log \mathcal{L}(\boldsymbol{\theta}) = p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})$. Using the above obtained expression, $\log \mathcal{L}(\boldsymbol{\theta})$ is for FITC accordingly modified to

$$\log \mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^\top(\mathbf{G} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{G} + \sigma_n^2\mathbf{I}| - \frac{n}{2}\log 2\pi.$$

3.2.2 VFE

A more recent approach is to employ posterior (as opposed to prior) approximations performing inexact inference rather than approximating the model itself [30, 32]. Since the posterior approximations maintain the prior in its exact form, the inducing conditionals in VFE remain unchanged. The usage of approximate inference is indicated in the VFE model name itself through the word variational. In general, variational inference refers to methods for approximating distributions due to the exact distributions being computationally intractable for practical usage [34]. Again, the distribution that we use to approximate the intractable one is also called a variational distribution. Terms "approximate" and "variational" are both, often interchangeably, used in GP literature [13, 28, 32].

The methods using variational inference try to find a variational distribution from a family of tractable distributions that is the closest to the approximated distribution. Employing variational inference in GPs results in the approximation achieved by constructing a lower bound to the log marginal likelihood as

$$\begin{aligned}\log \mathcal{L}(\boldsymbol{\theta}) &= \log \int_{\mathbf{f},\mathbf{u}} p(\mathbf{y}|\mathbf{f}) p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z}) p(\mathbf{u}|\mathbf{Z}) \\ &\geq \int_{\mathbf{f},\mathbf{u}} q(\mathbf{f},\mathbf{u}) \log \frac{p(\mathbf{y}|\mathbf{f}) p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z}) p(\mathbf{u}|\mathbf{Z})}{q(\mathbf{f},\mathbf{u})},\end{aligned}$$

where $q(\mathbf{f},\mathbf{u})$ is variational posterior distribution to approximate $p(\mathbf{f},\mathbf{u})$. Within the VFE model, it is proposed to use $q(\mathbf{f},\mathbf{u}) = p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z})q(\mathbf{u})$ as the variational posterior, so the variational lower bound is given by

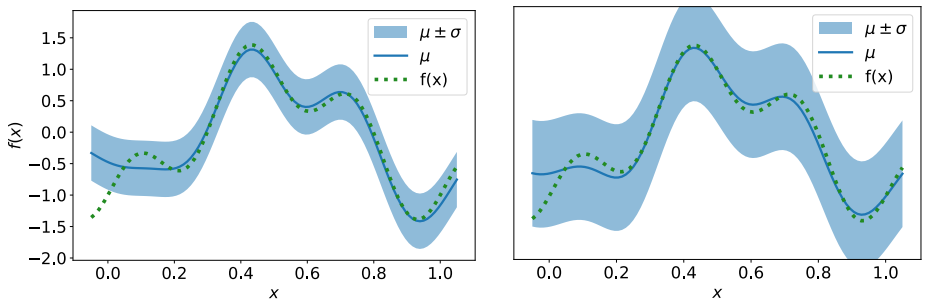
$$\begin{aligned}\log \mathcal{L}(\boldsymbol{\theta}) &\geq \int_{\mathbf{f},\mathbf{u}} p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z}) q(\mathbf{u}) \log \frac{p(\mathbf{y}|\mathbf{f}) \cancel{p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z})} p(\mathbf{u}|\mathbf{Z})}{\cancel{p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z})} q(\mathbf{u})} \\ &\geq \langle \log p(\mathbf{y}|\mathbf{f}) \rangle_{p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z})q(\mathbf{u})} - \mathrm{KL}(q(\mathbf{u}) \| p(\mathbf{u}|\mathbf{Z})),\end{aligned}$$

where KL stands for the Kullback-Leibler divergence used to measure how much one probability distribution differs from the second one. During the variational inference, KL terms needed for the computations are often intractable. In practice, a tractable term called the evidence lower bound (ELBO) is used instead. Maximising the ELBO is equivalent to minimising the KL divergence term.

The notation $\langle . \rangle_{p(.)}$ is adapted from the GP literature [13, 30, 35] and represents the expectation over the distribution $p(.)$. The cancellation of terms leads to an approximation [33] where the lower bound to the $\log \mathcal{L}(\boldsymbol{\theta})$ is constructed in a way that gets rid of all the terms containing $\mathbf{K}_{nn}^{-1}$ [32]. As a result of approximating the inference, the VFE method is trained by maximising

$$\log \mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^\top (\mathbf{Q}_{nn} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y} - \frac{1}{2} \log |\mathbf{Q}_{nn} + \sigma_n^2 \mathbf{I}| - \frac{n}{2} \log 2\pi - \frac{1}{2\sigma_n^2} \operatorname{tr}(\mathbf{T}),$$

where $\mathbf{T} = \mathbf{K}_{nn} - \mathbf{Q}_{nn}$ and $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_{nn} + \sigma_n^2 \mathbf{I})$. An additional trace term in the VFE is put in place to guarantee a lower bound of the true log marginal likelihood. Moreover, the trace term serves as a regulariser that makes the model more resilient to overfitting the training data and helps find suitable locations for inducing inputs [32].



(a) FITC (b) VFE

Figure 3.2: Visual comparison of variance using FITC (a) and VFE (b).

3.2.3 Differences between FITC and VFE

Apart from the obvious similarities that FITC and VFE models share due to their common nature, they differ in their levels of susceptibility to certain behaviour. This section attempts to summarise some of the known distinctions. Even with the additional trace term improving the training of VFE, this approximation method still faces optimisation issues, and it has been shown that it is generally harder to optimise than FITC [32]. However, in the light of the fact that numerous experiments demonstrated the tendency of prior approximations to provide poorer predictions than the posterior approximations [30, 32], researchers often suggest using VFE, albeit dedicating more time to fine-tuning the model. This drawback is also outweighed by other qualities of VFE, such as being able to approximate the standard GP more accurately than FITC.

Moreover, while VFE is likely to underfit if not appropriately adjusted, FITC is prone to overfitting, which seems to be more problematic [32]. Overfitting is often detectable from variance, which is one of the indicators of a good fit since it can be interpreted as the uncertainty of the model. If the uncertainty is small, it can be an undesirable outcome even when it may seem counterintuitive. This possibly culminates in an underestimation of uncertainty since it can reach near-zero values even in sparsely covered regions. An example of the FITC model being more confident in its predictions compared to the VFE model can be seen in Figure 3.2.

Another difference lies in the placement of the inducing points. Even when we would expect the model to improve with a higher number of inducing points, there is a chance we may not witness this behaviour using FITC. Inducing inputs are likely to be placed in the immediate proximity of each other, creating clusters or even overlapping. Mentioned characteristics and typical

exhibited behaviour of models are well explained and thoroughly justified in [32] along with other noteworthy differences and remarks regarding VFE and FITC models.

3.3 Recent Developments and Extensions

The extensive and successful application of scalable GPs has led to the growth of the GP community and an increased interest in improving the scalability even further. The desire to alleviate the computational complexity we face when working with GPs triggered an evolution of scalable GPs in several directions. With the demand for improvement, various extensions and hybrid models of approximate GPs emerged, the development of which is still a matter of ongoing research. One of the incentives to create new models and build on the present work is undoubtedly the possibility of employing scalable GPs in various fields, from finance and economics to engineering and computer science. However, many potential research avenues yet remain undiscovered.

3.3.1 SVGP

One of the standard approximate models that researchers use to base their novel ideas on is VFE. The introduction of the extended VFE model known under the name Stochastic Variational Gaussian Process (SVGP) [36] enabled it to handle large amounts of data in batches. The improvement lies in replacing the original so-called collapsed bound utilising integral with the uncollapsed bound using a summation term instead. Moreover, the KL term becomes data-independent. The uncollapsed bound is less tight [30] and is obtained as

$$\begin{aligned}\log \mathcal{L}(\boldsymbol{\theta}) &\geq \langle \log p(\mathbf{y}|\mathbf{f}) \rangle_{p(\mathbf{f}|\mathbf{u},\mathbf{X},\mathbf{Z})q(\mathbf{u})} - \mathrm{KL}(q(\mathbf{u})\|p(\mathbf{u})) \\ &\geq \sum_{i=1}^{n} \left\langle \log \mathcal{N}\left(y_i|f_i,\sigma^2\right) \right\rangle_{q(f_i|\mathbf{x}_i,\mathbf{Z})} - \mathrm{KL}(q(\mathbf{u})\|p(\mathbf{u})).\end{aligned}$$

The bound is further adjusted to be suitable for batch training by approximating the summation over the whole dataset with a summation over batches. Let $\tilde{p}$ denote the distribution of $\mathbf{x}, \mathbf{y}$ values within a single batch [13, 35], then we can write the bound as follows

$$\log \mathcal{L}(\boldsymbol{\theta}) \geq \frac{n}{b} \sum_{\mathbf{x}_j, y_j \sim \tilde{p}(\mathbf{x},\mathbf{y})} \left\langle \log \mathcal{N}\left(y_j|f_j,\sigma^2\right) \right\rangle_{q(f_j|\mathbf{x}_j,\mathbf{Z})} - \frac{n}{b}\mathrm{KL}(q(\mathbf{u})\|p(\mathbf{u})),$$

where n is the number of training samples and b is the batch size. The right-hand side term is rescaled accordingly.

Usage of the relaxed bound results in significantly reduced complexity of $\mathcal{O}(bm^3)$, where m is the number of inducing points. Nevertheless, since the

optimisation is performed within batches, it leads to a drawback of having additional $\mathcal{O}(m^2)$ variational parameters [37]. Naturally, having more parameters makes the model calibration more difficult.

Additionally, even though the SVGP model can be used for online data, it is not well-suited for this task [38]. Its effectiveness relies on having access to the entire dataset at each stage of training, but this may not be possible due to memory constraints. This problem can occur with any machine learning model, but it is particularly relevant for the SVGP model because its performance depends on having access to the whole dataset at any time, which is counterintuitive and not practical in the context of online streaming data. As a result, another weakness of the model is that it may not be possible to train it if computational resources are limited, even though its potential application to streaming settings suggests otherwise.

In addition, suppose we are receiving the data sequentially, and the number of data points is unknown. In that case, with each new data batch received, we can either add new data points to the previously obtained data or use only the latest data. Both approaches are suboptimal since the computational complexity is constantly increasing, or the information present in older data is lost. Even though the SVGP model was a breakthrough in enabling the training on larger datasets to be more affordable, it was unsuitable for their combination with an online training setting.

3.3.2 SSGP

The authors of the Streaming Sparse GP (SSGP) model [38] got inspired by VFE and SVGP approaches and tried to satisfy the demand by creating a global approximation suitable for the application in the streaming data regime. The SSGP model also considers the possible lack of resources and utilises the optimisation technique where each batch of data is single use only. Every new batch updates the model's hyperparameters and reflects itself into the locations of inducing points by replacing 30% of them with randomly selected data samples from the batch [39]. The exception is the initial inducing set consisting exclusively of randomly sampled data points. Utilising some of the samples from the batch as new inducing locations is supposed to help the model, particularly in cases where the data are not independent and identically distributed, and the input domain is not discovered at once. The SSGP model uses both the old inducing set $\mathbf{Z}_\mathrm{a}$ of size m_a and the adjusted inducing set $\mathbf{Z}_\mathrm{b}$ of size m_b created after receiving the new batch to compute the next update. All the conducted experiments presented in the paper [38] use the same size of inducing sets $m_\mathrm{a} = m_\mathrm{b}$, although it is not required.

Since the data are at our disposal only once, optimisation is performed by taking multiple gradient steps with carefully adjusted learning rates [38]. The described approach enables updating a previously fitted model without seeing or knowing the amount of the old or the new data. This means that at every training phase, information from the current batch is incorporated into the model's knowledge without directly looking at any other data. Unfortunately, the computational requirements are not reduced compared to the SVGP model and remain quadratic.

In this section, the already introduced notation for describing matrices in Chapter 2 will be neglected in favour of the notation used in [38]. The change was employed primarily due to the immense number of matrices used to describe the SSGP model and will hopefully help the reader avoid confusion. Thus, the indices will no longer describe the dimensions of the matrix but rather expressions to which the matrix relates. Furthermore, the notation for the training inputs $\mathbf{X}$ and training outputs $\mathbf{y}$ will be reused to describe the data from the current batch. At the same time, the equations listed below are equally applicable in and do not contradict the scenario where all the data come simultaneously. Online training of the SSGP model is performed by maximising the bound defined as

$$\log \mathcal{L}(\boldsymbol{\theta}) = \mathcal{F} + \Delta_1 + \Delta_2,$$

where $\mathcal{F}$ represents log marginal likelihood of $\hat{\mathbf{y}} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_{\hat{\mathrm{f}}\hat{\mathrm{f}}} + \Sigma_{\hat{\mathrm{y}}})$ and Δ_1, Δ_2 are regularisation terms, all given as

$$\begin{aligned}
\mathcal{F} &= \log \mathcal{N}(\mathbf{0}, \mathbf{K}_{\hat{\mathrm{f}}\hat{\mathrm{f}}} + \Sigma_{\hat{\mathrm{y}}}) \\
&= -\frac{n + m_{\mathrm{a}}}{2} \log(2\pi) - \frac{1}{2} \log \left| \mathbf{K}_{\hat{\mathrm{f}}\hat{\mathrm{f}}} + \Sigma_{\hat{\mathrm{y}}} \right| - \frac{1}{2} \hat{\mathbf{y}}^T \left(\mathbf{K}_{\hat{\mathrm{f}}\hat{\mathrm{f}}} + \Sigma_{\hat{\mathrm{y}}} \right)^{-1} \hat{\mathbf{y}}, \\
\Delta_1 &= \frac{1}{2} \Bigg(- \log \frac{|\mathbf{S}_{\mathrm{a}}|}{|\mathbf{K}_{\mathrm{aa}}| \, |\mathbf{D}_{\mathrm{a}}|} + \mathbf{m}_{\mathrm{a}}^T \mathbf{S}_{\mathrm{a}}^{-1} \mathbf{D}_{\mathrm{a}} \mathbf{S}_{\mathrm{a}}^{-1} \mathbf{m}_{\mathrm{a}} - \operatorname{tr}\left(\mathbf{D}_{\mathrm{a}}^{-1} \mathbf{Q}_{\mathrm{a}} \right) - \mathbf{m}_{\mathrm{a}}^T \mathbf{S}_{\mathrm{a}}^{-1} \mathbf{m}_{\mathrm{a}} \\
&\quad + m_{\mathrm{a}} \log(2\pi) \Bigg), \\
\Delta_2 &= -\frac{1}{2\sigma_{\mathrm{y}}^2} \operatorname{tr}\left(\mathbf{Q}_{\mathrm{f}} \right),
\end{aligned}$$

where

$$\mathbf{K}_{\hat{\mathrm{f}}\hat{\mathrm{f}}} = \mathbf{K}_{\hat{\mathrm{f}}\mathrm{b}} \mathbf{K}_{\mathrm{bb}}^{-1} \mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}}, \hat{\mathbf{y}} = \begin{bmatrix} \mathbf{y} \\ \mathbf{D}_{\mathrm{a}} \mathbf{S}_{\mathrm{a}}^{-1} \mathbf{m}_{\mathrm{a}} \end{bmatrix}, \mathbf{K}_{\hat{\mathrm{f}}\mathrm{b}} = \begin{bmatrix} \mathbf{K}_{\mathrm{fb}} \\ \mathbf{K}_{\mathrm{ab}} \end{bmatrix}, \Sigma_{\hat{\mathrm{y}}} = \begin{bmatrix} \sigma_{\mathrm{y}}^2 \mathrm{I} & 0 \\ 0 & \mathbf{D}_{\mathrm{a}} \end{bmatrix},$$

$\mathbf{K}_{\mathrm{fb}} = K(\mathbf{X}, \mathbf{Z}_{\mathrm{b}}), \mathbf{K}_{\mathrm{ab}} = K(\mathbf{Z}_{\mathrm{a}}, \mathbf{Z}_{\mathrm{b}}), \mathbf{D}_{\mathrm{a}} = (\mathbf{S}_{\mathrm{a}}^{-1} - \mathbf{K}_{\mathrm{aa}}^{-1})^{-1}, \mathbf{K}_{\mathrm{aa}} = K(\mathbf{Z}_{\mathrm{a}}, \mathbf{Z}_{\mathrm{a}})$, $\mathbf{Q}_{\mathrm{a}} = \mathbf{K}_{\mathrm{aa}} - \mathbf{K}_{\mathrm{ab}} \mathbf{K}_{\mathrm{bb}}^{-1} \mathbf{K}_{\mathrm{ba}}, \mathbf{K}_{\mathrm{bb}} = K(\mathbf{Z}_{\mathrm{b}}, \mathbf{Z}_{\mathrm{b}})$, and $\mathbf{Q}_{\mathrm{f}} = \mathbf{K}_{\mathrm{ff}} - \mathbf{K}_{\mathrm{fb}} \mathbf{K}_{\mathrm{bb}}^{-1} \mathbf{K}_{\mathrm{bf}}$.

