Bayesian Nonparametric Inference in Mechanistic Models of Complex Biological Systems

Umberto Noè

A thesis submitted to the
College of Science and Engineering
at the University of Glasgow
for the degree of
Doctor of Philosophy

December 2018

For my parents

Abstract

Parameter estimation in expensive computational models is a problem that commonly arises in science and engineering. With the increase in computational power, modellers started developing simulators of real life phenomena that are computationally intensive to evaluate. This, however, makes inference prohibitive due to the unit cost of a single function evaluation. This thesis focuses on computational models of biological and biomechanical processes such as the left-ventricular dynamics or the human pulmonary blood circulatory system. In the former model a single forward simulation is in the order of 11 minutes CPU time, while the latter takes approximately 23 seconds in our machines. Markov chain Monte Carlo methods or likelihood maximization using iterative algorithms would take days or weeks to provide a result. This makes them not suitable for clinical decision support systems, where a decision must be taken in a reasonable time frame.

I discuss how to accelerate the inference by using the concept of emulation, i.e. by replacing a computationally expensive function with a statistical approximation based on a finite set of expensive training runs. The emulation target could be either the output-domain, representing the standard approach in the emulation literature, or the loss-domain, which is an alternative and different perspective. Then, I demonstrate how this approach can be used to estimate the parameters of expensive simulators. First I apply loss-emulation to a nonstandard variant of the Lotka-Volterra model of prey-predator interactions, in order to assess if the approach is approximately unbiased. Next, I present a comprehensive comparison between output-emulation and loss-emulation on a computational model of left ventricular dynamics, with the goal of inferring the constitutive law relating the myocardial stretch to its strain. This is especially relevant for assessing cardiac function post myocardial infarction.

The results show how it is possible to estimate the stress-strain curve in just 15 minutes, compared to the one week required by the current best literature method. This means a reduction in the computational costs of 3 orders of magnitude.

Next, I review Bayesian optimization (BO), an algorithm to optimize a computationally expensive function by adaptively improving the emulator. This method is especially useful in scenarios where the simulator is not considered to be a "stable release". For example, the simulator could still be undergoing further developments, bug fixing, and improvements. I develop a new framework based on BO to estimate the parameters of a partial differential equation (PDE) model of the human pulmonary blood circulation. The parameters, being related to the vessel structure and stiffness, represent important indicators of pulmonary hypertension risk, which need to be estimated as they can only be measured with invasive experiments. The results using simulated data show how it is possible to estimate a patient's vessel properties in a time frame suitable for clinical applications.

I demonstrate a limitation of standard improvement-based acquisition functions for Bayesian optimization. The expected improvement (EI) policy recommends query points where the improvement is on average high. However, it does not account for the variance of the random variable Improvement. I define a new acquisition function, called ScaledEI, which recommends query points where the improvement on the incumbent minimum is expected to be high, with high confidence. This new BO algorithm is compared to acquisition functions from the literature on a large set of benchmark functions for global optimization, where it turns out to be a powerful default choice for Bayesian optimization. ScaledEI is then compared to standard non-Bayesian optimization solvers, to confirm that the policy still leads to a reduction in the number of forward simulations required to reach a given tolerance level on the function value. Finally, the new algorithm is applied to the problem of estimating the PDE parameters of the pulmonary circulation model previously discussed.

Acknowledgements

I am grateful to Professor Dirk Husmeier, my supervisor over the past four years, for the many discussions we have had and the guidance he has given me during my PhD. He also read a draft of the thesis and provided many helpful suggestions.

Thanks to the Biometrika Trust for awarding me the Biometrika PhD studentship, effectively making this project possible, and also for the opportunity to give a seminar at the Department of Statistics of University College London in front of the Trustees.

Thanks also to my friends and colleagues during these years of study, in particular to Christopher, Craig, Joan, Marnie, Michael and Yoana. I could not have asked for better lifelong friends.

Finally, I want to thank my parents, who always supported me and showed me what unconditional love really means.

Declaration of Authorship

I, Umberto Noè, hereby certify that this thesis titled "Bayesian Nonparametric Inference in Mechanistic Models of Complex Biological Systems" is entirely my own original work except where otherwise clearly attributed in collaborative projects. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

The contents of this thesis represent the work I carried out during my PhD, under the supervision of Professor Dirk Husmeier, and is part of the following papers:

1. Noè et al. (2015): **Noè, U.**, Filippone, M., and Husmeier, D. (2015). Emulation of ODEs with Gaussian processes. In *Proceedings of the 30th International Workshop on Statistical Modelling*, pages 191–194.

2. Noè et al. (2017): **Noè, U.**, Chen, W., Filippone, M., Hill, N., and Husmeier, D. (2017). Inference in a Partial Differential Equations Model of Pulmonary Arterial and Venous Blood Circulation Using Statistical Emulation. In Bracciali, A., Caravagna, G., Gilbert, D., and Tagliaferri, R., editors, *Computational Intelligence Methods for Bioinformatics and Biostatistics. CIBB 2016. Lecture Notes in Computer Science*, volume 10477, pages 184–198. Springer, Cham, Switzerland.

3. Noè and Husmeier (2018): **Noè, U.** and Husmeier, D. (2018). On a New Improvement-Based Acquisition Function for Bayesian Optimization. *eprint arXiv:1808.06918.*

4. Davies et al. (2018): Davies, V.*, **Noè, U.***, Lazarus, A., Gao, H., Macdonald, B., Berry, C., Luo, X., and Husmeier, D. (2018). Fast Parameter Inference

in a Computational Model of the Left Ventricle Using Emulation. *In submission*. *Joint first authors.

Papers 1, 2 and 3 were written by myself, with input from Dirk Husmeier (1,2,3), Maurizio Filippone (1,2), Weiwei Chen (2) and Nicholas Hill (2). Paper 4 is the result of a joint collaboration with Vinny Davies, Alan Lazarus, Hao Gao, Benn Macdonald, Xiaoyu Luo and Dirk Husmeier. All authors contributed to parts of the paper and in discussions.

Furthermore, additional publications which are not included in the contents of this thesis are:

5. Pasetto et al. (2017b): Pasetto, M. E., **Noè, U.**, Luati, A., and Husmeier, D. (2017). Inference with Unscented Kalman Filter and Optimization of Sigma Points. In Petrucci, A. and Verde, R., editors, *SIS 2017. Statistics and Data Science: new challenges, new generations. Proceedings of the Conference of the Italian Statistical Society*, pages 767–772. Florence. Firenze University Press.

6. Pasetto et al. (2017a): Pasetto, M. E., Husmeier, D., **Noè, U.**, and Luati, A. (2017). Statistical Inference in the Duffing System with the Unscented Kalman Filter. In *Proceedings of the 32nd International Workshop on Statistical Modelling*, pages 119–122. Groningen.

Contents

Chapter 1

Introduction

Mathematics has always been considered as an abstraction and rationalization of concepts and hypotheses based on experimental observations. However, by the construction of *models*, mathematics started to be seen as an investigative tool, to gain insight into a system of interest. In this approach, standard constructs typical of mathematics (equations, functions, ...) are being related to each other on the basis of the available knowledge on that system and assumptions which are often experimentally based. The combination of mathematical abstraction and experimentally-validated hypotheses gives rise to *mathematical models*. In 2004, J. E. Cohen published a thought-provoking article on the synergy between mathematics and biology, claiming that "mathematics is biology's next microscope, only better", and that mathematics is transforming biology in the same way it shaped physics in the previous centuries (Cohen, 2004). Now, almost fifteen years later, it is time to include statistics, and in particular modern computational statistics, in this synergy. Mathematics is providing powerful new tools to describe biological systems and processes in a more rigorous and quantitative manner, with parameters often representing interpretable quantities of interest which might be hard to measure experimentally. This opens up new challenging problems related to parameter inference, model selection and systems identification. To put it differently, mathematical contributions to biology have dealt with the forward (modelling) problem, while statistics aims to tackle the more challenging inverse (inference) problem.

This thesis is concerned with the latter problem of parameter estimation in mecha-

nistic models of biological systems. Chapter 2 formally defines a mathematical model as a *simulator* of a real world process. Typically, macro-level models are a complex combination of smaller scale models and involve multiple layers of differential equations, making the numerical solution computationally expensive to obtain. Parameter estimation requires many evaluations of the simulator for different parameter settings, hence it quickly becomes prohibitive considering the generic non-convexity of the estimation problem. To overcome this limitation, the mathematical model (simulator) is replaced by a statistical approximation of it, called *emulator*. Any inference based on the emulator will be an approximate, but computationally feasible, solution to the original problem. Chapter 3 reviews the type of statistical approximation most commonly used in the emulation literature: the Gaussian process. In Chapter 4, I show how emulation can be used to estimate the parameters of a nonlinear ordinary differential equation (ODE) known as the Lotka-Volterra model, without requiring at each likelihood evaluation step a numerical solution of the ODE for a given parameter setting. In Chapter 5, I estimate the parameters of a soft tissue mechanical model of the left ventricle of the heart, whose computational costs for a single output are in the order of 11 minutes CPU time[1]. Chapter 6 is a review of Bayesian optimization (BO): an estimation method where the emulator, instead of being fixed, is updated iteratively. Chapter 7 presents an application of BO to estimate the parameters of a partial differential equation (PDE) model of the human pulmonary circulation. Chapter 8 introduces a new acquisition function for Bayesian optimization and compares it with state-of-the-art algorithms from the BO literature on a large set of benchmark functions for global optimization. It then quantifies the reduction in function evaluations compared to standard global optimization solvers. The novel acquisition function is then used to estimate the parameters of the human pulmonary circulation PDE model described in Chapter 7. Finally, Chapter 9 summarizes the work done.

[1]Intel Xeon CPU, 2.9GHz, 32 cores and 32GB memory.

Notation

I conclude with a few words about the notation used throughout this thesis. Lowercase Latin and Greek letters denote ordinary scalar variables or functions. No distinction is made in the notation between variables and random variables, but the meaning should be clear from the context. Following Gentle (2009) I denote n-vectors either as

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \tag{1.1}$$

or

$$\boldsymbol{x} = (x_1, \ldots, x_n). \tag{1.2}$$

I make no distinction between the notations (1.1) and (1.2) as they represent the same entity. When used in combination with matrices, a vector is considered an $n \times 1$ column matrix:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix},$$

and by transposing the vector $\boldsymbol{x}$ using the superscript $^\top$ we obtain a row matrix of size $1 \times n$. Matrices are denoted using uppercase bold italic letters and square brackets: $\boldsymbol{A} = [a_{ij}]$. Sometimes it is useful to highlight the rows of an $n \times d$ matrix as follows: $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^\top$, where each $\boldsymbol{x}_i \in \mathbb{R}^d$ for $i = 1, \ldots, n$. If $\boldsymbol{A} = [\boldsymbol{a}_1, \ldots, \boldsymbol{a}_d]$ is an $n \times d$ matrix with columns $\boldsymbol{a}_j$ $(n \times 1)$, the *vectorization* of $\boldsymbol{A}$ is the nd-vector:

$$\text{vec}(\boldsymbol{A}) = (\boldsymbol{a}_1^\top, \ldots, \boldsymbol{a}_d^\top). \tag{1.3}$$

Uppercase calligraphic Latin letters such as $\mathcal{D}, \mathcal{X}, \mathcal{Y}$, usually denote sets; while blackboard bold style letters such as $\mathbb{R}$ are used for spaces. The letter $\mathcal{L}$ is used to denote the likelihood function, while L denotes the log likelihood. Throughout the text, a lowercase script ℓ denotes a generic loss function.

Part I

Emulation

Chapter 2

Simulators and Emulators

The main goal of this chapter is to introduce the concepts of *simulator* of a real world process (Section 2.1) and *emulator* of a computationally expensive simulator (Section 2.3). We overview the problem of estimating the parameters of a simulator in Section 2.2. This can be done by direct minimization of a loss function measuring the distance between the model output and experimental data; however this approach involves an expensive simulation at every iteration. To reduce the computational costs of the minimization problem we discuss two paradigms for approximate inference that, after an initial set of training runs, avoid further expensive simulations by predicting the desired value from a statistical emulator. The first approach involves emulating the simulator's output (Section 2.3.1), while the second entails direct emulation of the distances between the model output and the experimental data (Section 2.3.2). Section 2.4 discusses how to design the training runs.

2.1 Simulators

Modelling is the art of capturing the main features of a real world *system* or *process* and translating them into a mathematical or algorithmic form, the *model* or *simulator*. A model strips away the unnecessary low-level details by looking for regularity in the natural variability of things, in order to reach a more generic and widely applicable abstraction. Simulators are used to recreate the original system in silico, or to gain a deeper understanding of its constitutive elements and their interaction. Furthermore,

a simulator typically enjoys important features that are not present in the original process: it is easier to control, to reproduce, and less costly to observe. Simulators can be empirical or mechanistic models. The latter comprise models that encode a deep understanding about the system under investigation. However, mechanistic models include functional forms and parameters which are empirically determined.

In the last decades, with the increase in computational power, scientists started developing more complex simulators of real life processes. For example, by switching from linear to nonlinear differential equations (DEs), adding more layers of them, and interfacing many micro-level models in order to simulate macro-level phenomena. This research direction led to powerful multiscale computational models of entities that could not be described mathematically before, such as soft tissue mechanical models of the human heart (Wang et al., 2014) and the double-sided human pulmonary circulation (Qureshi et al., 2014). The downside of this, however, is that the cost of a single simulation (obtaining a model output) is the direct summation of the cost of simulating from each individual component, effectively making the simulation process computationally expensive. For example, a single simulation from the model presented by Wang et al. (2014) takes approximately 11 minutes CPU time[1].

Simulators are usually implemented as computational models spanning numerous lines of code and involving many tunable parameters, collected in a vector $\boldsymbol{q} \in \mathcal{Q} \subset \mathbb{R}^d$, which have a direct influence on the output. A simulator, m, can be thought of as a function taking a vector of inputs $\boldsymbol{q}$ and returning a possibly multivariate output $\boldsymbol{y} = \mathsf{m}(\boldsymbol{q}) \in \mathbb{R}^k$. We call $\mathsf{m}(\boldsymbol{q})$ the *simulation* at $\boldsymbol{q}$, and it represents the model's prediction of the real life phenomenon. Only this input-output relationship needs to be exploited in order to perform parameter estimation, hence the simulator can be effectively thought of as a *black-box function*. In this thesis we will only deal with deterministic simulators which will return the same output if run with the same input multiple times. On the other hand, stochastic simulators would return different outputs if run with the same input twice. Stochasticity is an inherent feature of natural phenomena, but stochastic simulators are substantially more complicated than deterministic ones, with added computational complexity. The likelihood function

[1]Intel Xeon CPU, 2.9GHz, 32 cores and 32GB memory.

for stochastic differential equation (SDE) models typically needs to be approximated as the transition density is often not available in closed-form, except in a few cases (Iacus, 2008). On top of that, we also have to run the inferential algorithm, e.g. maximum likelihood estimation, which iteratively calls for an approximation of the likelihood. Under certain conditions, deterministic models have proven to be a useful approximation to stochastic processes.

2.2 Estimation

Given experimental data $\boldsymbol{y}^{\text{obs}}$, assumed to come from the same generative model m, the goal is to find the optimal parameter vector $\hat{\boldsymbol{q}}$ leading to a prediction $\mathsf{m}(\hat{\boldsymbol{q}})$ as close as possible to the data $\boldsymbol{y}^{\text{obs}}$. Let the *target loss*[2] be the non-negative function

$$\ell_{\mathsf{m}}(\boldsymbol{q}) = d\left(\mathsf{m}(\boldsymbol{q}), \boldsymbol{y}^{\text{obs}}\right)^2, \tag{2.1}$$

where $d(\cdot,\cdot)$ is a metric measuring the distance between the simulation at $\boldsymbol{q}$ and the experimental data. The estimate $\hat{\boldsymbol{q}}$ is the value of $\boldsymbol{q}$ that minimizes the loss (2.1):

$$\hat{\boldsymbol{q}} = \arg\min_{\boldsymbol{q}\in\mathcal{Q}} \ell_{\mathsf{m}}(\boldsymbol{q}). \tag{2.2}$$

For any $\boldsymbol{y}_i, \boldsymbol{y}_j \in \mathbb{R}^k$, possible choices for $d(\cdot,\cdot)$ are the Euclidean distance:

$$d_2(\boldsymbol{y}_i, \boldsymbol{y}_j) = \|\boldsymbol{y}_i - \boldsymbol{y}_j\| = \left[\sum_{t=1}^{k} (y_{it} - y_{jt})^2\right]^{1/2}, \tag{2.3}$$

or, more generally, the Minkowski distance of order p:

$$d_p(\boldsymbol{y}_i, \boldsymbol{y}_j) = \left[\sum_{t=1}^{k} |y_{it} - y_{jt}|^p\right]^{1/p}. \tag{2.4}$$

[2]In this context, the word *loss* does not carry the same meaning as in decision theory. The target loss represents a generic real-valued function measuring the distance between the observed data and the simulation at $\boldsymbol{q}$, and hence it should be considered as an error measure. It is used to unify the notation and make it consistent across the thesis for different distance functions $d(\cdot,\cdot)$. Small values of the target loss are preferred to large values and mean that the input $\boldsymbol{q}$ gives rise to simulated data which is close or similar to the observed data, hence the value of $\boldsymbol{q}$ is plausible.

If the outputs are correlated and coming from a distribution with covariance $\boldsymbol{\Sigma}$, a possible choice is represented by the Mahalanobis distance:

$$d_M(\boldsymbol{y}_i, \boldsymbol{y}_j) = \left[(\boldsymbol{y}_i - \boldsymbol{y}_j)^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{y}_i - \boldsymbol{y}_j)\right]^{1/2}. \tag{2.5}$$

If the data $\boldsymbol{y}^{\text{obs}}$ are assumed to come from a Gaussian distribution centred at the true simulation with variance $\sigma^2 \boldsymbol{I}$, minimizing the squared Euclidean distance corresponds to maximizing the log likelihood. Each evaluation of the objective function $\ell_{\mathsf{m}}(\boldsymbol{q})$ involves a costly forward simulation $\mathsf{m}(\boldsymbol{q})$, hence we wish to use as few queries as possible. If the unit cost for a simulation is t seconds, the total waiting time to estimate $\hat{\boldsymbol{q}}$ will be $n_{\max} \times t$, with $n_{\max}$ being the total number of function evaluations required by the optimization algorithm. Many global optimization algorithms have been proposed in the literature, such as genetic algorithms, multistart and simulated annealing methods (Locatelli and Schoen, 2013). However, these methods require many function evaluations and are hence designed for functions that are cheap to query. The computational complexity of the problem also rules out any Markov chain Monte Carlo (MCMC) based inference method. In real-time decision making, such as in-clinic decision support systems, reservoir management and monitoring of volcanic activity, a decision has to be taken quickly. The unit cost of a single simulation $\mathsf{m}(\boldsymbol{q})$ sets a computational limit on the number of function evaluations allowed, effectively calling for a careful selection of each query point in order to maximize the information gained.

2.3 Emulators

In order to reduce the computational burden brought by the increasing complexity of the developed simulators, lots of attention has been drawn to the concept of *emulation* (Kennedy and O'Hagan, 2001; O'Hagan, 2006). An *emulator* $\hat{\mathsf{m}}$, also known as *surrogate model* or *metamodel*, is a statistical approximation of the black-box function m based on a set of costly *training runs*:

$$\mathcal{D} = \{\boldsymbol{q}_i, \boldsymbol{y}_i = \mathsf{m}(\boldsymbol{q}_i)\}_{i=1}^n. \tag{2.6}$$

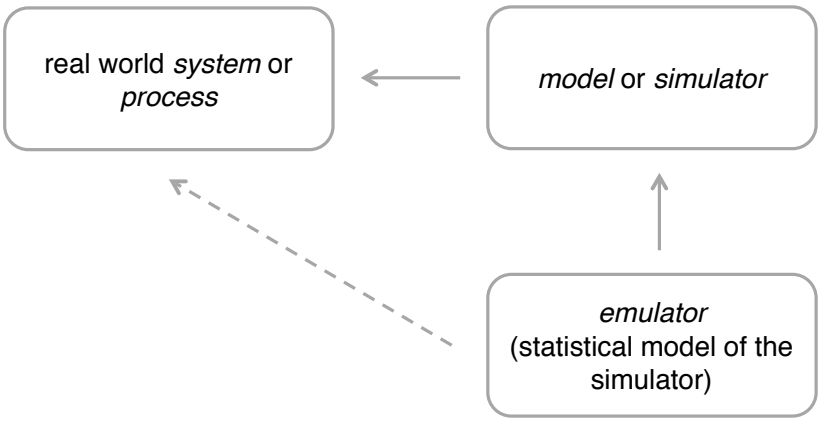


Figure 2.1: Diagram illustrating the concepts of simulator and emulator. A simulator approximates the real world system (solid arrow). At the same time, an emulator is an approximation to the simulator, hence the solid arrow. Being a double approximation, the emulator indirectly models the real world system (dashed arrow).

The training simulations should be obtained by exploiting the fact that all n runs used to fit the surface can be done in parallel, even before seeing any experimental data. Whenever a simulation from the black-box function is needed at a point which has not been run before, the costly value $\mathsf{m}(\boldsymbol{q})$ is replaced by a fast prediction from the surrogate model $\hat{\mathsf{m}}(\boldsymbol{q})$. Figure 2.1 shows a diagram representing the concepts of simulator and emulator. The solid arrows indicate that a simulator is an approximation to a real world process. At the same time, an emulator is a statistical approximation of the simulator. The dashed arrow, instead, shows the indirect effect of the emulator which, being a double approximation, indirectly models the real world system. More details about the type of statistical model used in the literature, called Gaussian process, can be found in Chapter 3. Section 2.3.1 and 2.3.2 discuss two different approaches to estimate $\hat{\boldsymbol{q}}$ while substantially decreasing the computational costs required to solve the minimization problem in (2.2).

2.3.1 Output Emulation

Output emulation represents the strategy of directly emulating the model output, i.e. replacing $\mathsf{m}(\boldsymbol{q})$ by $\hat{\mathsf{m}}(\boldsymbol{q})$. Different strategies have been considered in the literature

for emulating simulators which return a multivariate output (see Conti and O'Hagan (2010) for a review):

1. Ensemble of single-output emulators (MS)
2. Multivariate-output Gaussian processes (MO)
3. Input augmentation (IA)

Each strategy has some advantages over the others, either in terms of computational efficiency or modelling flexibility.

If the model is multivariate, i.e. $\mathsf{m} = (\mathsf{m}_1, \ldots, \mathsf{m}_k)$, the first approach (MS) fits k independent real-valued emulators $\hat{\mathsf{m}}_j(\boldsymbol{q})$ of $y_j = \mathsf{m}_j(\boldsymbol{q})$ for $j = 1, \ldots, k$, and considers the multivariate surrogate model as the vector $\hat{\mathsf{m}} = (\hat{\mathsf{m}}_1, \ldots, \hat{\mathsf{m}}_k)$. A prediction from $\hat{\mathsf{m}}(\boldsymbol{q})$ is then obtained by predicting from each univariate component $\hat{\mathsf{m}}_j(\boldsymbol{q})$ for $j = 1, \ldots, k$. If multiple cores are available on the machine, it is possible to take advantage of the parallel nature of the fitting and prediction tasks by fitting (or predicting from) a univariate emulator on each core and obtaining k emulators (or predictions) at the cost of one.

The second strategy (MO), discussed in Conti et al. (2009) and Conti and O'Hagan (2010), involves using a k-dimensional Gaussian process as the multivariate emulator of m. However, this comes with additional issues that make the inferential problem more challenging: the covariance model becomes more costly, there are frequent numerical instabilities[3], and a larger number of hyperparameters (full matrices) have to be inferred. Furthermore, starting from a k-dimensional Gaussian process prior

[3]As an example, consider a given number of points uniformly covering a sphere in a d-dimensional Euclidean space. As we increase the dimensionality of the space, the points tend to move more and more towards the outer shell. In higher dimensions, most of the volume is contained on the surface. The data now lie on a lower-dimensional submanifold, leading to rank deficiency and lots of eigenvalues near zero. For Gaussian processes, the issue resides in the inversion of the training covariance matrix. When the points are highly correlated, the condition number of the matrix is high, meaning that the solution of a linear system involving that matrix is highly sensitive and prone to numerical errors. This leads to numerical issues in the inversion of the covariance matrix. A common solution to improve the condition number is to add a small value, e.g. 10^{-6}, to the diagonal of the matrix.

on m, the conditional posterior distribution of m given the kernel hyperparameters and the data is a k-dimensional t-process instead of a Gaussian process.

Input augmentation, discussed by Roberts et al. (2012), considers the output label j as an extra input, where $y_j = \mathsf{m}_j(\boldsymbol{q})$ is represented as $y = \mathsf{m}^*(\boldsymbol{q}, j)$, with $j = 1, \ldots, k$. Instead of building a multivariate-output emulator of the simulator $\boldsymbol{y} = \mathsf{m}(\boldsymbol{q})$, this approach builds a single-output emulator of $\mathsf{m}^*(\cdot, \cdot)$ with domain $\mathcal{Q} \times \{1, \ldots, k\}$. A similar approach was tried in the MSc project by Huang (2016), which I co-supervised, where the label $j = 1, \ldots, k$ was replaced by an ordering induced by the location along the first principal component of the multivariate data matrix. However, the results were not encouraging. This might be due to the information loss incurred by mapping the outputs to a linear subspace. Rather then trying non-linear variants of PCA, like self-organizing maps or generative topographic maps, it was decided to pursue the simpler approach of fitting independent univariate emulators. We further remark that the application presented in Figure 17 of Roberts et al. (2012) uses a dataset with large intervals of missing data. While in that scenario there is a clear benefit in sharing information between the multiple outputs, especially when predicting future values of time series over a left-bounded and right-unbounded interval, in our work we will only deal with compact sets, i.e. closed and bounded, where the inputs are fairly regularly spaced and dense. In this scenario independent real-valued Gaussian processes work well as any test point will always have at least one training point in a sufficiently small neighbourhood. Furthermore, because MS fits separate independent emulators of each output variable, the Gaussian process hyperparameters are allowed to be different across the k models, hence allowing for more flexibility. On the contrary, MO and IA assume sharing of the kernel hyperparameters for all $j = 1, \ldots, k$. In general, we have no reason to believe that all the k responses to changes of a given input will share the same smoothness behaviour.

The estimation problem in (2.2) can be approximated by replacing any query to the expensive simulator $\mathsf{m}(\boldsymbol{q})$ by a call to the surrogate model $\hat{\mathsf{m}}(\boldsymbol{q})$. This leads to a loss function which does not involve any further costly simulations and can be optimized using standard optimization algorithms found e.g. in Locatelli and Schoen

(2013). The *surrogate-based loss*, given a metric $d(\cdot, \cdot)$, is the positive function:

$$\ell_{\hat{\mathsf{m}}}(\boldsymbol{q}) = d\left(\hat{\mathsf{m}}(\boldsymbol{q}), \boldsymbol{y}^{\text{obs}}\right)^2. \tag{2.7}$$

The estimate

$$\hat{\boldsymbol{q}} = \arg\min_{\boldsymbol{q} \in \mathcal{Q}} \ell_{\hat{\mathsf{m}}}(\boldsymbol{q})$$

represents an approximate, but computationally feasible, solution to the minimization of the target loss (2.2).

Emulating the output has two drawbacks: (1) the multivariate emulator $\hat{\mathsf{m}}$ requires fitting k independent emulators $\hat{\mathsf{m}}_j$. If the outputs are correlated, this approach clearly does not use any of the information from the other variables; (2) each evaluation of $\ell_{\hat{\mathsf{m}}}$ involves predicting from k univariate emulators. If the computer does not have enough cores, ideally k, the cost for a prediction from $\hat{\mathsf{m}}$ will be the sum of the cost of predicting from each $\hat{\mathsf{m}}_j$ for $j = 1, \ldots, k$.

2.3.2 Loss Emulation

Recall that the final goal is to estimate the vector of parameters by minimizing the expensive-to-evaluate objective function ℓ_{m}, defined in (2.1), using only a few costly simulations. *Loss emulation* overcomes both problems mentioned at the end of Section 2.3.1 by reducing the dimensionality of the outputs in $\mathcal{D}$. It entails emulating the real-valued objective function $\ell_{\mathsf{m}}(\boldsymbol{q})$ instead of the multivariate output $\boldsymbol{y} = \mathsf{m}(\boldsymbol{q})$. This requires an additional postprocessing step of the simulations in $\mathcal{D}$. This step reduces the dimensionality of the training outputs from kD to 1D, using the mapping

$$\mathsf{m}(\boldsymbol{q}_i) \in \mathbb{R}^k \mapsto \ell_{\mathsf{m}}(\boldsymbol{q}_i) \in \mathbb{R}, \tag{2.8}$$

which does not involve any further expensive simulations. Then, instead of fitting a multivariate emulator of the outputs $y_1, \ldots, y_k$, the target of the emulation is the univariate loss $\ell_{\mathsf{m}}(\boldsymbol{q})$. The statistical approximation of the data $\mathcal{D} = \{\boldsymbol{q}_i, \ell_{\mathsf{m}}(\boldsymbol{q}_i)\}_{i=1}^n$, is denoted by $\hat{\ell}_{\mathsf{m}}(\boldsymbol{q})$ and will be called *emulated or surrogate loss*. It is worth noting that emulation of the surrogate-based loss $\ell_{\hat{\mathsf{m}}}(\boldsymbol{q})$, instead, would be meaningless as it would entail approximating a quantity which is already fast to query.

2.4 Training Runs

We are now left with the discussion on how to design the inputs of the training data (2.6) which are used to fit the emulator. Because of the computational complexity of each simulation, we aim to pick each training input $\boldsymbol{q}_i$ in order to cover the whole parameter domain $\mathcal{Q}$ as effectively as possible. Let $\boldsymbol{q} = (q_1, \ldots, q_d)$ denote a generic element of $\mathcal{Q}$. The simplest approach involves defining a grid $\boldsymbol{g}_k \in \mathbb{R}^G$ between a lower and upper bound for each coordinate q_k $(k = 1, \ldots, d)$:

$$\boldsymbol{g}_k : \mathrm{lb}_k = q_{k1} < \cdots < q_{kG} = \mathrm{ub}_k.$$

The total number of points $\boldsymbol{q}_i$ at which a simulation is required equals G^d, which quickly becomes prohibitive. For example, for $G = 100$ the number of required forward simulations would become one million for a simple 3D Euclidean space.