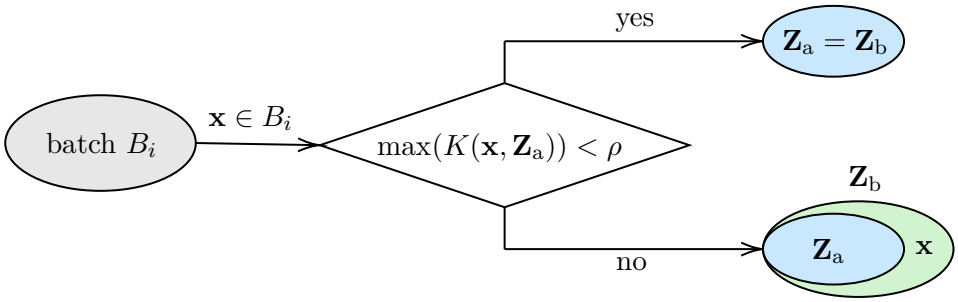


Figure 3.3: Visualisation of the OIPS decision process on adding a new sample to the inducing set.

The terms $\mathbf{m}_\mathrm{a}, \mathbf{S}_\mathrm{a}$ are obtained from the model by predicting the mean and covariance of the old inducing set $\mathbf{Z}_\mathrm{a}$ before seeing the current batch. The expressions that are used to compute the predictive mean and covariance matrix are stated in Appendix B.1.

The bound proposed above is not used in the implementation as is, but is instead rewritten to provide better numerical stability. The derivation of the bound adjusted for practical use from the theoretical bound is demonstrated step-by-step in Appendix B.2.

3.3.3 OIPS

As indicated earlier, the purpose of inducing points is to represent all of the data that have been observed accurately. At the same time, this should be accomplished while keeping the number of inducing points significantly smaller than the number of training samples, thus reducing computational complexity. In a streaming setting, the objective can be more challenging to achieve when the number of inducing points is fixed beforehand. We may not know the final number of data points that will be received, making it hard to estimate the size of the inducing set. Therefore, increasing the number of inducing points with the growing dataset is likely necessary.

The Online Inducing Points Selection (OIPS) [40] addresses this problem and extends the SSGP model to scale the model's capacity as required. The OIPS extension embraces the changing number of inducing points with each batch by adding new data points to the inducing set based on their novelty. Data sample $\mathbf{x}$ from the current batch is added to the inducing set if the maximum value of $K(\mathbf{x}, \mathbf{Z}_\mathrm{a})$ is smaller than a threshold parameter ρ [40]. The matrix computation does not cause an increase in the computational complexity, and it is the same as in the SSGP model utilising its original inducing points selection.

However, the determination of whether to use the data sample $\mathbf{x}$ as an inducing point is not based solely on the selected threshold ρ, where $0 < \rho < 1$. The decision is also affected by the kernel, already received data and their structure. The decision making process is schematically depicted in Figure 3.3. Furthermore, it has been demonstrated that the inducing set grows more rapidly with larger dimensions, smaller lengthscales of the kernel, and a higher ρ value [40]. The authors also guarantee the deterministic properties of the OIPS technique for fixed ordering of the data points.

Therefore, instead of creating a new inducing set of the same size by replacing some of the old inducing points with a fraction of new data samples, the new inducing set is enriched by new points and contains all of the previously optimised inducing points. If no data points are added, the initial inducing set $\mathbf{Z}_\mathrm{b}$ used for training on the current batch is identical to the old inducing set $\mathbf{Z}_\mathrm{a}$ after the optimisation on the previous batch.

3.3.4 PIPS

How many and which inducing points to use was a question that also bothered the authors of Probabilistic Selection of Inducing Points in Sparse Gaussian Processes [41], hereinafter referred to as the Probabilistic Inducing Points Selection abbreviated to PIPS. The responsibility of setting a number of inducing points is, in this case, not entirely left to the user. The determination of the inducing set and its size is accomplished with the help of two point processes introduced below, and the dataset itself.

Let Ω be a probability space and $T \subseteq \mathbb{R}$ an index set. The system of random variables

$$\boldsymbol{X} = \{X_t \mid t \in T\}, \quad X_t : \Omega \to \mathbb{R},$$

is called a random process [42]. The index set T can be used to represent some measure, e.g. time or space, and is classified as being at most countable or uncountable, in other words, discrete or continuous. The feasible set of values of X_t is called the set of states and is noted as $S \subseteq \mathbb{R}$. Similarly, the set of states S can be discrete or continuous.

A point process is a random process with a continuous index set T and a discrete set of states S. One of the point processes of great importance is a Poisson point (counting) process, which, thanks to its properties, has become a common tool for mathematical modelling [43].

We say that $\{N_t \mid t \in [0, +\infty)\}$ is a Poisson point process with intensity λ if

- $N_0 = 0$ almost surely,

- $N_t - N_s \sim \text{Poisson}(\lambda(t-s))$ for all $t \geq s \geq 0$
- N_t has independent increments, i.e. $\forall k \in \mathbb{N}$ and for all $0 \leq t_0 \leq ... \leq t_k$

$$N_{t_1} - N_{t_0}, N_{t_2} - N_{t_1}, ..., N_{t_k} - N_{t_{k-1}} \text{ are independent.}$$

As stated earlier, the final selection of inducing points is performed using two random processes; the prior point process and the variational Poisson point process (PPP). The prior point process employed in the inducing points selection is supplied with a candidate inducing set $\mathbf{Z}_c$ randomly selected from the training data and a rate α. The α value should express our estimate or belief of how many points the inducing set should contain. Higher α values result in stronger pruning, thus the number of inducing points that remain at the end of the training is lower. Another way of looking at α is to see it as a tool for adjusting the computational complexity as desired. The goal of the PIPS model is to recognise a subset $\mathbf{Z} \subseteq \mathbf{Z}_c$ that is informative while reasonable in size. Favouring a smaller number of inducing points is ensured by assigning a probability to each subset according to the squared cardinality of $\mathbf{Z}$ [41] in the following way

$$p_\alpha(\mathbf{Z}) = Ce^{-\alpha|\mathbf{Z}|^2},$$

where C is an optional normalisation term unused in the PIPS implementation[1]. The authors also suggest that squared cardinality is only one of several possible ways of penalising the size of the inducing set.

The PPP assigns each point in the predefined candidate inducing set $\mathbf{Z}_c$ a probability of inclusion based on its affiliation to $\mathbf{Z}$

$$q_\lambda(\mathbf{Z}) = \prod_{\mathbf{z}_k \in \mathbf{Z}} \lambda_k \prod_{\mathbf{z}_k \notin \mathbf{Z}} (1 - \lambda_k),$$

with $\lambda = \{\lambda_k\}_{k=1}^K$ where $K = |\mathbf{Z}_c|$. The assignment of respective probabilities is carried out repeatedly in iterations as the model gradually learns which points explain the data well. At the end of the procedure, the inducing points with zero probabilities are removed from the inducing set.

To sum up, the prior point process is responsible for assigning probabilities to sets of inducing points, while the PPP is responsible for assigning probabilities to the individual points. This approach allows the model to pick an inducing set based on the nature of the data. The model utilising the PIPS technique employs the bound from the SVGP and augments it by adding a KL divergence term $\text{KL}(q_\lambda(\mathbf{Z})||p_\alpha(\mathbf{Z}))$ between the two point processes, which

[1] https://github.com/akuhren/selective_gp

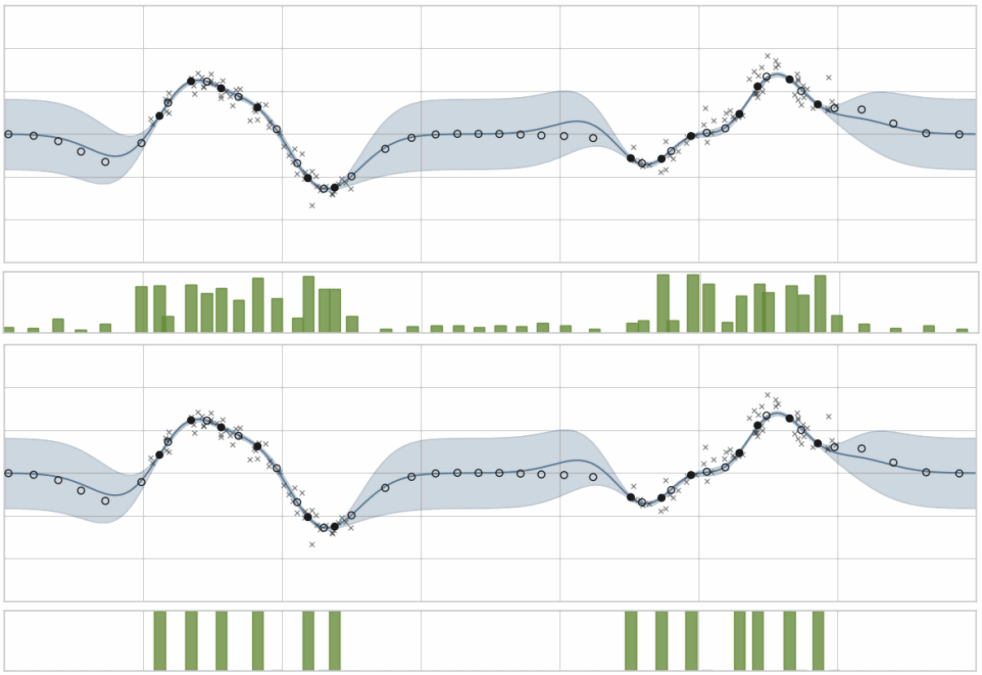


Figure 3.4: The PIPS prediction (light blue), the initial inducing set (black dots and circles) and the final inducing set (black dots) visualised. Each inducing point is associated with a probability of inclusion (green) that is set by the model, as presented in [41]. The upper part of the figure shows the probabilities assigned to the inducing points during the training and the lower part of the figure shows the probabilities of those inducing points that remained after the training and best explain the observed data (grey crosses).

is closed-form computable [41]. Such a constructed bound can be optimised through score function estimation. The score function is defined as a gradient of the log-likelihood function.

The training of the model has proven to be efficient when carried out in three phases. The initial inducing set is created by random selection from the training data while respecting the inducing set size set by a user. Then, the first training stage is conducted without the use of the PPP. The second, main phase, is executed by including the PPP in training. Ultimately, the uninformative points with zero probability are completely removed, and the model is additionally trained with the final inducing set without the PPP. Note that the so-called pre-training and post-training phases are as important as the main phase and should not be omitted. Lastly, the original PIPS implementation builds upon the SVGP, but it is possible to be employed in other sparse approximation models as well.

3.4 Other Research Directions

As the selection of the GP models presented in this chapter suggests, global approximations are a common choice to base novel ideas on. However, posterior approximations are not the only group of models growing in size. For example, some GP models were inspired by the prior approximation FITC [44] or were developed by combining global and local approximations into a single framework [45].

Some researchers were highly motivated by the historical application of various Gaussian models to non-Gaussian problems. This approach seemed promising, also thanks to some well-known use cases, such as Kalman filters for the Apollo program [46]. Hence, another research direction covers attempts to bend the GP model, trick it into forgetting its Gaussian nature and use it to model non-Gaussian data.

Furthermore, a popular field of application is modelling hierarchical data using multilayer GPs with a hierarchical structure. With enough computational resources, the whole model can learn faster when training the GPs in the same hierarchical levels in parallel. Besides, it is possible to employ different approximation types and thus utilise GPs with desired properties at each level. Nevertheless, one issue that needs to be considered is dividing data into chunks for the GPs within one level. The data division can be performed using some clustering technique of choice, which can improve along with the model itself, as seen in [45].

Up to this point, we have considered only single output GPs. Nonetheless, a large number of multioutput regression problems gave rise to Multioutput Gaussian Processes (MOGPs), which became popular and largely studied [47]. One fundamental and valuable property of MOGPs is their ability to leverage information from one output to predict another. The MOGPs use correlations between the outputs to deliver more reliable predictions than obtaining them with one GP for one output. Even though the correlations improve the prediction quality, they put an even bigger computation burden on an already computationally expensive model. When using standard GP, the computational complexity $\mathcal{O}(n^3)$ given for single output datasets changes to $\mathcal{O}(n^3p^3)$ for datasets with p outputs. Similarly to training on large datasets, there is a need for approximate modelling. To put this scenario into a more general machine learning context, sharing valuable information across outputs leads to MOGPs being categorised as transfer learning methods [47].

Chapter 4

Experiments

In this chapter, we describe the experiments that were carried out to meet the objectives of this thesis. We provide a detailed description of the experimental design and the obtained results and offer possible interpretations of the findings.

One of the aims of this thesis is to experimentally evaluate the predictive performance and computational complexity of scalable Gaussian Process (GP) architectures trained on several publicly available datasets. The chosen models include standard GP as well as state-of-the-art approximations such as VFE, FITC, SVGP, SSGP, OIPS and PIPS. The motivation for choosing these methods is that they are among the most feasible and state-of-the-art approaches for scalable GP architectures. By conducting the experiments, we aim to gain a better understanding of the performance and capabilities of these methods in real-world applications. The results of these experiments will be presented both numerically and visually in Section 4.3.

This chapter not only assesses the performance of selected GP architectures on regression tasks, but also investigates their effectiveness in the context of simulated Bayesian optimisation, which is another established goal of this work. The GP models chosen for this task are be a subset of those used in the regression task, and the reasoning behind this choice is be provided later in the thesis. The demonstration of this scenario is supported by several visualisations along with the results of the conducted experiments in Section 4.4.

In addition to detailing our experiments, this chapter also includes information about the datasets and tools used. In Section 4.1, we provide a thorough description of the datasets employed in our study. In Section 4.2, we outline the software and hardware that were used to conduct the experiments. By providing this information, we aim to make our procedure and results transparent and replicable.

4.1 Datasets

The experiments for the regression task were conducted using three publicly available datasets: kin40k [48], 3D Road Network [49] and Airline dataset [36]. All of the datasets are multidimensional, with one target variable, and contain thousands of samples. The datasets were used in a way to be as comparable as possible with other publications employing them. The fourth dataset was manually assembled for the Bayesian optimisation task.

The kin40k dataset is a widely used benchmark for GP regression models [33, 48, 50] and consists of 40000 samples describing the position of a robotic arm. The data collected from the control input have 8 input dimensions. We obtained the data from a repository published at GitHub[2]. Employment of this dataset in several studies allows fine comparison with the results obtained in our experiments.

The second dataset we experimented with is the 3D Road Network that comes from the UCI repository[3], which contains many datasets used in the machine learning community. The dataset contains more than 400000 very precise measurements of the height above the sea level covering the region of North Jutland in Denmark. It is stated that the dataset is especially appropriate for tasks such as eco-routing and constructing cyclist routes, but it is also suitable for regression tasks. We use longitude and latitude as predictors and altitude as the predicted variable, but exclude the ID of the landscape segment.

The Airline dataset has almost 2 million flight records, making it the largest dataset we work with in terms of sample size. The records contain detailed information about every commercial flight in the USA from January 2008 to April 2008. The Airline data were introduced in a paper demonstrating Gaussian Processes for big data [36] in regression task predicting delay, which is the difference in minutes from the original planned arrival. The authors of the paper made the dataset available via the pods software[4] and thus enabled other researchers to replicate the experiments accurately [37]. By using the default settings in the download function, you will get a dataset with 700000 training and 100000 testing records, selected randomly from the larger dataset. This data matches the one used in the original paper. This dataset also has 8 input dimensions which represent the month, day of the month, day of the week, departure time, arrival time, time spent in the air, distance to travel and the age of the aircraft.

[2] https://github.com/trungngv/fgp

[3] https://archive.ics.uci.edu/ml/datasets/3D+Road+Network+%28North+Jutland%2C+Denmark%29

[4] https://github.com/sods/ods

Table 4.1: Categorisation of the Gaussian Process models based on frameworks.

Framework	Pyro				GPflow		GPyTorch
Model	GP	VFE	FITC	SVGP	SSGP	OIPS	PIPS

The last two datasets have been additionally preprocessed to have zero mean and unit variance. Standardisation was performed independently on each input and output variable. This operation is performed by computing the relevant means and standard deviations using exclusively the training samples and then subtracting the mean and dividing by the standard deviation to ensure the described properties. Even though standardisation is not absolutely necessary, it is deemed to be a standard procedure [30, 33] and good practice for numerical reasons, as argued in [51]. No further adjustments were made to the kin40k dataset since all features, including the target variable, are spread around zero and have a unit standard deviation.

The synthetic dataset used for the Bayesian optimisation task is generated from a one-dimensional function sampled from a multivariate normal distribution with a zero mean and a covariance matrix created using SE kernel with $\ell = 0.6$ and $\sigma = 5$. Each evaluation of the function f at an arbitrary point $\mathbf{x}$ is corrupted by additive Gaussian noise in the following manner: $\mathbf{y} = f(\mathbf{x}) + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, 0.02^2)$.