Another possibility would be drawing samples from a uniform distribution in the d-dimensional domain. However, this can easily lead to points being clustered together. In the emulation point of view, unlike Monte Carlo theory, this would be a sub-optimal design choice. Computer codes are often deterministic, i.e. by running the code with the same inputs twice we get the same output, and furthermore the outputs at inputs which are close together are often similar, hence implying some sort of correlation based on the distance between the inputs (Jones et al., 1998). In light of these observations, once we have waited for a lengthy computation and observed an output at $\boldsymbol{q}_i$, we would not gain much information on the function behaviour by adding another evaluation in a small neighbourhood of $\boldsymbol{q}_i$. It would be rather more informative querying in areas that have been less explored. The emulation literature suggests the use of Latin hypercube designs or Sobol sequences, see Jones et al. (1998); Santner et al. (2003) and Fang et al. (2006).

Figure 2.2 shows a comparison of four different choices for the inputs $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n$. The plots show 100 points in the 2D Euclidean space $[0, 1]^2$ using (from top left to bottom right) a grid-based approach, uniform random points, a Latin hypercube design and points from the Sobol sequence. The grid points cover the space very regularly, without clustering of points. The resolution of the grid depends on the number of points to be generated, and vice versa. A grid-based approach is mostly

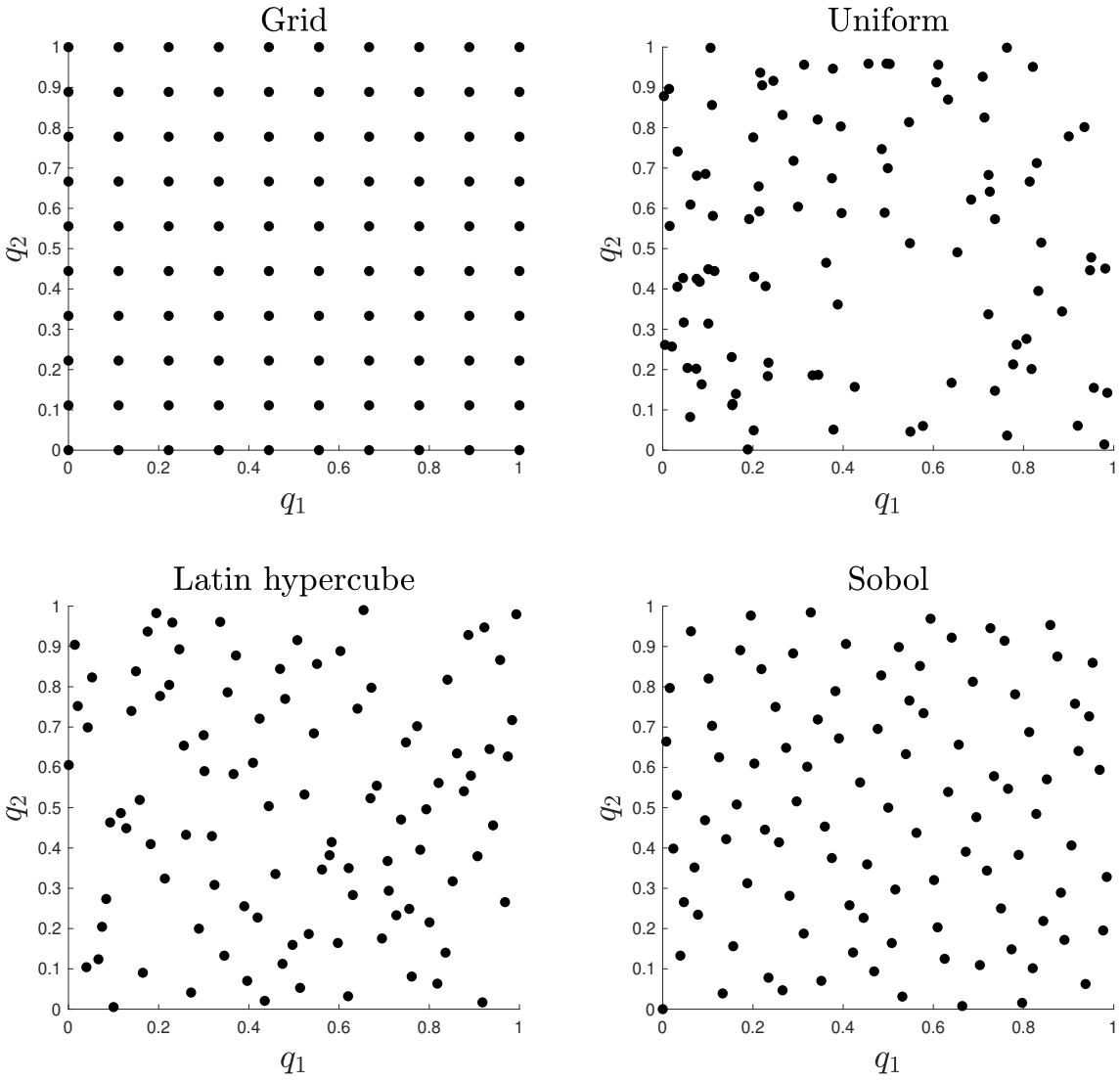


Figure 2.2: A comparison of different design choices for the training inputs. The plots show 100 points $\{\boldsymbol{q}_i\}$ in the 2D space $[0, 1]^2$ using different design choices.

efficient in low dimensional Euclidean spaces as the number of gridpoints increases exponentially with the space dimensionality, limiting the applicability of this method when each simulation is expensive. Sampling from a uniform distribution (top right panel) can lead to points that are next to previously run ones, hence not using the computational time efficiently to explore the whole domain. In contrast, better approaches that scale to high dimensions are represented by Latin hypercubes and Sobol sequences. They both try to cover the space more thoroughly by using a lower number of points compared to uniform random samples. Latin hypercube sampling and Sobol sequences are widely used in the emulation literature, but with different goals. The advantage of the Sobol sequence is that it can be easily extended by adding more points to the shorter sequence. This happens in cases when, once the emulator has been fitted, the estimation is very poor and hence more information about the underlying computer algorithm is needed. However, this is not so straightforward for Latin hypercubes. For the same space dimensionality d and random number generator seed, if we need to go from n to $n + m$ points, we need to generate a new Latin hypercube from scratch and the newly generated $n + m$ points do not include as a subset the first n ones (Santner et al., 2003). The Sobol sequence is widely used in the emulation literature (Santner et al., 2003), while the Latin hypercube design is the standard choice in the Bayesian optimization literature (Jones et al., 1998), where the emulator instead of being fixed is updated iteratively. Both are valid space-filling design choices. Furthermore, Bayesian optimization (discussed in Part 2) is used when the simulator is not considered as fixed, but is undergoing developments. In that scenario it would be sub-optimal spending months, computational power and electricity to generate training runs for a given computer code when a new version of the code is due to be released soon as the fitted emulator would not be an image of the improved mathematical simulator.

When the bottleneck is obtaining outputs from the simulator, also compressive sampling (Candès and Wakin, 2008) could be considered. This is based on the observation that signals could have a sparse representation in a suitable basis. Then, by only recording as measurements random linear combinations of the signal, it is possible to reconstruct the original signal with far less samples than those required

by the Nyquist–Shannon sampling theorem.

2.5 Summary

This chapter defined the concepts of simulator of a real-world process and emulator of a computationally expensive simulator. The problem of estimating the parameters of an expensive simulator was discussed, and two alternative strategies to approximate and speed up the inference have been described. Output emulation (strategy 1) involves fitting a statistical model to the simulator's output. Loss emulation (strategy 2) entails direct emulation of the distances between the training runs and the experimental data. Either emulation strategy can be used to accelerate the inference as when a simulation (or the loss) is requested at a point that is not part of the training runs, the value is predicted by the corresponding statistical model.

Chapter 3

Gaussian Processes

This chapter reviews nonparametric regression using Gaussian processes (GPs), which represents the type of emulator commonly used in the literature. Section 3.1 shows that the conditional expectation function (CEF) is the best predictor, hence we will consider the prediction from the emulator to be the CEF of the predictive distribution. Section 3.2 summarizes the simple Bayesian linear regression model and its generalization using basis functions in order to model nonlinear input-output relationships. Section 3.3 explains the link between Bayesian linear regression and GPs. Section 3.4 gives the definition of a stochastic process and Section 3.5 formally introduces GPs, which are completely specified by a mean and a covariance function. Section 3.6 describes the classes of covariance functions commonly used in the literature. Section 3.7 illustrates how to sample from a Gaussian process prior over functions, while Section 3.8 discusses how to update the prior in light of the data. Section 3.9 shows how to obtain samples from the posterior GP, while estimation of the model hyperparameters is discussed in Section 3.10. For a general introduction to GPs see Roberts et al. (2012), while for more details refer to the book by Rasmussen and Williams (2006). The mathematically-oriented reader interested in the link between Gaussian processes and reproducing kernel Hilbert spaces can also look at Appendix E for a short description or, for more details, at Chapter 6 in Rasmussen and Williams (2006) and Wahba (1990, 1999).

3.1 Best Predictor

Suppose that we are interested in a phenomenon involving several observable variables. The goal of supervised learning is to learn a mapping from a vector of d inputs $\boldsymbol{x} \in \mathcal{X}$ to an output $y \in \mathcal{Y}$, using a set of training data $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$. The learned mapping is then used to predict (estimate) the value of the response y from given values of the regressors $\boldsymbol{x} = (x_1, \ldots, x_d)$. Depending on the nature of the response variable, the problem is called *regression* if the outcome variable is continuous or *classification* if the response is categorical.

A predictive model is a relationship linking the output y to the input $\boldsymbol{x}$ according to the generative model:

$$y = f(\boldsymbol{x}) + \varepsilon, \qquad \mathbb{E}(\varepsilon \mid \boldsymbol{x}) = 0. \tag{3.1}$$

The term ε represents a random error which is not observable and makes sure that to a given value of $\boldsymbol{x}$ corresponds a variety of values of y. Equation (3.1), along with the zero conditional mean assumption on the error, implies that $f(\boldsymbol{x}) = \mu(\boldsymbol{x})$, where $\mu(\boldsymbol{x}) = \mathbb{E}(y \mid \boldsymbol{x})$ is known as the conditional expectation function (CEF). In regression, $f(\cdot)$ predicts the average value of y when the observed value of the independent random variables equals $\boldsymbol{x}$. In a binary classification problem, i.e. $y \in \{0, 1\}$, the function $f(\cdot)$ is interpreted as $f(\boldsymbol{x}) = \mathbb{E}(y \mid \boldsymbol{x}) = \mathbb{P}(y = 1 \mid \boldsymbol{x})$. Knowing $f(\cdot)$ means being able to predict the probability of y belonging to class 1 (success), when the observed value of the independent variables is $\boldsymbol{x}$.

The estimation of $f(\cdot)$ is performed using n observations for which the values of all the variables in the problem are known: $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$. The estimated function $\hat{f}(\boldsymbol{x})$ will be used to predict the value of y when only the value of the independent variables $\boldsymbol{x}$ is known. There are two approaches to the problem. The *parametric* one assumes that f belongs to a given family of functions $\{f_{\boldsymbol{w}}(\cdot)\}_{\boldsymbol{w} \in \mathcal{W}}$ with functional form depending on a number of unknown constants $\boldsymbol{w}$, called parameters. In this case the estimation of the regression function involves estimating the parameters: $\hat{f}_{\boldsymbol{w}}(\cdot) = f_{\hat{\boldsymbol{w}}}(\cdot)$. The *nonparametric* approach does not put any major structural constraints on the functional form of $f(\cdot)$, and its shape is informed by the data only. In this approach there are no such structural parameters

$\boldsymbol{w}$ to estimate, but rather $f(\boldsymbol{x})$ will have to be estimated pointwise for all $\boldsymbol{x}$.

A predictor is any function $f(\boldsymbol{x})$ of the independent variable $\boldsymbol{x}$. Define the prediction error to be $y - f(\boldsymbol{x})$ and quantify the magnitude of the error in terms of the mean squared prediction error (MSPE): $\mathbb{E}[y - f(\boldsymbol{x})]^2$.

Theorem 3.1. *The CEF $\mu(\boldsymbol{x}) = \mathbb{E}(y \mid \boldsymbol{x})$ is the best predictor of y for a given value of the independent variables $\boldsymbol{x}$. Any other predictor $f(\boldsymbol{x})$ will have a higher MSPE:*

$$\mathbb{E}[y - f(\boldsymbol{x})]^2 \geq \mathbb{E}[y - \mu(\boldsymbol{x})]^2. \tag{3.2}$$

Proof. If $\mathbb{E}[y - f(\boldsymbol{x})]^2 < \infty$, then:

$$\begin{aligned}
\mathbb{E}[y - f(\boldsymbol{x})]^2 &= \mathbb{E}[y - \mu(\boldsymbol{x}) + \mu(\boldsymbol{x}) - f(\boldsymbol{x})]^2 \\
&= \mathbb{E}[y - \mu(\boldsymbol{x})]^2 + \mathbb{E}[\mu(\boldsymbol{x}) - f(\boldsymbol{x})]^2 + 2\mathbb{E}\left\{[y - \mu(\boldsymbol{x})][\mu(\boldsymbol{x}) - f(\boldsymbol{x})]\right\} \\
&= \mathbb{E}[y - \mu(\boldsymbol{x})]^2 + \mathbb{E}[\mu(\boldsymbol{x}) - f(\boldsymbol{x})]^2 + 2\mathbb{E}\left\{[\mu(\boldsymbol{x}) - f(\boldsymbol{x})]\mathbb{E}[y - \mu(\boldsymbol{x}) \mid \boldsymbol{x}]\right\} \\
&= \mathbb{E}[y - \mu(\boldsymbol{x})]^2 + \mathbb{E}[\mu(\boldsymbol{x}) - f(\boldsymbol{x})]^2 + 2\mathbb{E}\left\{[\mu(\boldsymbol{x}) - f(\boldsymbol{x})]0\right\} \\
&= \mathbb{E}[y - \mu(\boldsymbol{x})]^2 + \mathbb{E}[\mu(\boldsymbol{x}) - f(\boldsymbol{x})]^2 \\
&\geq \mathbb{E}[y - \mu(\boldsymbol{x})]^2.
\end{aligned}$$

□

Because of Theorem 3.1, whenever a prediction of y is required for a given value of the independent variables $\boldsymbol{x}$, this will be taken to be the value of the conditional expectation function.

3.2 From Bayesian Linear Models to Gaussian Processes

This section focuses on the regression problem, where the outputs y_i are assumed to be noisy realizations of a latent function $f_{\boldsymbol{w}}(\boldsymbol{x}_i)$, with $\boldsymbol{w} = (w_1, \ldots, w_d)$ representing unknown structural parameters of $f(\cdot)$. Equation (3.2) shows that the best predictor is the CEF; any other $f(\cdot)$ will have a higher MSPE. The CEF is often a complex

nonlinear function of $\boldsymbol{x}$, but in some applications it can be approximated with a linear function of both inputs and parameters:

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{w}, \qquad y = f_{\boldsymbol{w}}(\boldsymbol{x}) + \varepsilon,$$

where ε is assumed to be an additive, independent and identically distributed $\mathsf{N}(0, \sigma^2)$ noise term. The linearity in the inputs, however, imposes a significant limitation to the model. If the true generative model is not linear in the inputs, this would lead to poor predictive power.

To overcome this problem, a solution is to map the d-dimensional input to a p-dimensional feature space, where usually $p \geq d$, using a fixed set of basis functions $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_p)$. Now, by replacing $\boldsymbol{x}$ with the feature vector $\boldsymbol{\phi}(\boldsymbol{x})$, it is possible to model nonlinear relationships between input and output variables while still having a functional form for $f(\cdot)$ which is linear in the (now p-dimensional) parameter vector $\boldsymbol{w} = (w_1, \ldots, w_p)$:

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{w} = \sum_{j=1}^{p} w_j \phi_j(\boldsymbol{x}).$$

As an example, if x is univariate, the set of basis functions $\boldsymbol{\phi}(x) = (1, x, x^2, \ldots, x^p)$ leads to polynomial regression. Usually the first basis function is $\phi_1(\boldsymbol{x}) = 1$ in order to allow for an intercept term in the model.

A fully Bayesian treatment of the linear regression model

$$y = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{w} + \varepsilon, \qquad \varepsilon \sim \mathsf{N}(0, \sigma^2) \text{ independently}, \tag{3.3}$$

requires assuming a prior on the regression weights; for example a multivariate Gaussian:

$$\boldsymbol{w} \sim \mathsf{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}). \tag{3.4}$$

Given data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, we need to obtain the posterior distribution $\boldsymbol{w} \mid \mathcal{D}$ in order to find the predictive distribution of an outcome y given the data. Let $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^\top$ denote the $n \times d$ matrix of training inputs, $\boldsymbol{y} = (y_1, \ldots, y_n)$ the

n-vector of training outputs, and $\boldsymbol{\Phi}$ the $n \times p$ matrix of feature vectors:

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^\top \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_n)^\top \end{bmatrix} = \begin{bmatrix} \phi_1(\boldsymbol{x}_1) & \cdots & \phi_p(\boldsymbol{x}_1) \\ \vdots & \ddots & \vdots \\ \phi_1(\boldsymbol{x}_n) & \cdots & \phi_p(\boldsymbol{x}_n) \end{bmatrix}.$$

The linear model (3.3) at the training inputs becomes:

$$\boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{w} + \boldsymbol{\varepsilon}, \qquad \boldsymbol{\varepsilon} \sim \mathsf{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I}). \tag{3.5}$$

Define $\boldsymbol{f} = (f_{\boldsymbol{w}}(\boldsymbol{x}_1), \ldots, f_{\boldsymbol{w}}(\boldsymbol{x}_n)) = \boldsymbol{\Phi}\boldsymbol{w}$. Being a linear transformation of $\boldsymbol{w}$, its distribution is again Gaussian:

$$\boldsymbol{f} \mid \boldsymbol{X} \sim \mathsf{N}(\boldsymbol{\Phi}\boldsymbol{\mu}, \boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Phi}^\top).$$

The marginal distribution of $\boldsymbol{y}$ is obtained by integrating over $\boldsymbol{f}$:

$$p(\boldsymbol{y} \mid \boldsymbol{X}) = \int p(\boldsymbol{y} \mid \boldsymbol{f}) p(\boldsymbol{f} \mid \boldsymbol{X}) \mathrm{d}\boldsymbol{f}. \tag{3.6}$$

Since $p(\boldsymbol{f} \mid \boldsymbol{X})$ is a Gaussian marginal distribution and $p(\boldsymbol{y} \mid \boldsymbol{f})$ a conditional Gaussian distribution with mean being a linear function of $\boldsymbol{f}$ and variance independent of $\boldsymbol{f}$ (see 3.5), we can apply standard results found e.g. in (2.115) of Bishop (2006):

$$\boldsymbol{y} \mid \boldsymbol{X} \sim \mathsf{N}(\boldsymbol{\Phi}\boldsymbol{\mu}, \boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Phi}^\top + \sigma^2 \boldsymbol{I}).$$

This shows that $\boldsymbol{y} = \boldsymbol{f} + \boldsymbol{\varepsilon}$, being the sum of two independent multivariate Gaussian random variables, inherits randomness from both of them, and the variances are simply added because of the independence assumption.

The covariance between $\boldsymbol{y}$ and $\boldsymbol{w}$ is:

$$\begin{aligned} \mathrm{Cov}(\boldsymbol{y}, \boldsymbol{w}) &= \mathrm{Cov}(\boldsymbol{\Phi}\boldsymbol{w} + \boldsymbol{\varepsilon}, \boldsymbol{w}) \\ &= \boldsymbol{\Phi}\mathrm{Cov}(\boldsymbol{w}, \boldsymbol{w}) + \mathrm{Cov}(\boldsymbol{\varepsilon}, \boldsymbol{w}) \\ &= \boldsymbol{\Phi}\boldsymbol{\Sigma}, \end{aligned}$$

from the properties of the covariance and the independence between the noise $\boldsymbol{\varepsilon}$ and $\boldsymbol{w}$. Using the quantities derived above, the joint distribution of $\boldsymbol{y}$ and $\boldsymbol{w}$ is:

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{w} \end{bmatrix} \mid \boldsymbol{X} \sim \mathsf{N}\left(\begin{bmatrix} \boldsymbol{\Phi}\boldsymbol{\mu} \\ \boldsymbol{\mu} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Phi}^\top + \sigma^2 \boldsymbol{I} & \boldsymbol{\Phi}\boldsymbol{\Sigma} \\ \boldsymbol{\Sigma}\boldsymbol{\Phi}^\top & \boldsymbol{\Sigma} \end{bmatrix} \right).$$

The posterior distribution of $\boldsymbol{w}$ given $\mathcal{D}$ is obtained by applying the conditioning formulas for multivariate Gaussians, see Appendix C.2 in Davidson (2000):

$$\begin{aligned}\boldsymbol{w} \mid \mathcal{D} &\sim \mathsf{N}(\hat{\boldsymbol{w}}, \boldsymbol{S}_{\boldsymbol{w}}) \\ \hat{\boldsymbol{w}} &= \boldsymbol{\mu} + \boldsymbol{\Sigma}\boldsymbol{\Phi}^\top \left(\boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Phi}^\top + \sigma^2 \boldsymbol{I}\right)^{-1} (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\mu}) \\ \boldsymbol{S}_{\boldsymbol{w}} &= \boldsymbol{\Sigma} - \boldsymbol{\Sigma}\boldsymbol{\Phi}^\top \left(\boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Phi}^\top + \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{\Phi}\boldsymbol{\Sigma}.\end{aligned}$$

Conditioning on $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$ is equivalent to conditioning on $\boldsymbol{y}$, since $\boldsymbol{X}$ is assumed to be given and fixed.

Let $\boldsymbol{X}_* = [\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_m^*]^\top$ denote an $m \times d$ matrix of test inputs. We wish to predict the unseen values of the response variable y at $\boldsymbol{X}_*$ using our model:

$$\boldsymbol{f}_* = \begin{bmatrix} f_{\boldsymbol{w}}(\boldsymbol{x}_1^*) \\ \vdots \\ f_{\boldsymbol{w}}(\boldsymbol{x}_m^*) \end{bmatrix} = \begin{bmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1^*)^\top \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_m^*)^\top \end{bmatrix} \boldsymbol{w} = \boldsymbol{\Phi}_* \boldsymbol{w}.$$

The posterior distribution of the regression function at the test inputs (*predictive distribution*) is:

$$\begin{aligned}\boldsymbol{f}_* \mid \mathcal{D} &\sim \mathsf{N}(\hat{\boldsymbol{f}}_*, \boldsymbol{S}_{\boldsymbol{f}_*}) \\ \hat{\boldsymbol{f}}_* &= \boldsymbol{\Phi}_* \hat{\boldsymbol{w}} = \boldsymbol{\Phi}_* \boldsymbol{\mu} + \boldsymbol{\Phi}_* \boldsymbol{\Sigma}\boldsymbol{\Phi}^\top \left(\boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Phi}^\top + \sigma^2 \boldsymbol{I}\right)^{-1} (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\mu}) \\ \boldsymbol{S}_{\boldsymbol{f}_*} &= \boldsymbol{\Phi}_* \boldsymbol{S}_{\boldsymbol{w}} \boldsymbol{\Phi}_*^\top = \boldsymbol{\Phi}_* \boldsymbol{\Sigma}\boldsymbol{\Phi}_*^\top - \boldsymbol{\Phi}_* \boldsymbol{\Sigma}\boldsymbol{\Phi}^\top \left(\boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Phi}^\top + \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Phi}_*^\top\end{aligned} \tag{3.7}$$

and corresponds to integrating out the parameters $\boldsymbol{w}$, i.e. averaging the prediction $p(\boldsymbol{f}_* \mid \boldsymbol{w})$ for a given value of the parameters $\boldsymbol{w}$ over the posterior $p(\boldsymbol{w} \mid \mathcal{D})$:

$$p(\boldsymbol{f}_* \mid \mathcal{D}) = \int p(\boldsymbol{f}_* \mid \boldsymbol{w}) p(\boldsymbol{w} \mid \mathcal{D}) \mathrm{d}\boldsymbol{w}.$$

In (3.7) the inputs enter only through the inner products $\boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Phi}^\top$, $\boldsymbol{\Phi}_*\boldsymbol{\Sigma}\boldsymbol{\Phi}^\top$, $\boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Phi}_*^\top$, and $\boldsymbol{\Phi}_*\boldsymbol{\Sigma}\boldsymbol{\Phi}_*^\top$, which have as generic $(i,j)^{\text{th}}$ entry an inner product of the form $\boldsymbol{\phi}(\boldsymbol{x}_i)^\top \boldsymbol{\Sigma} \boldsymbol{\phi}(\boldsymbol{x}_j)$. Since $\boldsymbol{\Sigma}$ is the covariance matrix of $\boldsymbol{w}$, it is a symmetric and positive semidefinite matrix. By applying the Spectral theorem, we can find a decomposition $\boldsymbol{\Sigma} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top$, with $\mathbf{Q}$ orthogonal and $\mathbf{D}$ diagonal with nonnegative entries, so that the matrix square root is $\boldsymbol{\Sigma}^{1/2} = \mathbf{Q}\mathbf{D}^{1/2}\mathbf{Q}^\top$. Let $\boldsymbol{\gamma}(\boldsymbol{x}) = \boldsymbol{\Sigma}^{1/2}\boldsymbol{\phi}(\boldsymbol{x})$ and define the function:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\phi}(\boldsymbol{x}_i)^\top \boldsymbol{\Sigma} \boldsymbol{\phi}(\boldsymbol{x}_j) = \boldsymbol{\gamma}(\boldsymbol{x}_i)^\top \boldsymbol{\gamma}(\boldsymbol{x}_j). \tag{3.8}$$

Then, the posterior distribution of the regression function becomes:

$$\boldsymbol{f}_* \mid \mathcal{D} \sim \mathsf{N}(\hat{\boldsymbol{f}}_*, \boldsymbol{S}_{\boldsymbol{f}_*}) \tag{3.9}$$

$$\hat{\boldsymbol{f}}_* = \boldsymbol{m}_* + \boldsymbol{K}_*^\top \left(\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right)^{-1} (\boldsymbol{y} - \boldsymbol{m}) \tag{3.10}$$

$$\boldsymbol{S}_{\boldsymbol{f}_*} = \boldsymbol{K}_{**} - \boldsymbol{K}_*^\top \left(\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{K}_*, \tag{3.11}$$

where $\boldsymbol{m}_* = \boldsymbol{\Phi}_* \boldsymbol{\mu}$ represents the prior mean of $\boldsymbol{f}_*$, the vector $\boldsymbol{m} = \boldsymbol{\Phi}\boldsymbol{\mu}$ is the prior mean of $\boldsymbol{f}$, $\boldsymbol{K}_{**} = [k(\boldsymbol{x}_i^*, \boldsymbol{x}_j^*)]_{i,j=1}^m$ is the $m \times m$ covariance matrix of f at the test points, $\boldsymbol{K}_* = [k(\boldsymbol{x}_i, \boldsymbol{x}_j^*)]_{i=1,j=1}^{n,m}$ is the $n \times m$ covariance between f at the training and test points and $\boldsymbol{K} = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1}^n$ is the $n \times n$ covariance matrix of $\boldsymbol{f}$.

This section showed that the Gaussian prior (3.4) on $\boldsymbol{w}$ induces a prior distribution on the regression function $f_{\boldsymbol{w}}(\cdot) = \boldsymbol{\phi}(\cdot)^\top \boldsymbol{w}$ which is still Gaussian, with mean and covariance:

$$\mathbb{E}[f_{\boldsymbol{w}}(\boldsymbol{x})] = \boldsymbol{\phi}(\boldsymbol{x})^\top \mathbb{E}[\boldsymbol{w}] = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\mu} \tag{3.12}$$

$$\mathrm{Cov}[f_{\boldsymbol{w}}(\boldsymbol{x}_i), f_{\boldsymbol{w}}(\boldsymbol{x}_j)] = \boldsymbol{\phi}(\boldsymbol{x}_i)^\top \mathrm{Cov}[\boldsymbol{w}, \boldsymbol{w}] \boldsymbol{\phi}(\boldsymbol{x}_j) = \boldsymbol{\phi}(\boldsymbol{x}_i)^\top \boldsymbol{\Sigma} \boldsymbol{\phi}(\boldsymbol{x}_j). \tag{3.13}$$

In vector notation, the prior distribution of $\boldsymbol{f} = (f_{\boldsymbol{w}}(\boldsymbol{x}_1), \ldots, f_{\boldsymbol{w}}(\boldsymbol{x}_n))$ is $\boldsymbol{f} \mid \boldsymbol{X} \sim \mathsf{N}(\boldsymbol{m} = \boldsymbol{\Phi}\boldsymbol{\mu}, \boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Phi}^\top)$. The variance-covariance matrix of $\boldsymbol{f}$ is the square and symmetric matrix $\boldsymbol{K} = [\boldsymbol{\gamma}(\boldsymbol{x}_i)^\top \boldsymbol{\gamma}(\boldsymbol{x}_j)]_{i,j=1}^n$, where $\boldsymbol{\gamma}(\boldsymbol{x}) = \boldsymbol{\Sigma}^{1/2} \boldsymbol{\phi}(\boldsymbol{x})$. The predictive distribution was derived in (3.9) and shows that the feature vectors enter in the formulas only through inner products.

3.3 The Kernel Trick

Instead of going through the process of defining a set of p features $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_p)$, computing the feature vectors $\boldsymbol{\phi}(\boldsymbol{x})$, and then calculating their inner products, the *kernel trick* directly defines the covariance between the random variables $f(\boldsymbol{x}_i)$ and $f(\boldsymbol{x}_j)$ in terms of a generic positive definite function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ called *kernel* or *covariance function*. The kernel trick states that we are not restricted to the functional form in (3.8), which is specific for the Bayesian linear regression model (3.3) with weight prior (3.4), but we can choose any covariance function $k(\cdot, \cdot)$ that satisfies the following conditions:

1. *Symmetry*: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \text{Cov}[f(\boldsymbol{x}_i), f(\boldsymbol{x}_j)] = \text{Cov}[f(\boldsymbol{x}_j), f(\boldsymbol{x}_i)] = k(\boldsymbol{x}_j, \boldsymbol{x}_i)$

2. *Positivity*: for all n, indices $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathcal{X}$ and $\boldsymbol{a} = (a_1, \ldots, a_n) \in \mathbb{R}^n$:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0. \tag{3.14}$$

Let $\boldsymbol{K} = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1}^n$, then (3.14) corresponds to requiring that $\boldsymbol{a}^\top \boldsymbol{K} \boldsymbol{a} \geq 0$, i.e. $\boldsymbol{K}$ is a positive semidefinite matrix. In particular, by setting $\boldsymbol{a} = (0, \ldots, 1, \ldots, 0)$ we obtain $k(\boldsymbol{x}_i, \boldsymbol{x}_i) = \text{Cov}[f(\boldsymbol{x}_i), f(\boldsymbol{x}_i)] = \mathbb{V}[f(\boldsymbol{x}_i)] \geq 0$.

Definition 3.1. (Kernel function). A kernel function is any real-valued function of two inputs that corresponds to an inner product in some feature space $\mathcal{F}$:

$$(\boldsymbol{x}, \boldsymbol{x}') \in \mathcal{X} \times \mathcal{X} \mapsto k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}') \in \mathcal{F},$$

where $\boldsymbol{\phi}$ maps an input $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d$ to an element $\boldsymbol{\phi}(\boldsymbol{x})$ of an inner product feature space $\mathcal{F} \subset \mathbb{R}^p$.

The Bayesian linear regression kernel, shown in (3.8), satisfies this definition. In Section 3.2 we followed the parametric approach to the estimation of the regression function f. After (1) specifying a structural form for f which is linear in the parameters $\boldsymbol{w}$, (2) assuming a Gaussian prior on the regression weights, (3) integrating out the weights; we obtained a prior distribution on the unknown predictor $f_{\boldsymbol{w}}(\cdot)$ which is still Gaussian. Instead of assuming a given parametric form for the regression function, the nonparametric approach lets the data only guide its shape. This means, however, that the function needs to be estimated pointwise, for all $\boldsymbol{x} \in \mathcal{X}$, hence the number of parameters grows to infinity. Section 3.5 discusses a widely used nonparametric model, called the Gaussian process, which directly specifies a distribution on the unknown regression function $f(\cdot)$ and uses a kernel function, as discussed in Section 3.3, to define the covariance between the random variables $f(\boldsymbol{x}_i)$ and $f(\boldsymbol{x}_j)$ for any $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{X}$.

3.4 Stochastic Processes

Probability deals with univariate random variables (RVs) and finite sets of RVs $\boldsymbol{y} = (y_1, \ldots, y_p)$ known as random vectors. In principle, there is no reason why this

should be limited to finite sequences. The theory of stochastic processes (SPs) deals with infinite sequences of random variables, which include as special cases the ones mentioned above.

Let $(E, \mathcal{E})$ be a measurable space and $\mathcal{T}$ an arbitrary set, countable or uncountable. For each t in $\mathcal{T}$, let y_t be a random variable defined on the measurable space $(\Omega, \mathcal{H})$ and taking values in $(E, \mathcal{E})$. The collection of random variables $y = \{y_t\}_{t \in \mathcal{T}}$ is a stochastic process with *state space* $(E, \mathcal{E})$ and *index* or *parameter set* $\mathcal{T}$. For a fixed outcome $\omega \in \Omega$, the function $t \mapsto y_t(\omega)$ is called *trajectory* or *sample path*, and represents a realization of the random process. For a given index $t \in \mathcal{T}$, the function $\omega \mapsto y_t(\omega)$ is just a *random variable*. For fixed index t and outcome ω, the quantity $y_t(\omega)$ is a *number*.

In summary, a stochastic process can be interpreted as:

1. a random function, when both t and ω are variables;
2. a single function (sample path), for fixed ω;
3. a random variable, for fixed index t;
4. a single number, when both t and ω are fixed.

When the parameter set $\mathcal{T}$ is discrete, e.g. the set of natural numbers $\mathbb{N}$, a common notation is $\{y_n\}_{n \in \mathbb{N}}$. When dealing with continuous parameter spaces, such as subsets of higher-dimensional Euclidean spaces, the stochastic process is usually denoted by $\{y(\boldsymbol{x})\}_{\boldsymbol{x} \in \mathcal{X}}$, where $\mathcal{X}$ is a subset of $\mathbb{R}^d$.

3.5 Gaussian Processes

Definition 3.2. (Gaussian Process). A stochastic process $f = \{f(\boldsymbol{x})\}_{\boldsymbol{x} \in \mathcal{X}}$ is said to be a Gaussian process (GP) if the RVs $\boldsymbol{f} = (f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n))$ are jointly normal for any n and inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$:

$$\boldsymbol{f} \sim \mathsf{N}(\boldsymbol{m}, \boldsymbol{K}),$$

where $\boldsymbol{m} = (m(\boldsymbol{x}_1), \ldots, m(\boldsymbol{x}_n))$ is the mean n-vector and $\boldsymbol{K} = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1}^n$ represents the $n \times n$ variance-covariance matrix of $\boldsymbol{f}$.

Similarly to a Gaussian distribution, which is parametrized by a mean vector and a covariance matrix, a GP is fully specified by a mean function $m : \mathcal{X} \to \mathbb{R}$ and a covariance function $k : \mathcal{X}^2 \to \mathbb{R}^+$ returning the mean $m(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})]$ and the covariance $k(\boldsymbol{x}, \boldsymbol{x}') = \mathrm{Cov}[f(\boldsymbol{x}), f(\boldsymbol{x}')]$ of the random variables $f(\boldsymbol{x})$ as function of the index only. If f is a Gaussian process, this is denoted $f \sim \mathsf{GP}(m, k)$. A Gaussian process is a nonparametric model in that the cardinality of the parameter set $\mathcal{X}$ is uncountable.

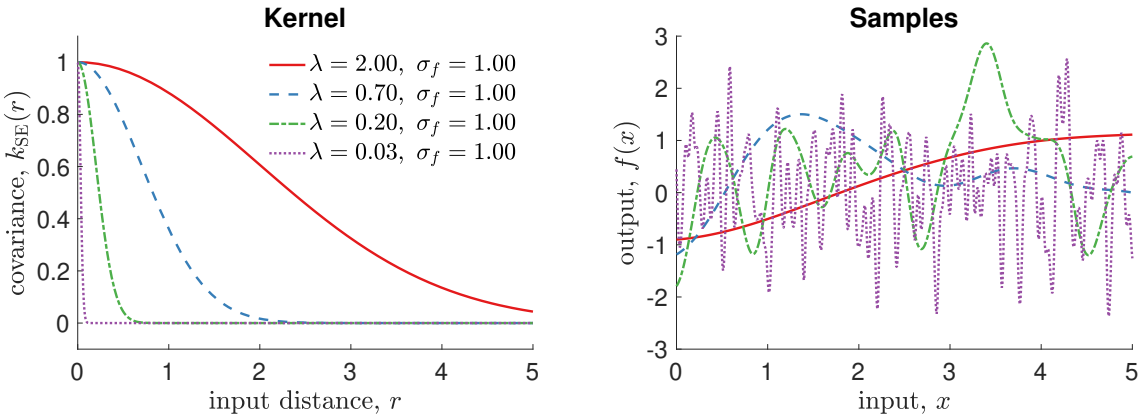
3.6 Covariance Functions

Many kernels are function of the difference between the inputs only, so that $k(\boldsymbol{x}, \boldsymbol{x}') = k_{\mathrm{S}}(\boldsymbol{x} - \boldsymbol{x}')$. These are known as stationary kernels because they are invariant to translations in the input space (Bishop, 2006). Another type of covariance functions is the *homogenous* or *radial basis function* (RBF) kernels, which depend on the magnitude of the distance (typically Euclidean) between the inputs, so that $k(\boldsymbol{x}, \boldsymbol{x}') = k_{\mathrm{RBF}}(r)$, where $r = \|\boldsymbol{x} - \boldsymbol{x}'\|$.

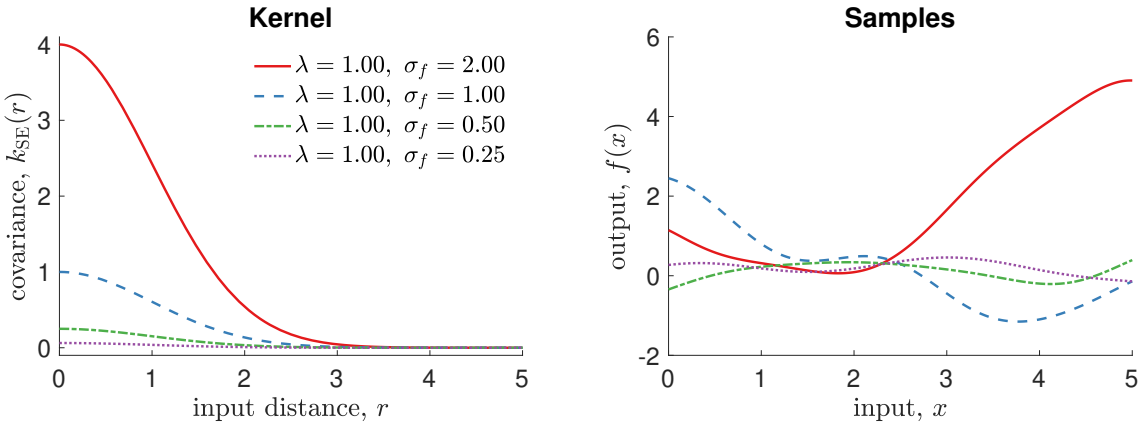
A commonly used RBF kernel is the isotropic (ISO) Squared Exponential (SE) covariance function, which depends on two parameters, the *lengthscale* λ and the *signal standard deviation* or *amplitude* σ_f:

$$k_{\mathrm{SE}}(r) = \sigma_f^2 \exp\left(-\frac{r^2}{2\lambda^2}\right), \qquad r = \|\boldsymbol{x} - \boldsymbol{x}'\|. \tag{3.15}$$

The mean and kernel functions are the parameters of the GP; so the parameters of the kernel are called *hyperparameters* and collectively denoted by $\boldsymbol{\theta}$. The left panel of Figure 3.1a shows a plot of the SE covariance function, $k_{\mathrm{SE}}(r)$, vs the distance between the inputs, $r = \|\boldsymbol{x} - \boldsymbol{x}'\|$, for different values of the lengthscale. On the right panel are shown samples from a GP prior using the corresponding kernel from the left panel. With a small lengthscale, the correlation between the random variables $f(\boldsymbol{x})$ and $f(\boldsymbol{x}')$ decreases quickly, hence allowing the GP to model more erratic functions. By increasing the lengthscale, the GP can model smoother processes. In Figure 3.1b a similar plot is shown, but varying the amplitude parameter. From the left panel it is possible to see that σ_f^2 represents the marginal variance of a RV



(a) The SE kernel (left) and GP samples (right) for varying lengthscales λ.



(b) The SE kernel (left) and GP samples (right) for varying amplitudes σ_f.

Figure 3.1: The Squared Exponential Kernel.

$f(\boldsymbol{x})$. If $r = 0$, then $k(r) = k(\boldsymbol{x}, \boldsymbol{x}) = \mathbb{V}[f(\boldsymbol{x})] = \sigma_f^2$. The right panel shows that by increasing the amplitude, the GP can model processes having a larger y-axis variation. A Gaussian process with the Squared Exponential covariance function gives rise to sample paths which are infinitely differentiable. The Fourier transform of the SE kernel is a Gaussian spectral density:

$$S(s) = (2\pi\lambda^2)^{d/2} \exp(-2\pi^2\lambda^2 s^2), \tag{3.16}$$

see Section 4.2.1 in Rasmussen and Williams (2006). This effectively means that the SE kernel does not model high-frequency functions because its spectral density places most of its support on low frequencies.

Another choice is the Matérn class of kernel functions with positive hyperparame-

ters $\boldsymbol{\theta} = (\lambda, \sigma_f)$. It also depends on an extra parameter, the degree ν:

$$k_\nu(r) = \sigma_f^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu} r}{\lambda} \right)^\nu K_\nu \left(\frac{\sqrt{2\nu} r}{\lambda} \right).$$

The degree gives rise to a wide class of functions ranging from quickly decaying correlations to almost linear relations. Here K_ν represents a modified Bessel function of the second order. The most interesting kernels of this class for nonlinear regression are represented by special cases for rational ν values (Rasmussen and Williams, 2006):

1. Exponential kernel or Ornstein-Uhlenbeck process (for $\nu = 1/2$):

$$k_{1/2}(r) = \sigma_f^2 \exp\left(-r/\lambda\right); \tag{3.17}$$

2. Matérn 3/2 (for $\nu = 3/2$):

$$k_{3/2}(r) = \sigma_f^2 \left(1 + \frac{\sqrt{3} r}{\lambda} \right) \exp\left(-\frac{\sqrt{3} r}{\lambda} \right); \tag{3.18}$$

3. Matérn 5/2 (for $\nu = 5/2$):

$$k_{5/2}(r) = \sigma_f^2 \left(1 + \frac{\sqrt{5} r}{\lambda} + \frac{5r^2}{3\lambda^2} \right) \exp\left(-\frac{\sqrt{5} r}{\lambda} \right). \tag{3.19}$$

This class of functions converges to the Squared Exponential kernel as $\nu \to \infty$, as shown in Figure 3.2. For the same hyperparameter values, by increasing the degree ν it is possible to model smoother processes. Let $\lceil \nu \rceil$ denote the ceiling of ν, i.e. the smallest integer that is greater than or equal ν. A Gaussian process having a Matérn covariance function with degree ν gives rise to sample paths which are almost surely $\lceil \nu \rceil - 1$ differentiable. Hence, the degree ν is also called the smoothness parameter of the Matérn class (Santner et al., 2003). In particular, the Matérn 5/2 kernel leads to twicely-differentiable paths, which is the standard assumption required, for example, by the Quasi-Newton methods. Recall that the SE kernel has a Gaussian spectral density (3.16). The Wiener-Khintchine theorem states that kernels and spectral densities are Fourier duals of each other, see Rasmussen and Williams (2006). Hence, we can also build a kernel starting from a spectral density.

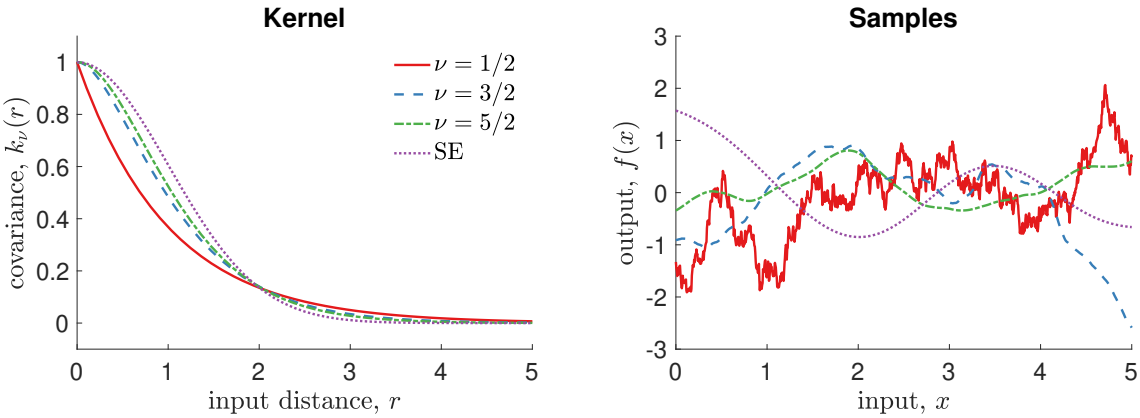


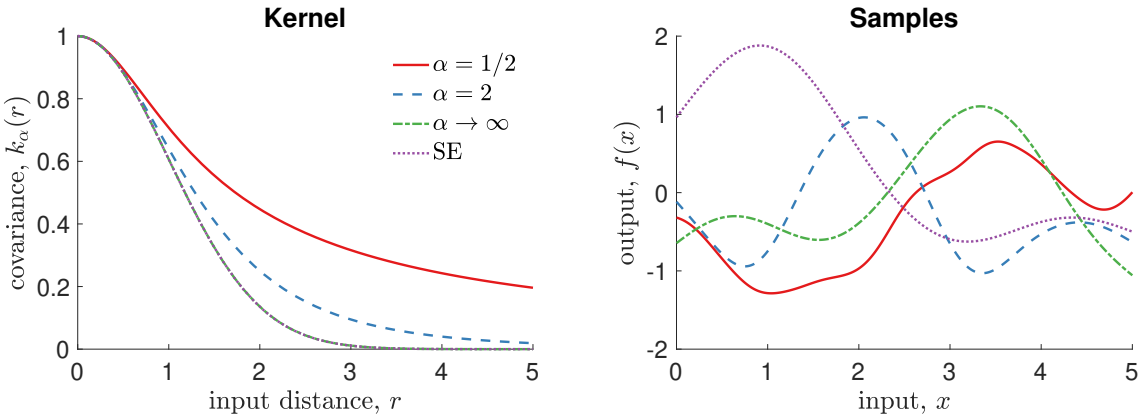
Figure 3.2: Convergence of the Matérn class to the SE kernel. Decay in correlation over the input distance (left) and samples from a GP prior (right) using the corresponding kernel from the left panel. Hyperparameters fixed to $\lambda = 1,\ \sigma_f = 1$.

Instead of a Gaussian, consider a heavier-tailed distribution for $S(s)$, such as a t-spectral density, in order to give more weight to higher frequencies. By taking the inverse Fourier transform we recover the Matérn class. It is also worth remarking the link between a Gaussian distribution and the t-distribution. Consider a random variable $x \sim \mathsf{N}(\mu, \sigma^2)$ with unknown variance, and place an Inverse Gamma prior on the variance: $\sigma^2 \sim \mathsf{InverseGamma}(a = \nu/2, b = \nu\sigma^2/2)$. By integrating out σ^2, the marginal distribution of x is a t-distribution with parameters (μ, σ^2, ν). In a similar way, the SE and the Matérn kernels are related as the spectral density of the Matérn kernel can be obtained by assuming a Gamma prior on the lengthscale of the Gaussian spectral density of the SE kernel.

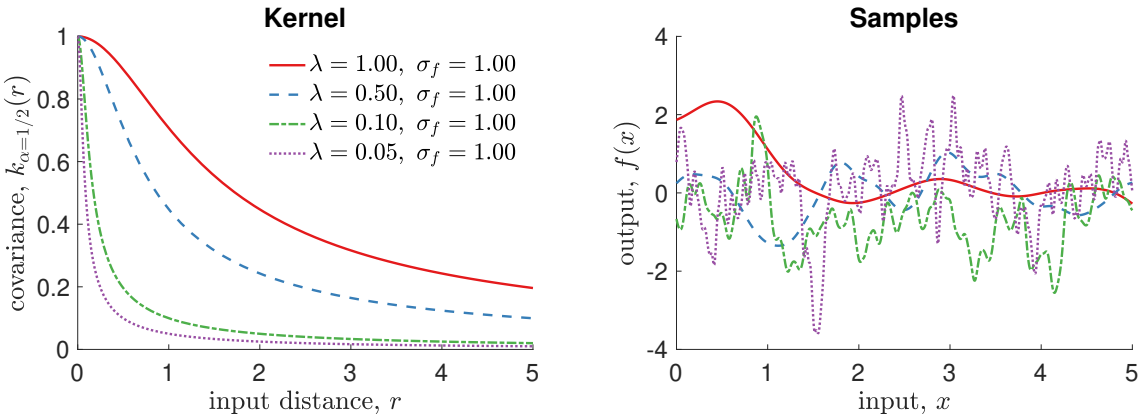
The Rational Quadratic class with amplitude σ_f, lengthscale λ and exponent α is defined as (see (3.14) in Roberts et al. (2012)):

$$k_{\mathrm{RQ}}(r) = \sigma_f^2 \left(1 + \frac{r^2}{2\alpha\lambda^2}\right)^{-\alpha}. \tag{3.20}$$

This class converges to the SE kernel as $\alpha \to \infty$, as shown in Figure 3.3a. Low values of α lead to higher tails and hence a correlation which decays more slowly. Figure 3.3b shows different kernels (left) and GP samples (right) from a RQ covariance function with $\alpha = 1/2,\ \sigma_f = 1$ and varying lengthscales. From the left panel we can see that given any two inputs at distance $r = \|\boldsymbol{x} - \boldsymbol{x}'\|$, their correlation decreases by decreasing the lengthscale. The right panel shows that the samples from the GP



(a) Convergence of the RQ kernel with $\lambda = 1,\ \sigma_f = 1$ to the SE kernel for different values of α.



(b) RQ kernel (left) and GP samples (right) for $\alpha = 1/2,\ \sigma_f = 1$ and different values of λ.

Figure 3.3: The Rational Quadratic kernel.

prior get smoother by increasing the correlation, i.e. by increasing the lengthscale.

The RQ kernel can be obtained as a mixture of SE kernels with varying lengthscales distributed according to a Gamma density. Define $\tau = \lambda^{-2}$, with prior $p(\tau \mid \alpha, \beta) \propto \tau^{\alpha-1}\exp(-\alpha\tau/\beta)$. It can be shown that (Rasmussen and Williams, 2006):

$$\begin{aligned} k_{\mathrm{RQ}}(r) &= \int_0^\infty k_{\mathrm{SE}}(r \mid \tau) p(\tau \mid \alpha, \beta) \mathrm{d}\tau \\ &\propto \sigma_f^2 \left(1 + \frac{r^2}{2\alpha\lambda^2}\right)^{-\alpha}, \end{aligned}$$

where we defined $\beta^{-1} = \lambda^2$.

A generalization of the isotropic SE kernel is the ARD Squared Exponential covariance function:

$$k_{\mathrm{SE}}(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp\left\{-\frac{1}{2}\sum_{i=1}^{d}\frac{(x_i - x_i')^2}{\lambda_i^2}\right\}. \tag{3.21}$$

The acronym ARD stands for automatic relevance determination, and it refers to kernels that allow for a different lengthscale parameter λ_i $(i = 1, \ldots, d)$ in each dimension. When an input dimensionality has a large lengthscale, the function is effectively flat along that direction, meaning that the input is not relevant. The isotropic SE kernel (3.15) can be obtained by setting $\lambda_i = \lambda$ for $i = 1, \ldots, d$.

Also the Matérn class can be extended to have a different lengthscale in each dimension:

1. ARD Exponential kernel:

$$k_{1/2}(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp(-r), \qquad r = \sqrt{\sum_{i=1}^{d} \frac{(x_i - x_i')^2}{\lambda_i^2}};$$

2. ARD Matérn 3/2 kernel:

$$k_{3/2}(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \left(1 + \sqrt{3}r\right) \exp\left(-\sqrt{3}r\right), \qquad r = \sqrt{\sum_{i=1}^{d} \frac{(x_i - x_i')^2}{\lambda_i^2}};$$

3. ARD Matérn 5/2 kernel:

$$k_{5/2}(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \left(1 + \sqrt{5}r + \frac{5}{3}r^2\right) \exp\left(-\sqrt{5}r\right), \qquad r = \sqrt{\sum_{i=1}^{d} \frac{(x_i - x_i')^2}{\lambda_i^2}}. \tag{3.22}$$

Similarly, the Rational Quadratic class can be extended to the ARD case as follows:

$$k_{\mathrm{RQ}}(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \left(1 + \frac{1}{2\alpha}r^2\right)^{-\alpha}, \qquad r = \sqrt{\sum_{i=1}^{d} \frac{(x_i - x_i')^2}{\lambda_i^2}}.$$

Among those, the ARD Squared Exponential and ARD Matérn 5/2 kernels are the most commonly used in the emulation literature.

When modelling nonstationary functions, common choices for the kernel are the Neural Network covariance function or, if there is evidence of periodicity, the Periodic kernel; see (4.29) and (4.31) in Rasmussen and Williams (2006) respectively.

As discussed in Jones et al. (1998), deterministic simulators (representing the focus of this thesis) typically return similar values for inputs which are close together in the domain. Given the usually small number of training samples due to the fact

that each simulation is expensive, for example in Bayesian optimization we start with $n = 10 \times d$ training samples, it is hard to understand if the underlying process is periodic or highly erratic. In order to assess periodicity, we would need lots of training data. In the emulation of expensive computer codes literature, a common prior assumption is to use the SE kernel. This takes into account the correlation between the outputs of computer simulators for inputs which are close together. Furthermore, a GP with a SE kernel is known as a *universal approximator*: for enough training data it can approximate any function arbitrarily well (van der Vaart and van Zanten, 2009).

Regarding the mean function, in the GP literature it is common to assume a zero-mean process after standardizing the training data. We prefer to use a constant mean function, whose constant value is inferred jointly with the remaining GP hyperparameters. In practice, it is hard to specify a prior mean function for the same reason described above: limited training data. Considering that we work in compact spaces, i.e. closed and bounded, the choice of the mean function is also not very substantial. On the other hand, for prediction beyond the training data, it would play a rather substantial role as, with lack of data, the GP tends to go back to the prior.

3.7 Sampling From a Gaussian Process Prior

Let $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be a set of n points from $\mathcal{X}$. In order to generate a sample from a GP prior $f \sim \mathsf{GP}(m, k)$:

Algorithm 3.1. *Sampling from a Gaussian process prior.*

1. Calculate the mean vector $\boldsymbol{m} = (m(\boldsymbol{x}_1), \ldots, m(\boldsymbol{x}_n))$ and the covariance matrix $\boldsymbol{K} = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1}^n$ by pointwise evaluation of the mean function $m(\cdot)$ and kernel function $k(\cdot, \cdot)$ respectively.

2. Obtain the Cholesky decomposition of $\boldsymbol{K} = \boldsymbol{U}^\top \boldsymbol{U}$.

3. Generate a sample $z_1, \ldots, z_n$ independently from $z \sim \mathsf{N}(0, 1)$ and let $\boldsymbol{z} = (z_1, \ldots, z_n)$.

4. Return $\boldsymbol{f} = \boldsymbol{m} + \boldsymbol{U}^\top \boldsymbol{z}$.

Figure 3.4 uses $n = 1000$ linearly spaced points between 0 and 10. Each panel uses the prior mean function $m(\boldsymbol{x}) = 0$ for all $\boldsymbol{x} \in \mathcal{X}$ and shows, for a given kernel, three samples from a GP prior (blue lines), the prior mean $\boldsymbol{m}$ (red line), and in gray the approximate 95% confidence interval $m(\boldsymbol{x}) \pm 2\sqrt{k(\boldsymbol{x}, \boldsymbol{x})}$ for all $\boldsymbol{x} \in \mathcal{X}$.

3.8 Posterior Gaussian Process

Until now we have shown how to specify a Gaussian process prior over functions. This section shows how to update the prior distribution in light of the data, in order to make predictions at unseen cases.

Given observations $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, assumed to come from the generative model

$$y = f(\boldsymbol{x}) + \varepsilon, \qquad \varepsilon \sim \mathsf{N}(0, \sigma^2) \text{ independently}, \tag{3.23}$$

the goal is to predict y at an unseen point $\boldsymbol{x}$.

In matrix form, the model is $\boldsymbol{y} = \boldsymbol{f} + \boldsymbol{\varepsilon}$, with $\boldsymbol{\varepsilon} \sim \mathsf{N}(0, \sigma^2 \boldsymbol{I})$. Similarly to (3.6), the marginal distribution of $\boldsymbol{y}$ is:

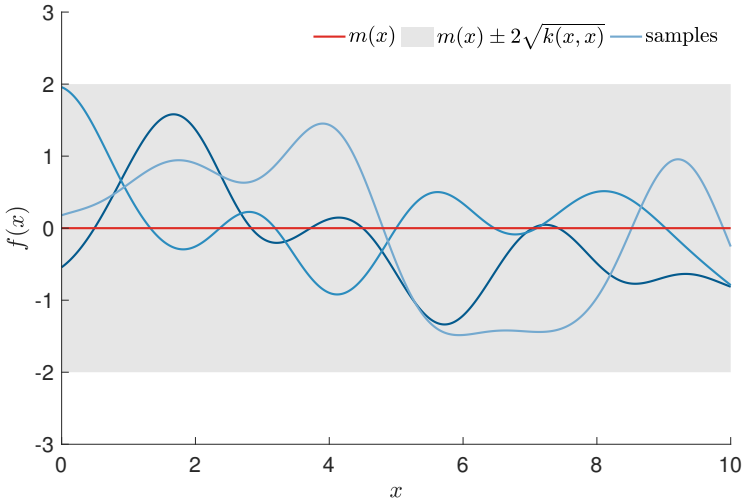
$$\boldsymbol{y} \mid \boldsymbol{X} \sim \mathsf{N}(\boldsymbol{m}, \boldsymbol{K} + \sigma^2 \boldsymbol{I}). \tag{3.24}$$

Hence, the joint distribution of $\boldsymbol{y}$ and $f(\boldsymbol{x})$ is:

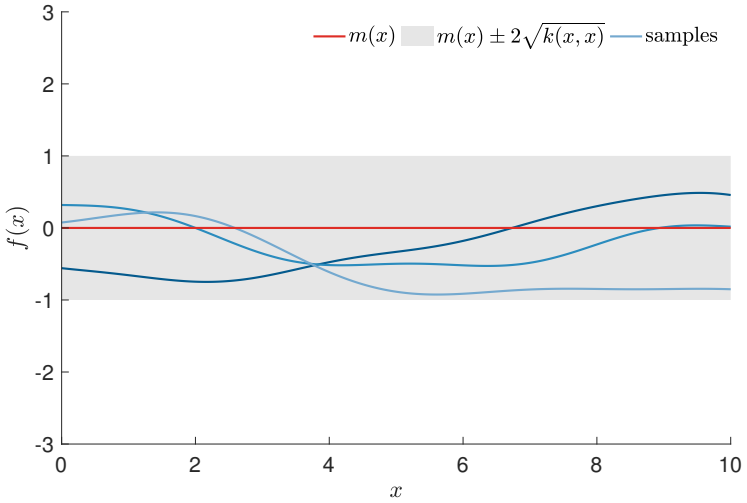
$$\begin{bmatrix} \boldsymbol{y} \\ f(\boldsymbol{x}) \end{bmatrix} \sim \mathsf{N}\left(\begin{bmatrix} \boldsymbol{m} \\ m(\boldsymbol{x}) \end{bmatrix}, \begin{bmatrix} \boldsymbol{K} + \sigma^2 \boldsymbol{I} & \boldsymbol{k}(\boldsymbol{x}) \\ \boldsymbol{k}(\boldsymbol{x})^\top & k(\boldsymbol{x}, \boldsymbol{x}) \end{bmatrix} \right).$$

Here, $\boldsymbol{K} = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1}^n$ is the $n \times n$ covariance matrix of f evaluated at the training inputs, $\boldsymbol{k}(\boldsymbol{x}) = (k(\boldsymbol{x}_1, \boldsymbol{x}), \ldots, k(\boldsymbol{x}_n, \boldsymbol{x}))$ is the column n-vector containing the covariances between f at each training input and the test point $\boldsymbol{x}$, while $\boldsymbol{m} = (m(\boldsymbol{x}_1), \ldots, m(\boldsymbol{x}_n))$ represents the prior mean at the training inputs.