4.2 Technologies

This section briefly describes the software and hardware used for the purposes of design and evaluation of the experiments. All computations regarding regression experiments are run on a computer with an Intel Xeon E5-2650 v4 processor with 24 cores/48 threads, 270 GB available RAM and NVIDIA Tesla V100 graphics card with 16 GB of memory. The computations needed for the Bayesian optimisation task are run on a computer with IntelCore i9-9900K processor with 8 cores/16 threads, 64 GB available RAM and GeForce GTX 1080 Ti graphics card with 11 GB of memory. The code for evaluating the experiments is written in Python 3.7 [52], which provides the necessary tools for numerical computations and data handling. The choice of this version was based on the version of Python used in the implementation of the PIPS model.

Regarding the standard GP, VFE, FITC and SVGP models, we use the implementations available in a GP dedicated module within the Pyro probabilistic language[5], which is built upon the PyTorch framework [53]. For the rest of the selected models, we use original implementations from their authors.

[5] https://pyro.ai/

Table 4.2: Number of data samples in training, validation and test set used in fine-tuning experiments.

Training data	Validation data	Test data
15000	7500	7500

The SSGP model was implemented using a TensorFlow[6] based library called GPflow [54], and the implementation of the PIPS model utilises the GPyTorch [55] library with a PyTorch backend. The summary of models and frameworks they are implemented in is provided in Table 4.1. The standard GP model is in the provided overview abbreviated and referred to as GP.

4.3 Regression Task

This section describes the details of the conducted regression experiments, the metrics used to evaluate the models' performance and the obtained results. All numerical results are presented in tables, and their interpretation supported with several visualisations.

4.3.1 Design of Experiments

The experiments described in this section are divided into subsections primarily according to the frameworks in which the GP models are implemented, as shown in Table 4.1. The SSGP model and its expansion using OIPS each have their own dedicated section.

Despite being outlined in separate sections, the regression experiments all share the same data. The sizes of the datasets were chosen with the time and computational limitations in mind, while still being considered large. Concerning Gaussian Processes, a large dataset consists of at least a few thousand data points [36]. Each dataset was split into the train, validation and test set. Sizes of each set are invariant with respect to datasets and are presented in Table 4.2. The subsets were created by randomly sampling the data in a way that ensures there is no overlap between them.

All of the experiments conducted in this study followed the same two-phase structure, with some models potentially undergoing additional experimental stages. During the fine-tuning phase, the models were trained until their performance on the validation data stopped improving, at which point the training process was terminated. The final follow-up stage involved evaluating the best-performing models using the test set. The selection and evaluation of these models will be discussed in greater detail in Section 4.3.2. This

[6] https://github.com/tensorflow/tensorflow

section focuses on the evaluation phase and the metrics used to assess the performance of the models. Additional experimental stages beyond these two phases, if applicable, will be discussed in further detail in the relevant sections of the thesis.

4.3.1.1 Pyro GPs

The Pyro library was utilised to deploy the standard GP and three benchmark state-of-the-art approximations VFE, FITC and SVGP. We used the SVGP in both full and batch training regimes, giving us a total of 5 models to train on 3 datasets, resulting in 15 scenarios overall.

We divided our experiments with Pyro GP models into three parts and conducted them in consecutive order. Therefore, in contrast to the experiments with other models, we included an additional stage prior to the common experimental stages conducted for all regression experiments. The first part, the coarse experiments, used smaller, randomly selected data sets that did not overlap with the data used in the fine-tuning experiments to explore the hyperparameter space. The second part, the fine-tuning experiments, used the findings from the coarse experiments to train models on the data described earlier in Section 4.3.1. The best of these models are then evaluated in the final stage of the experiments, which is conducted in the same manner for all regression experiments. The described experimental flow is visualised in Figure 4.1.

To begin, we perform coarse experiments for several options of each model setting. However, a vast number of possible settings regarding the training and the model itself suggests a need for an enormous number of experiments in each scenario. Performing an exhaustive search in hyperparameter space with numerous options for each setting, such as trying out five various optimizers and ten different learning rates, is impossible within the scope of this thesis due to limited time and resources. With countless options of settings' values available, we acknowledge that the number of combinations we try out is relatively small, but we do not believe this harms the overall message of our results. However, we want to make it clear to the reader that given the constraints of this work, we have not rigorously explored the hyperparameter space.

To give an example, in spite of being aware of many different kernels and the possibility of combining them, we settled on employing no more than three: SE, Matérn 3/2 and Matérn 5/2. Besides, the kernel's parameters, lengthscales and variances, need to be selected as well. The selection of these parameters can be made using various techniques, e.g. heuristics or drawing values from different prior distributions. At the same time, some of the strate-

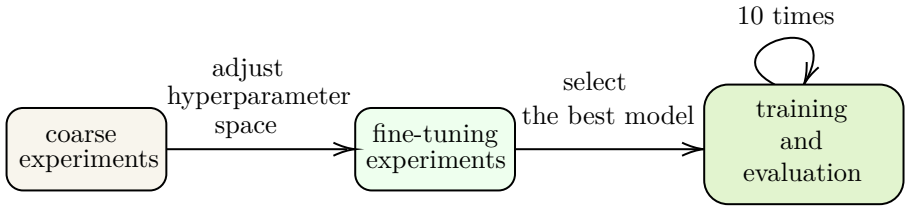


Figure 4.1: Schematically summarised flow of experiments for Pyro models.

gies for the parameter selection are not always efficient, as was shown in [23], where the authors observed that priors of the parameters have only a little impact in relation to GP models trained on real-world regression datasets. Thus, we opted for the simplest option and decided to use default kernel parameter values.

In addition to settings that all models have in common, sparse models have more parameters to be set, such as settings related to inducing points. Not only their number but also the locations of the initial inducing points can play a crucial role in the model's capacity and performance. Regarding the number of inducing points, we followed the strategy presented in [33] and tried out sizes of inducing sets changing in powers of two ranging from 2^7 to 2^{13}. The upper bound was determined with regard to the training set size, which never exceeded ten thousand samples. The initialisation of the locations was performed with a simple grid in order to save computational resources in comparison with more elaborate techniques such as k-means or Latin hypercube sampling.

We split each data chunk for the coarse experiments into a training and test set and trained the models via variational inference for a fixed number of epochs, which were set to be large enough to converge on the given dataset. As mentioned in the theoretical part of this thesis, variational inference typically uses ELBO loss during its computation. The Pyro framework offers several different modifications of ELBO that allow the losses and gradients to be computed in different ways. We employ two types of losses, one of which is `Trace_ELBO`, the most commonly used within this framework. The second one is `TraceMeanField_ELBO`, the only one in Pyro that uses analytic KL divergence terms when available, which will be later referred to as `TMF_ELBO`. Regarding the optimisers, we opted for two commonly used options, Adam and RMSprop. We test out four different learning rates ranging from 0.001 to 0.01 for both of these optimisers. Additionally, since GP models are prone to numerical instability, we utilised jitter added to the diagonal elements of the covariance matrix. If training fails even after adding jitter, we try to increase it and train the model again. This attempt is repeated at most two times.

In conclusion, the evaluation of the test data chunk helped us identify less promising settings values for each model and determine which settings have little or no impact on their performance. For instance, coarse experiments helped reveal that kernel type often has negligible influence on performance. Therefore, we opted for the SE kernel as a default choice for fine-tuning experiments. Regarding the inducing set, sparse approximation models FITC, VFE, and SVGP for the kin40k and the 3D Road Network datasets performed better with a higher number of inducing points. However, the models trained on the Airline dataset behaved differently, indicating that a smaller size of the inducing set may be preferable. Therefore, the domain of inducing set size was only slightly altered for fine-tuning experiments.

The influence of the values of the hyperparameters not mentioned in the previous paragraph, which summarised several findings, did not remain undiscovered. To sum up, we thoroughly examined all of them and adjusted the domain of each hyperparameter for each model based on its results across the three datasets. The modifications made to the hyperparameter space and the code for the coarse experiments are part of the repository belonging to this thesis.

Another observation made in the course of the first experimental part is that the results obtained from the SVGP model using batch training were mostly worse than those produced by other models. In an effort to improve the results, we decided to try out an alternative strategy that involved taking more gradient steps per batch rather than adhering to the standard approach of making only a single gradient update per batch [38] as was also presented in Pyro's demonstration of the SVGP usage[7]. This alternative approach proved to be slightly or significantly worse depending on the dataset and is not employed in the fine-tuning experiments.

Finally, we performed the fine-tuning experiments. In these experiments, we limited the possible initialisation and training configurations to a smaller set of hyperparameters based on the findings from the coarse experiments. This allowed us to focus on potential model configurations that were more likely to yield successful results. Even with the constrained hyperparameter space, the number of models trained is still significant. For each dataset, we train 2 standard GPs, 16 VFE and FITC each, 20 SVGPs and 50 SVGP* models.

[7] https://docs.pyro.ai/en/stable/contrib.gp.html

4.3.1.2 SSGP

The authors of the SSGP made the implementation of the model publicly available in a GitHub repository[8]. However, the versions of libraries stack primarily used for the implementation, GPflow and Tensorflow, was too old and could not be used. The GPflow version 0.4.0 used for the implementation is not backward compatible with the latest version and, to the best of our knowledge, is no longer available.

For the purposes of direct comparison and consistency, we decided to implement the SSGP model in the Pyro framework. The implementation was based on the equations given for the theoretical bound. Unfortunately, the performance of the SSGP rewritten to Pyro was very poor due to frequent training failures. Every attempt of training the model on the 3D Road Network dataset failed even when trying more settings than presented in the paper. For instance, all the demonstrated experiments employed SE kernel, while we also used Matérn 3/2 and Matérn 5/2, all without success. The performance of the model on the other two datasets was also highly unsatisfactory.

After a repeated and thorough examination of the equations and the original code, we thought we discovered a mistake in the equation for predictive covariance. Our suspicion was confirmed by the authors in a publicly available issue[9] that we submitted to their repository. The correct version can be found in Appendix B.1. After fixing this error in the code, the performance of our reimplementation slightly improved but was still inferior.

The SSGP model in Pyro was additionally partially rewritten based on the practical bound presented in B.2 as an effort to increase its numerical stability. The practical bound emerges from the theoretical one by adjusting terms to obtain desirable forms for terms to cancel out or be calculated easily and more stable. Therefore, the practical bound is more compact and cannot be split into parts that clearly belong to the distribution and the two regularisers. Since the Pyro framework requires separation of regularisation terms and terms related to the distribution, we considered fully rewriting the model to be unattainable. The partially numerically optimised SSGP Pyro model proved to be more stable than the model rewritten solely based on theoretical bound, but the number of training failures due to numerical errors was still significant and the performance very poor.

After all, attempts to stay within one framework proved to be both incredibly time-consuming and challenging. The plan to have SSGP implemented in Pyro was abandoned in favour of rewriting the original implementation with

[8] https://github.com/thangbui/streaming_sparse_gp
[9] https://github.com/thangbui/streaming_sparse_gp/issues/5

newer versions of the libraries. The model was rewritten with the minimum changes possible to transfer to an upgraded version and was supplemented by enhancement of the former brief documentation. The intention to later contribute to the original repository with our reimplementation turned out to be unnecessary. Surprisingly, after approximately four years of inactivity, the authors published the upgraded version in October 2022. We found out about this latest contribution to the repository only a few days after it was released only by accident when reviewing the correctness of the training procedure that we replicated based on the published experiments. However, since we were already familiar with the changes that we made in our newer SSGP, tested its functionality and already started with our experiments, we decided to keep using our reimplementation. The investigation of their code has shown only very few changes were made to switch to newer versions and confirmed its similarity with our reimplementation.

Since we utilised principally the same implementation as used in the paper, we took inspiration from the conducted experiments and training procedure used by the authors. Some of the settings were fixed and used throughout all the demonstrated experiments with both synthetic and real-world datasets, and we did not see a reason to do it another way. We believe that the settings and variational inference used across all experiments by the creators of the model themselves would not be of any limitation for a fair comparison. Adapted techniques include utilising only the SE kernel or using 1000 training data samples as an initial training set for pre-modelling with the VFE to obtain reasonable initial settings for the SSGP. The pre-modelling is considered to be a part of the whole training procedure and is included in the time measurements of the SSGP algorithm. It should be stressed that there might be better suited values of the settings regarding performance optimality. Also, in order to make all the employed GP models perfectly comparable it would be required using the same or akin frameworks and as much as possible overlapping hyperparameter space in every case.

The parameters that we experimented with are batch size and the number of inducing points. The considered range of these parameters match the default sets for coarse experiments. Specifically, default values for batch size range from 200 to 1000 with a step size of 200, and the number of samples in an inducing set range from 2^7 to 2^{13} changing in powers of two. Accordingly, we train $5 \times 7 = 35$ models for each dataset. As employed in all fine-tuning experiments, the convergence of the models is monitored using validation data. However, the maximum number of epochs is set lower compared to other models since the SSGP was designed for a streaming setting where the data can be processed only once. Accordingly, the introduced training procedure includes very thorough processing and learning from each batch and should perform well after a single epoch.

4.3.1.3 OIPS

Another group of experiments with the SSGP model was performed in combination with OIPS extension. Such an extended model does not have a fixed size of the inducing set. However, there is a need to set an initial size, for which we try values from the same range as from the original inducing set range, i.e. increasing powers of 2. The initial size is set at the beginning of the training procedure and remains unchanged during pre-modelling with the VFE model.

Once the training of the SSGP model starts, the inducing set grows based on data and model characteristics and a threshold parameter ρ. Its domain is set to $\{0.3, 0.6, 0.9\}$, where larger values cause the inducing set to grow more rapidly. However, it needs to be stated that models with higher ρ values are more prone to training failure due to insufficient memory capacity compared to the rest of the employed GPs. The selection of the particular points added to the current set of inducing points is implemented based on the pseudocode available in [40]. The number of SSGP models enhanced by OIPS that we test out is for each dataset equal to $5 \times 7 \times 3 = 105$.

4.3.1.4 PIPS

The last model utilised in the regression experiments is PIPS. We followed the experimental setup proposed in the paper [41] and used the same values for the settings fixed in all demonstrated experiments as the authors described in the appendix dedicated to their design. We adapted settings such as kernel type, optimiser and learning rates. Since the training is split into three phases which include pre-training and post-training, early stopping is not included in the training procedure. The number of epochs is set individually for each training phase. We utilise numbers of epochs for each of the three stages presented in the paper for experiments carried out on real-world datasets.

The settings we experimented with include the initial size of the inducing set, α value and the inducing points prior probability of inclusion. Again, the initial sizes of the inducing set are the same as the default inducing set sizes. The domain of the prior probability of inclusion set before the second training phase and the domain of the α value are limited to $\{0.3, 0.6, 0.9\}$ and $\{0.1, 0.2\}$, respectively. Altogether, we train $7 \times 3 \times 2 = 42$ PIPS models for each dataset. We fully utilised the original implementation available in a publicly available GitHub repository[10].

[10] https://github.com/akuhren/selective_gp

4.3.2 Evaluation

In the course of the regression experiments, we evaluated the performance of the model using three metrics. The first technique used to measure the prediction quality is Root Mean Squared Error (RMSE), which is commonly employed for evaluating the error of regression models. The RMSE metric is computed as

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2},$$

where $\hat{y}_i$ is a predicted value, y_i is a true value and n is the number of predicted samples. To put the above formula into words, RMSE is a square root of Mean Squared Error (MSE) computed as the average value of squared residuals between the estimated and the true values.

Another metric used for the evaluation of the GP models is Standardized Mean Squared Error (SMSE), which is obtained by dividing the MSE by the variance of the true target variable values. Using SMSE rather than MSE is sometimes the preferred option since MSE is sensitive to the range of values that the target variable can acquire [10].

However, both previously mentioned metrics ignore that we are also provided with the model's uncertainty in its prediction. The third performance measure utilised for the evaluation considers that the regression models used in this thesis are probabilistic, and thus they do not output only predictive mean $\hat{y}$ but also predictive standard deviation $\hat{\sigma}$. The Mean Standardised Log Loss (MSLL) value can be understood as the average value of the negative log likelihood of predicted samples under the model [10]. The expression for MSLL is given as

$$\text{MSLL} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\log\left(2\pi{\hat{\sigma}_i}^2\right) + \frac{(y_i - \hat{y}_i)^2}{2{\hat{\sigma}_i}^2}.$$

In fine-tuning experiments, each type of model is trained on various setting combinations from hyperparameter space. Afterwards, the models that take longer than one day to train are filtered out, and the rest is sorted based on RMSE and MSLL in this exact order. For each of these metrics, a lower value means a better fit. In case of multiple models being equally good, the models are additionally sorted based on the time they needed to converge. The model with the best metrics values and lowest training time is repeatedly trained with the same settings. Its final performance is obtained as an average of these fits.