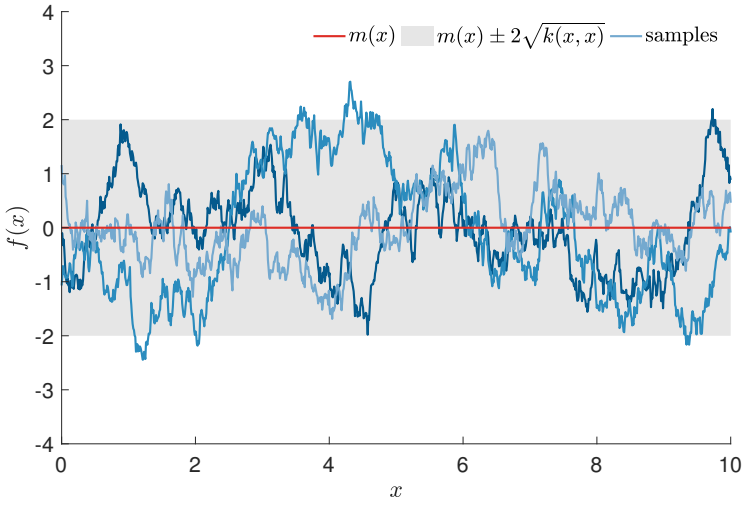
Using the formulas for the conditional distribution of a Gaussian random vector, see Appendix C.2 in Davidson (2000), we obtain the predictive distribution of $f(\boldsymbol{x})$



(a) SE kernel with $\lambda = 1,\ \sigma_f = 1$.



(b) SE kernel with $\lambda = 2,\ \sigma_f = 1/2$.



(c) Matérn 1/2 kernel with $\lambda = 1,\ \sigma_f = 1$.

Figure 3.4: Samples from a GP prior.

given data $\mathcal{D}$:

$$
\begin{aligned}
f(\boldsymbol{x}) \mid \mathcal{D} &\sim \mathsf{N}(\hat{f}(\boldsymbol{x}), s^2(\boldsymbol{x})) \\
\hat{f}(\boldsymbol{x}) &= m(\boldsymbol{x}) + \boldsymbol{k}(\boldsymbol{x})^\top \left[\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right]^{-1} (\boldsymbol{y} - \boldsymbol{m}) \\
s^2(\boldsymbol{x}) &= k(\boldsymbol{x}, \boldsymbol{x}) - \boldsymbol{k}(\boldsymbol{x})^\top \left[\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right]^{-1} \boldsymbol{k}(\boldsymbol{x}).
\end{aligned} \tag{3.25}
$$

As mentioned before, conditioning on $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$ is equivalent to conditioning on $\boldsymbol{y}$ only, since $\boldsymbol{X}$ is assumed to be given and fixed. This can be easily generalized to obtain the posterior Gaussian process:

$$
\begin{aligned}
f(\boldsymbol{x}) \mid \mathcal{D} &\sim \mathsf{GP}(\hat{f}(\boldsymbol{x}), s(\boldsymbol{x}, \boldsymbol{x}')) \\
\hat{f}(\boldsymbol{x}) &= m(\boldsymbol{x}) + \boldsymbol{k}(\boldsymbol{x})^\top \left[\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right]^{-1} (\boldsymbol{y} - \boldsymbol{m}) \\
s(\boldsymbol{x}, \boldsymbol{x}') &= k(\boldsymbol{x}, \boldsymbol{x}') - \boldsymbol{k}(\boldsymbol{x})^\top \left[\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right]^{-1} \boldsymbol{k}(\boldsymbol{x}').
\end{aligned} \tag{3.26}
$$

For simplicity, assume that the prior mean is zero: $m(\boldsymbol{x}) = 0$ for all $\boldsymbol{x} \in \mathcal{X}$. We collectively denote by $\boldsymbol{\theta}$ all model hyperparameters: mean, kernel and noise standard deviation σ. Figure 3.5 shows the process of updating the GP prior by conditioning on the observed data.

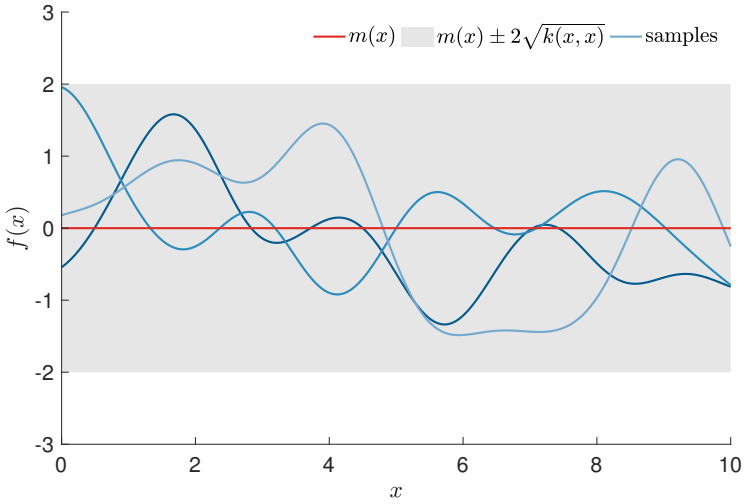
It is worth noting that the predictive mean can be written in two different but equivalent ways:

$$
\begin{aligned}
\hat{f}(\boldsymbol{x}) &= \boldsymbol{k}(\boldsymbol{x})^\top \left[\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right]^{-1} \boldsymbol{y} \\
&= \boldsymbol{k}(\boldsymbol{x})^\top \boldsymbol{a} && (3.27) \\
&= \boldsymbol{b}^\top \boldsymbol{y}, && (3.28)
\end{aligned}
$$

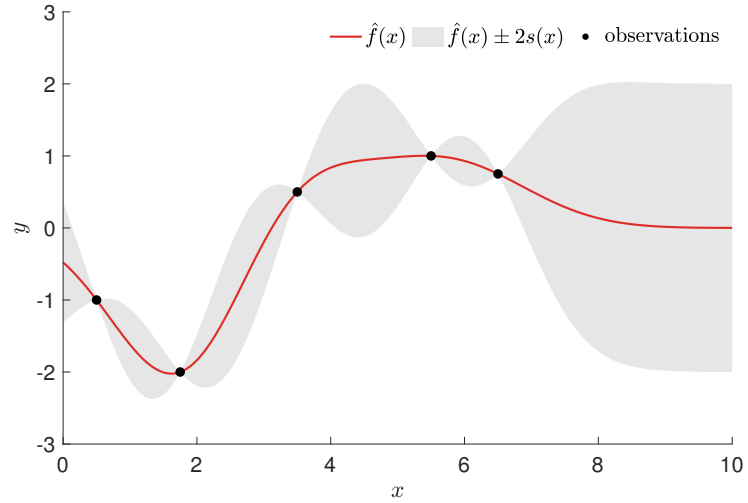
where $\boldsymbol{a} = \left[\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right]^{-1} \boldsymbol{y}$ and $\boldsymbol{b} = \left[\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right]^{-1} \boldsymbol{k}(\boldsymbol{x})$. The predictive mean is a linear combination of n kernel functions $\boldsymbol{k}(\boldsymbol{x}) = (k(\boldsymbol{x}_1, \boldsymbol{x}), \ldots, k(\boldsymbol{x}_n, \boldsymbol{x}))$, see (3.27). The other identity, (3.28), is often referred to by saying that the mean is a linear predictor, in the sense of being a linear combination of the observations in $\boldsymbol{y}$.

3.9 Sampling From the Posterior

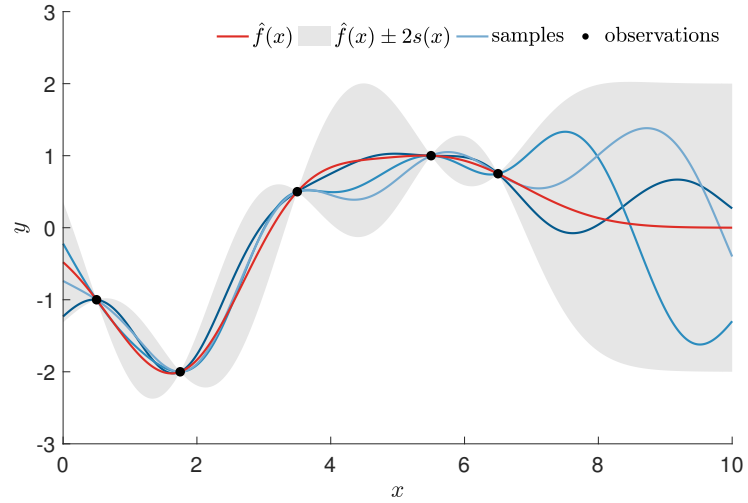
The process of sampling from the posterior essentially follows the same steps described in Algorithm 3.1, but using the mean and covariance function of the posterior process. Let $\{\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_m^*\}$ be a set of unseen cases to predict.



(a) GP prior.



(b) GP posterior.



(c) Samples from the GP posterior.

Figure 3.5: Conditioning the GP on data.

Algorithm 3.2. *Sampling from a Gaussian process posterior.*

1. Compute the m-vector $\hat{\boldsymbol{f}}_* = (\hat{f}(\boldsymbol{x}_1^*), \ldots, \hat{f}(\boldsymbol{x}_m^*))$ and the $m \times m$ covariance matrix $\boldsymbol{S}_* = [s(\boldsymbol{x}_i^*, \boldsymbol{x}_j^*)]_{i,j=1}^m$ by pointwise evaluation of the predictive mean $\hat{f}(\cdot)$ and covariance function $s(\cdot, \cdot)$ respectively.

2. Obtain the Cholesky decomposition of $\boldsymbol{S}_* = \boldsymbol{U}^\top \boldsymbol{U}$.

3. Generate a sample $z_1, \ldots, z_m$ independently from $z \sim \mathsf{N}(0, 1)$ and let $\boldsymbol{z} = (z_1, \ldots, z_m)$.

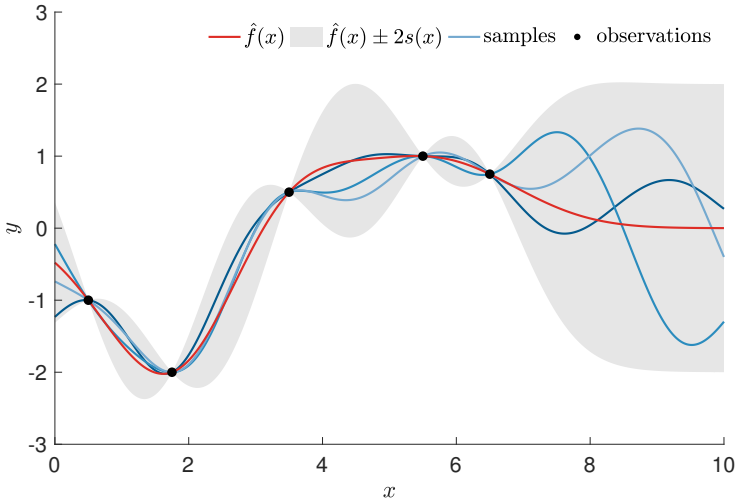
4. Return $\boldsymbol{f}_* = \hat{\boldsymbol{f}}_* + \boldsymbol{U}^\top \boldsymbol{z}$.

Figure 3.6 shows samples from the posterior process (3.26) in blue, the predictive mean in red, and in gray the approximate 95% confidence interval $\hat{f}(\boldsymbol{x}) \pm 2s(\boldsymbol{x})$ for all $\boldsymbol{x}$. The chosen test points $\{x_1^*, \ldots, x_m^*\}$ were $m = 1000$ linearly spaced points between 0 and 10. Each panel shows samples from the posterior for a different level of the noise standard deviation $\sigma \in \{0, 0.1, 0.5\}$. In Figure 3.6a it is interesting that the GP model also works in the noise-free scenario, unlike most parametric methods that can not interpolate the training data exactly.

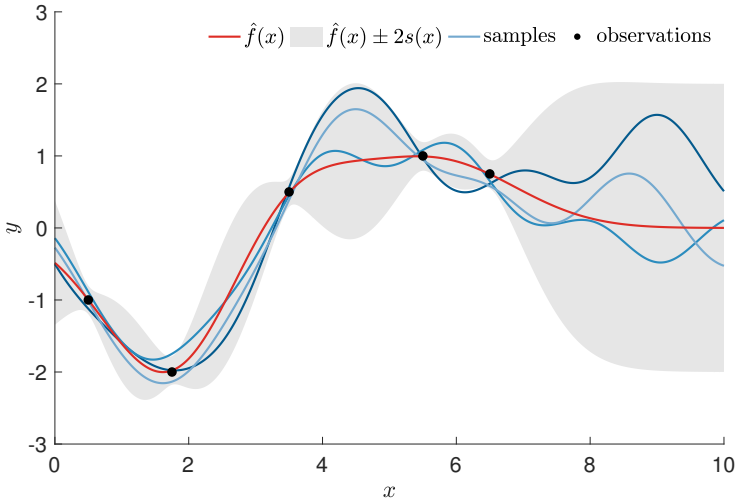
3.10 Training a Gaussian Process

Section 3.8 showed how to update the Gaussian process prior in the light of the data $\mathcal{D}$ to obtain a posterior distribution. Then, Section 3.9 discussed how to sample from the posterior GP. This approach is feasible only if you have a strong prior knowledge about the data generating process (DGP), as it requires fully specifying a mean and covariance function (including the hyperparameters).

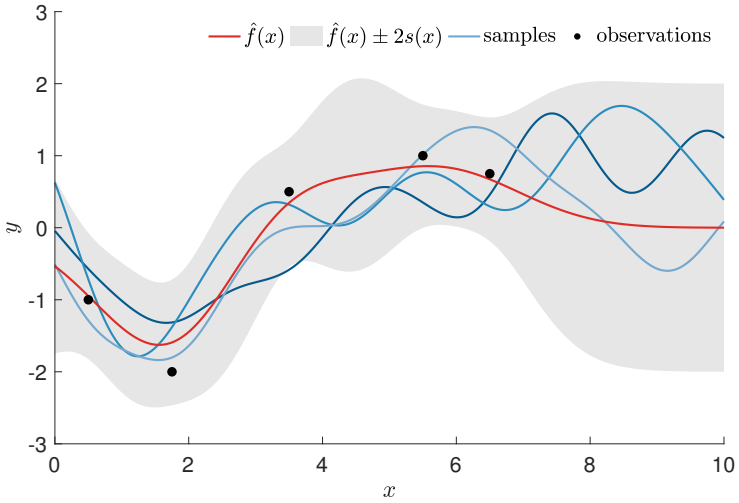
Typically we only have vague prior information and, in order for the method to be of practical use, we need to be able to choose between different mean and covariance functions in light of the data. We refer to this process as *training or fitting the Gaussian process*. In practice, by looking at the data it is possible to select a parametric family of mean and kernel functions. This corresponds to specifying a hierarchical prior where the mean and covariance functions are parametrized



(a) $\sigma = 0$



(b) $\sigma = 0.1$



(c) $\sigma = 0.5$

Figure 3.6: Samples from the posterior GP with the SE kernel.

by some hyperparameters and the ultimate goal of the training is to infer these hyperparameters from the data. The hyperparameters correspond, as shown in Section 3.6, to how fast the correlation decays, the amplitude, and the noise.

As an example, if the data show that the DGP is very smooth, good choices for the covariance function are the SE or the Matérn 5/2 kernels. If, instead, the pattern is very erratic and rough, a better choice would be the Matérn 1/2 (Ornstein-Uhlenbeck) kernel. As remarked above, choosing the kernel allows us to specify our vague prior information easily. However, we still allow for the data to guide the selection of the kernel hyperparameters.

Let $\boldsymbol{C} = \boldsymbol{K} + \sigma^2 \boldsymbol{I}$ denote the covariance of the noisy outputs $\boldsymbol{y}$. From (3.24) we can obtain the *log marginal likelihood*:

$$L(\boldsymbol{\theta}) = \log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) = -\frac{1}{2}\log|\boldsymbol{C}| - \frac{1}{2}(\boldsymbol{y}-\boldsymbol{m})^\top \boldsymbol{C}^{-1}(\boldsymbol{y}-\boldsymbol{m}) - \frac{n}{2}\log(2\pi), \quad (3.29)$$

where the name "marginal" is due to the marginalization over the unknown function values:

$$\begin{aligned} p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) &= \int p(\boldsymbol{y} \mid \boldsymbol{f}) p(\boldsymbol{f} \mid \boldsymbol{X}) \mathrm{d}\boldsymbol{f} \\ &= \int \underbrace{\mathsf{N}(\boldsymbol{y} \mid \boldsymbol{f}, \sigma^2 \boldsymbol{I})}_{\text{i.i.d. noise}} \underbrace{\mathsf{N}(\boldsymbol{f} \mid \boldsymbol{m}, \boldsymbol{K})}_{\text{GP prior}} \mathrm{d}\boldsymbol{f} \\ &= \mathsf{N}(\boldsymbol{y} \mid \boldsymbol{m}, \boldsymbol{K} + \sigma^2 \boldsymbol{I}). \end{aligned}$$

The partial derivatives of the log marginal likelihood with respect to the model hyperparameters are (Rasmussen, 2004):

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\theta}_m} L(\boldsymbol{\theta}) &= -(\boldsymbol{y}-\boldsymbol{m})^\top \boldsymbol{C}^{-1} \frac{\partial}{\partial \boldsymbol{\theta}_m} m \\ \frac{\partial}{\partial \boldsymbol{\theta}_k} L(\boldsymbol{\theta}) &= \frac{1}{2}\mathrm{tr}\left(\boldsymbol{C}^{-1}\frac{\partial \boldsymbol{C}}{\partial \boldsymbol{\theta}_k}\right) + \frac{1}{2}(\boldsymbol{y}-\boldsymbol{m})^\top \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{\theta}_k} \boldsymbol{C}^{-1} \frac{\partial \boldsymbol{C}}{\partial \boldsymbol{\theta}_k}(\boldsymbol{y}-\boldsymbol{m}), \end{aligned} \quad (3.30)$$

where $\boldsymbol{\theta}_m$ and $\boldsymbol{\theta}_k$ represent the hyperparameters of the mean and kernel function respectively.

Estimation of the kernel hyperparameters can be done by selecting the $\boldsymbol{\theta}$ that maximizes the log marginal likelihood given in (3.29), with partial derivatives (3.30), using for example the Quasi-Newton method, see Givens and Hoeting (2012). This is the approach used throughout the thesis, where the Quasi-Newton method stops

iterating when either the tolerance on the log marginal likelihood value is less than 10^{-6}, the tolerance on the $\boldsymbol{\theta}$ value is less than 10^{-12} or the maximum number of iterations exceeds 10000, see Appendix A for more details.

Define the following summaries of the observed data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$:

$$\begin{aligned} \bar{y} &= \frac{1}{n}\sum_{i=1}^{n} y_i \\ s_y &= \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2} \\ \bar{x}_j &= \frac{1}{n}\sum_{i=1}^{n} x_{ij} \\ s_{x_j} &= \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2}. \end{aligned}$$

As intelligent starting guesses for the optimization of the model hyperparameters I recommend the following settings which, from experience, work well in practice:

- Constant mean function $m(\boldsymbol{x}) = c$ for all $\boldsymbol{x} \in \mathcal{X}$: $c^0 = \bar{y}$;
- ARD lengthscales: $\lambda_j^0 = s_{x_j}$ for $j = 1, \ldots, d$;
- ISO lengthscale: $\lambda^0 = \frac{1}{d}\sum_{j=1}^{d} s_{x_j}$;
- Signal standard deviation: $\sigma_f^0 = s_y$;
- Noise standard deviation: $\sigma^0 = \sigma_f^0/\sqrt{2}$ or lower for noise-free data, e.g. 10^{-3}.

The initial value for the noise standard deviation is equal to the signal standard deviation divided by $\sqrt{2}$, i.e. the noise variance is initialized to half of the signal variance. This is because, intuitively, the noise should be smaller in magnitude than the variation in function range.

The log marginal likelihood (3.29) would make it easy for the predictive mean to fit the data exactly: this happens by setting $\sigma = 0$. However, the first term, $-\frac{1}{2}\log|\boldsymbol{C}|$, is a penalty on the complexity of the model and balances the second term representing a measure of data fit. Indeed, the second term is the only one depending on the training outputs. The last term is a log normalization constant.

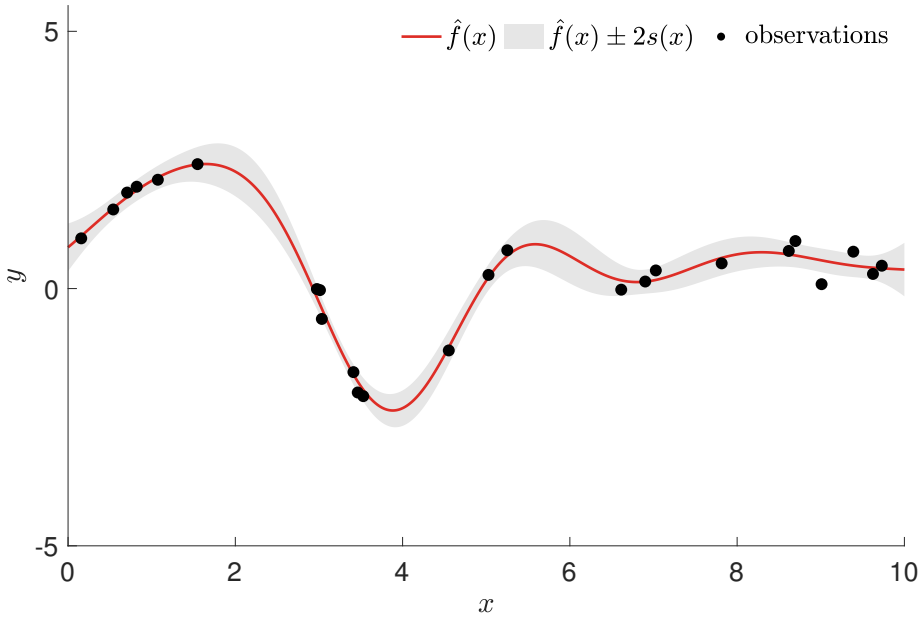
From (3.29) it is clear that the tradeoff between data-fit and model complexity is automatic and there is no need to estimate an additional regularization parameter, for example using generalized cross validation, as in penalized regression splines. The hierarchical specification of the GP prior also allows the user to set hyperpriors on the GP hyperparameters if they wish. The estimation of $\boldsymbol{\theta}$ could then be performed for example by maximum a posteriori (MAP).

Figure 3.7 shows, for the same set of training data, the posterior GP for different hyperparameter settings, along with their respective log marginal likelihood score. Figure 3.7a appears to be the best fit, while the GP in Figure 3.7b is just tracking the noise. The log marginal likelihood criterion chooses as the best vector of hyperparameters the first one, which has the highest score.

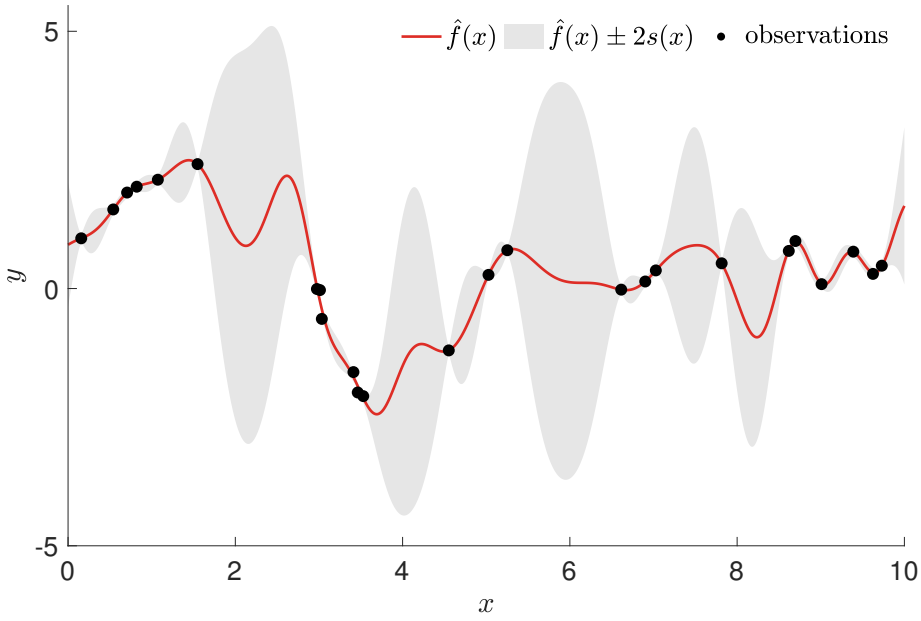
The problem of model selection, i.e. selecting the type of covariance function, can be done by sampling hyperparameters from the posterior and then using advanced information criteria like WAIC or WBIC (Watanabe, 2010, 2013).

3.11 Summary

This chapter discussed nonparametric regression using Gaussian processes: a powerful regression method that, unlike parametric models, is also able to interpolate the training data exactly. It then established the link between Bayesian linear regression models and GPs using the kernel trick. A GP is completely specified by a mean and a covariance function; so Section 3.6 discussed several families of covariance functions and the effect they have on the sample paths. Next, it showed how the GP prior on functions can be updated in light of data, to obtain the posterior Gaussian process. Finally, Section 3.10 discussed how to train a GP model, i.e. how to estimate the model hyperparameters from the observed data. The Gaussian process represents the statistical model that will be used in the next chapters to build emulators of expensive simulators or of their functions.



(a) $\boldsymbol{\theta} = (\lambda, \sigma_f, \sigma) = (1, 1, 1/5) \qquad L(\boldsymbol{\theta}) = -19.5$



(b) $\boldsymbol{\theta} = (\lambda, \sigma_f, \sigma) = (1/3, 2, 1/20) \qquad L(\boldsymbol{\theta}) = -47.8$

Figure 3.7: Two posterior GPs and the corresponding log marginal likelihood scores.

Chapter 4

Parameter Estimation in Nonlinear ODEs

Inference in nonlinear ordinary differential equations (ODEs) is challenging due to the many numerical integrations at different parameter settings required by global optimization algorithms or MCMC schemes. In this chapter I explore an emulation-based approach for approximating the likelihood, based on Gaussian processes, with the objective to reduce the number of numerical integration steps. This is a different approach from the standard emulation literature, which entails direct emulation of the output (Kennedy and O'Hagan, 2001; O'Hagan, 2006). The viability of the scheme is assessed on a nonstandard variant of the Lotka-Volterra model of predator-prey interactions.

Notes This chapter is adapted from: Noè, U., Filippone, M., and Husmeier, D. (2015). Emulation of ODEs with Gaussian processes. In *Proceedings of the 30th International Workshop on Statistical Modelling*, pages 191–194.

4.1 Motivation

Ordinary Differential Equations (ODEs) arise in the study of several areas of science and engineering. They describe the evolution over time of a set of state variables of a system of interest in a concise and elegant manner while providing means for

interpreting the underlying dynamics of the system. However, carrying out parameter inference is challenging for a number of reasons. First, the likelihood can be highly multimodal. Second, direct numerical integration of ODEs for several settings of the parameters can be prohibitively expensive to be feasible. Motivated by encouraging results reported in Wilkinson (2014), I explore the feasibility of emulation based on Gaussian processes (GPs) for accelerated inference in ODEs such that an explicit numerical solution is only required for a comparatively small set of parameters. I report an experimental evaluation of the proposed emulation approach on a two-parameter Lotka-Volterra (LV) model where it is possible to gain insights into the potential and the limitations of the considered approach.

4.2 Numerical Solution of Differential Equations

A general continuous-time dynamical system described by the interaction of S state variables can be modelled by a functional equation of the form:

$$\frac{\mathrm{d}\boldsymbol{u}(t)}{\mathrm{d}t} = \boldsymbol{g}(\boldsymbol{u}(t), t; \boldsymbol{q}), \qquad \boldsymbol{u}(t_0) = \text{given}, \qquad \boldsymbol{q} \in \mathcal{Q} \subseteq \mathbb{R}^D,$$

where the S states at time t are $\boldsymbol{u}(t) = (u_1(t), \ldots, u_S(t))$, $\boldsymbol{g} = (g_1, \ldots, g_S)$ is the vector-valued function describing their evolution over time, and $\boldsymbol{q}$ is a vector of parameters.

If the state-space form of the differential equation can be expressed as $\mathrm{d}\boldsymbol{u}(t)/\mathrm{d}t = \boldsymbol{G}(t)\boldsymbol{u}(t) + \boldsymbol{L}(t)\boldsymbol{w}(t)$, where $\boldsymbol{w}(t)$ represents a forcing function, then it is called *linear differential equation* and it belongs to a class of differential equations that can be solved analytically unlike general nonlinear ones. If the forcing term is not present, the differential equation is said to be *homogeneous*. In order to solve linear time-invariant homogeneous differential equations we recall the separation of variables method for the scalar case and the series based approach (matrix exponential) in the multivariate case. Inhomogeneous linear time-invariant ODEs can be solved using the integrating factor method and the Fourier transform. The integrating factor method can also be used in the case of linear time-varying homogenous differential equations.

For nonlinear ODEs having the form $\mathrm{d}\boldsymbol{u}(t)/\mathrm{d}t = \boldsymbol{g}(\boldsymbol{u}(t), t; \boldsymbol{q}),\ \boldsymbol{u}(t_0) = \text{given}$, there is no general rule to find an analytic solution, but we can approximate it numerically using e.g. Euler's method, Heun's method or Runge-Kutta methods, see Särkkä and Solin (2018). Getting an approximate solution can be computationally expensive for ODEs describing complex systems which comprise high dimensional parameter vectors.

4.3 GP Emulation for ODEs

Let $\boldsymbol{U}(\boldsymbol{q}) \in \mathbb{R}^{T \times S}$ denote the numerical solution of the ODE at times $\boldsymbol{t} = (t_1, \ldots, t_T)$ for a given parameter vector $\boldsymbol{q}$. Assume that the data matrix $\boldsymbol{Y}$ comes from the data generating process $\boldsymbol{U}(\cdot)$, corrupted by additive i.i.d. Gaussian noise:

$$\boldsymbol{Y} = \boldsymbol{U}(\boldsymbol{q}^*) + \boldsymbol{E}, \qquad [\boldsymbol{E}]_{ij} \sim \mathsf{N}(0, \sigma^2) \text{ independently.}$$

Any optimization or inference scheme for the ODE parameters would entail repeatedly solving the ODE for different parameter configurations. Consider, as discussed in Chapter 2, the squared Euclidean loss function measuring the distance between a numerical solution $\boldsymbol{U}(\boldsymbol{q})$ and the data $\boldsymbol{Y}$:

$$\ell(\boldsymbol{q}) = \textsc{rss}(\boldsymbol{q}) = \|\mathrm{vec}(\boldsymbol{Y}) - \mathrm{vec}(\boldsymbol{U}(\boldsymbol{q}))\|^2. \tag{4.1}$$

The residual sum of squares (RSS) function is the one which is most often used empirically, see for example the DREAM challenges[1]. The RSS function in output-space also corresponds to the negative log likelihood under the i.i.d. noise assumption. Each evaluation of ℓ involves an explicit numerical solution of the ODE, hence direct minimization of $\ell(\boldsymbol{q})$ might not be feasible. The goal is to estimate the parameters by numerically solving the ODE only at a limited number of parameter configurations. Unlike the standard emulation literature, which focuses on emulating the outputs (Kennedy and O'Hagan, 2001; O'Hagan, 2006), let us explore an emulation of the loss approach based on GPs as follows. Consider N parameter configurations $\boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_N]^\top$. For each of the N parameter configurations, compute the

[1] `http://dreamchallenges.org/`

corresponding numerical solution $\boldsymbol{U}(\boldsymbol{q}_n)$ of the ODE. Then compare these numerical solutions with the data $\boldsymbol{Y}$ using the squared Euclidean loss (4.1), obtaining the vector $\boldsymbol{\ell} = (\ell(\boldsymbol{q}_1), \ldots, \ell(\boldsymbol{q}_N))$. The next step involves fitting a GP to the training data $\mathcal{D} = \{\boldsymbol{Q}, \boldsymbol{l}\} = \{(\boldsymbol{q}_n, l_n)\}_{n=1}^N$, where $\boldsymbol{l}$ is the normalized RSS vector $\boldsymbol{\ell}$. The n^{th} component of $\boldsymbol{l}$ is:

$$l_n = \frac{\ell_n - \bar{\ell}}{s_\ell}, \qquad \bar{\ell} = \frac{1}{N}\sum_{n=1}^{N} \ell_n, \qquad s_\ell = \sqrt{\frac{1}{N-1}\sum_{n=1}^{N}(\ell_n - \bar{\ell})^2}.$$

This way we can infer the optimal ODE parameters relying on the GP emulator rather than numerically integrating the ODE every time we are interested in the plausibility of a different combination of parameters $\boldsymbol{q} \notin \boldsymbol{Q}$.

The regression model $l_n = l(\boldsymbol{q}_n) + \epsilon_n$ assumes the training outputs as observations from a latent function $l(\boldsymbol{q})$, which is given a Gaussian process prior, corrupted by additive i.i.d. $\mathsf{N}(0, \sigma_l^2)$ noise. The loss function is deterministic, but allowing for a small noise value can improve the conditioning number of the GP kernel matrix. We collectively denote all model hyperparameters (mean, kernel and noise standard deviation) by $\boldsymbol{\theta}$. The proposed hierarchical nonparametric Bayesian model makes use of the ARD Squared Exponential kernel[2] (3.21) and is given by:

$$\begin{aligned} \boldsymbol{l} \mid l(\boldsymbol{Q}), \sigma_l^2 &\sim \mathsf{N}(l(\boldsymbol{Q}), \sigma_l^2 \boldsymbol{I}) \\ l(\boldsymbol{q}) &\sim \mathsf{GP}(m(\boldsymbol{q}), k(\boldsymbol{q}, \boldsymbol{q}')) \\ k(\boldsymbol{q}, \boldsymbol{q}') &= \sigma_f^2 \exp\left\{-\frac{1}{2}\sum_{d=1}^{D}\frac{(q_d - q_d')^2}{\lambda_d^2}\right\}, \end{aligned} \tag{4.2}$$

where $\boldsymbol{q}, \boldsymbol{q}' \in \mathcal{Q}$, the latent values are $l(\boldsymbol{Q}) = (l(\boldsymbol{q}_1), \ldots, l(\boldsymbol{q}_N))$ and, as a consequence of normalization, we assume that $m(\boldsymbol{q}) = 0$ for all $\boldsymbol{q} \in \mathcal{Q}$. The hierarchical model in (4.2) consists of 2 levels of randomness:

1. A Gaussian likelihood: $p(\boldsymbol{l} \mid l(\boldsymbol{Q}), \sigma_l^2) = \mathsf{N}(\boldsymbol{l} \mid l(\boldsymbol{Q}), \sigma_l^2 \boldsymbol{I})$;

[2] A Gaussian process with the SE kernel gives rise to sample functions which are infinitely-differentiable. However, it is important to notice that finite-order ODEs can have infinitely-differentiable solutions. For example, the solution of $\mathrm{d}u/\mathrm{d}t = -u$ is $u(t) = c \times \exp(-t)$, which is clearly infinitely-differentiable. Then, taking the L2 loss, we still have a function which is infinitely-differentiable.

2. A GP prior on the regression function: $p(l(\boldsymbol{Q}) \mid \boldsymbol{Q}) = \mathsf{N}(l(\boldsymbol{Q}) \mid \boldsymbol{0}, \boldsymbol{K})$.

The GP formulation yields predictions for the normalized RSS score $l(\boldsymbol{q})$ corresponding to any ODE parameters $\boldsymbol{q} \in \mathcal{Q}$ using standard properties of GPs, see (3.26):

$$\begin{aligned} l(\boldsymbol{q}) \mid \boldsymbol{Q}, \boldsymbol{l} &\sim \mathsf{GP}(\hat{l}(\boldsymbol{q}), s(\boldsymbol{q}, \boldsymbol{q}')) \\ \hat{l}(\boldsymbol{q}) &= \boldsymbol{k}(\boldsymbol{q})^\top [\boldsymbol{K} + \sigma_l^2 \boldsymbol{I}]^{-1} \boldsymbol{l} \\ s(\boldsymbol{q}, \boldsymbol{q}') &= k(\boldsymbol{q}, \boldsymbol{q}') - \boldsymbol{k}(\boldsymbol{q})^\top [\boldsymbol{K} + \sigma_l^2 \boldsymbol{I}]^{-1} \boldsymbol{k}(\boldsymbol{q}'), \end{aligned} \tag{4.3}$$

where $\boldsymbol{k}(\boldsymbol{q}) = (k(\boldsymbol{q}_1, \boldsymbol{q}), \ldots, k(\boldsymbol{q}_N, \boldsymbol{q}))$ is the N-vector of kernels between each training input and the test input, and $\boldsymbol{K} = [k(\boldsymbol{q}_i, \boldsymbol{q}_j)]_{i,j=1}^N$ is the $N \times N$ training covariance matrix.

In the case of observations $\mathrm{vec}(\boldsymbol{Y})$ assumed to be distributed as a Gaussian centred at the solution of the ODE with variance $\sigma^2 \boldsymbol{I}$, we can interpret this approach as emulating a negative tempered log likelihood. The log likelihood of the model is:

$$L(\boldsymbol{q}) = \log p(\mathcal{D} \mid \boldsymbol{q}) = \text{const} - \frac{1}{2\sigma^2} \ell(\boldsymbol{q}), \tag{4.4}$$

and from $l(\boldsymbol{q}) = \{\ell(\boldsymbol{q}) - \bar{\ell}\}/s_\ell$ follows that $\ell(\boldsymbol{q}) = \bar{\ell} + s_\ell \times l(\boldsymbol{q})$. Substituting in the log likelihood equation we get:

$$\begin{aligned} L(\boldsymbol{q}) &= \text{const} - \frac{1}{2\sigma^2} \ell(\boldsymbol{q}) \\ &= \left(\text{const} - \frac{\bar{\ell}}{2\sigma^2} \right) - \frac{s_\ell}{2\sigma^2} l(\boldsymbol{q}) \\ &= \text{const} - \frac{s_\ell}{2\sigma^2} l(\boldsymbol{q}). \end{aligned} \tag{4.5}$$

This means that modelling the normalized RSS score $l(\boldsymbol{q})$ corresponds to modelling the negative logarithm of a power of the likelihood of the model. The approach discussed above is in spirit different from the Latent Force Models by Álvarez et al. (2009) and Särkkä et al. (2017). It is important to stress that instead of modelling the output $\boldsymbol{Y}$, or latent variables $\boldsymbol{F}$ which are linearly related to the output ($\boldsymbol{Y} = \boldsymbol{F}\boldsymbol{W} + \boldsymbol{E}$) as in Álvarez et al. (2009), we instead focus on emulating the residual sum of squares function.

4.4 Experimental Evaluation

In this section I explore the performance of the approach discussed above on a non-standard variant of the Lotka-Volterra (LV) system, introduced by Mingari Scarpello and Ritelli (2003). The ODE describing the evolution over time of the $S = 2$ states is:

$$\begin{cases} x_1'(t) = x_1(t)\,[a - bx_2(t)] \\ x_2'(t) = x_2(t)\,[-c + dx_1(t)]\,. \end{cases} \tag{4.6}$$

Let us define

$$\begin{cases} u_1(t) := \log\left[\frac{d}{c}x_1(t)\right] \\ u_2(t) := \log\left[\frac{b}{a}x_2(t)\right], \end{cases}$$

obtaining the following reparametrization of (4.6):

$$\begin{cases} u_1'(t) = a\,[1 - \exp\{u_2(t)\}] \\ u_2'(t) = -c\,[1 - \exp\{u_1(t)\}]\,. \end{cases} \tag{4.7}$$

Here the components of $\boldsymbol{u} = (u_1, u_2)$ represent the populations of "log preys" and "log predators" respectively, but for simplicity the word "log" is considered as implicit. In all the experiments that follow, the initial conditions are fixed to $\boldsymbol{u}(0) = (0, 1)$. Assume that the true population parameters are $\boldsymbol{q}^* = (a^*, c^*) = (2, 1)$, and let $\boldsymbol{U}(\boldsymbol{q}^*)$ denote the numerical solution at $T = 50$ linearly spaced times $\boldsymbol{t} = (t_1, \ldots, t_{50})$ with $t_1 = 0$ and $t_{50} = 20$. The simulated data $\boldsymbol{Y}$ are obtained by adding i.i.d. Gaussian noise to $\boldsymbol{U}(\boldsymbol{q}^*)$ with signal-to-noise ratio SNR $= 10$. Figure 4.1 shows the two columns of the data matrix $\boldsymbol{Y}$, which represent the observed preys and the observed predators, as well as the underlying true signal for preys and predators.

As discussed in Section 2.4, in order to fit a GP emulator to the normalized RSS scores, we need a set of training runs (2.6). Given the low-dimensionality of the parameter space ($D = 2$), consider a grid of $G = 20$ values for each parameter in $[0.75, 5]$. The $N \times D$ matrix $\boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_N]^\top$ contains all the possible $N = G^2 = 400$ configurations of the parameters (a, c), and is given by:

$$\boldsymbol{Q} = \begin{bmatrix} a_1 & \ldots & a_1 & a_2 & \ldots & a_2 & \ldots & a_G & \ldots & a_G \\ c_1 & \ldots & c_G & c_1 & \ldots & c_G & \ldots & c_1 & \ldots & c_G \end{bmatrix}^\top .$$

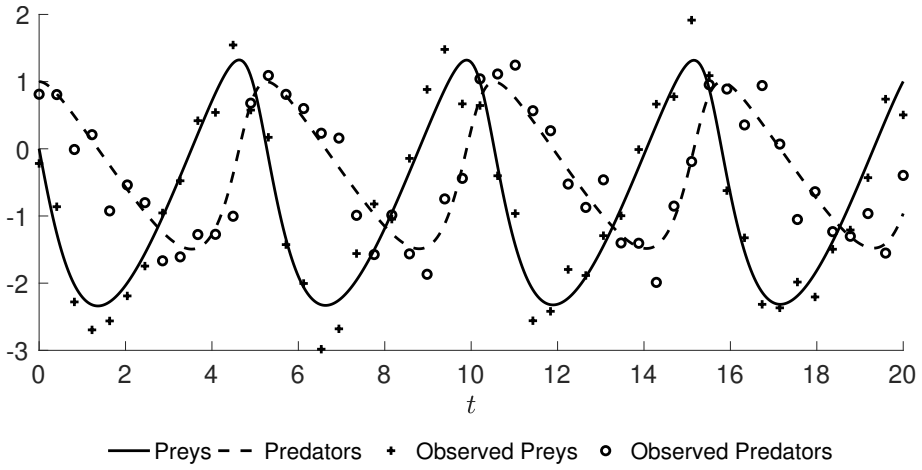


Figure 4.1: The true numerical solution of the two-parameters LV model for $t \in [0, 20]$, and $T = 50$ linearly spaced observations.

The GP hyperparameters $\boldsymbol{\theta}$ are set by maximizing the log marginal likelihood starting from the initial values discussed in Section 3.10. In particular, the initial value for the noise standard deviation is set to 10^{-3} as the function to be emulated is deterministic.

Given the GP posterior mean $\hat{l}(\boldsymbol{q})$ from (4.3), the *emulated log likelihood* and the *emulated likelihood* are transformations of $\hat{l}(\boldsymbol{q})$, see (4.5). Furthermore, the emulated log likelihood and the emulated likelihood are scaled, for plotting purposes, to have a maximum equal to zero and one respectively:

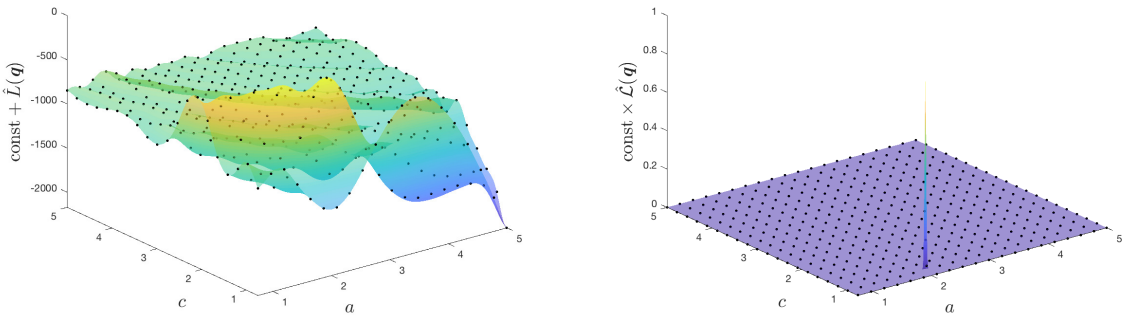
$$\begin{aligned} \text{const} + \hat{L}(\boldsymbol{q}) &= -\frac{s_\ell}{2\sigma^2}\hat{l}(\boldsymbol{q}) - \max[-\frac{s_\ell}{2\sigma^2}\hat{l}(\boldsymbol{q})] \\ \text{const} \times \hat{\mathcal{L}}(\boldsymbol{q}) &= \exp\{\text{const} + \hat{L}(\boldsymbol{q})\}. \end{aligned}$$

Figure 4.2 (a) shows the emulated log likelihood and (b) the emulated likelihood for the data in Figure 4.1. Using multiple cycles leads to very spiky likelihood landscapes, as also found by other authors (see e.g. Lazarus et al. (2018)).

We can estimate the ODE parameters by maximizing the emulated log likelihood or, equivalently, by minimizing the Gaussian process predictive mean $\hat{l}(\boldsymbol{q})$. In formulas:

$$\begin{aligned} \hat{\boldsymbol{q}} \quad &= \quad \arg\min_{\boldsymbol{q} \in \mathcal{Q}} \hat{l}(\boldsymbol{q}) \\ &\text{s.t.} \quad \boldsymbol{q} \in [m, M]^2 \end{aligned} \tag{4.8}$$

where we assume $m = \min(\{a_g\}_{g=1}^G) = \min(\{c_g\}_{g=1}^G) = 0.75$ and $M = \max(\{a_g\}_{g=1}^G) =$



(a) **(b)**

Figure 4.2: The emulated log likelihood (a) and the emulated likelihood (b) for the $T = 50$ linearly spaced observations shown in Figure 4.1.

$\max(\{c_g\}_{g=1}^{G}) = 5$. From the properties of GPs, we have an analytical form for the predictive mean $\hat{l}(\boldsymbol{q})$, its gradient $\nabla\hat{l}(\boldsymbol{q})$, and the Hessian $\boldsymbol{H}(\boldsymbol{q})$, see Appendix B. Hence, the minimization problem in (4.8) can be solved using the trust-region-reflective algorithm (Byrd et al., 1988; Branch et al., 1999) implemented in MATLAB's `fmincon` function, which requires up to the second order derivatives.

The training inputs $\boldsymbol{Q}$ represent a fixed and pre-specified design. Consider generating $K = 1000$ different datasets $\{\boldsymbol{Y}_k\}_{k=1}^{K}$, using different random number generator seeds, from the true signal $\boldsymbol{U}(\boldsymbol{q}^*)$. To each dataset $\boldsymbol{Y}_k$ corresponds a vector of training normalized RSS scores $\boldsymbol{l}_k$ which, together with the pre-specified design $\boldsymbol{Q}$, form the GP training data $\mathcal{D}_k = \{\boldsymbol{Q}, \boldsymbol{l}_k\}$. The posterior Gaussian process, given data $\mathcal{D}_k$, has predictive mean $\hat{l}_k(\boldsymbol{q})$. Minimize each function $\hat{l}_k(\boldsymbol{q})$ using a set of 50 starting points designed as follows. The starting points should include the training inputs $\boldsymbol{q}^{(1)}, \boldsymbol{q}^{(2)}, \boldsymbol{q}^{(3)} \in \boldsymbol{Q}$ giving rise to the three lowest training RSS scores in $\boldsymbol{l}_k$. Next, include 12 randomly sampled points from a $\mathsf{N}(\boldsymbol{q}^{(i)}, 0.1^2\boldsymbol{I})$, for $i = 1, 2, 3$. Finally, add 11 points sampled from a uniform distribution on $[0.75, 5]^2$ in order to explore the domain.

A local solver is run from each starting point, and the best minimum is kept, discarding the remaining ones. The optimal value is denoted $\hat{\boldsymbol{q}}_k$ and represents the optimal parameter vector for the k^{th} dataset $\boldsymbol{Y}_k$. The sample of optimal parameters

for the $K = 1000$ datasets is denoted $\{\hat{\boldsymbol{q}}_k\}_{k=1}^K$.

In order to understand more clearly the distribution of $\{\hat{\boldsymbol{q}}_k\}_{k=1}^K$, consider a multivariate kernel density estimator (KDE):

$$\hat{p}(\boldsymbol{q}) = \frac{1}{K|\boldsymbol{H}|} \sum_{k=1}^{K} \kappa\left(\boldsymbol{H}^{-1}(\boldsymbol{q}_k - \boldsymbol{q})\right),$$

where $\kappa(\cdot)$ is the standard Gaussian kernel function $\kappa(\boldsymbol{q}) = (2\pi)^{-D/2} \exp\{-\frac{1}{2}\boldsymbol{q}^\top \boldsymbol{q}\}$, $\boldsymbol{H} = \text{diag}(h_1, \ldots, h_D)$ is the diagonal matrix having the bandwidth parameters as diagonal elements and in our case $D = 2$. The bandwidth parameters are chosen according to the Normal reference rule (Bowman and Azzalini, 1997):

$$h_d = \sigma_d \left\{ \frac{4}{(D+2)K} \right\}^{\frac{1}{D+4}},$$

where σ_d represents the standard deviation of dimension d, which is replaced by a sample estimate.

The optimization results are shown in Figure 4.3. Panel (a) is a scatterplot of the optimal parameters for the 1000 different datasets (red dots) and the truth $\boldsymbol{q}^*$ (black diamond). The distribution of the optimal parameters is not scattered around the true value, suggesting some sort of bias. However, the true value is not distant from the cluster of optimal parameters, suggesting a good reconstruction in function-space. We notice that for one dataset (corresponding to the random seed 301) the estimated parameter vector $\hat{\boldsymbol{q}}_{301}$ lies outside of the bulk of the distribution. Panel (b) shows the kernel density estimate along with the truth (black diamond). In order to assess the performance in function-space, Panel (c) displays in black the true underlying signal and in red the solution $\boldsymbol{U}(\boldsymbol{q}^{\text{KDE}})$ using the parameter vector which maximizes the KDE. We can see that the reconstruction of the true signal in function-space is very accurate even though the distribution in parameter-space is not centred around $\boldsymbol{q}^*$.

Figure 4.4 analyzes in more detail the dataset $\boldsymbol{Y}_k$ giving rise to the outlier $\hat{\boldsymbol{q}}_k$, which happens for $k = 301$. Panel (a) shows the underlying true signal $\boldsymbol{U}(\boldsymbol{q}^*)$ and the two columns of $\boldsymbol{Y}_{301}$, representing the observed preys and predators respectively. Panel (b) plots the emulated log likelihood and (c) the emulated likelihood. We can see how the GP, trying to predict between the training data, generated ripples which have a higher value than the training outputs. Hence, when exponentiating the log

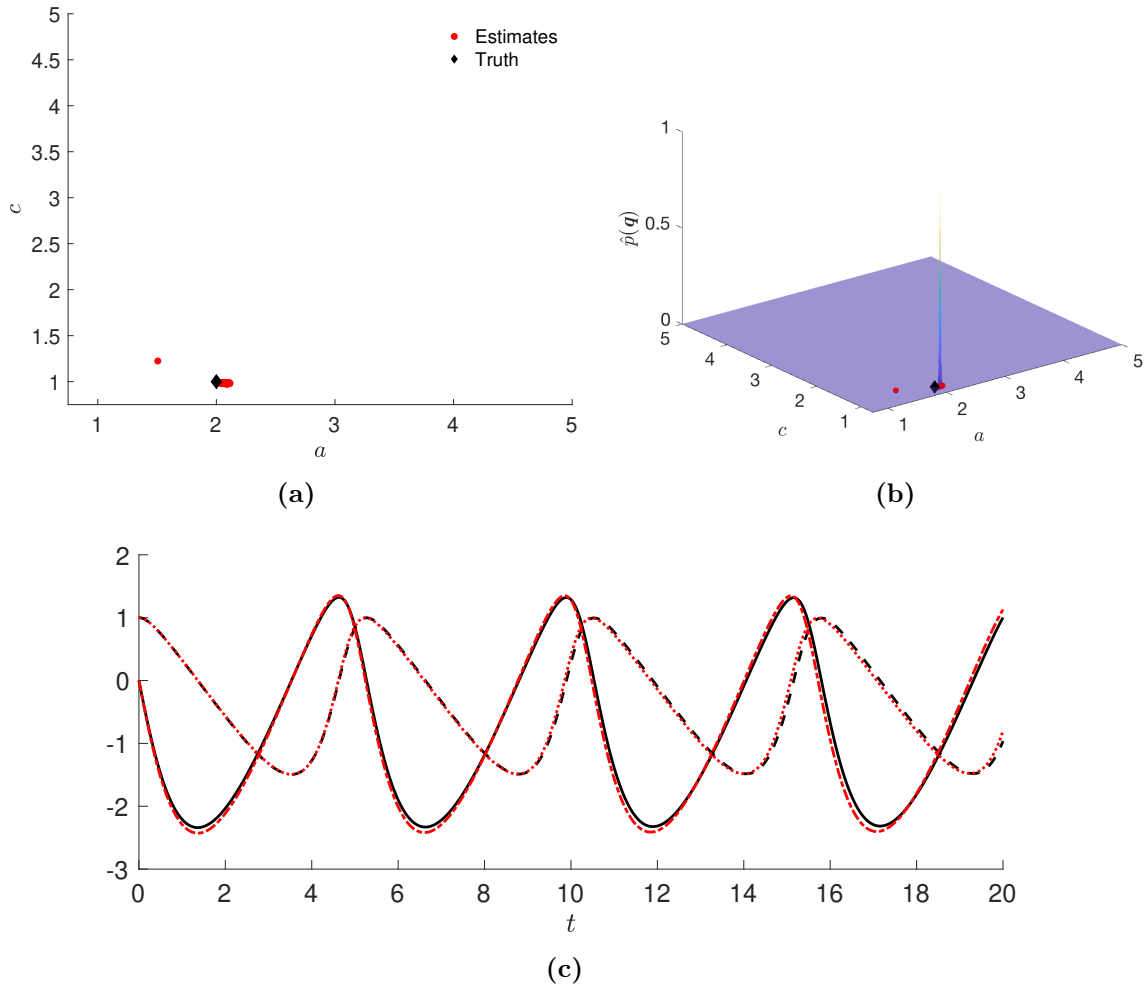


Figure 4.3: (a) The optimal parameters from 1000 different datasets (red circles) and the true parameter vector (black diamond); (b) the kernel density estimate of the sample of optimal parameters; (c) the true signal (in black) and the ODE solution for $\boldsymbol{q}^{\text{KDE}} = \arg\max_{\boldsymbol{q}} \hat{p}(\boldsymbol{q})$ (in red).

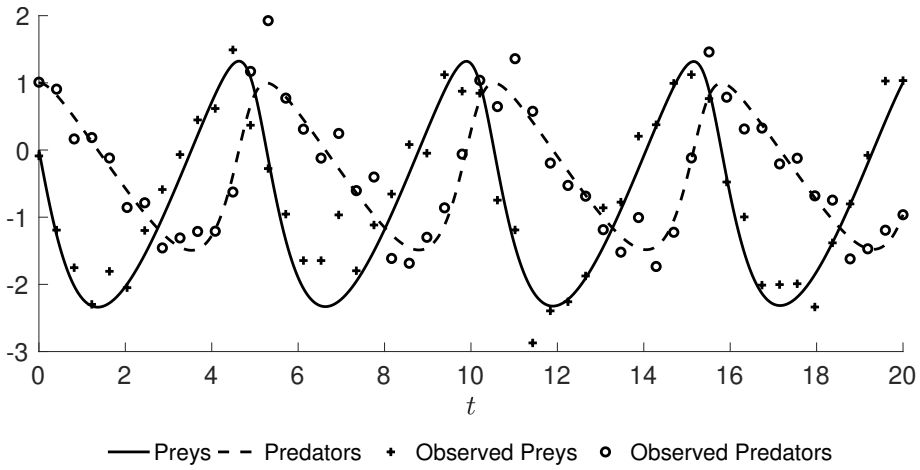
likelihood, we obtain a higher spurious peak. Panel (d) shows the solution $\boldsymbol{U}(\hat{\boldsymbol{q}}_{301})$ using the parameter vector corresponding to the spurious likelihood peak in red, and the true signal in black. We see that the estimate of the predators signal is accurate, while the preys signal is less satisfactory.

As a benchmark, we also consider direct minimization of the true loss function $\ell(\boldsymbol{q})$, defined in (4.1), which does not involve emulation, but rather a numerical solution of the ODE at every evaluation. Minimization is performed using the same initial design strategy on the same 1000 datasets $\{\boldsymbol{Y}_k\}$. The distribution of the sample of optimal parameters is displayed in Figure 4.5. Panel (a) shows a scatterplot of $\{\hat{\boldsymbol{q}}_k\}$ as red circles, along with the truth $\boldsymbol{q}^*$ as a black diamond. The distribution is scattered around the true value, proving that this approach is unbiased. Panel (b) plots the kernel density estimate of the sample of optimal parameters. In Panel (c) we notice that the solution $\boldsymbol{U}(\boldsymbol{q}^{\mathrm{KDE}})$ for the parameter vector which maximizes the KDE (shown in red) is a very accurate estimate of the true signal $\boldsymbol{U}(\boldsymbol{q}^*)$ (shown in black). The computational costs required to obtain the 1000 optima by direct minimization of the true loss function are in the order of 1 hour CPU time[3], while minimization of the GP predictive mean required 20 minutes only. Because minimization of the residual sum of squares (RSS) between the numerical solution of the ODE and the data (4.1) is an unbiased estimator of the true parameter vector, we can argue that the RSS function (L2 loss) is *not only* the most commonly used loss measure, but also an appropriate measure for this task due to the unbiasedness of the procedure.

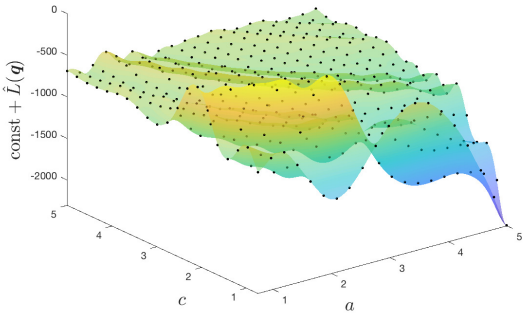
The sample of optimal parameters in Figure 4.3 (a) is not centred on the true parameter vector $\boldsymbol{q}^*$. This might be due to the fact that the Lotka-Volterra system gives rise to very rugged likelihood landscapes when considering multiple cycles, see Lazarus et al. (2018) for more examples. Therefore, the GP might be slightly smoother than the underlying likelihood landscape and, while interpolating the training points, the GP can lead to ripples having a higher predicted value than the maximum of the training data.

Let us decrease the number of cycles explored by the numerical solution of the ODE by considering the time interval $t \in [0, 5]$. This leads to an exploration of the

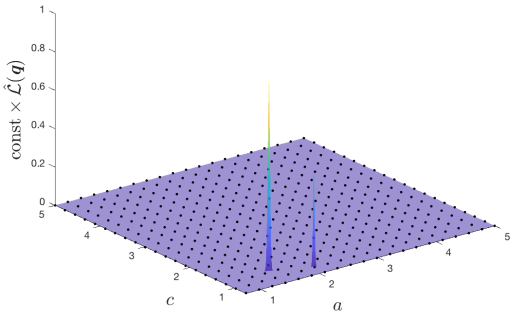
[3]On a MacBook Pro with a 2.6GHz 6-core Intel Core i7 processor.



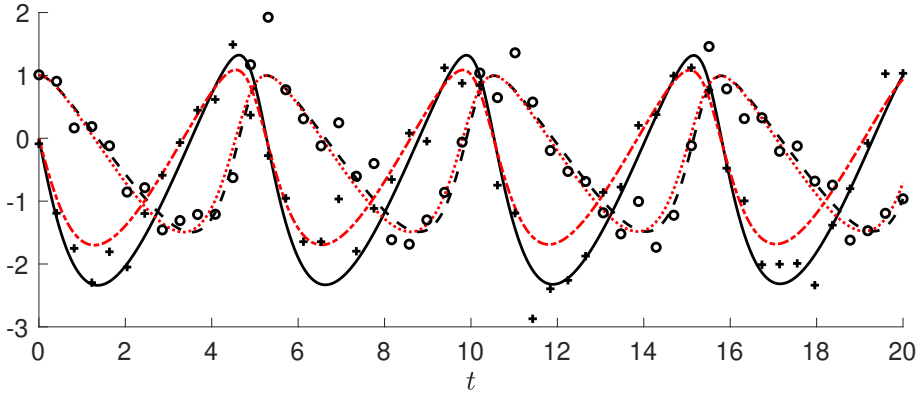
(a)



(b)



(c)



(d)

Figure 4.4: (a) The dataset $\boldsymbol{Y}_{301}$ giving rise to the outlier $\hat{\boldsymbol{q}}_{301}$ in Figure 4.3; (b) the emulated log likelihood for this dataset; (c) the emulated likelihood; (d) the true signal (in black) and the estimated signal $\boldsymbol{U}(\hat{\boldsymbol{q}}_{301})$ (in red) .