Another indicator of the goodness of the fit employed for informative purposes is the percentage of data points that lie within two standard deviations $\hat{\sigma}$ of the

mean $\hat{y}$. For any given Gaussian distribution, data points within the bounds should cover approximately 95% of the data. We treat this performance measure as a metric, though it is actually more of an empirical heuristic.

4.3.3 Results and Discussion

This section presents the procedure used to obtain the results and their overview in tables along with visualisations and their possible interpretations. The best model selection for each GP type was accomplished based on the performance achieved on the validation data in fine-tuning experiments. Finally, the best-performing models were trained ten times and evaluated on the test data. Their hyperparameters are demonstrated in Appendix D. The main objective of the experiments was to examine the quality of each model's predictions and detect GPs with superior performance.

Performances and training times measured for the best models are for each dataset considered available in Tables 4.3 to 4.5. Each cell represents the mean value for the corresponding metric calculated by averaging the results obtained from repeated training and evaluation of the best model. For formatting purposes, in every table and visualisation demonstrating the models' performance, the standard GP will be referred to as GP, SSGP with OIPS extension simply as OIPS and SVGP in batch training regime as SVGP*. In addition, the best value within each metric is highlighted. The tables' versions extended with standard deviation measurements for each of the evaluation criteria are presented in Appendix C.

Firstly, let us present the results obtained for the kin40k dataset used commonly as a benchmark in GP literature. It is of no surprise that the standard GP model is superior in RMSE, SMSE and MSLL metrics. On the other hand, it is also one of two models that required significantly more training time than the rest. Regarding the percentage of samples covered within the prediction bounds, the computed value is very close to the desired 95%, but the SVGP model performed even better. Also, the SVGP took the least amount of time to train. In terms of RMSE, SMSE and MSLL, the second-best model is the VFE, and the third-best is the FITC. Both of these models required less than a third of the time that was needed to train the standard GP. The model with the poorest performance according to each criterion is undoubtedly PIPS. The performance of each of the eight GPs is summarised in Table 4.3.

Experiments with the 3D Road Network dataset have also shown that the standard GP model has the best prediction quality, as demonstrated by its superior performance in terms of RMSE, SMSE, and MSLL metrics, as well as the higher percentage of samples within its prediction bounds. Although the VFE and OIPS models have slightly lower prediction quality than the

Table 4.3: Mean metrics values obtained by repeated training and evaluation of the best models on the kin40k dataset.

Model	Performance measure				Training time (s)
	RMSE	SMSE	MSLL	Covered (%)	
GP	**0.0971**	**0.0093**	**-1.0448**	94.2133	22309.2132
VFE	0.1109	0.0122	-0.8362	96.3413	6045.4770
FITC	0.1303	0.0168	-0.7403	98.6853	702.2540
SVGP	0.1641	0.0267	-0.7093	**95.0493**	**432.1855**
SVGP*	0.1911	0.0361	-0.1253	99.5840	553.3067
SSGP	0.1725	0.0294	0.0973	99.9320	9859.5440
OIPS	0.1385	0.0190	-0.1563	99.7413	26689.6728
PIPS	0.2130	0.0449	-0.0265	87.9413	557.6066

Table 4.4: Mean metrics values obtained by repeated training and evaluation of the best models on the 3D Road Network dataset.

Model	Performance measure				Training time (s)
	RMSE	SMSE	MSLL	Covered (%)	
GP	**0.2719**	**0.0729**	**-0.0039**	**94.9867**	9383.9991
VFE	0.2960	0.0864	0.1852	94.9400	6224.6407
FITC	0.4033	0.1604	0.0180	93.3627	**927.0813**
SVGP	0.4825	0.2296	0.2751	97.6893	2462.5867
SVGP*	0.4242	0.1775	0.1931	97.3787	31855.4244
SSGP	0.5221	0.2689	3.7316	81.1920	1307.2037
OIPS	0.3072	0.0931	1.8539	79.0107	6444.0499
PIPS	0.5346	0.2821	15.7997	52.7200	499.9772

Table 4.5: Mean metrics values obtained by repeated training and evaluation of the best models on the Airline dataset.

Model	Performance measure				Training time (s)
	RMSE	SMSE	MSLL	Covered (%)	
GP	**0.9305**	**0.8286**	1.3440	95.3467	5424.2358
VFE	0.9375	0.8409	1.3530	95.3107	2835.3271
FITC	0.9381	0.8421	1.1763	95.0413	466.4226
SVGP	0.9498	0.8633	1.3315	92.5520	983.9290
SVGP*	0.9568	0.8760	**1.1522**	**95.0200**	**352.0852**
SSGP	0.9419	0.8490	26.5962	42.7440	7579.0588
OIPS	0.9420	0.8492	39.6042	37.0533	687.6577
PIPS	0.9409	0.8471	58.4339	29.7493	516.5287

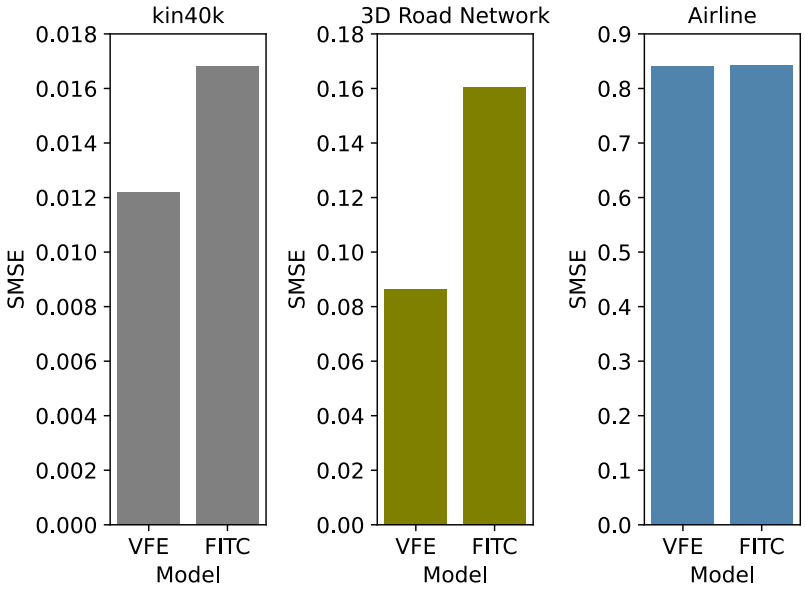


Figure 4.2: A visual comparison of the mean SMSE values obtained with the best-performing VFE and FITC models.

standard GP model, they have faster training times. In contrast, the FITC model may have the quickest training time among all the models tested, but its prediction quality falls short compared to many of the other models in our experiments. The results are summarized in Table 4.4.

It is clear from the results in Table 4.5 that the models performed worse on the Airline dataset compared to the other two datasets. Additionally, the performance of the models on the Airline dataset was relatively consistent, with the standard GP, VFE, and FITC models performing the best. While the standard GP model took longer to train, the VFE and FITC models showed similar performance to the standard GP model in a much shorter amount of time. The fastest model was SVGP*, which also had the best performance in terms of the MSLL metric and the percentage of data points covered.

Now, let's take a closer look at each model and discuss its performance and results. To begin with, it is clear that the standard GP model was the standout performer, surpassing the other models in every scenario. Although this impressive performance came at the cost of longer training times compared to the majority of the other models. This outcome was to be expected.

Generally, the second best performing model was VFE, which consistently came in second place after the standard GP model. The predictive power

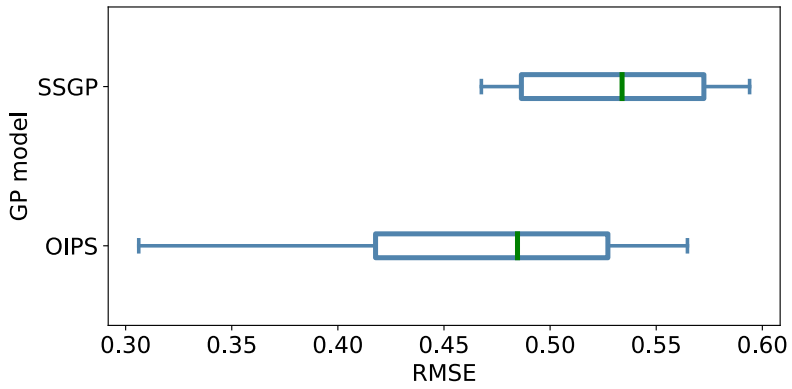


Figure 4.3: Comparison of all SSGP and OIPS models' performance after being trained for one epoch. The RMSE values are measured during fine-tuning experiments using validation data for the 3D Road Network dataset.

of VFE was almost as good as that of a standard GP, but at a much lower computational cost. Also, it has been previously observed through coarse experiments that VFE models with higher numbers of inducing points tend to perform better on the kin40k and 3D Road Network datasets, while the opposite is true for the Airline dataset, where lower numbers of inducing points tend to produce better results. These findings are supported by the sizes of inducing sets in the best VFE models, as shown in Table D.2.

In the best-performing FITC models, the sizes of inducing sets also match earlier discoveries. Based on the obtained MSLL values, it appears that the possible underestimation of uncertainty, as mentioned in Section 3.2.3, did not occur in this case. The percentages of points lying within two standard deviations from the predictive means also do not indicate any significant issues. Regarding the overall predictive power, the FITC model consistently performs worse than the VFE model, as shown in Figure 4.2. The performances vary, with some instances showing only a slight difference. Additionally, FITC tends to favour higher learning rates compared to the VFE model.

The evaluation of the results of the SVGP model showed that it did not stand out as either the best or the worst among the models considered. In terms of convergence time, it was not the fastest but also not the slowest. Interestingly, the sizes of the inducing sets of the best SVGP models matched those of the best VFE and FITC models.

Next, we will discuss the last one of Pyro models. In the Airline dataset, the SVGP* model performed the best in terms of MSLL and the percentage of points covered, and was the fastest among all models. However, in the 3D

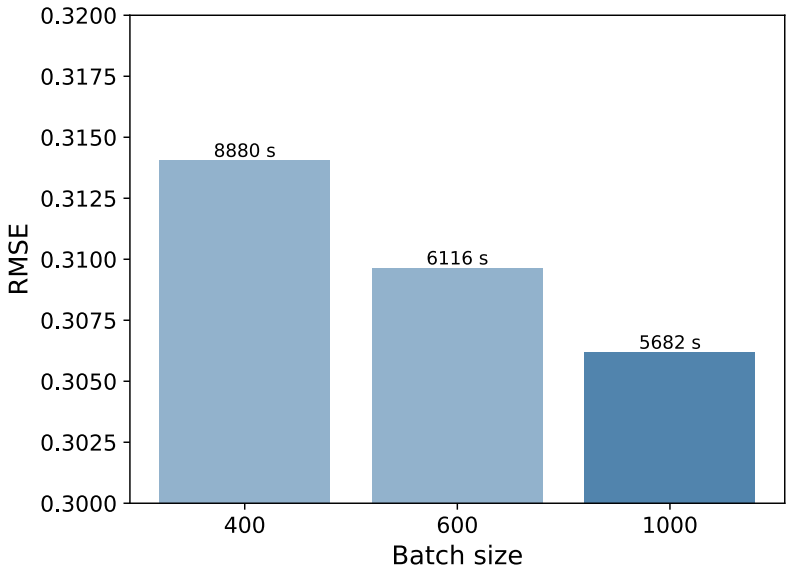


Figure 4.4: Comparison of the best-performing OIPS model (dark blue) with OIPS models with the same settings except the batch size. The RMSE values are measured during fine-tuning experiments using validation data for the 3D Road Network dataset.

Road Network and kin40k datasets, the SVGP* model was among the worst performers. Additionally, its training process took an exceptionally long time on the 3D Road Network dataset compared to the other models. The differences in the number of inducing points for different datasets were particularly pronounced for the SVGP* model, as can be seen in Table D.5.

The SSGP model, similar to the SVGP model, generally performed at a mediocre level, neither the best nor the worst among the models considered. As shown in Table D.6, the hyperparameters of the best models used in the final repeated evaluation do not follow a clear pattern. Both the inducing set and batch sizes varied, ranging from smaller to larger values.

Since the OIPS model builds upon SSGP, it would be reasonable to expect its superiority. This assumption has turned out to be true in two times out of three. The OIPS model performed worse than SSGP only when predicting the Airline dataset, though the difference is not significant. The RMSE values computed for all SSGP and OIPS models trained on the 3D Road Network dataset for one epoch during fine-tuning phase are depicted in Figure 4.3. The visualisation indicates that the OIPS model is able to achieve much lower RMSE values compared to the SSGP model.

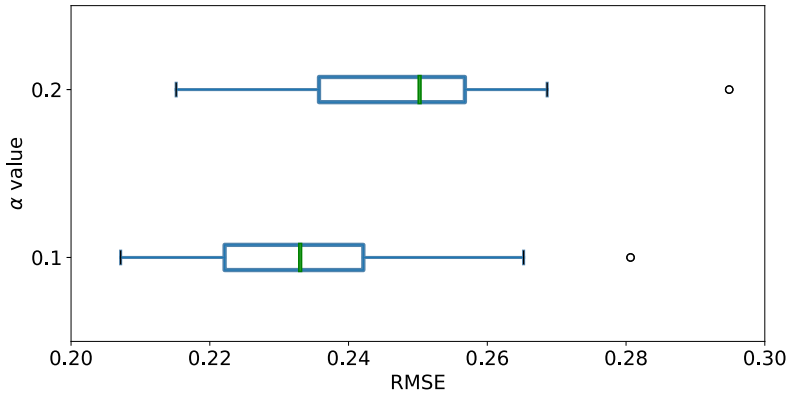


Figure 4.5: The RMSE values measured for PIPS models during the fine-tuning experiments grouped by the α value. The models were trained on the kin40k dataset.

The hyperparameter values of the best-performing OIPS models, shown in Table D.7, indicate that it performs better with higher batch size values, which we tested in the range of 200 to 1000. This assumption is supported by Figure 4.4 , which shows a comparison of the best-performing OIPS model with OIPS models that have the same settings on the Airline dataset, except for the batch size. Some batch sizes are not shown due to training failures. The values of the hyperparameter ρ show a trend, indicating that the OIPS model performs better with larger ρ values. Additionally, the OIPS model appears to prefer very small initial inducing set sizes, such as 128 and 256. Even when starting only with a small number of inducing points, the final inducing set size for the kin40k dataset exceeds 2300 points, 900 points for the Airline dataset and 4600 point for the 3D Road Network dataset.

The last GP model that we experimented with in the regression experiments is PIPS. Based on the results from the fine-tuning experiments, the hyperparameter settings chosen for the final repeated training and evaluation of PIPS models demonstrate that some hyperparameter values are generally better than others. For every dataset, the best-performing models have α equal to 0.1. All models that were trained during the fine-tuning experiments for the kin40k dataset are split into two groups based on the α value, and their performance in RMSE metric is visualised in Figure 4.5. The RMSE values obtained from the evaluation of the validation data show a notable difference between the two groups and demonstrate a significant influence of the α hyperparameter. Furthermore, PIPS models performed better with smaller inducing set sizes and bigger prior probabilities of inclusion. The influence of the different values used for the prior probability of inclusion is demonstrated for the kin40k dataset in Figure 4.6. The overview of all three hyperparameter

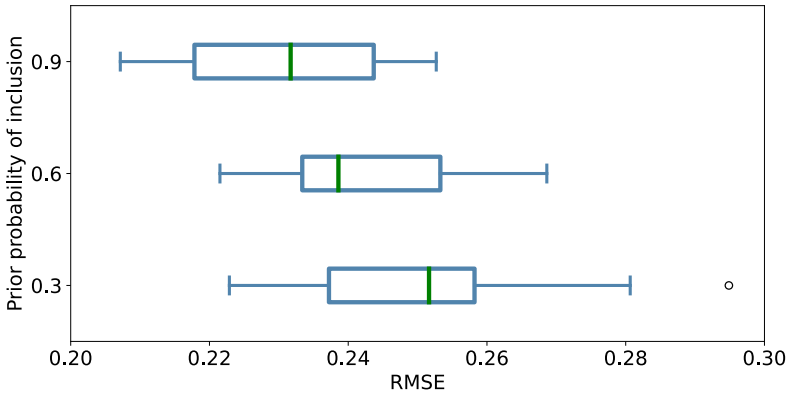


Figure 4.6: The RMSE values measured for PIPS models during the fine-tuning experiments grouped by the prior probability of inclusion. The models were trained on the kin40k dataset.

values used for the best PIPS models is presented in Table D.8. Regarding performance, PIPS models achieved generally very poor results in the MSLL metric. The obtained results also show that PIPS covers fewer data points than the rest of the employed models. Such inferior performance is surprising primarily because PIPS, together with OIPS, is among the latest published models employed in this work. However, PIPS models are among the models with lower training times in every case.

4.3.3.1 Trade-off between Speed and Accuracy

In this section, we discuss the performance of the GP models from the perspective of a trade-off between speed and accuracy. We analyse the training times and RMSE values in an attempt to identify the best compromise between these two factors. We show comparisons of GP models in all three datasets and, additionally few comparisons within a model type.