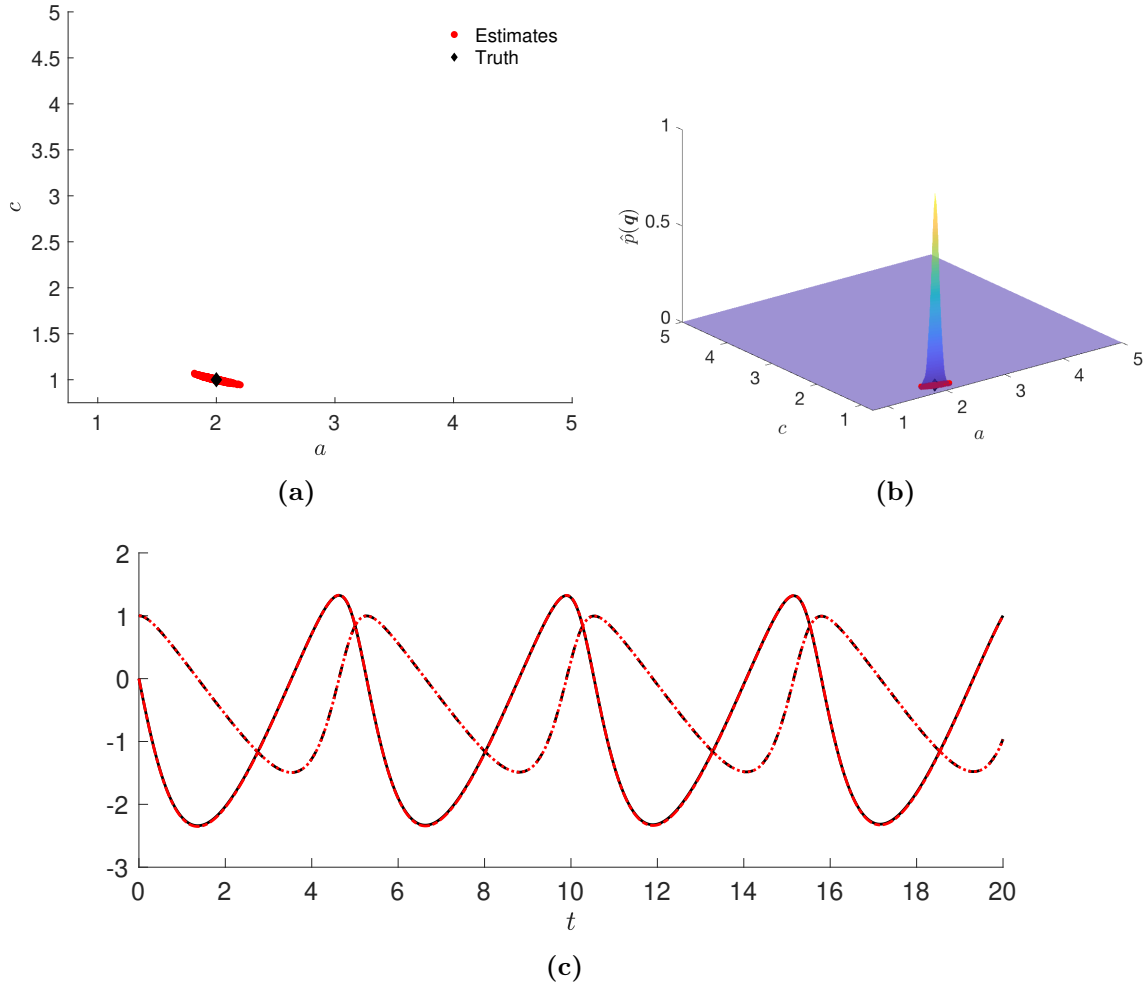


Figure 4.5: (a) The sample of optimal parameters obtained by direct minimization of the true RSS function (no emulation involved) for 1000 different datasets (red circles) and $\boldsymbol{q}^*$ (black diamond); (b) the kernel density estimate of the sample of optimal parameters; (c) the ODE solution for $\boldsymbol{q}^{\text{KDE}} = \arg\max_{\boldsymbol{q}} \hat{p}(\boldsymbol{q})$ in red and the true signal in black.

first cycle of the ODE. We study two different T and SNR settings:

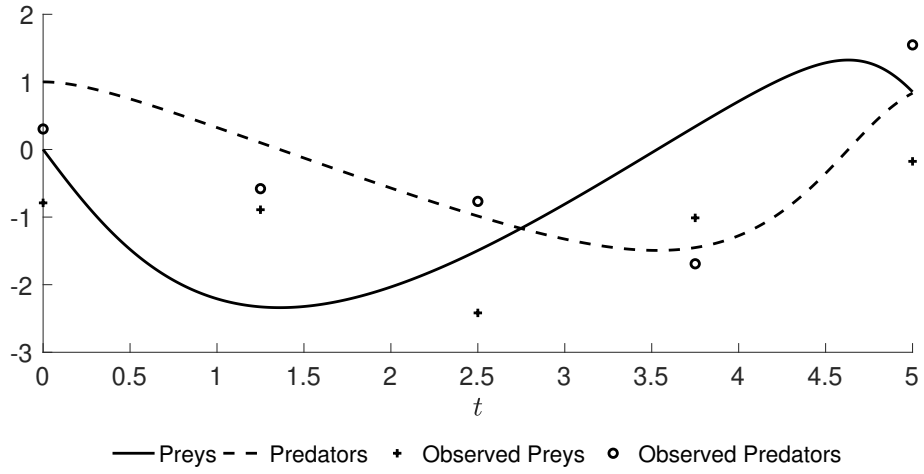
S1 High uncertainty: $T = 5$ and SNR $= 1$;

S2 Low uncertainty: $T = 50$ and SNR $= 10$.

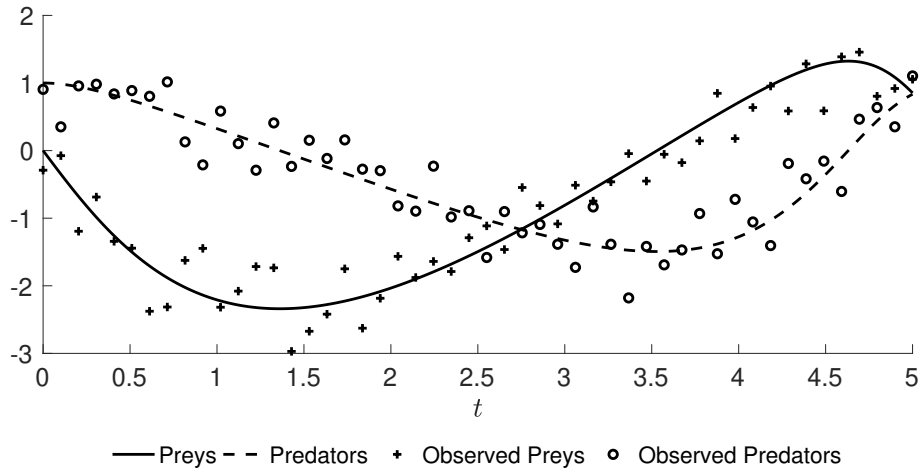
In this new experiment we consider $K = 5000$ different datasets $\{\boldsymbol{Y}_k\}$ for each setting, and we still minimize the set of predictive means $\{\hat{l}_k(\boldsymbol{q})\}$ using the 50 starting points discussed beforehand. This leads to two samples of optimal parameters $\{\hat{\boldsymbol{q}}_k\}_{k=1}^{5000}$, one for each scenario. Figure 4.6 (a) plots the first dataset from scenario **S1**, while (b) the first dataset from scenario **S2**.

Figure 4.7 shows that the emulated (log) likelihood for a particular dataset is characterized by many local optima in the case of higher uncertainty (**S1**) while in the latter scenario (**S2**) we find a more pronounced peak. By optimizing the emulated log likelihood for different datasets, we obtain a distribution of optimal parameters scattered around the true configuration $\boldsymbol{q}^* = (2, 1)$. A multivariate kernel density estimator is shown in Figure 4.7 (centre right). The argument that maximizes the density estimate in the first scenario is $\boldsymbol{q}^{\text{KDE}} = (2.18, 0.96)$ and $\boldsymbol{q}^{\text{KDE}} = (1.98, 1.01)$ in the last. This allows us to make a comparison in the parameter space with the true configuration $\boldsymbol{q}^* = (2, 1)$. In order to compare the estimates with the true parameter in the function space instead, Figure 4.7 (right) plots the solution of the ODE for each $\boldsymbol{q}^{\text{KDE}}$ (in red) and the true signal using $\boldsymbol{q}^*$ (in black). It is worth remarking that the inference using only $T = 5$ timepoints shown in the top right panel of Figure 4.7 is based on $\boldsymbol{q}^{\text{KDE}}$. This is the parameter having the highest kernel density estimate in the sample of optimal parameters from 5000 different datasets. We would not expect such a good reconstruction of the original signal with just one dataset comprising 5 noisy observations only.

From the distribution of the optimized parameters shown in Figure 4.8 (a) we can see that the maximum emulated likelihood is an approximately unbiased estimator of the true ODE parameter vector. Panel (b) shows the optimal parameters obtained by direct minimization of the true RSS function. While solving the $500,000$ optimization tasks (2 different T and SNR settings, 50 starting points and 5000 datasets) using



(a)



(b)

Figure 4.6: The underlying true signal for one ODE cycle and one observed dataset. Panel (a) shows the high uncertainty scenario **S1**, while Panel (b) the low uncertainty scenario **S2**.

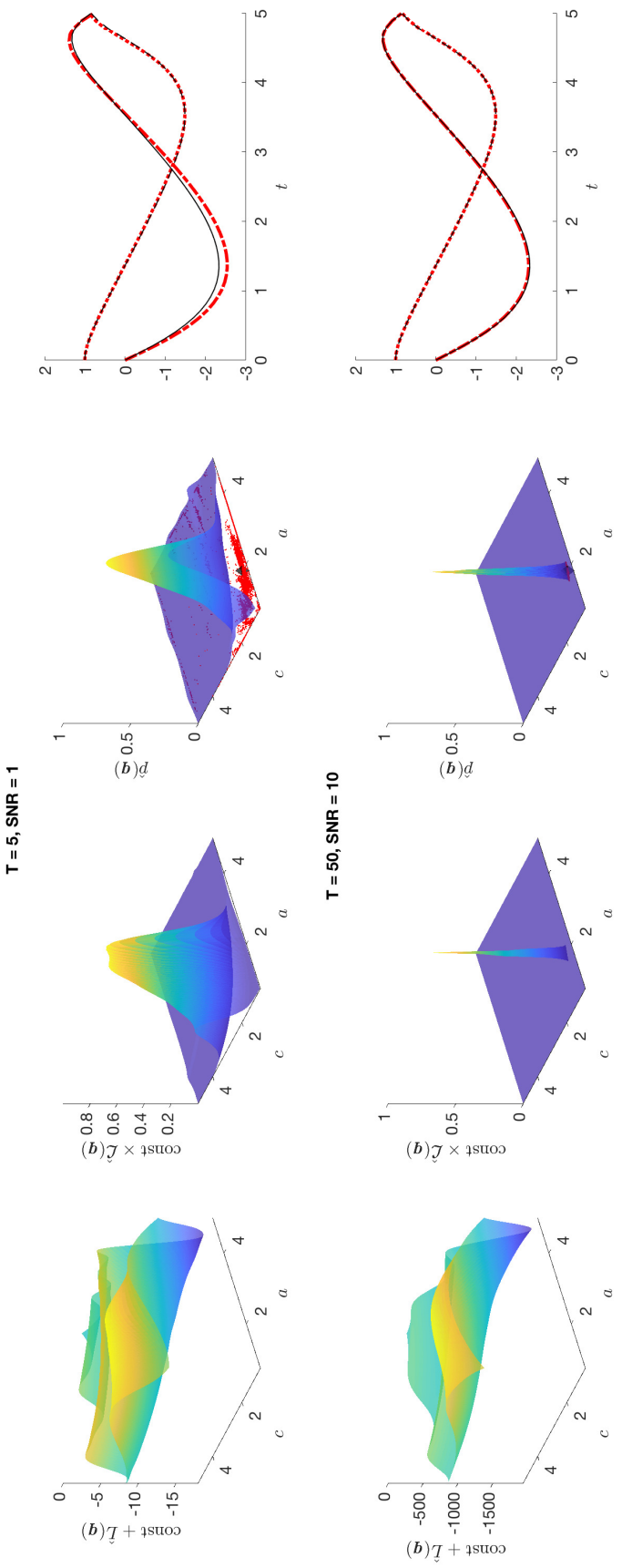


Figure 4.7: For two different T and SNR settings: the emulated log likelihood (left) and the emulated likelihood (centre left) for one dataset; the kernel density estimate of the sample of optimal parameters from 5000 different datasets (centre right); the ODE solution for $\boldsymbol{q}^{\text{KDE}} = \arg\max_{\boldsymbol{q}} \hat{p}(\boldsymbol{q})$ shown in red and the true signal shown in black (right).

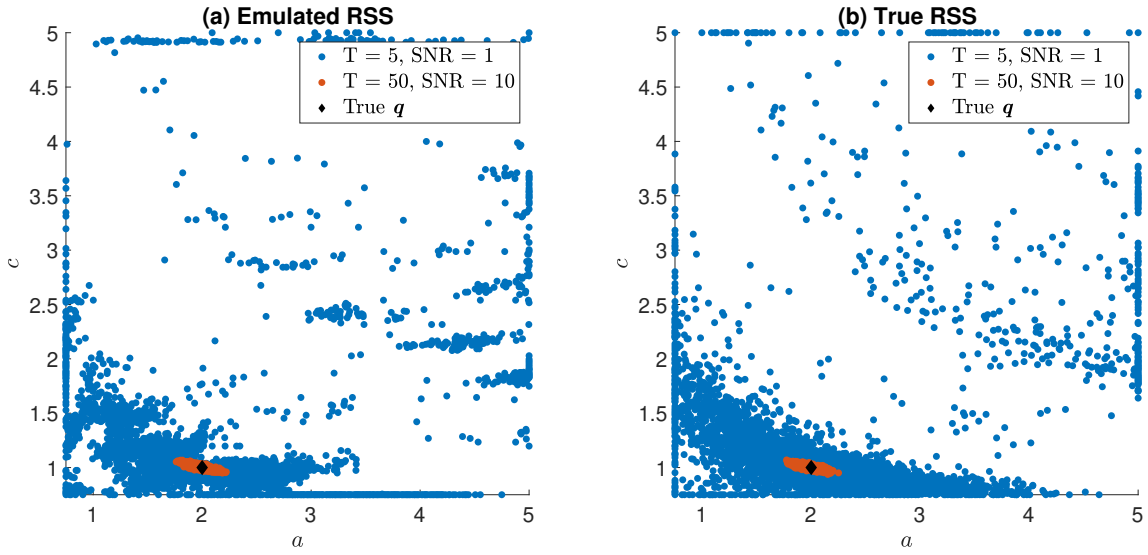


Figure 4.8: (a) The sample of optimal parameters obtained by minimizing the GP predictive mean in 5000 different datasets. (b) The optimal parameters obtained by direct minimization of the true RSS on the same 5000 datasets.

the emulated RSS took 1 hour and a half[4], in order to solve the same problem with the true RSS (4.1) we needed 5 hours and a half.

We now compare the performance of GP emulation over brute-force grid search. In other words, we are interested in quantifying the gain obtained by fitting a GP and then minimizing the posterior mean, as opposed to considering as $\hat{\boldsymbol{q}}$ the training input $\boldsymbol{q}_n$ giving rise to the lowest training l_n score; gain which, of course, depends on the resolution of the grid, G. I ran a simulation study with 4 different T and SNR settings, 4 different grid sizes in each dimension and varying the true parameter configuration $\boldsymbol{q}^*$ as the interest is not in the inference of the true parameter vector anymore. Given the training data $\mathcal{D} = \{(\boldsymbol{q}_n, l_n)\}_{n=1}^N$, define $\boldsymbol{q}^{\mathrm{G}} = \arg\min(l_1, \ldots, l_N)$ and $\hat{\boldsymbol{q}} = \arg\min \hat{l}(\boldsymbol{q})$. Figure 4.9 shows the differences of the norms $\|\boldsymbol{q}^{\mathrm{G}} - \boldsymbol{q}^*\|^2$ and $\|\hat{\boldsymbol{q}} - \boldsymbol{q}^*\|^2$, where $\boldsymbol{q}^*$ represents the true parameter vector. The plots show that emulation outperforms grid search in the scenarios of many timepoints in the solution of the ODE and small grid resolutions, see Panel (c) and (d). Hence, emulation turns out to be of particular importance in problems with high-dimensional parameter spaces, where a dense grid would be computationally too onerous to evaluate. In

[4]On a MacBook Pro with a 2.6GHz 6-core Intel Core i7 processor.

higher dimensional spaces, the density of the grid would be comparable to the scenarios presented in the first two boxplots of Panel (c) or (d). When increasing the resolution of the grid, as expected, the difference becomes not significant. In the cases of few timepoints (Panel (a) and (b)) the value 0 is included in the boxplots, showing a non significant improvement in optimizing the GP posterior mean rather than doing grid search, due to the very low amount of information available. For this particular application, considering the low-dimensionality of the parameter space and the high number of gridpoints ($N = G^2 = 400$), we see no significant gain by going from grid search to GP emulation. However, as previously discussed, the importance of emulation becomes much clearer as the dimensionality of the parameter space increases.

In retrospect, it is worth investigating what would have happened in the problem considered at the beginning of Section 4.4, i.e. for $t \in [0, 20]$ and $T = 50$, if the ARD Matérn 5/2 kernel (3.22) had been used. Would the estimates be centred around the true parameter vector $\boldsymbol{q}^*$? As we can see in Figure 4.10, this is not the case. The only difference lies in one point being in the bulk of the distribution instead of being an outlier. However, since a change in the kernel only affected one out of 1000 points, we do not consider this as a strong evidence to re-run the whole analysis with the ARD Matérn 5/2 kernel. Furthermore, the reconstruction in function space is identical when either using the ARD Squared Exponential or the ARD Matérn 5/2 kernel, see Panel (c) of Figures 4.3 and 4.10 respectively.

4.5 Summary

In this chapter, I investigated an emulation approach based on GPs to estimate ODE parameters by numerically solving the differential equations only at a small pre-selected set of parameter configurations. The emulation entails fitting a GP to a normalized version of the RSS between the observed data and the finite set of explicit ODE numerical solutions.

Working with a GP-based emulator of the normalized RSS has strong advantages over direct numerical integration of the ODEs. The GP formulation leads to analytical

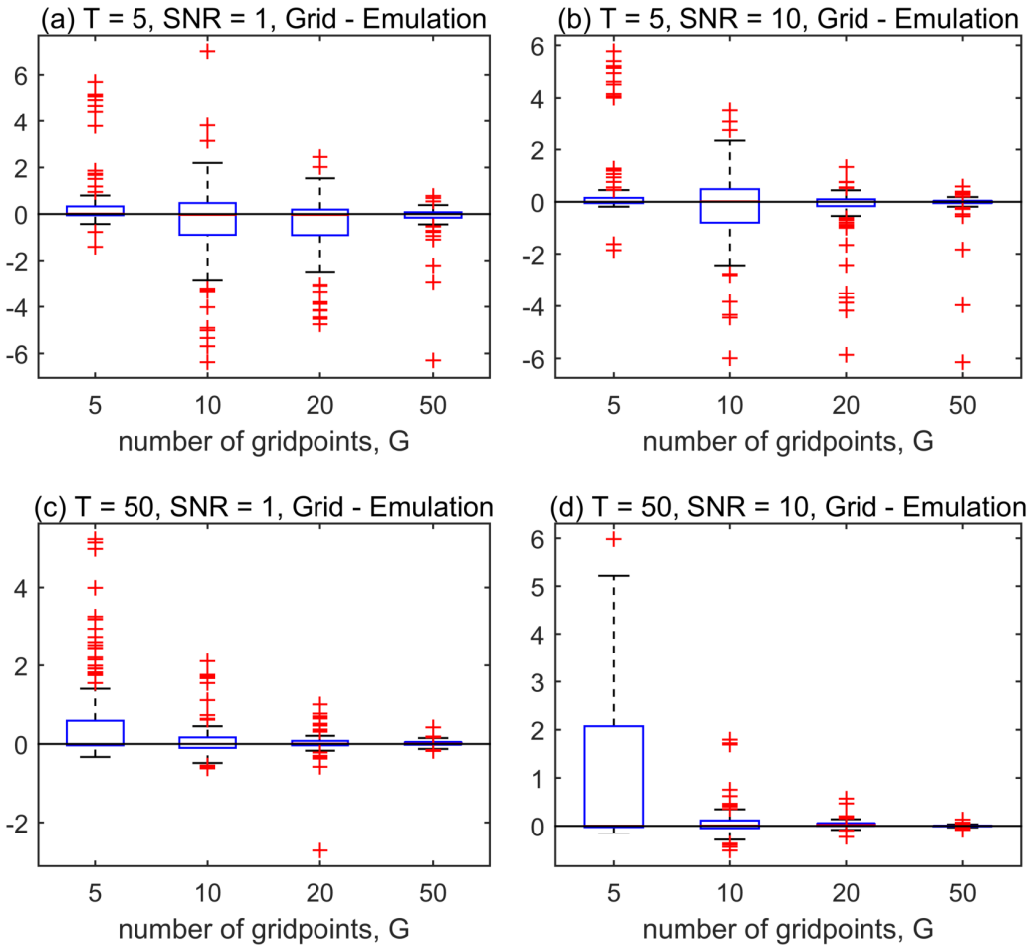


Figure 4.9: Boxplots of the differences of the norms $\|\boldsymbol{q}^{\mathrm{G}} - \boldsymbol{q}^*\|^2$ and $\|\hat{\boldsymbol{q}} - \boldsymbol{q}^*\|^2$, representing the distance of the grid-based approach estimate to the true parameter vector and the distance of the emulation-based approach estimate to the true parameter vector respectively. Here $\boldsymbol{q}^{\mathrm{G}} = \arg\min(l_1, \ldots, l_N)$, $\hat{\boldsymbol{q}} = \arg\min_{\boldsymbol{q}} \hat{l}(\boldsymbol{q})$ and $\boldsymbol{q}^*$ is the true parameter configuration. In the figure, "Grid - Emulation" is a reminder that we subtract the distance of the emulation-based approach estimate to the true parameter vector from the distance of the grid-based approach estimate to the true parameter vector. If the difference is positive, emulation outperforms the grid-based approach.

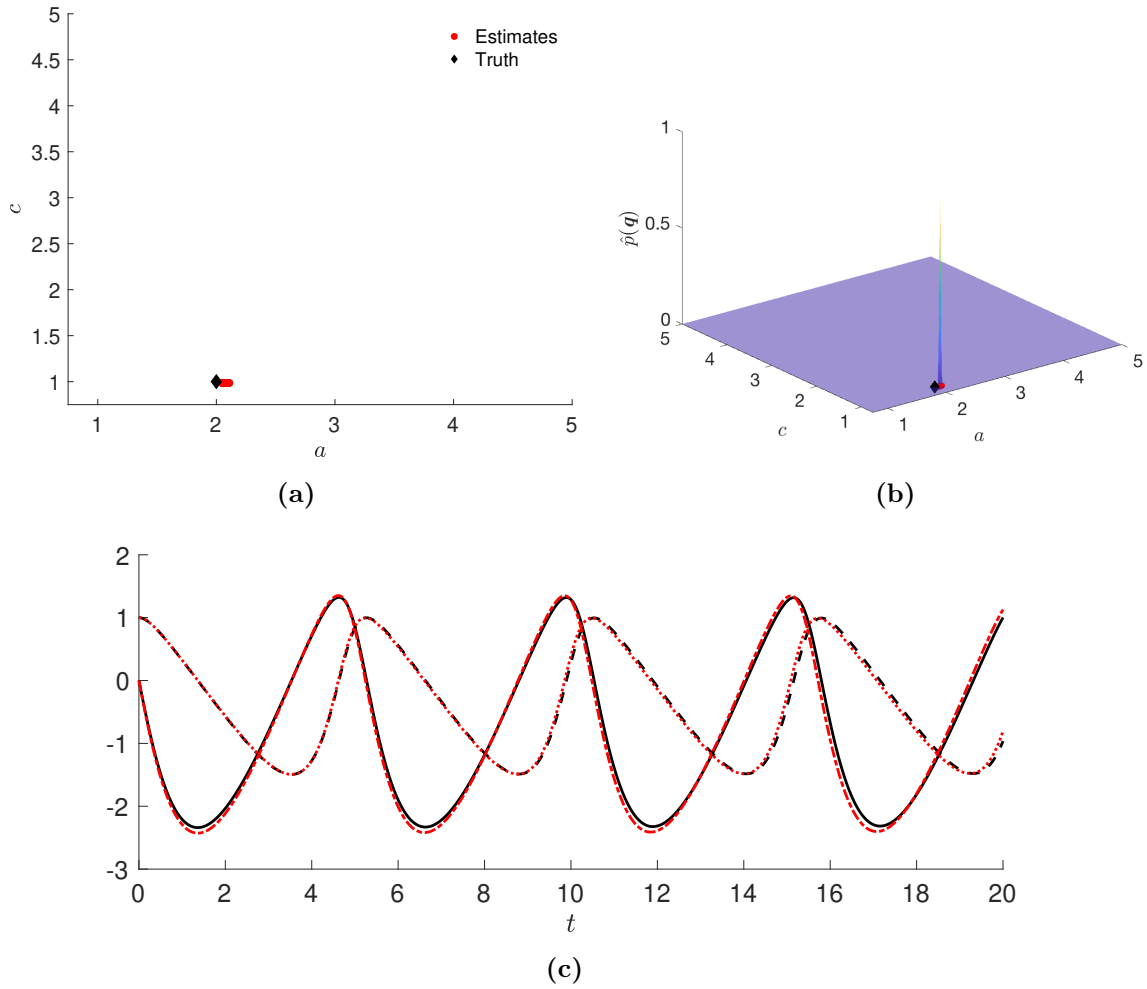


Figure 4.10: Using the ARD Matérn 5/2 kernel: (a) the sample of optimal parameters from 1000 different datasets (red circles) and the true parameter vector (black diamond); (b) the kernel density estimate of the sample of optimal parameters; (c) the ODE solution for $\boldsymbol{q}^{\mathrm{KDE}} = \arg\max_{\boldsymbol{q}} \hat{p}(\boldsymbol{q})$ in red and the true signal in black.

predictive formulas, along with gradients and Hessians, and the computational time was reduced. To solve the same problem by minimizing the true RSS I needed 5 hours and a half, while in only 1 hour and a half I solved half a million optimization tasks which involved fitting 10,000 different GPs. The Lotka-Volterra system, however, is not very computationally expensive to integrate numerically, hence this chapter represents a proof-of-concept study done in my first year of PhD in order to get familiar with Gaussian processes and emulation.

The grid-based approach turns out to be as effective as emulation in low-dimensional parameter spaces, when the number of gridpoints can be high. However, covering higher dimensional Euclidean spaces as effectively with a grid would be difficult, and these are the scenarios where GP emulation is of particular help. The results in Figure 4.7 show that the parameter configuration that has the highest estimated density of the optimized parameters is a very good estimate of the true parameter vector, and the estimator appears to be approximately unbiased (Figure 4.8).

In the next chapter, I demonstrate how GP emulation can be used to estimate the parameters of a soft tissue mechanical model of the left ventricular dynamics, where a single output takes approximately 11 minutes CPU time[5]. That application promises more substantial savings in terms of computational time.

[5]Intel Xeon CPU, 2.9GHz, 32 cores and 32GB memory.

Chapter 5

Fast Inference in a Computational Model of the Left Ventricle Using Emulation

A central topic in biomechanics is modelling of the human left ventricle (LV). From a personalized model of the LV it is possible to estimate properties of the cardiac soft tissues using in-vivo clinical measurements. These properties aim to provide insight into heart function or dysfunction without the need for invasive measurements. However, finding a solution to the differential equations which mathematically describe the myocardium through numerical integration can be computationally expensive. In order to provide estimates in a time frame suitable for clinical decision support systems, in this chapter we use the concept of emulation (discussed in Section 2.3) to infer the properties of the cardiac muscle of a healthy volunteer from non-invasive magnetic resonance imaging (MRI) data. We compare and contrast two emulation targets: (1) emulation of the computational model output and (2) emulation of the loss between the observed data and the computational model output. Both strategies are tested with two different statistical approximations, as well as with two different loss functions. The best combination of methods is found by comparing the accuracy of the parameter inference on simulated test data for which the true parameters are known. Finally, the best method is used to estimate the material parameters for a healthy volunteer. The best approach provides accurate parameter inference in both

simulated and clinical data, with a reduction in the computational costs of about 3 orders of magnitude compared to numerical integration of the differential equations using finite element discretization techniques.

Notes This chapter is adapted from Davies et al. (2018), in submission. Vinny Davies and I are joint first authors of the paper, but the project is the cumulative effort of different people. Hao Gao and Xiaoyu Luo are the authors of the computational model of the left ventricle. Dirk Husmeier overviewed all of the statistics work packages which follow. Benn Macdonald focused on the reparametrization of the LV model, designed the training simulations and carried out the numerical solution of the differential equations for the space-filling training inputs by massive parallelization. I fitted the local Gaussian process emulators of the training runs. Once the best strategy was found, I estimated the myocardial parameters from real data of a healthy volunteer. I take full responsibility for the results of this method only. Vinny Davies independently carried out another study fitting low rank Gaussian processes using the R package `mgcv` by Wood (2017). His results are reported side-by-side for comparison only, and to check for agreement. Alan Lazarus is extending the work to generic left ventricular geometries by investigating the dimensionality reduction approaches discussed in Section 5.7. All members participated in discussions.

5.1 Motivation

Computational modelling of cardiac biomechanics, when integrated with in vivo imaging, can provide means to understand cardiac function for both healthy and diseased individuals (Smith et al., 2011; Wang et al., 2015; Chabiniok et al., 2016). Recent mathematical studies have demonstrated that passive myocardial stiffness is higher in diastolic heart failure patients compared to healthy volunteers (Xi et al., 2014). Similarly, patients who had a heart attack (myocardial infarction) need their heart muscle to contract more than healthy people. This is to compensate for the damage in their heart (Gao et al., 2017). Myocardial passive properties not only affect left ventricular (LV) diastolic filling, but also influence the heart pumping

function during systole through the Frank–Starling law: the relationship between stroke volume and end diastolic volume (Widmaier et al., 2016). From recent studies it is recognised that myocardial passive stiffness provides diagnostic information for patient risk stratification (Xi et al., 2014; Gao et al., 2017). Therefore, it is helpful to quantify passive myocardial stiffness in order to assess LV function. Traditionally, myocardial properties are determined by a series of ex vivo or in vitro experiments (Dokos et al., 2002). The Holzapfel–Ogden (HO) constitutive law (Holzapfel and Ogden, 2009), widely used in the biomechanics literature, gives a detailed description of the myocardial response, including the effects of fibre structure. However, to be of practical use for clinical applications, the model requires specifying the values of the material parameters. This is challenging due to the need for invasive experiments (Dokos et al., 2002). The biomechanical model considered in this chapter describes the LV dynamics during the diastolic filling process, starting from early-diastole and finishing at end-diastole (the point of maximum LV expansion). Both early and late diastolic states can be measured by magnetic resonance imaging (MRI). We can therefore estimate the parameters non-invasively by formulating an inverse problem and matching the individual measurements of a healthy volunteer to the output of the biomechanical model (Gao et al., 2015).