From the results of the kin40k dataset visualised in Figure 4.7, it is clear that the PIPS, SVGP, and SVGP* models have slightly lower training times than FITC but significantly worse performance. The VFE and standard GP models also offer a good balance between performance and time. The SSGP and OIPS models, on the other hand, provide the worst trade-offs. Overall, these results indicate that the SVGP, FITC, VFE and standard GP models form the Pareto front and thus are efficient performers for the kin40k dataset.

For the 3D Road dataset, using the PIPS, SSGP, or SVGP models would not be beneficial as they have higher RMSE values than the FITC model with very similar training times as is apparent from Figure 4.8. However, the PIPS belongs to the Pareto front as well. A sound trade-off is also offered by the

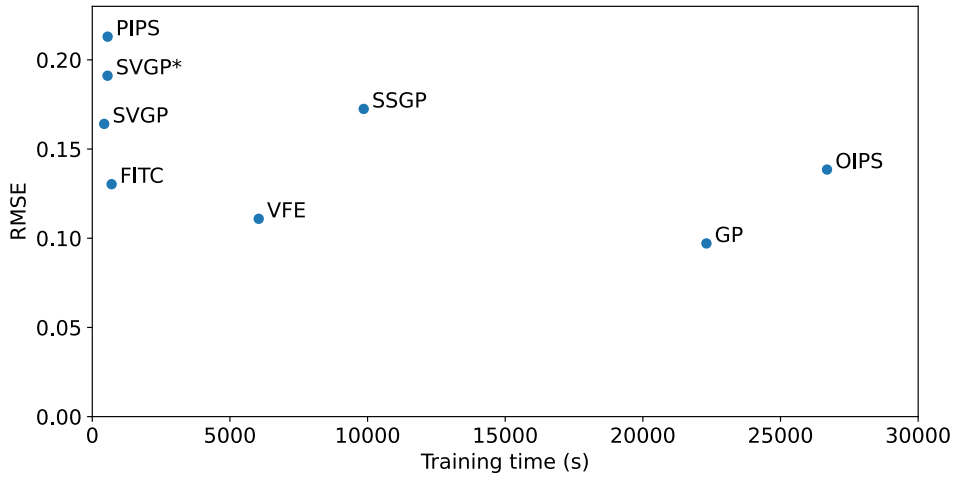


Figure 4.7: Comparison of GP models in terms of trade-off between prediction quality and training time for the kin40k dataset.

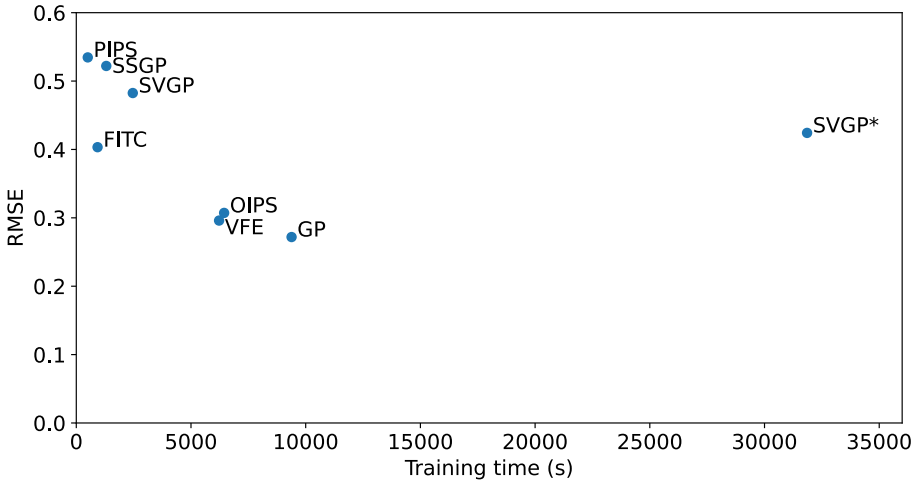


Figure 4.8: Comparison of GP models in terms of trade-off between prediction quality and training time for the 3D Road Network dataset.

VFE and standard GP models. The OIPS model is not worthwhile due to its longer training time and inferior performance compared to the VFE model. The very poor performance of the SVGP* model once again confirms that a longer training time does not necessarily lead to better results.

The Figure 4.9 for the Airline dataset shows that the differences in RMSE values are negligible, while the training time values are more diverse. Based on this visualisation, the SSGP model may not be the best choice for this

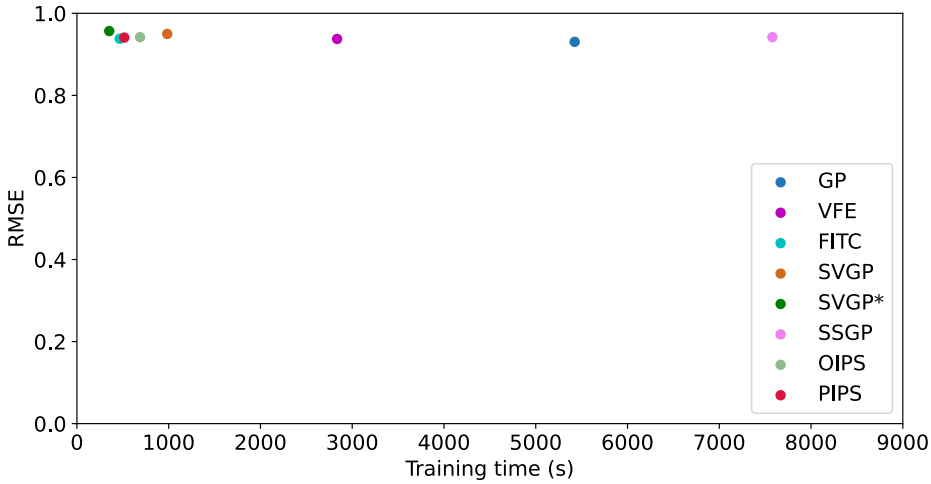


Figure 4.9: Comparison of GP models in terms of trade-off between prediction quality and training time for the Airline dataset.

dataset as it has the worst balance of speed and prediction quality among all models. When considering the numerical results in Table 4.5, we can see that the SVGP*, FITC, VFE, and standard GP models offer the highest level of accuracy for the Airline dataset within the same or shorter training time compared to the other models.

In conclusion, our results show that the Pareto front always contains the standard GP, VFE, and FITC models, which offer the best trade-off in terms of prediction quality and training speed and are superior performers compared to the other models considered in this study. We consider examining the trade-off between training time and prediction quality of GP models essential, as a model with high prediction quality but a long training time may not be practical for many applications. On the other hand, a model with a shorter training time and slightly worse performance may still be useful. By examining the models from this perspective, we can identify which models offer the best balance between training time and prediction quality and thus make informed decisions about which model to use for a given task rather than deciding solely based on their performance.

4.4 Bayesian Optimisation Task

This section covers the specifics of the experiments with several GPs in simulated Bayesian optimisation that were carried out, the methods employed to determine the sufficient quality of the found solution and the results along with their interpretation.

The problem we are addressing involves finding the maximum of a function when only a few evaluations are available initially. Since we lack the training data, the fact that SSGP, OIPS and PIPS each sample an initial inducing set from them is an obstacle. At the same time, SSGP and OIPS require data for the pre-modeling phase, and this data should not overlap with the data used later during model's training. Given that we have only units of data samples at the beginning, we cannot afford to separate a subset, let alone big enough to be able to obtain a reasonable starting point for the models during the pre-modelling phase. Therefore, due to these reasons, the experiments in this chapter are conducted using standard GP, VFE, FITC, SVGP and SVGP*.

4.4.1 Design of Experiments

The experiments presented in this section were conducted after those related to the regression task. At this point, we already had an intuitive understanding of the models and the influence of the hyperparameters' values on their performance. We exploited the knowledge gained from previous experiments with the GPs and reused the settings used for the the fine-tuning phase of regression task.

Our optimised objective function is multimodal, and we will attempt to find a data point in an immediate proximity of its maximum. Therefore, it will most likely require evaluating a large number of samples during the Bayesian optimisation run, and the training data will grow to thousands of data points, as was the case with regression datasets. For this reason, reusing the settings should not pose an obstacle. In Section 4.1, we present the optimised function alongside other datasets, and through the use of Bayesian optimisation, we constrain the function's domain to the range of 0 to 35, as depicted in Figure 4.10.

Prior experiments with GPs allowed us to obtain most of the settings that are experimented with in relation to the surrogate models easily. In the Bayesian optimisation task, it is necessary to additionally set the number of epochs for which the model is trained:

- on the initial training data,
- after each dataset augmentation.

The number of epochs used for the training on the initial dataset of size 15 is fixed at 100. All 15 data points are equally spread across the whole domain. During each iteration of the Bayesian optimisation algorithm, the model is updated by being trained on the dataset augmented with a new objective function's evaluation for 10 epochs. The exception is the SVGP* model for

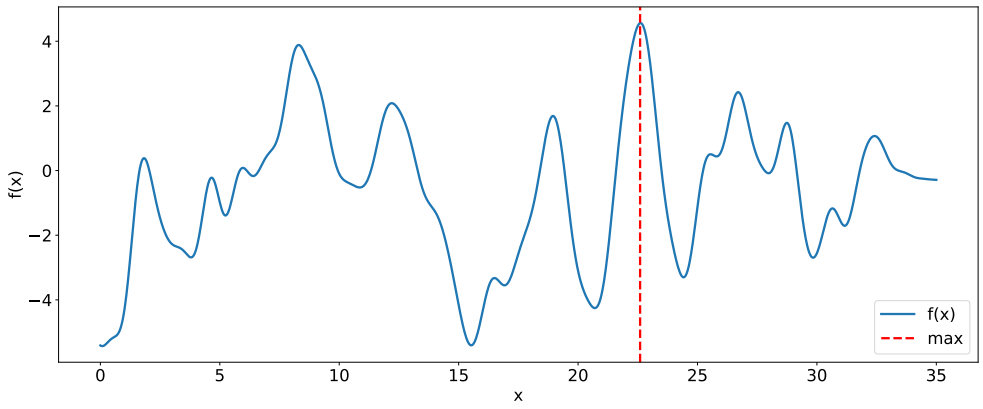


Figure 4.10: Function optimised in Bayesian optimisation task within a constrained domain.

which is the number of epochs per iteration set to 3. Moreover, as described in Section 4.1, all function evaluations are corrupted by noise by default. To sum up, the simulation is performed by gradually supplying GP models with data points from the synthetic dataset.

Still, there is a need to make design choices associated with the second key ingredient of Bayesian optimisation, an acquisition function. We decided to test out three types of acquisition functions presented in Section 1.2: UCB, PI and EI. In addition, each of the selected acquisition functions has a hyperparameter that needs to be specified. For UCB, we set $\kappa = 2$, and for PI and EI, hyperparameter ξ was fixed to 0.01 as suggested in [56].

Since we deal with maximum optimisation, the acquisition function is maximised in order to decide where to evaluate the objective function next. In a sense, we have an optimisation problem within another optimisation problem. However, there are several possibilities of approaching this. Instead of using gradient-based optimisation algorithms, we opted for two simpler strategies. The first is a random selection, and the second uses a grid with additive noise. In both cases, we generate 80 candidates for subsequent evaluation of the objective function. The selected point is the candidate with a maximum value when evaluated using an acquisition function. Repeated evaluation of the acquisition function does not constitute an issue since it is cheap to evaluate.

Altogether, with all possible combinations of setting related to the surrogate model and the acquisition function, we run Bayesian optimisation 12 times with the standard GP, 96 times with VFE and FITC each, 120 times for SVGP and 300 times for SVGP*.

Table 4.6: Minimum time and number of iterations required to find an adequate solution for the most successful runs of the Bayesian optimisation within individual models.

	Model				
	GP	VFE	FITC	SVGP	SVGP*
Time (s)	95.6946	9.0157	9.2546	11.1699	4.1552
Number of iterations	521	41	41	51	21

4.4.2 Evaluation

For Bayesian optimisation experiments, the number of evaluations of the objective function is upper-bounded. Specifically, the maximum number of iterations for each Bayesian optimisation run is set to 8000. The algorithm is stopped if the found solution is close to the maximum, which is defined as being within a distance of 2×10^{-4}. If no such solution is found, the search is considered unsuccessful. For each GP type, the best combination of the model and the acquisition function is selected from the successful runs based on the time required to find the point close enough to the true maximum.

4.4.3 Results and Discussion

In this section, we present the results of our experiments on Bayesian optimisation using selected GP models. We conducted a series of experiments to evaluate the performance of different GP models and acquisition functions in finding the maximum of an objective function. The results of these experiments allowed us to identify the most effective combinations of GP models and acquisition functions and compare their performance in terms of speed and the required number of evaluations.

Table 4.6 indicates that all GP model types except for the standard GP were able to find an adequate solution in a relatively short time. However, this does not necessarily mean that using these models in Bayesian optimisation will consistently lead to quick solution finding. On the other hand, Figure 4.11 suggests that each best solution presented in Table 4.6 is only the result of one successful setting combination among many. Additionally, it is possible that the success of these solutions is influenced by the random sampling of candidate points for evaluation. In fact, the best solutions found using the standard GP, VFE, and FITC models all employed the random sampling strategy, and they all utilised the UCB acquisition function. The solutions found using the SVGP and SVGP* models, on the other hand, were found using the EI acquisition function and the grid sampling strategy. In all of the sparse models, the number of inducing points was set to 8192.

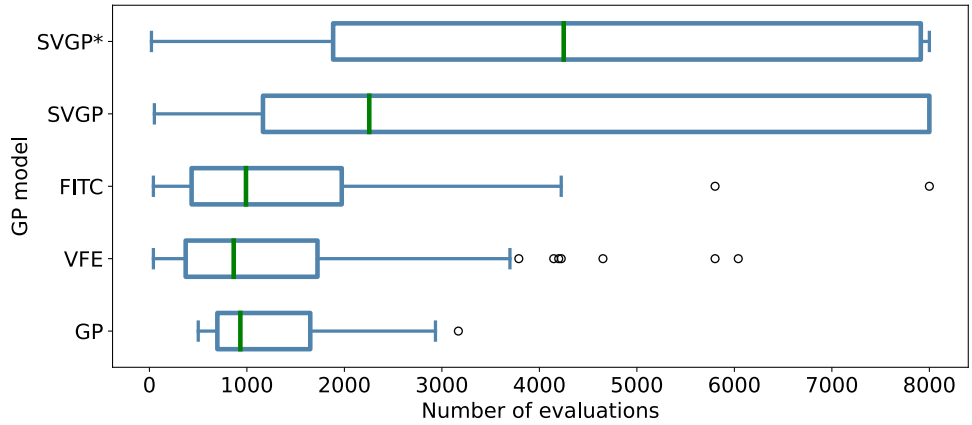


Figure 4.11: A visualisation showing the difference in the number of evaluations required for Bayesian optimisation using the employed GP models.

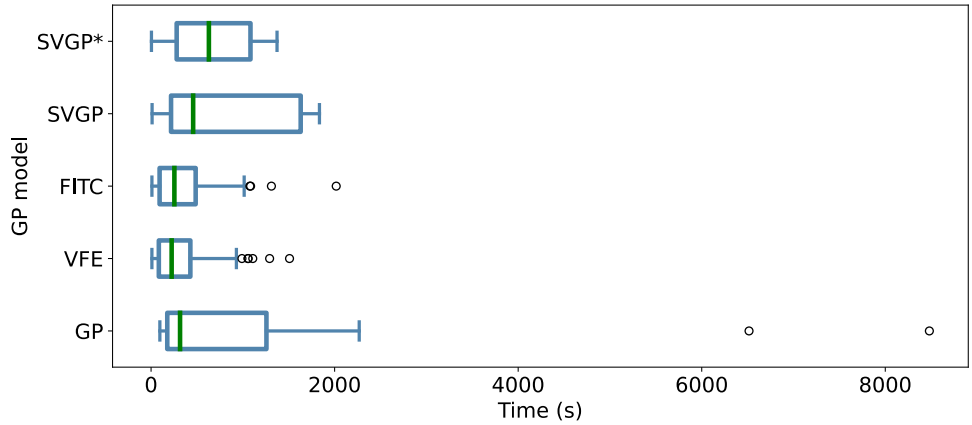


Figure 4.12: A visualisation showing the variation in run times for Bayesian optimization using the employed GP models.

Despite the initial impression that the standard GP model is the least effective due to the higher minimum number of evaluations needed to find a solution compared to other models, Figure 4.11 does not support this conclusion. In fact, the standard GP model was the only one that consistently required fewer than 3500 objective function evaluations. Although the standard GP model generally required fewer evaluations to find a solution, it took longer to do so compared to the other models, as demonstrated in Figure 4.12. Three iterations of the Bayesian optimisation process using a standard GP model, which ultimately resulted in a solution after 521 iterations, are illustrated in Figure 4.13.

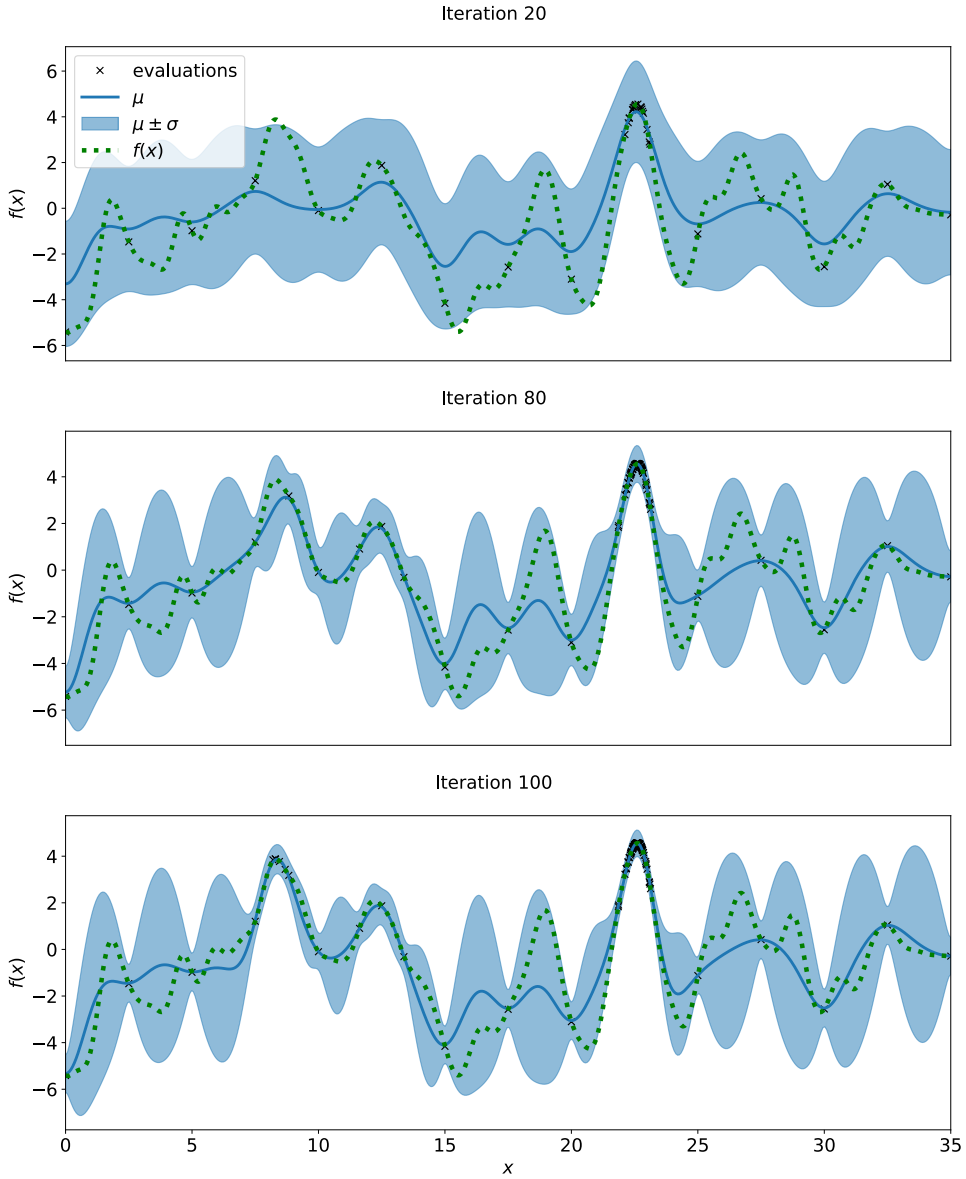


Figure 4.13: A visualisation of three iterations of the Bayesian optimisation using the standard GP surrogate model.

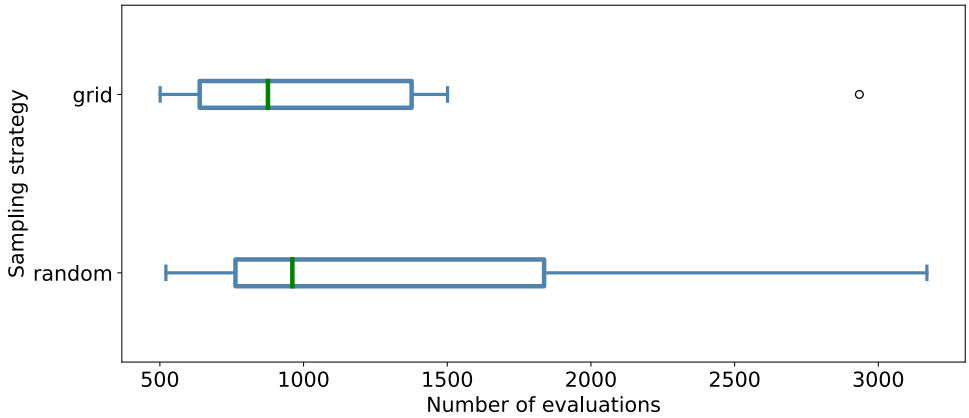


Figure 4.14: Comparison of different sampling strategies used for the Bayesian optimisation runs with the standard GP model.

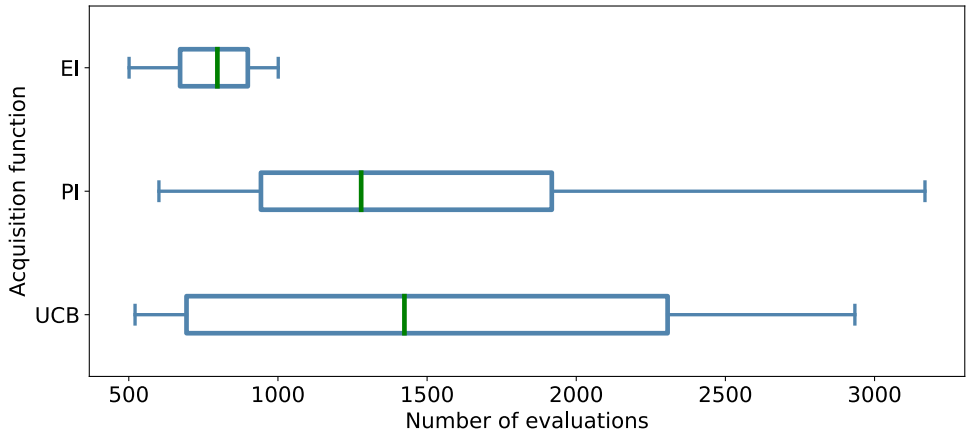


Figure 4.15: Comparison of different acquisition functions used for the Bayesian optimisation runs with the standard GP model.

While the VFE and FITC models were able to find solutions in less time on some occasions, they also needed more time to find solutions on other occasions. There were also some outliers that required significantly more evaluations to approach the true maximum, with one instance of Bayesian optimisation using the FITC model being unsuccessful in finding a solution (the maximum number of allowed evaluations was 8000). Despite the varying number of evaluations required to find a solution, the VFE and FITC models consistently needed less time than the standard GP model. Lastly, the SVGP and SVGP* models often failed to find points close to the optimum, as seen in Figure 4.11. In addition, Figure 4.12 suggests that they required a longer runtime.

In order to gain a more detailed understanding of the runs with the three models (standard GP, VFE, and FITC) that were able to find a solution each time (with one exception), it is worth examining the influence of different settings. Regarding sampling strategies, random sampling sometimes allowed for quick solution finding, but it also sometimes led to slow solution finding, as can be seen in Figure 4.14, which visualises runs of Bayesian optimisation with the standard GP model divided by sampling strategy. The differences in the use of various sampling strategies are also evident, but to a lesser degree, in instances using VFE and FITC models.

The differences in the use of various acquisition functions were not significant in most cases. The only noticeable effect was with the standard GP model, where the EI acquisition function was the most successful as shown in Figure 4.15. However, it should be noted that there were only 12 instances of Bayesian optimisation using the standard GP model, so if more experiments were conducted, it is possible that the differences would be less pronounced.

Conclusion

This chapter reviews the undertaken steps and work that was performed to fulfil our goals and discusses possible future work directions. As described at the very beginning of this thesis, we established three objectives.

The first goal was to conduct a survey of state-of-the-art scalable GPs, which is provided in Chapter 3. We present one of possible taxonomies of scalable GP models and provide a concise description of their main representatives. Among all the introduced groups, special attention is given to the global sparse approximation methods and models belonging to this category. A detailed description is provided for several influential and recent global sparse GP models, some of which are later employed in the experimental part of the thesis.

Our contribution primarily lies in adapting multiple selected GPs and their application in tasks of regression and Bayesian optimisation. The experiments and evaluation of models' performance in both contexts fulfil the second and third objectives stated in this thesis. In the regression task, we demonstrate the models' prediction quality on various datasets and compare their performance measured using several metrics. The results of the conducted experiments indicate that standard GP, VFE, and FITC models, in that order, have superior predictive capabilities compared to the other GP models used in the experiments. However, there were some datasets, such as the Airline dataset, that proved challenging for all of the GP models, resulting in similar prediction quality for each model.

In the Bayesian optimisation task, we evaluate the ability of various models to search for the extrema of a function. Several GP models that we employed in the regression task are unsuitable for usage in our Bayesian optimisation setup and were thus omitted for this scenario, as explained in Section 4.4. We found that most of the models performed similarly when using different sam-

pling strategies or acquisition functions. However, the choice of a surrogate model appears to have a more significant impact on the results. The standard GP, VFE, and FITC models were consistently better at finding an adequate solution with fewer evaluations.

Nevertheless, there is an immense number of intricacies related to GPs left unexplored. The following text suggests several directions for extending this work, which were neglected due to the time and computational limitations.

Firstly, there are many state-of-the-art GP models utilising novel ideas and approaches, which were not mentioned nor employed. On account of GPs being appealing to many researchers, their application to various fields is on the rise. Few examples of other usage are covered in Section 3.4, which includes only a mere fraction of possible application areas and purposes they can serve. One of the listed models is the hierarchical GP model proposed in [45], which is able to combine global and local information in a way that could be beneficial for certain problems. It might be interesting to consider using and exploiting the capabilities of this GP model for for addressing the challenges of our datasets, especially the spatial 3D Road Network dataset.

Even though the experiments we performed took a considerable amount of time, it would be reasonable to explore the hyperparameter space more thoroughly. Particularly interesting might be testing out more kernel types and their various combinations. Another possible experiment could utilise and evaluate different strategies for the initial values of kernel's parameters. Our strategy used default values, which is not a limitation but does not ensure a good starting point for the model. Of course, none of the more complex strategies can guarantee the best possible parameter values, but can be helpful in providing initialisation values more advantageous for the model. For instance, one such technique might be using heuristics to infer the parameter values directly from the training data.

Another possible future work direction could aim to investigate different hyperparameter values of SSGP, OIPS and PIPS models, which were considered to be fixed within our experiments. The hyperparameter values we used were taken from experiment descriptions provided by their authors, which does not necessarily mean there is no possible improvement. It could be daring to find the best performing values or extend these models and work in some novel ideas.

Each one of the models listed in the previous paragraph was left out of our experiments related to the Bayesian optimisation task. The omission of these models is a consequence of the models' unfitness for the problem we addressed. Suppose the scenario is changed to having a lot of points evaluated beforehand

and thus having an initial dataset big enough. In that case, these models can also be employed in the context of Bayesian optimisation. Another experimental design that can be utilised when we lack the data might include the possibility of obtaining the initial training dataset with different strategies, such as random selection and evaluation of a few hundred of data points.

Investigating the wealth of potentially promising areas offered by this work and the current GP literature could take us a step further in our attempts to overcome the challenges associated with GPs. At the end of the day, what matters is continuous research and development, striving to improve current solutions and thus paving the way for groundbreaking discoveries in the near future.

Bibliography

1. BROCHU, Eric; CORA, Vlad M.; FREITAS, Nando de. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *arXiv:1012.2599 [cs]* [online]. 2010 [visited on 2021-08-28]. Available from: http://arxiv.org/abs/1012.2599.
2. WORTMANN, Thomas {and} Giacomo Nannicini. Black-Box Optimisation Methods for Architectural Design. In: *Living Systems and Micro-Utopias: Towards Continuous Designing, Proceedings of the 21st International Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA 2016) / Melbourne 30 March–2 April 2016, pp. 177-186* [online]. CUMINCAD, 2016 [visited on 2021-09-28]. Available from: http://papers.cumincad.org/cgi-bin/works/Show?caadria2016_177.
3. HASANÇEBI, OĞUZHAN; ÇARBAŞ, S; DOĞAN, E; ERDAL, FERİDE; SAKA, MP. Performance evaluation of metaheuristic search techniques in the optimum design of real size pin jointed structures. *Computers & Structures*. 2009, vol. 87, no. 5-6, pp. 284–302.
4. RIOS, Luis Miguel; SAHINIDIS, Nikolaos V. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*. 2013, vol. 56, no. 3, pp. 1247–1293.
5. SHAHRIARI, Bobak; SWERSKY, Kevin; WANG, Ziyu; ADAMS, Ryan P.; FREITAS, Nando de. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proc. IEEE* [online]. 2016, vol. 104, no. 1, pp. 148–175 [visited on 2021-09-01]. ISSN 0018-9219, ISSN 1558-2256. Available from DOI: 10.1109/JPROC.2015.2494218.
6. SNOEK, Jasper; LAROCHELLE, Hugo; ADAMS, Ryan P. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*. 2012, vol. 25.

7. CANDELIERI, Antonio; PEREGO, Riccardo; GIORDANI, Ilaria; PONTI, Andrea; ARCHETTI, Francesco. Modelling human active search in optimizing black-box functions. *Soft Computing*. 2020, vol. 24, no. 23, pp. 17771–17785.
8. NANDY, Abhilash; KUMAR, Chandan; MEWADA, Deepak; SHARMA, Soumya. Bayesian Optimization–Multi-Armed Bandit Problem. *arXiv preprint arXiv:2012.07885*. 2020.
9. BERGSTRA, James; BARDENET, Rémi; BENGIO, Yoshua; KÉGL, Balázs. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*. 2011, vol. 24.
10. WILLIAMS, Christopher K; RASMUSSEN, Carl Edward. *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006. No. 3.
11. BREIMAN, Leo. Random forests. *Machine learning*. 2001, vol. 45, no. 1, pp. 5–32.
12. HOOF, Jeroen van; VANSCHOREN, Joaquin. Hyperboost: Hyperparameter Optimization by Gradient Boosting surrogate models. *arXiv preprint arXiv:2101.02289*. 2021.
13. LIU, Haitao; ONG, Yew-Soon; SHEN, Xiaobo; CAI, Jianfei. When Gaussian Process Meets Big Data: A Review of Scalable GPs. *arXiv:1807.01065 [cs, stat]* [online]. 2019 [visited on 2021-09-24]. Available from arXiv: 1807.01065.
14. SRINIVAS, Niranjan; KRAUSE, Andreas; KAKADE, Sham M; SEEGER, Matthias. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*. 2009.
15. KUSHNER, Harold J. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. 1964.
16. MOČKUS, Jonas. On Bayesian methods for seeking the extremum. In: *Optimization techniques IFIP technical conference*. 1975, pp. 400–404.
17. HENNIG, Philipp; SCHULER, Christian J. Entropy Search for Information-Efficient Global Optimization. *Journal of Machine Learning Research*. 2012, vol. 13, no. 6.
18. THOMPSON, William R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*. 1933, vol. 25, no. 3/4, pp. 285–294.
19. HOFFMAN, Matthew; BROCHU, Eric; DE FREITAS, Nando, et al. Portfolio Allocation for Bayesian Optimization. In: *UAI*. 2011, pp. 327–336.

20. LYU, Wenlong; YANG, Fan; YAN, Changhao; ZHOU, Dian; ZENG, Xuan. Batch Bayesian optimization via multi-objective acquisition ensemble for automated analog circuit design. In: *International conference on machine learning*. 2018, pp. 3306–3314.

21. WANG, Jie. An intuitive tutorial to Gaussian processes regression. *arXiv preprint arXiv:2009.10862*. 2020.

22. SCHULZ, Eric; SPEEKENBRINK, Maarten; KRAUSE, Andreas. A tutorial on Gaussian process regression: Modelling, exploring, and exploiting functions. *Journal of Mathematical Psychology* [online]. 2018, vol. 85, pp. 1–16 [visited on 2021-11-19]. ISSN 0022-2496. Available from DOI: 10.1016/j.jmp.2018.03.001.

23. CHEN, Zexun; WANG, Bo. How priors of initial hyperparameters affect Gaussian process regression models. *Neurocomputing*. 2018, vol. 275, pp. 1702–1710.

24. WENDLAND, Holger. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in computational Mathematics*. 1995, vol. 4, no. 1, pp. 389–396.

25. KIM, Hyoung-Moon; MALLICK, Bani K; HOLMES, Chris C. Analyzing nonstationary spatial data using piecewise Gaussian processes. *Journal of the American Statistical Association*. 2005, vol. 100, no. 470, pp. 653–668.