Many studies have demonstrated that it is possible to estimate the constitutive material parameters from in vivo measurements even with very complex constitutive equations (Guccione et al., 1991; Remme et al., 2004; Sermesant et al., 2006; Sun et al., 2009). However, because of complex interdependencies among the material parameters and sparse noisy data, the formulated inverse problem is highly nonlinear (Xi et al., 2011; Gao et al., 2015). Determining the unknown parameters is very time consuming, with the direct inference process taking days or weeks to converge, even on a modern multi-core workstation (Gao et al., 2015; Nikou et al., 2016). The primary reason for this is the high computational cost of a single simulation from the biomechanical model (approximately 11 minutes CPU time[1]) and the thousands of steps required to optimize the loss function, each step involving an expensive simulation from the LV model. Direct estimation of the myocardial properties

[1]Intel Xeon CPU, 2.9GHz, 32 cores and 32GB memory.

using the simulator is not suitable for real-time clinical diagnosis, hence calling for emulation approaches.

This chapter presents a proof-of-concept study that aims to demonstrate how emulation can be used to successfully to learn the myocardial properties of a healthy volunteer from *non-invasive* in vivo MRI data only. To this end, we use a simplified simulator which is limited in applicability to a single patient, rather than being a general tool, and at this stage we focus on developing the statistical methodology in this simplified setting which can then be applied to more complex tasks in the future. The considered biomechanical model uses a fixed LV geometry from a healthy volunteer and assumes a fixed population-based value for the end-diastolic pressure of 8 mmHg as in Gao et al. (2015) to set the mathematical boundaries. Additionally, we consider a reduced parametrization of the HO law in the biomechanical model presented by Gao et al. (2015). The main focus of this chapter is not in having the most general simulator or emulator, but rather on testing on simulated data different statistical emulation strategies, comparing different loss functions, as well as interpolation methods. The best strategy is then used to estimate the myocardial properties of a healthy volunteer for which the MRI data and the LV geometry are available, to assess its ability to estimate the material properties in a time frame suitable for clinical decision support systems.

5.2 The Left Ventricle Model

The LV biomechanical model describes the diastolic filling process from early-diastole to end-diastole. Different models have been proposed in the literature, and a review is given by Chabiniok et al. (2016). In this chapter we consider the model used in Wang et al. (2013a) and Gao et al. (2015), which assumes that the stress-strain

constitutive equation is given by the Holzapfel–Ogden law:

$$\begin{aligned}\Psi = &\frac{a}{2b}\{\exp[b(I_1-3)]-1\}\\ &+\sum_{i\in\{f,s\}}\frac{a_i}{2b_i}\{\exp[b_i(I_{4i}-1)^2]-1\}\\ &+\frac{a_{fs}}{2b_{fs}}[\exp(b_{fs}I_{8fs}^2)-1]\\ &+\frac{1}{2}K(J-1)^2,\end{aligned} \tag{5.1}$$

where, among other quantities, $a, b, a_f, b_f, a_s, b_s, a_{fs}, b_{fs}$ are the eight non-negative unknown material parameters to be estimated. For the purpose of statistical inference, the simulator is considered as a *black-box function* taking as input a vector of parameters $(a, b, a_f, b_f, a_s, b_s, a_{fs}, b_{fs})$ and returning a multivariate output $\boldsymbol{y} \in \mathbb{R}^{25}$ which includes a prediction of the LV chamber volume and 24 predicted strain measurements along the chamber wall. For more details about the other quantities in the law, see Wang et al. (2013a).

Gao et al. (2015) performed sensitivity analysis and found that the 8 parameters of the HO law are strongly correlated, hence their identification from limited and noisy in vivo measurements is challenging. For example, an increase in a can be compensated by decreasing b or the remaining correlated parameters. They further demonstrated that myofibre stiffness, the final quantity of interest, can be estimated well by considering a reduced parametrization. Similarly to Gao et al. (2015), we group the eight parameters of (5.1) into four, so that:

$$\begin{aligned} a &= q_1\, a_0 & b &= q_1\, b_0\\ a_f &= q_2\, a_{f0} & a_s &= q_2\, a_{s0}\\ b_f &= q_3\, b_{f0} & b_s &= q_3\, b_{s0}\\ a_{fs} &= q_4\, a_{fs0} & b_{fs} &= q_4\, b_{fs0}\end{aligned} \tag{5.2}$$

where $\boldsymbol{q} = (q_1, \ldots, q_4) \in [0.1, 5]^4$ are the parameters to be inferred from in vivo data, and the reference values $a_0 = 0.224\,\text{kPa}$, $b_0 = 1.62$, $a_{f0} = 2.427\,\text{kPa}$, $b_{f0} = 1.83$, $a_{s0} = 0.556\,\text{kPa}$, $b_{s0} = 0.775$, $a_{fs0} = 0.391\,\text{kPa}$, $b_{fs0} = 1.695$ are the estimated parameters from the multi-stage approach of Gao et al. (2017).

The black-box function can be considered as the input-output relationship $\boldsymbol{y} = \mathsf{m}(\boldsymbol{q}) \in \mathbb{R}^{25}$ for $\boldsymbol{q} = (q_1, \ldots, q_4)$ and fixed reference values $a_0, \ldots, b_{fs0}$. Equation (5.2)

assumes that the ratio between a and b is constant. This is motivated by the fact that a and b represent the isotropic response of the myocardium, or the extracellular matrix, and are assumed to be similar for different healthy subjects. The ratio between a_{fs} and b_{fs} is also fixed due to the general low mechanical response of the fibre-sheet cross-link (Dokos et al., 2002). Then, as the two terms involving a_f and a_s both describe collagen structures with different types, we further assume a_f and a_s share the same scale from the original values, and similarly for b_f and b_s. However, the assumptions in (5.2) may be only applicable to a sub-population of healthy volunteers, i.e. the same age group with similar ventricular size and blood pressure.

A simulation $\mathsf{m}(\boldsymbol{q})$ from the computational model without using parallelization takes about 15 minutes in our local Linux workstation[2], or around 11 minutes with parallelization on 6 CPUs. Note that the 15 or 11 minutes are required to obtain just a single (multivariate) output from the simulator.

5.3 Comparative Study

The goal of this section and the next one is to compare the performance of different emulation strategies on simulated test data. The simulated data are obtained by considering the LV computational model, using the left ventricular geometry of a healthy volunteer, as a generative model. The best strategy is then used in Section 5.5 to infer the myocardial parameters of a healthy volunteer for which observed data are available.

In order to fit any emulator we need a set of training runs $\mathcal{D} = \{(\boldsymbol{q}_i, \boldsymbol{y}_i)\}_{i=1}^n$. The training inputs $\boldsymbol{Q} = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n]^\top$ represent $n = 10,000$ points from a Sobol sequence in $[0.1, 5]^4$. The corresponding outputs have been obtained by Benn Macdonald using massive parallelization, solving the LV model for each input: $\boldsymbol{y}_i = \mathsf{m}(\boldsymbol{q}_i)$ for $i = 1, \ldots, n$. In addition to the $n = 10,000$ training points, he also generated $m = 100$ additional test points by extending the Sobol sequence and simulating from the model at each of the test inputs. The test points are used as an out-of-sample dataset to test the performance of the different emulation strategies.

[2]Intel Xeon CPU, 2.9GHz, 32 cores and 32GB memory.

We compare different types of statistical approximation: *Local GP* and *Low-Rank GP*; two different emulation targets: emulation of the model output (*output emulation*) vs emulation of the loss between a simulation and the data (*loss emulation*); as well as two different loss functions: *Euclidean* and *Mahalanobis* with $\boldsymbol{\Sigma} = \mathrm{Cov}(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$. The methods using Local GPs were run by myself, while Low-Rank GPs by Vinny Davies as part of a joint paper. Because of the large number of training data ($n = 10,000$) and the $O(n^3)$ computational complexity of standard GP regression which is due to the inversion of the $n \times n$ training covariance matrix $\boldsymbol{K}$, we can not apply exact GP regression as in (3.26). The statistical models considered in this study are Local Gaussian processes, also discussed by Gramacy and Apley (2015), and Low-Rank GPs as described in Wood (2017).

5.3.1 Local Gaussian Processes

When the sample size n is large, it is not feasible to use exact GP regression on the full dataset as described in Section 3.5, due to the $O(n^3)$ computational complexity of the $n \times n$ training covariance matrix $\boldsymbol{K}$ inversion. A possible approach is to use sparse GPs as in Titsias (2009), or the more recent approaches by Gal and Turner (2015) and Hensman et al. (2018). Titsias (2009) considers a fixed number of m inducing variables $\boldsymbol{u} = (u_1, \ldots, u_m)$, with $m \ll n$, corresponding to inputs $\boldsymbol{Z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m]^\top$. The locations of the inducing points and the kernel hyperparameters are chosen by maximizing using variational inference methods the evidence lower bound (ELBO), i.e. a lower bound on the log marginal likelihood. The ELBO can be derived by applying Jensen's inequality:

$$\begin{aligned}
\log p(\boldsymbol{y}) &= \log \int\int p(\boldsymbol{y}, \boldsymbol{f}, \boldsymbol{u}) \mathrm{d}\boldsymbol{u} \mathrm{d}\boldsymbol{f} \\
&= \log \int\int q(\boldsymbol{f}, \boldsymbol{u}) \frac{p(\boldsymbol{y}, \boldsymbol{f}, \boldsymbol{u})}{q(\boldsymbol{f}, \boldsymbol{u})} \mathrm{d}\boldsymbol{u} \mathrm{d}\boldsymbol{f} \\
&\geq \underbrace{\int\int q(\boldsymbol{f}, \boldsymbol{u}) \log \frac{p(\boldsymbol{y}, \boldsymbol{f}, \boldsymbol{u})}{q(\boldsymbol{f}, \boldsymbol{u})} \mathrm{d}\boldsymbol{u} \mathrm{d}\boldsymbol{f}}_{\mathcal{F}(q(\boldsymbol{u}))} .
\end{aligned}$$

The computational costs of this approach are $O(nm^2)$. I initially tried sparse GPs with 100, 500 and 1000 inducing points but, using the code accompanying the paper

by Titsias (2009), the prediction time was between 0.5 and 0.6 seconds for 100 inducing points, around one second for 500, and in the order of a few seconds for 1000 inducing points[3]. This means that minimization of the surrogate-based loss (2.7) would still be slow as approximately 1 second is required for a single evaluation. The optimization time would exceed two and a half hours for 500 inducing points when using $10,000$ function evaluations. Consider now the variational sparse GP model using 100 inducing points only, which was the fastest between the three cases considered, taking approximately 0.5 seconds for a prediction at a given input. Figure 5.1 shows the predictive accuracy of the sparse GP model on the test data. In other words, each plot shows the true test outputs vs the prediction of the sparse GP at the test inputs. We can see that the fit on some variables like the LV chamber volume and Strains 17, 19, 20, 21, 22 and 23 are slightly off the perfect prediction line. The predictive accuracy improves by increasing the number of inducing points, but at the cost of a slower prediction time.

Keeping in mind the goal of the project: real-time in-clinic decision making, a local Gaussian process approach based on the K-nearest-neighbours was used instead (Gramacy and Apley, 2015). This method uses the standard GP prediction formulas described in Chapter 3, but subsetting the training data. Whenever we require a prediction at a given input, we find the training inputs representing the K-nearest-neighbours in input-domain, which will form the local set of training inputs, and the corresponding outputs will represent the local training outputs. Note that every time we ask for a prediction at a different input, the training sets need to be re-computed and the GP needs to be trained again. However, because of the small number of neighbours $K \ll 1000$ usually selected, this method is computationally fast and accurate, see Gramacy and Apley (2015) for a discussion.

Gramacy and Apley (2015) further discuss adding a fixed number of distant points in order to help in the estimation of the lengthscale parameters, but this comes with extra computational costs required by the iterative choice of which point to add to the set of neighbours. Given the time limitations required by our goal (real-time clinical decision support systems) we do not pursue this approach. Furthermore, this

[3]Dual Intel Xeon CPU E5-2699 v3, 2.30GHz, 36 cores and 128GB memory.

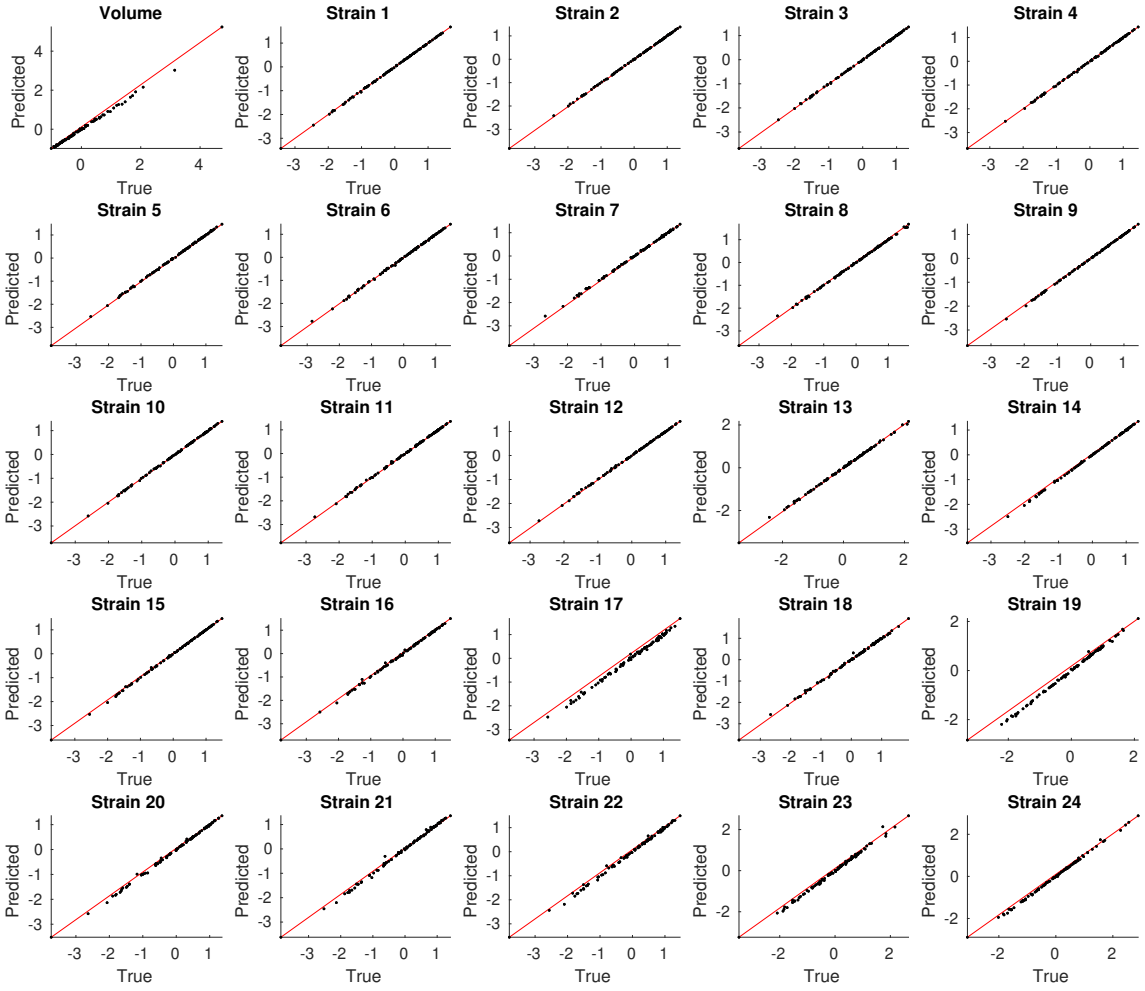


Figure 5.1: True vs predicted test outputs using Output Emulation with variational sparse GPs and $K = 100$ inducing points.

is mostly relevant when the interest lies in building predictive models able to make good predictions when the training data are distant from each other. Since we are working on a compact set which is very densely covered by the Sobol sequence, this is not necessary. For generic training data $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\} = \{\boldsymbol{X}, \boldsymbol{y}\}$, we can summarize the algorithm as follows:

Algorithm 5.1. *Predicting from a local Gaussian process at* $\boldsymbol{x}_*$:

1. Find the indices $\mathcal{N}(\boldsymbol{x}_*)$ of the points in $\boldsymbol{X}$ having the K smallest Euclidean distances from $\boldsymbol{x}_*$;
2. Training inputs: $\boldsymbol{X}_K(\boldsymbol{x}_*) = \{\boldsymbol{x}'_1, \ldots, \boldsymbol{x}'_K\} = \{\boldsymbol{x}_i : i \in \mathcal{N}(\boldsymbol{x}_*)\}$;
3. Training outputs: $\boldsymbol{y}_K(\boldsymbol{x}_*) = \{y'_1, \ldots, y'_K\} = \{y_i : i \in \mathcal{N}(\boldsymbol{x}_*)\}$;
4. Train a GP using the data $\mathcal{D}_K(\boldsymbol{x}_*) = \{\boldsymbol{X}_K(\boldsymbol{x}_*), \boldsymbol{y}_K(\boldsymbol{x}_*)\}$;
5. Predictive mean: $\hat{f}(\boldsymbol{x}_*) = m(\boldsymbol{x}_*) + \boldsymbol{k}(\boldsymbol{x}_*)^\top [\boldsymbol{K} + \sigma^2 \boldsymbol{I}]^{-1} (\boldsymbol{y}_K(\boldsymbol{x}_*) - \boldsymbol{m})$;
6. Predictive variance: $s^2(\boldsymbol{x}_*) = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}(\boldsymbol{x}_*)^\top [\boldsymbol{K} + \sigma^2 \boldsymbol{I}]^{-1} \boldsymbol{k}(\boldsymbol{x}_*)$.

In the algorithm above, the $K \times K$ training covariance matrix $\boldsymbol{K} = [k(\boldsymbol{x}'_i, \boldsymbol{x}'_j)]_{i,j=1}^K$, the $K \times 1$ vector of covariances between the training points and the test point is $\boldsymbol{k}(\boldsymbol{x}_*) = (k(\boldsymbol{x}'_1, \boldsymbol{x}_*), \ldots, k(\boldsymbol{x}'_K, \boldsymbol{x}_*))$ and $\boldsymbol{m} = (m(\boldsymbol{x}'_1), \ldots, m(\boldsymbol{x}'_K))$ is the $K \times 1$ prior mean vector. We consider a constant mean function $m(\boldsymbol{x}) = c$ and the ARD Squared Exponential kernel as used in the emulation of computer codes literature, see Santner et al. (2003); Fang et al. (2006). The model hyperparameters are estimated by maximizing the log marginal likelihood using the Quasi-Newton method, as described in Section 3.10, with σ initialized to 10^{-2} since we are modelling deterministic computer code.

The CPU time required to get a prediction from the local Gaussian process is approximately 0.18 seconds[4] using the $K = 100$ nearest neighbours of a given point. The number of neighbours K needs to be selected on the basis of the computational time allowed to reach a decision in a viable time frame, but keeping in mind that

[4]Dual Intel Xeon CPU E5-2699 v3, 2.30GHz, 36 cores and 128GB memory.

K also controls the accuracy of the emulation. In our experiments we found that $K = 100$ was sufficiently fast. Next, we evaluated the predictive accuracy of the chosen local GP approach ($K = 100$) on the test data. Figure 5.2 shows that, unlike the fastest sparse GPs approach, local GP regression using the $K = 100$ nearest-neighbours leads to very accurate predictions at the test inputs, as the predicted and true test outputs all lie on the perfect prediction line $y = x$.

Given the LV training runs $\mathcal{D} = \{(\boldsymbol{q}_i, \boldsymbol{y}_i)\}_{i=1}^n$, Output Emulation consists in fitting a multivariate emulator $\hat{\mathsf{m}} = (\hat{\mathsf{m}}_1, \ldots, \hat{\mathsf{m}}_{25})$ which comprises 25 local GPs estimated in parallel on 25 CPU cores. For data $\boldsymbol{y}^{\text{obs}}$, the estimated parameter vector $\hat{\boldsymbol{q}}$ is obtained by minimizing the *surrogate-based loss* (2.7):

$$\ell_{\hat{\mathsf{m}}}(\boldsymbol{q}) = d\left(\hat{\mathsf{m}}(\boldsymbol{q}), \boldsymbol{y}^{\text{obs}}\right)^2,$$

which does not involve any further expensive simulation from m. Loss Emulation (Section 2.3.2), instead, entails fitting a single real-valued local GP to the data $\mathcal{D} = \{(\boldsymbol{q}_i, \ell_{\mathsf{m}}(\boldsymbol{q}_i))\}_{i=1}^n$, and estimation is performed by minimizing the *surrogate or emulated loss*, i.e. the predictive mean of the GP: $\hat{\ell}_{\mathsf{m}}(\boldsymbol{q})$.

In this work, the surrogate-based loss and the emulated loss are optimized using the Global Search algorithm by Ugray et al. (2007), implemented in MATLAB's Global Optimization toolbox[5], with 2000 trial points and 400 stage one points. In order to describe the Global Search algorithm, we need to define the concept of basin of attraction first. Consider running a local solver from a given starting point $\boldsymbol{q}_0$, ending up at the point of local minimum $\hat{\boldsymbol{q}}$. The *basin of attraction* corresponding to that minimum is defined as the sphere[6] centred at $\hat{\boldsymbol{q}}$ and having radius equal to $\|\boldsymbol{q}_0 - \hat{\boldsymbol{q}}\|$. All starting points falling inside the sphere are assumed to lead to the same local minimum $\hat{\boldsymbol{q}}$, hence no local solver is run and they are discarded.

The Global Search algorithm requires as inputs: the function $f(\cdot)$ to be minimized, a starting point $\boldsymbol{q}_0$, and a set of constraint violation functions $c_i(\cdot)$, $i = 1, \ldots, m$. The first step involves running an interior-point local solver (Byrd et al., 2000) from the user-provided starting point $\boldsymbol{q}_0$, ending up at the minimizer $\hat{\boldsymbol{q}}_0$. The distance between the initial point and the minimizer is recorded in order to construct a basin

[5] `https://uk.mathworks.com/products/global-optimization.html`

[6] The basins of attraction are assumed to be spherical in the Global Search algorithm.

of attraction at $\hat{\boldsymbol{q}}_0$ with radius $\|\boldsymbol{q}_0 - \hat{\boldsymbol{q}}_0\|$. The algorithm also records the value of the score function

$$\text{Score}(\boldsymbol{q}) = f(\boldsymbol{q}) + \gamma \sum_{i=1}^{m} c_i(\boldsymbol{q}),$$

which is the sum between the objective function and a multiple of the sum of the constraint violations. This means that a feasible point, i.e. one that satisfies the constraints, must have $\text{Score}(\boldsymbol{q}) = f(\boldsymbol{q})$. The parameter γ is initially set to 1000 but it is updated during the algorithm. The next step involves generating a set of trial points using the scatter search algorithm (Glover, 1998). They represent potential start points. The score function is evaluated at a subset of the trial points called the "stage one" points. The point with the lowest score, $\boldsymbol{q}_1$, is taken as the next starting point. A local solver is run from this point, ending up at a new minimizer $\hat{\boldsymbol{q}}_1$ and leading to another basin of attraction. The "stage one" points are removed from the list of points to be examined, and the remaining trial points are called the "stage two" points. Global Search then examines each remaining trial point $\boldsymbol{q}$ from the list, running a local solver from $\boldsymbol{q}$ if:

1. $\|\boldsymbol{q} - \hat{\boldsymbol{q}}_i\| > \phi \times \|\boldsymbol{q}_i - \hat{\boldsymbol{q}}_i\|$, where i runs over the number of basins of attraction and $\phi = 0.75$ is a parameter of the algorithm. This can be interpreted as running a local solver from $\boldsymbol{q}$ if the point does not fall inside any existing basin of attraction.

2. $\text{Score}(\boldsymbol{q}) < \min\{\text{Score}(\hat{\boldsymbol{q}}_0), \text{Score}(\hat{\boldsymbol{q}}_1)\}$.

If a local solver is run from $\boldsymbol{q}$, ending at the minimizer $\hat{\boldsymbol{q}}$, then the solution is accepted if all distances $\|\hat{\boldsymbol{q}} - \hat{\boldsymbol{q}}_i\| > \text{InputTolerance}$ and $|f(\hat{\boldsymbol{q}}) - f(\hat{\boldsymbol{q}}_i)| > \text{OutputTolerance}$. If accepted, the new basin of attraction is added to the list of previously found basins of attraction. If, while examining the "stage two" points, a local solver is not run for many consecutive times, the algorithm makes sure to do more exploration by reducing the radii of the basins of attraction in order to run more local solvers.

5.3.2 Low-Rank GPs

Along with local GPs based on the K-nearest-neighbours, described in Section 5.3.1, we report results for another type of statistical approximation: low-rank GPs, as

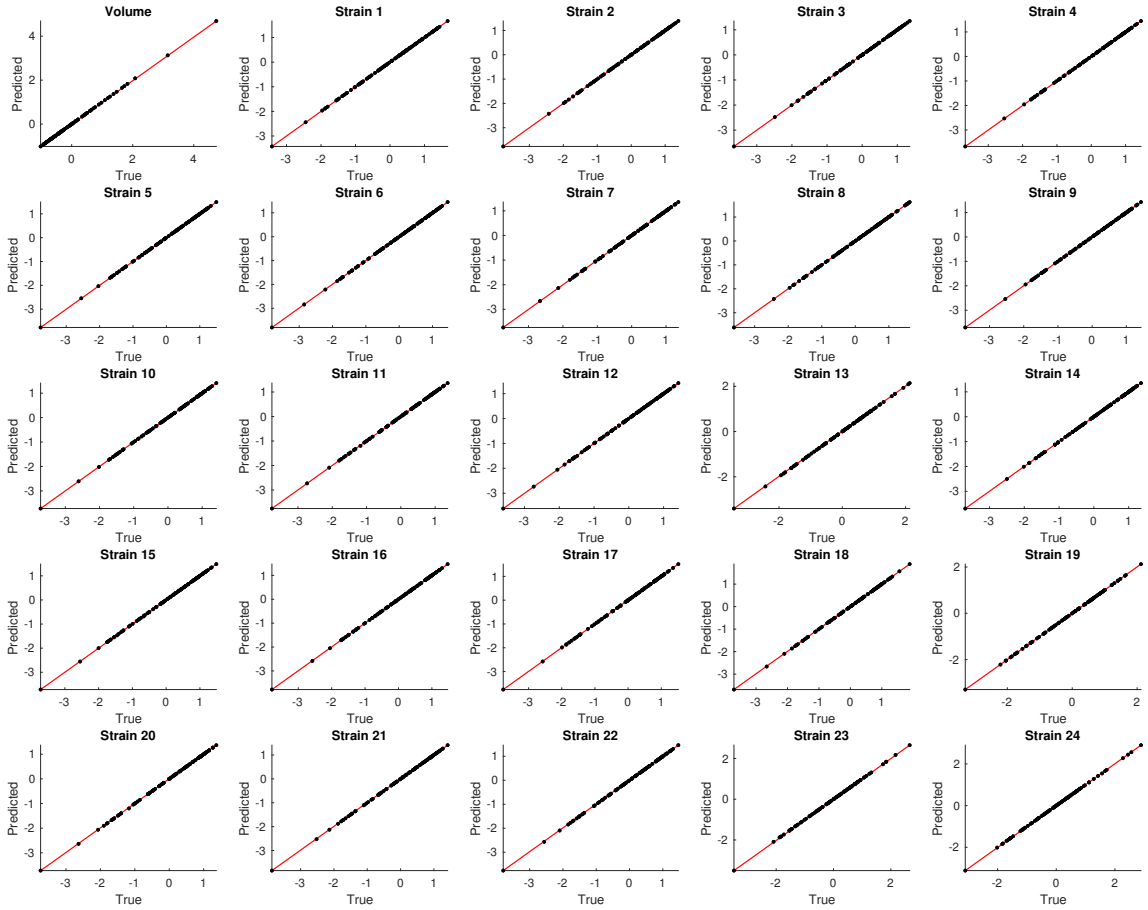


Figure 5.2: True vs predicted test outputs using Output Emulation with Local GPs and $K = 100$ nearest neighbours.

described in Section 5.8.2 of Wood (2017), whose main ideas are summarized here for generic training data $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\} = \{\boldsymbol{X}, \boldsymbol{y}\}$.

Let $\boldsymbol{C} = \boldsymbol{K} + \sigma^2 \boldsymbol{I}$ be the $n \times n$ covariance matrix of $\boldsymbol{y}$ and consider its eigen-decomposition $\boldsymbol{C} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^\top$ with eigenvalues $|D_{i,i}| \geq |D_{i+1,i+1}|$. Denote by $\boldsymbol{U}_k$ the submatrix consisting of the first k eigenvectors of $\boldsymbol{U}$, corresponding to the top k eigenvalues in $\boldsymbol{D}$. Similarly, $\boldsymbol{D}_k$ is the diagonal matrix containing all eigenvalues greater than or equal $D_{k,k}$. Wood (2017) considers replacing $\boldsymbol{C}$ with the rank k approximation $\boldsymbol{U}_k \boldsymbol{D}_k \boldsymbol{U}_k^\top$ obtained from the eigen-decomposition. Now, the main issue is how to find $\boldsymbol{U}_k$ and $\boldsymbol{D}_k$ efficiently enough. A full eigen-decomposition of $\boldsymbol{C}$ requires $O(n^3)$ operations, which somewhat limits the applicability of the rank-reduction approach. A solution is to use the Lanczos iteration method to find $\boldsymbol{U}_k$ and $\boldsymbol{D}_k$ at the substantially lower cost of $O(n^2 k)$ operations, see Section B.11 in Wood (2017). Briefly, the algorithm is an adaptation of power methods to obtain the truncated rank k eigen-decomposition of an $n \times n$ symmetric matrix in $O(n^2 k)$ operations. However, for large n, even $O(n^2 k)$ becomes prohibitive. In this scenario the training data are randomly subsampled by keeping n_r inputs and an eigen-decomposition is obtained for this random selection with $O(n_r^2 k)$ computational cost.

This algorithm has been independently run by Vinny Davies as part of a joint collaboration for a paper (Davies et al., 2018) of which I am joint first author. His results are reported in this chapter for a comparison only. He used the implementation found in the R package `mgcv` by Wood (2017), with the following settings: $n_r = 2000$ (the package default), $k = 2000$ for Output Emulation, while $k = 1000$ for Loss Emulation. The kernel used was an isotropic Matérn 3/2 kernel, with lengthscale set to the default of Kammann and Wand (2003): $\lambda = \max_{ij} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$. The remaining model hyperparameters are estimated by maximizing the log marginal likelihood.