26. JACOBS, Robert A; JORDAN, Michael I; NOWLAN, Steven J; HINTON, Geoffrey E. Adaptive mixtures of local experts. *Neural computation*. 1991, vol. 3, no. 1, pp. 79–87.

27. HINTON, Geoffrey E. Training products of experts by minimizing contrastive divergence. *Neural computation*. 2002, vol. 14, no. 8, pp. 1771–1800.

28. QUINONERO-CANDELA, Joaquin; RASMUSSEN, Carl Edward. A unifying view of sparse approximate Gaussian process regression. *The Journal of Machine Learning Research*. 2005, vol. 6, pp. 1939–1959.

29. WILSON, Andrew; NICKISCH, Hannes. Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In: *International conference on machine learning*. 2015, pp. 1775–1784.

30. LIU, Haitao; CAI, Jianfei; ONG, Yew-Soon; WANG, Yi. Understanding and comparing scalable Gaussian process regression for big data. *Knowledge-Based Systems*. 2019, vol. 164, pp. 324–335.

31. SNELSON, Edward; GHAHRAMANI, Zoubin. Sparse Gaussian processes using pseudo-inputs. *Advances in neural information processing systems*. 2005, vol. 18.

32. BAUER, Matthias; WILK, Mark van der; RASMUSSEN, Carl Edward. Understanding probabilistic sparse Gaussian process approximations. *Advances in neural information processing systems*. 2016, vol. 29.

33. TITSIAS, Michalis. Variational learning of inducing variables in sparse Gaussian processes. In: *Artificial intelligence and statistics*. 2009, pp. 567–574.

34. BLEI, David M; KUCUKELBIR, Alp; MCAULIFFE, Jon D. Variational inference: A review for statisticians. *Journal of the American statistical Association*. 2017, vol. 112, no. 518, pp. 859–877.

35. DAI, Zhenwen. *Scalability of Gaussian Process* [The Gaussian Process Summer School 2021, online]. 2021. Available also from: `https://zhenwendai.github.io/slides/gpss2021_slides.pdf`.

36. HENSMAN, James; FUSI, Nicolo; LAWRENCE, Neil D. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*. 2013.

37. SCHÜRCH, Manuel; AZZIMONTI, Dario; BENAVOLI, Alessio; ZAFFALON, Marco. Recursive estimation for sparse Gaussian process regression. *Automatica*. 2020, vol. 120, p. 109127.

38. BUI, Thang D; NGUYEN, Cuong; TURNER, Richard E. Streaming sparse Gaussian process approximations. *Advances in Neural Information Processing Systems*. 2017, vol. 30.

39. BUI, Thang D; NGUYEN, Cuong; TURNER, Richard E; JOHN, S T. *Streaming sparse Gaussian process approximations implementation* [comp. software]. GitHub, 2022. Available also from: `https://github.com/thangbui/streaming_sparse_gp`.

40. GALY-FAJOU, Théo; OPPER, Manfred. Adaptive inducing points selection for gaussian processes. *arXiv preprint arXiv:2107.10066*. 2021.

41. UHRENHOLT, Anders Kirk; CHARVET, Valentin; JENSEN, Bjørn Sand. Probabilistic selection of inducing points in sparse Gaussian processes. In: *Uncertainty in Artificial Intelligence*. 2021, pp. 1035–1044.

42. BLAŽEK, Rudolf B. et al. *NI-VSM – Vybrané statistické metody: soubor handoutů*. 2022. Available also from: `https://courses.fit.cvut.cz/NI-VSM/lectures/files/NI-VSM-TextBook-Handout.pdf`.

43. BLISCHKE, Wallace R; MURTHY, DN Prabhakar. *Reliability: modeling, prediction, and optimization*. John Wiley & Sons, 2011.

44. JANKOWIAK, Martin; PLEISS, Geoff; GARDNER, Jacob. Parametric gaussian process regressors. In: *International Conference on Machine Learning*. 2020, pp. 4702–4712.

45. NGUYEN, Thi Nhat Anh; BOUZERDOUM, Abdesselam; PHUNG, Son Lam. Stochastic variational hierarchical mixture of sparse Gaussian processes for regression. *Machine Learning*. 2018, vol. 107, no. 12, pp. 1947–1986.

46. RIOS, Gonzalo; TOBAR, Felipe. Compositionally-warped Gaussian processes. *Neural Networks*. 2019, vol. 118, pp. 235–246.

47. LIU, Haitao; CAI, Jianfei; ONG, Yew-Soon. Remarks on multi-output Gaussian process regression. *Knowledge-Based Systems*. 2018, vol. 144, pp. 102–121.

48. NGUYEN, Trung; BONILLA, Edwin. Fast allocation of Gaussian process experts. In: *International Conference on Machine Learning*. 2014, pp. 145–153.

49. KAUL, Manohar; YANG, Bin; JENSEN, Christian S. Building accurate 3d spatial networks to enable next generation intelligent transportation systems. In: *2013 IEEE 14th International Conference on Mobile Data Management*. 2013, vol. 1, pp. 137–146.

50. TERRY, Nick; CHOE, Youngjun. Splitting Gaussian Process Regression for Streaming Data. *arXiv preprint arXiv:2010.02424*. 2020.

51. DEISENROTH, Marc P; LUO, Yicheng; WILK, Mark van der. A practical guide to Gaussian processes. *Distill (cit. on pp. 79, 100)*. 2019.

52. VANROSSUM, Guido. Python reference manual. *Department of Computer Science [CS]*. 1995, no. R 9525.

53. PASZKE, Adam et al. Automatic differentiation in pytorch. 2017.

54. MATTHEWS, Alexander G de G et al. GPflow: A Gaussian Process Library using TensorFlow. *J. Mach. Learn. Res.* 2017, vol. 18, no. 40, pp. 1–6.

55. GARDNER, Jacob; PLEISS, Geoff; WEINBERGER, Kilian Q; BINDEL, David; WILSON, Andrew G. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. *Advances in neural information processing systems*. 2018, vol. 31.

56. LIZOTTE, Daniel James. *Practical Bayesian Optimization*. CAN: University of Alberta, 2008. ISBN 9780494463659. PhD thesis. AAINR46365.

Appendix A

Acronyms

EI Expected improvement

ELBO Evidence lower bound

ES Entropy search

FITC Fully Independent Training Conditional

GP Gaussian Process

MOGP Multioutput Gaussian Process

MS Mean square

MSE Mean Squared Error

MSLL Mean Standardised Log Loss

MoE Mixture-of-experts

OIPS Online Inducing Points Selection

PI Probability of improvement

PIPS Probabilistic Inducing Points Selection

PoE Product-of-experts

RF Random Forests

RMSE Root Mean Squared Error

RQ Rational quadratic

SE Squared exponential

SMSE Standardised Mean Squared Error

SSGP Streaming Sparse Gaussian Process

SVGP Stochastic Variational Gaussian Process

SKI Structured Kernel Interpolation

TPE Tree-Parzen Estimators

UCB Upper confidence bound

VFE Variational Free Energy

Appendix B

Supplementary Material to SSGP

B.1 Prediction

Let $\mathbf{X}_\mathrm{s}$ be a matrix of test data points. The predictive mean $\mathbf{m}_\mathrm{s}$ and predictive covariance $\mathbf{V}_\mathrm{ss}$ are calculated as [38]

$$\begin{aligned}
\mathbf{V}_\mathrm{ss} &= \mathbf{K}_\mathrm{ss} - \mathbf{K}_\mathrm{sb}\mathbf{K}_\mathrm{bb}^{-1}\mathbf{K}_\mathrm{bs} + \mathbf{K}_\mathrm{sb}\mathbf{K}_\mathrm{bb}^{-1}\left(\mathbf{K}_\mathrm{bb}^{-1} + \mathbf{K}_\mathrm{bb}^{-1}\mathbf{K}_\mathrm{bf}\Sigma_{\hat{\mathrm{y}}}^{-1}\mathbf{K}_{\hat{\mathrm{f}}\mathrm{b}}\mathbf{K}_\mathrm{bb}^{-1}\right)^{-1}\mathbf{K}_\mathrm{bb}^{-1}\mathbf{K}_\mathrm{bs} \\
&= \mathbf{K}_\mathrm{ss} - \mathbf{K}_\mathrm{sb}\mathbf{K}_\mathrm{bb}^{-1}\mathbf{K}_\mathrm{bs} + \mathbf{K}_\mathrm{sb}\mathbf{L}_\mathrm{b}^{-T}\left(\mathrm{I} + \mathbf{L}_\mathrm{b}^{-1}\mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}}\Sigma_{\hat{\mathrm{y}}}^{-1}\mathbf{K}_{\hat{\mathrm{f}}\mathrm{b}}\mathbf{L}_\mathrm{b}^{-T}\right)^{-1}\mathbf{L}_\mathrm{b}^{-1}\mathbf{K}_\mathrm{bs} \\
&= \mathbf{K}_\mathrm{ss} - \mathbf{K}_\mathrm{sb}\mathbf{K}_\mathrm{bb}^{-1}\mathbf{K}_\mathrm{bs} + \mathbf{K}_\mathrm{sb}\mathbf{L}_\mathrm{b}^{-T}\mathbf{D}^{-1}\mathbf{L}_\mathrm{b}^{-1}\mathbf{K}_\mathrm{bs}, \\
\mathbf{m}_\mathrm{s} &= \mathbf{K}_\mathrm{sb}\mathbf{K}_\mathrm{bb}^{-1}\left(\mathbf{K}_\mathrm{bb}^{-1} + \mathbf{K}_\mathrm{bb}^{-1}\mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}}\Sigma_{\hat{\mathrm{y}}}^{-1}\mathbf{K}_{\hat{\mathrm{f}}\mathrm{b}}\mathbf{K}_\mathrm{bb}^{-1}\right)^{-1}\mathbf{K}_\mathrm{bb}^{-1}\mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}}\Sigma_{\hat{\mathrm{y}}}^{-1}\hat{\mathbf{y}} \\
&= \mathbf{K}_\mathrm{sb}\mathbf{L}_\mathrm{b}^{-T}\left(\mathrm{I} + \mathbf{L}_\mathrm{b}^{-1}\mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}}\Sigma_{\hat{\mathrm{y}}}^{-1}\mathbf{K}_{\hat{\mathrm{f}}\mathrm{b}}\mathbf{L}_\mathrm{b}^{-T}\right)^{-1}\mathbf{L}_\mathrm{b}^{-1}\mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}}\Sigma_{\hat{\mathrm{y}}}^{-1}\hat{\mathbf{y}} \\
&= \mathbf{K}_\mathrm{sb}\mathbf{L}_\mathrm{b}^{-T}\mathbf{D}^{-1}\mathbf{L}_\mathrm{b}^{-1}\mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}}\Sigma_{\hat{\mathrm{y}}}^{-1}\hat{\mathbf{y}}.
\end{aligned}$$

Note that, $\mathbf{K}_\mathrm{ss} = K(\mathbf{X}_\mathrm{s}, \mathbf{X}_\mathrm{s}), \mathbf{K}_\mathrm{sb} = K(\mathbf{X}_\mathrm{s}, \mathbf{Z}_\mathrm{b}), \mathbf{K}_\mathrm{bs} = \mathbf{K}_\mathrm{sb}^T, \mathbf{K}_\mathrm{bf} = \mathbf{K}_\mathrm{fb}^T$, $\mathbf{K}_\mathrm{bb} = K(\mathbf{Z}_\mathrm{b}, \mathbf{Z}_\mathrm{b}) = \mathbf{L}_\mathrm{b}\mathbf{L}_\mathrm{b}^T$ and $\mathbf{D} = \mathrm{I} + \mathbf{L}_\mathrm{b}^{-1}\mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}}\Sigma_{\hat{\mathrm{y}}}^{-1}\mathbf{K}_{\hat{\mathrm{f}}\mathrm{b}}\mathbf{L}_\mathrm{b}^{-T}$. The remaining terms were already introduced in Section 3.3.2, which is a core section for this supplementary material, and their redefinition will be therefore omitted.

B.2 Step-by-step from Theoretical to Practical Bound

The authors of the SSGP model [38] presented both theoretical bound and bound adapted for practical usage. The practical bound is given as

$$\begin{aligned}\log \mathcal{L}(\boldsymbol{\theta}) = &-\frac{n}{2}\log\left(2\pi\sigma_{\mathrm{y}}^2\right) - \frac{1}{2}\log|\mathbf{D}| - \frac{1}{2\sigma_{\mathrm{y}}^2}\mathbf{y}^T\mathbf{y} + \frac{1}{2}\mathbf{c}^T\mathbf{L}_{\mathrm{b}}^{-T}\mathbf{D}^{-1}\mathbf{L}_{\mathrm{b}}^{-1}\mathbf{c} \\ &- \frac{1}{2}\log|\mathbf{S}_{\mathrm{a}}| + \frac{1}{2}\log|\mathbf{K}_{\mathrm{aa}}| - \frac{1}{2}\operatorname{tr}\left(\mathbf{D}_{\mathrm{a}}^{-1}\mathbf{Q}_{\mathrm{a}}\right) - \frac{1}{2}\mathbf{m}_{\mathrm{a}}^T\mathbf{S}_{\mathrm{a}}^{-1}\mathbf{m}_{\mathrm{a}} - \frac{1}{2\sigma_{\mathrm{y}}^2}\operatorname{tr}\left(\mathbf{Q}_{\mathrm{f}}\right).\end{aligned}$$

It can be derived from the theoretical bound as follows:

$$\begin{aligned}
\log \mathcal{L}(\boldsymbol{\theta}) &= \log \mathcal{N}(\mathbf{0}, \mathbf{K}_{\hat{\mathrm{f}}\hat{\mathrm{f}}} + \Sigma_{\hat{\mathrm{y}}}) + \Delta_1 + \Delta_2, \\
\log \mathcal{L}(\boldsymbol{\theta}) &= -\frac{n+m_{\mathrm{a}}}{2}\log(2\pi) - \frac{1}{2}\log\left|\mathbf{K}_{\hat{\mathrm{f}}\hat{\mathrm{f}}} + \Sigma_{\hat{\mathrm{y}}}\right| - \frac{1}{2}\hat{\mathbf{y}}^T\left(\mathbf{K}_{\hat{\mathrm{f}}\hat{\mathrm{f}}} + \Sigma_{\hat{\mathrm{y}}}\right)^{-1}\hat{\mathbf{y}} + \Delta_1 + \Delta_2, \\
\log \mathcal{L}(\boldsymbol{\theta}) &= -\frac{n+m_{\mathrm{a}}}{2}\log(2\pi) - \frac{1}{2}\left(\log|\Sigma_{\hat{\mathrm{y}}}| + \log\left|\mathrm{I} + \mathbf{L}_{\mathrm{b}}^{-1}\mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}}\Sigma_{\hat{\mathrm{y}}}^{-1}\mathbf{K}_{\hat{\mathrm{f}}\mathrm{b}}\mathbf{L}_{\mathrm{b}}^{-T}\right|\right) \\
&\quad - \frac{1}{2}\hat{\mathbf{y}}^T\left(\mathbf{K}_{\hat{\mathrm{f}}\mathrm{b}}\mathbf{K}_{\mathrm{bb}}^{-1}\mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}} + \Sigma_{\hat{\mathrm{y}}}\right)^{-1}\hat{\mathbf{y}} + \Delta_1 + \Delta_2, \\
\log \mathcal{L}(\boldsymbol{\theta}) &= -\frac{n+m_{\mathrm{a}}}{2}\log(2\pi) - \frac{1}{2}\left(n\log\sigma_{\mathrm{y}}^2 + \log|\mathbf{D}_{\mathrm{a}}| + \log|\mathbf{D}|\right) \\
&\quad - \frac{1}{2}\hat{\mathbf{y}}^T\left(\Sigma_{\hat{\mathrm{y}}}^{-1} - \Sigma_{\hat{\mathrm{y}}}^{-1}\mathbf{K}_{\hat{\mathrm{f}}\mathrm{b}}\mathbf{L}_{\mathrm{b}}^{-T}\mathbf{D}^{-1}\mathbf{L}_{\mathrm{b}}^{-1}\mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}}\Sigma_{\hat{\mathrm{y}}}^{-1}\right)\hat{\mathbf{y}} + \Delta_1 + \Delta_2, \\
\log \mathcal{L}(\boldsymbol{\theta}) &= -\frac{n+m_{\mathrm{a}}}{2}\log(2\pi) - \frac{1}{2}\left(n\log\sigma_{\mathrm{y}}^2 + \log|\mathbf{D}_{\mathrm{a}}| + \log|\mathbf{D}|\right) \\
&\quad - \frac{1}{2}\left(\hat{\mathbf{y}}^T\Sigma_{\hat{\mathrm{y}}}^{-1}\hat{\mathbf{y}} - \hat{\mathbf{y}}^T\Sigma_{\hat{\mathrm{y}}}^{-1}\mathbf{K}_{\hat{\mathrm{f}}\mathrm{b}}\mathbf{L}_{\mathrm{b}}^{-T}\mathbf{D}^{-1}\mathbf{L}_{\mathrm{b}}^{-1}\mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}}\Sigma_{\hat{\mathrm{y}}}^{-1}\hat{\mathbf{y}}\right) + \Delta_1 + \Delta_2, \\
\log \mathcal{L}(\boldsymbol{\theta}) &= -\frac{n+m_{\mathrm{a}}}{2}\log(2\pi) - \frac{1}{2}\left(n\log\sigma_{\mathrm{y}}^2 + \log|\mathbf{D}_{\mathrm{a}}| + \log|\mathbf{D}|\right) \\
&\quad - \frac{1}{2}\left(\frac{1}{\sigma_{\mathrm{y}}^2}\hat{\mathbf{y}}^T\hat{\mathbf{y}} + \mathbf{m}_{\mathrm{a}}^T\mathbf{S}_{\mathrm{a}}^{-1}\mathbf{D}_{\mathrm{a}}\mathbf{S}_{\mathrm{a}}^{-1}\mathbf{m}_{\mathrm{a}} - \hat{\mathbf{y}}^T\Sigma_{\hat{\mathrm{y}}}^{-1}\mathbf{K}_{\hat{\mathrm{f}}\mathrm{b}}\mathbf{L}_{\mathrm{b}}^{-T}\mathbf{D}^{-1}\mathbf{L}_{\mathrm{b}}^{-1}\mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}}\Sigma_{\hat{\mathrm{y}}}^{-1}\hat{\mathbf{y}}\right) \\
&\quad + \Delta_1 + \Delta_2, \\
\log \mathcal{L}(\boldsymbol{\theta}) &= -\frac{n+m_{\mathrm{a}}}{2}\log(2\pi) - \frac{1}{2}\left(n\log\sigma_{\mathrm{y}}^2 + \log|\mathbf{D}_{\mathrm{a}}| + \log|\mathbf{D}|\right) \\
&\quad - \frac{1}{2}\left(\frac{1}{\sigma_{\mathrm{y}}^2}\hat{\mathbf{y}}^T\hat{\mathbf{y}} + \mathbf{m}_{\mathrm{a}}^T\mathbf{S}_{\mathrm{a}}^{-1}\mathbf{D}_{\mathrm{a}}\mathbf{S}_{\mathrm{a}}^{-1}\mathbf{m}_{\mathrm{a}} - \hat{\mathbf{y}}^T\Sigma_{\hat{\mathrm{y}}}^{-1}\mathbf{K}_{\hat{\mathrm{f}}\mathrm{b}}\mathbf{L}_{\mathrm{b}}^{-T}\mathbf{D}^{-1}\mathbf{L}_{\mathrm{b}}^{-1}\mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}}\Sigma_{\hat{\mathrm{y}}}^{-1}\hat{\mathbf{y}}\right) \\
&\quad + \frac{1}{2}\Bigg(-\log\frac{|\mathbf{S}_{\mathrm{a}}|}{|\mathbf{K}_{\mathrm{aa}}|\,|\mathbf{D}_{\mathrm{a}}|} + \mathbf{m}_{\mathrm{a}}^T\mathbf{S}_{\mathrm{a}}^{-1}\mathbf{D}_{\mathrm{a}}\mathbf{S}_{\mathrm{a}}^{-1}\mathbf{m}_{\mathrm{a}} - \operatorname{tr}\left(\mathbf{D}_{\mathrm{a}}^{-1}\mathbf{Q}_{\mathrm{a}}\right) - \mathbf{m}_{\mathrm{a}}^T\mathbf{S}_{\mathrm{a}}^{-1}\mathbf{m}_{\mathrm{a}} \\
&\quad + m_{\mathrm{a}}\log(2\pi)\Bigg) - \frac{1}{2\sigma_{\mathrm{y}}^2}\operatorname{tr}\left(\mathbf{Q}_{\mathrm{f}}\right),
\end{aligned}$$

$$
\begin{aligned}
\log \mathcal{L}(\boldsymbol{\theta}) = &-\frac{n+m_{\mathrm{a}}}{2}\log(2\pi) - \frac{1}{2}\left(n\log\sigma_{\mathrm{y}}^2 + \log|\mathbf{D}_{\mathrm{a}}| + \log|\mathbf{D}|\right) \\
&- \frac{1}{2}\left(\frac{1}{\sigma_{\mathrm{y}}^2}\hat{\mathbf{y}}^T\hat{\mathbf{y}} - \hat{\mathbf{y}}^T\Sigma_{\hat{\mathrm{y}}}^{-1}\mathbf{K}_{\hat{\mathrm{f}}\mathrm{b}}\mathbf{L}_{\mathrm{b}}^{-T}\mathbf{D}^{-1}\mathbf{L}_{\mathrm{b}}^{-1}\mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}}\Sigma_{\hat{\mathrm{y}}}^{-1}\hat{\mathbf{y}}\right) \\
&+ \frac{1}{2}\left(-\log\frac{|\mathbf{S}_{\mathrm{a}}|}{|\mathbf{K}_{\mathrm{aa}}|\,|\mathbf{D}_{\mathrm{a}}|} - \operatorname{tr}\left(\mathbf{D}_{\mathrm{a}}^{-1}\mathbf{Q}_{\mathrm{a}}\right) - \mathbf{m}_{\mathrm{a}}^T\mathbf{S}_{\mathrm{a}}^{-1}\mathbf{m}_{\mathrm{a}} + m_{\mathrm{a}}\log(2\pi)\right) - \frac{1}{2\sigma_{\mathrm{y}}^2}\operatorname{tr}\left(\mathbf{Q}_{\mathrm{f}}\right), \\
\log \mathcal{L}(\boldsymbol{\theta}) = &-\frac{n+m_{\mathrm{a}}}{2}\log(2\pi) - \frac{n}{2}\log\sigma_{\mathrm{y}}^2 - \frac{1}{2}\log|\mathbf{D}_{\mathrm{a}}| - \frac{1}{2}\log|\mathbf{D}| - \frac{1}{2}\log|\mathbf{S}_{\mathrm{a}}| + \frac{1}{2}\log|\mathbf{K}_{\mathrm{aa}}| \\
&+ \frac{1}{2}\log|\mathbf{D}_{\mathrm{a}}| + \frac{1}{2}m_{\mathrm{a}}\log(2\pi) - \frac{1}{2\sigma_{\mathrm{y}}^2}\mathbf{y}^T\mathbf{y} + \frac{1}{2}\hat{\mathbf{y}}^T\Sigma_{\hat{\mathrm{y}}}^{-1}\mathbf{K}_{\hat{\mathrm{f}}\mathrm{b}}\mathbf{L}_{\mathrm{b}}^{-T}\mathbf{D}^{-1}\mathbf{L}_{\mathrm{b}}^{-1}\mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}}\Sigma_{\hat{\mathrm{y}}}^{-1}\hat{\mathbf{y}} \\
&- \frac{1}{2}\operatorname{tr}\left(\mathbf{D}_{\mathrm{a}}^{-1}\mathbf{Q}_{\mathrm{a}}\right) - \frac{1}{2}\mathbf{m}_{\mathrm{a}}^T\mathbf{S}_{\mathrm{a}}^{-1}\mathbf{m}_{\mathrm{a}} - \frac{1}{2\sigma_{\mathrm{y}}^2}\operatorname{tr}\left(\mathbf{Q}_{\mathrm{f}}\right), \\
\log \mathcal{L}(\boldsymbol{\theta}) = &-\frac{n}{2}\log(2\pi\sigma_{\mathrm{y}}^2) - \frac{1}{2}\log|\mathbf{D}| - \frac{1}{2\sigma_{\mathrm{y}}^2}\mathbf{y}^T\mathbf{y} + \frac{1}{2}\hat{\mathbf{y}}^T\Sigma_{\hat{\mathrm{y}}}^{-1}\mathbf{K}_{\hat{\mathrm{f}}\mathrm{b}}\mathbf{L}_{\mathrm{b}}^{-T}\mathbf{D}^{-1}\mathbf{L}_{\mathrm{b}}^{-1}\mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}}\Sigma_{\hat{\mathrm{y}}}^{-1}\hat{\mathbf{y}} \\
&- \frac{1}{2}\log|\mathbf{S}_{\mathrm{a}}| + \frac{1}{2}\log|\mathbf{K}_{\mathrm{aa}}| - \frac{1}{2}\operatorname{tr}\left(\mathbf{D}_{\mathrm{a}}^{-1}\mathbf{Q}_{\mathrm{a}}\right) - \frac{1}{2}\mathbf{m}_{\mathrm{a}}^T\mathbf{S}_{\mathrm{a}}^{-1}\mathbf{m}_{\mathrm{a}} - \frac{1}{2\sigma_{\mathrm{y}}^2}\operatorname{tr}\left(\mathbf{Q}_{\mathrm{f}}\right), \\
\log \mathcal{L}(\boldsymbol{\theta}) = &-\frac{n}{2}\log\left(2\pi\sigma_{\mathrm{y}}^2\right) - \frac{1}{2}\log|\mathbf{D}| - \frac{1}{2\sigma_{\mathrm{y}}^2}\mathbf{y}^T\mathbf{y} + \frac{1}{2}\mathbf{c}^T\mathbf{L}_{\mathrm{b}}^{-T}\mathbf{D}^{-1}\mathbf{L}_{\mathrm{b}}^{-1}\mathbf{c} \\
&- \frac{1}{2}\log|\mathbf{S}_{\mathrm{a}}| + \frac{1}{2}\log|\mathbf{K}_{\mathrm{aa}}| - \frac{1}{2}\operatorname{tr}\left(\mathbf{D}_{\mathrm{a}}^{-1}\mathbf{Q}_{\mathrm{a}}\right) - \frac{1}{2}\mathbf{m}_{\mathrm{a}}^T\mathbf{S}_{\mathrm{a}}^{-1}\mathbf{m}_{\mathrm{a}} - \frac{1}{2\sigma_{\mathrm{y}}^2}\operatorname{tr}\left(\mathbf{Q}_{\mathrm{f}}\right),
\end{aligned}
$$

where $\mathbf{c} = \mathbf{K}_{\mathrm{b}\hat{\mathrm{f}}}\Sigma_{\hat{\mathrm{y}}}^{-1}\hat{\mathbf{y}}$. The definitions of other terms are left out since they are already stated in Sections 3.3.2 and B.1.

Appendix C

Experimental Results

Table C.1: Mean and standard deviation metrics values obtained by repeated training and evaluation of the best models on the kin40k dataset.

Model	Performance measure			
	RMSE	SMSE	MSLL	Covered (%)
GP	0.0971±0.0000	0.0093±0.0000	-1.0448±0.0000	94.2133±0.0000
VFE	0.1109±0.0000	0.0122±0.0000	-0.8362±0.0000	96.3413±0.0275
FITC	0.1303±0.0002	0.0168±0.0000	-0.7403±0.0003	98.6853±0.0169
SVGP	0.1641±0.0002	0.0267±0.0001	-0.7093±0.0021	95.0493±0.1263
SVGP*	0.1911±0.0000	0.0361±0.0000	-0.1253±0.0045	99.5840±0.0084
SSGP	0.1725±0.0000	0.0294±0.0000	0.0973±0.0013	99.9320±0.0042
OIPS	0.1385±0.0005	0.0190±0.0001	-0.1563±0.0114	99.7413±0.0169
PIPS	0.2130±0.0027	0.0449±0.0011	-0.0265±0.0225	87.9413±0.3909

Table C.2: Mean and standard deviation metrics values obtained by repeated training and evaluation of the best models on the 3D Road Network dataset.

Model	Performance measure			
	RMSE	SMSE	MSLL	Covered (%)
GP	0.2719±0.0000	0.0729±0.0000	-0.0039±0.0000	94.9867±0.0000
VFE	0.2960±0.0006	0.0864±0.0003	0.1852±0.0029	94.9400±0.0373
FITC	0.4033±0.0053	0.1604±0.0042	0.0180±0.0110	93.3627±0.1571
SVGP	0.4825±0.0020	0.2296±0.0019	0.2751±0.0093	97.6893±0.0503
SVGP*	0.4242±0.0018	0.1775±0.0015	0.1931±0.0006	97.3787±0.0169
SSGP	0.5221±0.0046	0.2689±0.0047	3.7316±0.0195	81.1920±0.0590
OIPS	0.3072±0.0008	0.0931±0.0005	1.8539±0.0268	79.0107±0.1349
PIPS	0.5346±0.0142	0.2821±0.0154	15.7997±2.1049	52.7200±2.0850

Table C.3: Mean and standard deviation metrics values obtained by repeated training and evaluation of the best models on the Airline dataset.

Model	Performance measure			
	RMSE	SMSE	MSLL	Covered (%)
GP	0.9305±0.0000	0.8286±0.0000	1.3440±0.0000	95.3467±0.0000
VFE	0.9375±0.0003	0.8409±0.0006	1.3530±0.0004	95.3107±0.0155
FITC	0.9381±0.0012	0.8421±0.0022	1.1763±0.0020	95.0413±0.0489
SVGP	0.9498±0.0008	0.8633±0.0015	1.3315±0.0083	92.5520±0.1012
SVGP*	0.9568±0.0006	0.8760±0.0012	1.1522±0.0010	95.0200±0.0211
SSGP	0.9419±0.0003	0.8490±0.0005	26.5962±0.1811	42.7440±0.1602
OIPS	0.9420±0.0000	0.8492±0.0000	39.6042±0.0000	37.0533±0.0000
PIPS	0.9409±0.0003	0.8471±0.0006	58.4339±2.4844	29.7493±0.5953

Appendix D

Best GP Hyperparameters

Table D.1: Hyperparameters of standard GP models used for final repeated training and evaluation on each dataset.

Hyperparameter	Dataset		
	kin40k	3D Road Network	Airline
Kernel	SE	SE	SE
Loss function	`Trace_ELBO`	`Trace_ELBO`	`Trace_ELBO`
Optimiser	Adam	Adam	Adam
Learning rate	0.002	0.002	0.002

Table D.2: Hyperparameters of VFE models used for final repeated training and evaluation on each dataset.

Hyperparameter	Dataset		
	kin40k	3D Road Network	Airline
Kernel	SE	SE	SE
Loss function	`Trace_ELBO`	`Trace_ELBO`	`Trace_ELBO`
Optimiser	Adam	Adam	Adam
Learning rate	0.002	0.001	0.001
Inducing set size	4096	4096	128

Table D.3: Hyperparameters of FITC models used for final repeated training and evaluation on each dataset.

Hyperparameter	Dataset		
	kin40k	3D Road Network	Airline
Kernel	SE	SE	SE
Loss function	`Trace_ELBO`	`Trace_ELBO`	`Trace_ELBO`
Optimiser	Adam	Adam	Adam
Learning rate	0.01	0.01	0.005
Inducing set size	4096	4096	128

Table D.4: Hyperparameters of SVGP models used for final repeated training and evaluation on each dataset.

Hyperparameter	Dataset		
	kin40k	3D Road Network	Airline
Kernel	SE	SE	SE
Loss function	`TMF_ELBO`	`TMF_ELBO`	`TMF_ELBO`
Optimiser	Adam	Adam	Adam
Learning rate	0.01	0.01	0.002
Inducing set size	4096	4096	128

Table D.5: Hyperparameters of SVGP* models used for final repeated training and evaluation on each dataset.

Hyperparameter	Dataset		
	kin40k	3D Road Network	Airline
Kernel	SE	SE	SE
Loss function	`TMF_ELBO`	`TMF_ELBO`	`TMF_ELBO`
Optimiser	Adam	Adam	Adam
Learning rate	0.001	0.002	0.002
Inducing set size	8192	8192	128
Batch size	1000	200	200

Table D.6: Hyperparameters of SSGP models used for final repeated training and evaluation on each dataset.

Hyperparameter	Dataset		
	kin40k	3D Road Network	Airline
Inducing set size	1024	512	8192
Batch size	1000	800	400

Table D.7: Hyperparameters of OIPS models used for final repeated training and evaluation on each dataset.

Hyperparameter	Dataset		
	kin40k	3D Road Network	Airline
Inducing set size	128	256	128
Batch size	600	1000	800
ρ value	0.9	0.6	0.6

Table D.8: Hyperparameters of PIPS models used for final repeated training and evaluation on each dataset.

Hyperparameter	Dataset		
	kin40k	3D Road Network	Airline
Inducing set size	512	512	256
α value	0.1	0.1	0.1
Prior probability of inclusion	0.9	0.6	0.9

Appendix E

Contents of enclosed SD card

- `README.md` ..... the file with SD card contents description
- `datasets` ..... the directory of datasets
- `experiments` ..... the directory of experiments
  - `PIPS` ..... the directory of experiments with the PIPS model
  - `Pyro_GPs` ..... the directory of experiments with Pyro models
  - `SSGP` ..... the directory of experiments with the SSGP model
- `thesis.pdf` ..... the thesis text in PDF format