Minimization of the surrogate-based loss $\ell_{\hat{\mathsf{m}}}(\cdot)$ and the emulated loss $\hat{\ell}_{\mathsf{m}}(\cdot)$ is performed by the Conjugate Gradient method implemented in the R function `optim` (Nash, 1990), with maximum number of iterations set to 100. To avoid being trapped in local minima, 50 different starting points from a Sobol sequence were used. The best minimum found was kept as the estimate, discarding the remaining 49 optima.

5.4 Comparison Results

The Local GP and Low Rank GP methods, described in Section 5.3.1 and Section 5.3.2, are applied to both Output Emulation (see Section 2.3.1) and Loss Emulation (see Section 2.3.2). We also compared different metrics $d(\boldsymbol{y}_i, \boldsymbol{y}_j)$: the Euclidean (2.3) and the Mahalanobis (2.5) distances. In summary, the competing methods are as follows:

M1 Output Emulation using Local GPs, and a Euclidean loss function;

M2 Output Emulation using Local GPs, and a Mahalanobis loss function;

M3 Euclidean Loss Emulation using Local GPs;

M4 Mahalanobis Loss Emulation using Local GPs;

M5 Output Emulation using Low-Rank GPs, and a Euclidean loss function;

M6 Output Emulation using Low-Rank GPs, and a Mahalanobis loss function;

M7 Euclidean Loss Emulation using Low-Rank GPs;

M8 Mahalanobis Loss Emulation using Low-Rank GPs;

where M1-M4 were run by myself and M5-M8 by Vinny Davies as part of a joint paper. Let $\mathcal{D} = \{(\boldsymbol{q}_i, \boldsymbol{y}_i)\}_{i=1}^{n}$ denote the $n = 10,000$ training runs and $\mathcal{D}_{\text{test}} = \{(\boldsymbol{q}_t, \boldsymbol{y}_t)\}_{t=n+1}^{n+m}$ denote the $m = 100$ test data which are not used to fit the GP models. For each test output $\boldsymbol{y}_t \in \mathcal{D}_{\text{test}}$ we estimate the corresponding parameter vector $\hat{\boldsymbol{q}}_t$ using the 8 methods summarized above. We now compare the estimated $\hat{\boldsymbol{q}}_t$ to the known test input $\boldsymbol{q}_t$ using the mean squared error (MSE) score[7]:

$$\text{MSE}_t = \frac{1}{d}\sum_{k=1}^{d}(\hat{q}_{tk} - q_{tk})^2,$$

obtaining a sample of 100 MSE_t scores for each method. Table 5.1 reports the median MSE, along with the 1st and 3rd quartiles, for the 8 different approaches,

[7]We can not compare the methods using the log likelihood score, as it is an inherent feature of this model to have an intractable and computationally expensive likelihood. The goal of the project is to avoid evaluating the likelihood as it takes approximately 15 minutes CPU time for a single evaluation.

Table 5.1: Comparison of the different emulation strategies. Median, 1st and 3rd quartiles of the mean squared error distribution for the out-of-sample points. In bold is highlighted the method with the lowest median MSE: Output Emulation using Local GP and the Euclidean metric.

Abbrev.	Statistical approximation	Emulation target	Distance	MSE Median (1st, 3rd) quartiles
M1	Local GP	Output	Euclidean	**0.0001 (0.0000,0.0003)**
M2	Local GP	Output	Mahalanobis	0.0009 (0.0003,0.0022)
M3	Local GP	Loss	Euclidean	0.2201 (0.0588,0.6777)
M4	Local GP	Loss	Mahalanobis	0.0013 (0.0002,0.0063)
M5	Low-Rank GP	Output	Euclidean	0.0048 (0.0012,0.0107)
M6	Low-Rank GP	Output	Mahalanobis	0.0030 (0.0011,0.0062)
M7	Low-Rank GP	Loss	Euclidean	0.6814 (0.2222,1.5234)
M8	Low-Rank GP	Loss	Mahalanobis	0.0113 (0.0041,0.0377)

and highlights in bold the best combination found. The table represents a summary of Figure 5.3 using three of Tukey's five numbers. Furthermore, we present the 1st and 3rd quartiles separately, as reporting plus or minus the IQR can lead to the misinterpretation of a negative MSE. The table shows that the best method involves emulating the simulator's output using a Local GP and then minimizing the surrogate-based Euclidean loss (M1). In Figure 5.3 we show the distribution of the 100 MSE_t scores for each method using boxplots. Panel (a) shows the original y-axis scale, while Panel (b) shows a reduces y-axis scale to focus on the lowest MSE scores.

From Table 5.1 we see that, for the same emulation target and distance, Local GPs always outperform Low-Rank GPs. For each statistical approximation strategy, we find that emulating the output always leads to a lower MSE than emulating the loss. Hence, by summarizing the 25D output into a scalar loss score we lose too much information. Furthermore, in all methods but the best, using a Mahalanobis distance leads to a lower MSE than using the Euclidean metric. Similarly, in Figure 5.3 we see that the MSE distributions in the first two boxplots (for Local GPs) are lower than boxplots 5-6 for the Low-Rank GP approach. This is also true for boxplots 3-4 vs

Table 5.2: The literature gold standard and the recovered 8D estimates from $\hat{\boldsymbol{q}}$.

	a	b	a_f	b_f	a_s	b_s	a_{fs}	b_{fs}
Literature	0.2245	1.6215	2.4267	1.8269	0.5562	0.7747	0.3905	1.6950
Estimated	0.2246	1.6224	2.4109	1.8414	0.5526	0.7809	0.4115	1.7860

7-8, where Local GP outperforms Low-Rank models. The fact that Low-Rank GPs do not perform as well as Local GPs could be due to a variety of reasons, such as the value of k chosen for the rank k eigen-approximation, the number n_r of subsampled training inputs and the sampled inputs too.

In summary: (1) the Local GP model outperforms the Low-Rank GP model and is therefore the better of the two types of statistical approximations; (2) emulating the output leads to a lower MSE than emulating the loss; (3) the best strategy involves emulating the simulator's output using Local GPs, and then minimizing the surrogate-based Euclidean loss function. This strategy is used in the next section to estimate the HO law parameters using MRI data of a healthy volunteer.

5.5 Application to Real Data

From Table 5.1 we found that the best strategy is represented by method 1: Output Emulation using Local GPs, followed by minimization of the surrogate-based Euclidean loss function (M1, in short).

In this section I apply M1 to estimate the parameter vector $\boldsymbol{q}$ for a healthy volunteer for which MRI data, LV chamber volume and circumferential strain measurements are available. With circumferential strains only, and not strain measurements in all directions, we can not recover all eight parameters, hence the reduced parametrization. The estimated parameter vector is given by $\hat{\boldsymbol{q}} = (1.0006, 0.9935, 1.0080, 1.0537)$. Since the components are close to one, if we recover the 8D parametrization by applying (5.2), we will be close to the literature gold standard method by Gao et al. (2015), see Table 5.2. The MSE between the literature gold standard and the estimated parameters using emulation is MSE $= 0.0012$.

The ultimate quantity of interest in biomechanics papers is the stress-strain

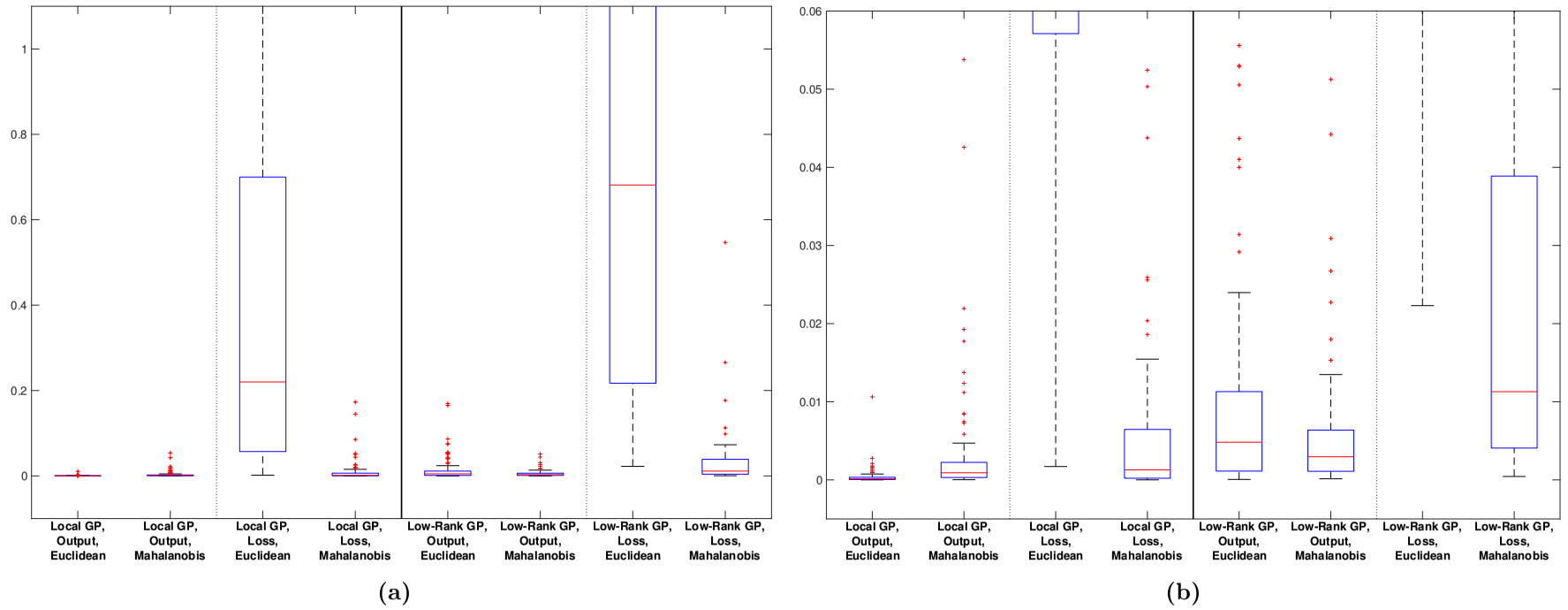
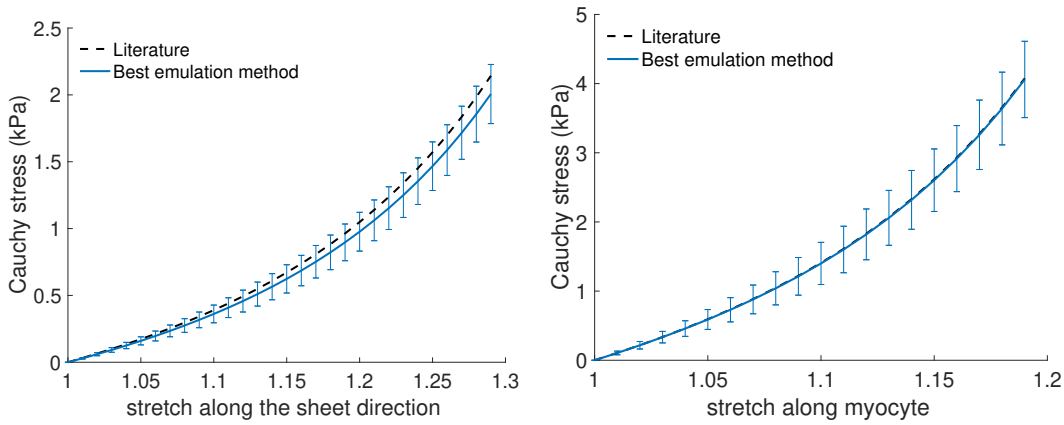


Figure 5.3: Distribution of the 100 out-of-sample MSE scores for each method. (a) Boxplots of the mean squared error for all 8 methods in the original y-axis scale and (b) with a reduced y-axis scale. The methods from left to right on each plot are as follows: Local GP emulation of the outputs with Euclidean loss function and Mahalanobis loss function, Local GP emulation of the Euclidean loss and the Mahalanobis loss, Low-Rank GP emulation of the outputs with Euclidean loss function and Mahalanobis loss function, and Low-Rank GP emulation of the Euclidean loss and the Mahalanobis loss. The outliers are due to convergence issues of the optimization algorithms and the highly correlated parameters of the HO law.



(a) **(b)**

Figure 5.4: Plots of the Cauchy stress against the stretch along (a) the sheet direction and (b) the myocyte. The current best estimate (i.e. the literature gold standard) from Gao et al. (2017) is reported as a dashed black line. Estimates of the curves using the best emulation approach (M1) are given as a blue solid line. The error bars show plus or minus one standard deviation, obtained by using the sampling methods described in Section 5.5.

curve, representing the constitute law of the material. Figure 5.4 shows in black the myofibre stress-stretch relationship for the healthy volunteer, where stretch is strain + 1, from the literature gold standard method by Gao et al. (2017). In their paper, the estimated parameters are obtained by direct minimization of the expensive full 8 parameters target loss, where a single evaluation takes 11 minutes CPU time[8], with convergence in a week time. The plot additionally shows, as a solid blue line, the estimated myofibre stress-stretch relationship corresponding to the parameter vector $\hat{\boldsymbol{q}}$ estimated using the best emulation strategy: Output Emulation with Local GPs, followed by minimization of the surrogate-based Euclidean loss function (M1). In order to obtain an indication of the uncertainty of our inference, we numerically estimated the Hessian $\boldsymbol{H}(\hat{\boldsymbol{q}})$ at the point of minimum of the surrogate-based loss, $\hat{\boldsymbol{q}}$. Its inverse represents a lower bound on the variance-covariance matrix. The 68% confidence interval can be obtained by sampling 1000 parameter vectors from

[8]Intel Xeon CPU, 2.9GHz, 32 cores and 32GB memory.

a $\mathsf{N}(\hat{\boldsymbol{q}}, \boldsymbol{H}(\hat{\boldsymbol{q}})^{-1})$ distribution[9]. To every sampled parameter vector corresponds a Cauchy stress-stretch curve. In Figure 5.4 we report the stress-stretch curve for $\hat{\boldsymbol{q}}$ (in blue) plus or minus the pointwise standard deviation of the sample of 1000 Cauchy stress-stretch curves.

From Figure 5.4 we can see that the emulation approach can accurately estimate the stress-stretch relationship for the healthy volunteer, with the gold standard lying inside the 68% confidence intervals. In particular, the Cauchy stress against the stretch along the myocyte, shown in Figure 5.4b, matches the gold standard almost exactly. It is also important to point out that the agreement between the stress-stretch curves from the literature gold standard method and the emulation approach is good even if the emulation has been carried out in a 4D subspace of the parameter space. This confirms the findings in Gao et al. (2015), where a good estimate of the curve can be obtained also with a reduced parametrization. We also notice that the uncertainty bands are not very tight, suggesting some sort of inflation in the uncertainty intervals due to the Gaussian approximation described above. To illustrate the concept in more detail, Figure 5.5 presents an illustration of the Laplace approximation. Panel (a) shows a right skewed density (in yellow) and its Laplace approximation (in red). If the true density is skewed, by approximating it with a Gaussian centred at the mode with precision matrix equal to the negative Hessian at the mode, we obtain a misleading approximation which can lead to an overestimation of the uncertainty at the point of maximum. Panel (b) presents the same concept but for the negative logarithm of the corresponding density from Panel (a).

The estimate $\hat{\boldsymbol{q}}$ giving rise to Figure 5.4 was obtained with a reduction in computational time of 3 orders of magnitude (1 week to 15 minutes). This makes emulation suitable for real-time decision support systems. When a patient comes into the clinic, it would be possible to estimate his or her myocardial stiffness in just 15 minutes. Then clinicians can make informed diagnoses on the basis of the

[9]The Laplace Approximation (LA) approximates at the mode, hence it's a local match. An alternative would be to use Variational Bayes methods or Expectation Propagation, which return global matches, but more analytical derivations are required. We prefer this simpler approach which is fast and returns an estimate of the uncertainty around the estimate.

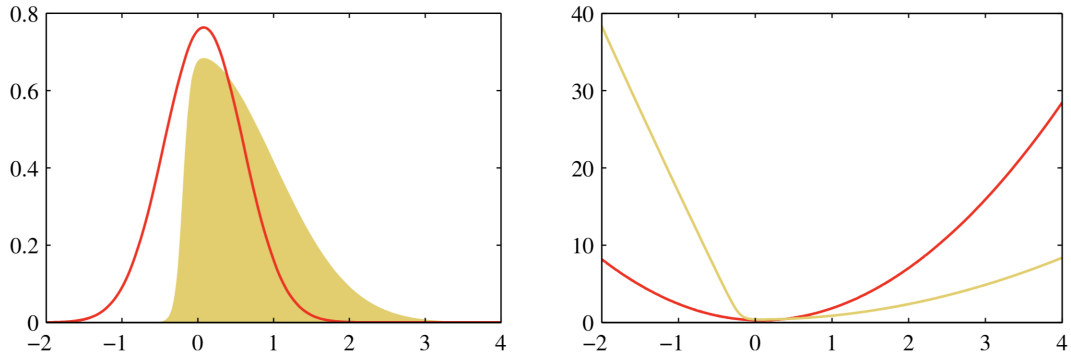


Figure 5.5: The Laplace Approximation. Left: the normalized density $p(z) \propto \exp(-z^2/2)\sigma(20z + 4)$ (in yellow), where $\sigma(z) = 1/(1 + \exp(-z))$, and the Laplace approximation centred at the mode (in red). Right: the negative logarithm of the corresponding densities from the left panel. *Source: Figure 4.14 of Bishop (2006).*

estimated myocardial properties. This is an important first step to open the path towards personalized diagnosis, prevention and informed treatment.

5.6 Summary

In this chapter we compared different emulation strategies by considering:

- different statistical approximations: Local GPs vs Low-Rank GPs;
- different emulation targets: Output Emulation vs Loss Emulation;
- different loss functions: Euclidean vs Mahalanobis.

We tested their accuracy in estimating the inputs for held-out test data for which the true inputs are known. The method incurring in the lowest MSE is given by emulating the output using Local GPs, and then minimizing a surrogate-based Euclidean loss, see Table 5.1.

The best strategy is then applied to real MRI data from a healthy volunteer, where the interest lies in estimating his myocardial properties (the parameters of the HO law). The estimated parameter vector $\hat{\boldsymbol{q}}$ can be used to obtain the final quantities of interest: the stress-strain curves along the sheet direction and the

myocyte. Comparing the estimated curves in Figure 5.4 with the current gold standard from the literature (Gao et al., 2015), we see a good agreement, with the gold standard being inside the 68% confidence interval. However, we suspect some sort of overestimation of the uncertainty bands because of the Gaussian approximation. The estimates in Gao et al. (2015) were obtained by direct minimization of the expensive target loss (2.1) using a multi-step approach, taking a week to converge. The emulation-based approach, instead, reaches almost identical solutions in only 15 minutes CPU time. This demonstrates the applicability of emulation-based methods to accelerate the estimation of indicators useful to support clinicians in their diagnostic work.

5.7 Future Work

One of the future goals for this project is to allow the emulator to estimate the material parameters of arbitrary patients having LV geometries on which the emulator has not been directly trained on. At the current stage, the computational model (simulator) requires the patient's left ventricular geometry as a fixed input, i.e. not to be optimized.

A possible solution is to simulate different datasets from the computational model for different LV geometries, given as inputs to the simulator. The next step involves finding a low-dimensional representation of each training geometry, which can then become an input to the emulator. Upon arrival in clinic, we can record the left ventricular geometry of the patient and then obtain its low-dimensional representation which can be included into the loss function.

Different approaches can be considered for reducing the dimensionality of the LV geometries, starting from principal component analysis (PCA) or independent component analysis (ICA) (Roberts and Everson, 2001). However, the limitation of PCA lies in the restriction to linear subspaces. Suppose that the training set of left ventricular geometries lies on a non-linear manifold of a higher dimensional space, then this approach would not be optimal. Many nonlinear extensions to PCA have been proposed in the literature, such as kernel PCA or autoencoder feed-

forward neural networks. In the latter, the LV geometries represent the input layer of the neural network, which are then passed through a bottleneck layer, and finally reconstructed at the output layer. Once trained, the bottleneck layer represents the low-dimensional representation of the left ventricular geometry.

Part II

Bayesian Optimization

Introduction to Part II

Part 1 focused on emulating the simulator's output or the inferential objective function. Both types of emulators are based on a set of training runs $\mathcal{D} = \{(\boldsymbol{q}_i, \boldsymbol{y}_i)\}_{i=1}^n$ for a given computational model $\boldsymbol{y} = \mathsf{m}(\boldsymbol{q})$. Hence, the emulator is model-specific. The training runs are obtained by simulating from the expensive computational model at the training inputs using massive parallelization. This is a substantial, a priori, time investment and is best done when the research group is confident that the model will not be changed for some time. Any update to the computational model would imply that the emulator is out-of-date and reflects a model which is no longer believed to be true.

When the simulator is undergoing further investigation and improvements, the substantial time invested in running the training simulations becomes wasted as soon as the version under development is released. The issue of model comparison is not the topic of this thesis which, instead, focuses on how to estimate the parameters of a chosen computationally expensive model. In order to do model selection with expensive simulators, a possibility involves emulating the posterior distribution and running Markov chain Monte Carlo on the predictive mean. The posterior samples can then be used to estimate the model marginal likelihood using, for example, Chib's method (Chib and Jeliazkov, 2001), bridge sampling (Meng and Wong, 1996) or thermodynamic integration (Friel and Pettitt, 2005; Grzegorczyk et al., 2017). The model with the highest marginal likelihood should be the one used for further inference.

Part 2 discusses how to estimate the inputs of an expensive simulator which is not deemed to be a "stable release" using Bayesian optimization (BO). In this scenario, we focus on emulating the objective function using a smaller set of training runs,

as a large number is not considered to be a wise time investment. However, the emulator of the objective function is *iteratively improved* by adding new training data using an adaptive strategy. While the goal is to use the algorithm to optimize a loss function for inference, $\ell(\boldsymbol{q})$, the algorithm is more general and can be used to optimize a generic expensive-to-evaluate function $f(\boldsymbol{x})$. For this reason, the BO algorithm is presented in its most generic form.

Chapter 6

Bayesian Optimization

Many real-life applications such as parameter estimation and decision making in science, engineering and economics require solving an optimization problem. Traditionally the aim has been to minimize a function which is fast to query at any given point and where the gradient information is readily available or easy to estimate. More recently, new research directions involve complex and multiscale computational models that do not meet these requirements. They typically are computationally expensive, perhaps without an exact functional form, the gradient information might not be available and outputs could be corrupted by noise. Examples of these applications include parameter estimation in robotics (Calandra et al., 2016; Lizotte et al., 2007), automatic tuning of machine learning algorithms (Hutter et al., 2011; Snoek et al., 2012; Wang et al., 2013b; Kotthoff et al., 2017), environmental monitoring and sensor placement (Garnett et al., 2010), soft tissue mechanical models of the pulmonary circulation (Noè et al., 2017) and more (Shahriari et al., 2016).

Bayesian optimization (BO) is a class of algorithms designed to solve these complex optimization tasks. It is not a recent field as it dates back to the 1970–1980s, when the Lithuanian mathematician Jonas Mockus published a series of papers and a book on the topic (Mockus, 1975, 1977, 1989), but it increased in popularity only recently due to the advances in computational resources.

With the increase in computational power, modellers started developing more complex *simulators* of real life phenomena. For example, by switching from linear to nonlinear differential equations, adding more layers of them, and interfacing many

micro-level models in order to recreate in silico macro-level phenomena. Simulators typically involve many tunable parameters. Setting these parameters by hand would be cumbersome, hence the need for an automated and principled framework to deal with them. In these models standard likelihood based inference is not straightforward due to the time required for a single forward simulation. This could involve, for example, the numerical solution of a system of nonlinear partial differential equations, hence calling for an iterative procedure. Furthermore, the maximum likelihood equations may not have an analytical solution and need to be solved iteratively, adding another level of computational complexity to the problem. Usually the likelihood landscape is highly multimodal, calling for multiple restarts, and effectively making the problem NP-hard.

This chapter reviews Bayesian optimization. Section 6.1 states the generic problem, while Section 6.2 recalls the main formulas of Gaussian processes, which were introduced in Chapter 3, that will be used throughout the chapter. Section 6.3 introduces Bayesian optimization and summarizes the popular classes of acquisition functions found in the literature, emphasizing the line of thought that led to their development. Finally, Section 6.4 demonstrates the BO algorithm on a one-dimensional objective function.

6.1 Problem Statement

Suppose that the task is to minimize globally a real-valued function $f(\boldsymbol{x})$, called *objective function,* over a compact domain $\mathcal{X} \subset \mathbb{R}^d$ and that observing $f(\boldsymbol{x})$ is costly due to the need to run long computer simulations or physical experiments. The global minimum is denoted $f_{\text{global}} = \min_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x})$, which is attained at $\boldsymbol{x}_{\text{global}}$. Here we will focus on the minimization problem as the conversion of a maximization problem into a minimization one is trivial. Many global optimization algorithms have been proposed in literature, e.g. genetic algorithms, multistart and simulated annealing methods (Locatelli and Schoen, 2013), but these algorithms require many function evaluations and hence are designed for functions that are cheap to query. Bayesian optimization (BO), instead, is an algorithm designed to optimize expensive-

to-evaluate functions by keeping the number of function evaluations as low as possible, hence saving computational time. To do so, BO uses all of the information collected so far (function values and corresponding locations) to internally maintain a model of the objective function. This is used to learn about the location of the minimum, and the model is continuously updated as new information arrives. The objective function f is approximated by a *surrogate model* or *emulator*, which is usually given a Gaussian process (GP) prior, see Rasmussen and Williams (2006). The values of the objective function are generally modelled according to the additive decomposition $y_i = f(\boldsymbol{x}_i) + \varepsilon_i$, where ε_i are i.i.d. $\mathsf{N}(0, \sigma^2)$ errors and $f \sim \mathsf{GP}(m, k)$ is the GP prior on the regression function.

6.2 Gaussian Processes Refresher

A random process $\{f(\boldsymbol{x}),\, \boldsymbol{x} \in \mathcal{X}\}$ is said to be Gaussian if and only if every finite dimensional distribution is a Gaussian random vector. Similarly to a multivariate Normal, parametrized by a mean vector and a covariance matrix, a Gaussian process is completely specified by a mean and a covariance function, denoted by $m(\boldsymbol{x})$ and $k(\boldsymbol{x}, \boldsymbol{x}')$ respectively. They return the mean $\mathbb{E}[f(\boldsymbol{x})] = m(\boldsymbol{x})$ and the covariance $\mathrm{Cov}(f(\boldsymbol{x}), f(\boldsymbol{x}')) = k(\boldsymbol{x}, \boldsymbol{x}')$ as function of the index only. A GP prior on the random function f is denoted $f(\boldsymbol{x}) \sim \mathsf{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$. We use a constant mean function: $m(\boldsymbol{x}) = c$, where c represents a constant to be estimated. The covariance functions considered are the ARD Matérn 5/2 kernel:

$$k_{5/2}(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \left(1 + \sqrt{5}r + \frac{5}{3}r^2\right) \exp\left(-\sqrt{5}r\right), \qquad r = \sqrt{\sum_{i=1}^{d} \frac{(x_i - x_i')^2}{\lambda_i^2}},$$

and the ARD Squared Exponential (SE) kernel:

$$k_{\mathrm{SE}}(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp\left\{-\frac{1}{2}\sum_{i=1}^{d} \frac{(x_i - x_i')^2}{\lambda_i^2}\right\},$$

see Section 3.6 for more details. The model hyperparameters, $\boldsymbol{\theta} = (c, \lambda_1, \ldots, \lambda_d, \sigma_f, \sigma)$, are estimated by maximizing the log marginal likelihood as discussed in Section 3.10. A GP with the ARD Squared Exponential kernel models infinitely differentiable

functions, while the ARD Matérn 5/2 kernel gives rise to sample paths which are only twice differentiable. Because of the computational complexity of the objective function, only a few training data are available. With very little information on the unknown function, it is hard to discover any kind of periodicity or unreasonable behaviour. To that end, we would need many training points. For this reason, the standard prior assumption in the literature is to use a SE kernel or the Matérn class (Jones et al., 1998; Kennedy and O'Hagan, 2001; Santner et al., 2003).

After having collected n data points, $\mathcal{D}_n = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, the predictive distribution of f is again a GP with mean $\hat{f}(\boldsymbol{x})$ and covariance $s(\boldsymbol{x}, \boldsymbol{x}')$:

$$\begin{aligned} f(\boldsymbol{x}) \mid \mathcal{D}_n &\sim \mathsf{GP}(\hat{f}(\boldsymbol{x}), s(\boldsymbol{x}, \boldsymbol{x}')) \\ \hat{f}(\boldsymbol{x}) &= m(\boldsymbol{x}) + \boldsymbol{k}(\boldsymbol{x})^\top [\boldsymbol{K} + \sigma^2 \boldsymbol{I}]^{-1} (\boldsymbol{y} - \boldsymbol{m}) \\ s(\boldsymbol{x}, \boldsymbol{x}') &= k(\boldsymbol{x}, \boldsymbol{x}') - \boldsymbol{k}(\boldsymbol{x})^\top [\boldsymbol{K} + \sigma^2 \boldsymbol{I}]^{-1} \boldsymbol{k}(\boldsymbol{x}'), \end{aligned} \tag{6.1}$$

where $\boldsymbol{K} = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1}^n$ is the $n \times n$ covariance matrix at the training inputs, $\boldsymbol{k}(\boldsymbol{x}) = [k(\boldsymbol{x}_1, \boldsymbol{x}), \ldots, k(\boldsymbol{x}_n, \boldsymbol{x})]^\top$ is the column vector of size $n \times 1$ containing the covariances of the process at each of the training inputs and the test point $\boldsymbol{x}$, while $\boldsymbol{m} = [m(\boldsymbol{x}_1), \ldots, m(\boldsymbol{x}_n)]^\top$ represents the mean at the training inputs. The posterior variance is readily obtained as $s^2(\boldsymbol{x}) = s(\boldsymbol{x}, \boldsymbol{x}) = \mathrm{Cov}(f(\boldsymbol{x}), f(\boldsymbol{x}))$.

It is worth remarking that we are not bound to GPs in order to build an emulator of the objective function. Shahriari et al. (2016) also discuss random forests (RFs), which scale better than GPs for large n. However, random forests underestimate the uncertainty, while GPs give an adequate representation of it, which is essential. This, together with the analytical formulas for the Gaussian process model, make the derivations needed for BO much easier. Furthermore, random forests lead to discontinuous and non-differentiable surrogate models, meaning that we could not make use of gradient based optimization algorithms.

6.3 Bayesian Optimization

The strength of BO lies in the following problem shift. Instead of directly optimizing the expensive objective function f, the optimization is performed on an inexpensive

Algorithm 6.1 Bayesian optimization.

Inputs:

Initial design: $\mathcal{D}_{n_{\text{init}}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n_{\text{init}}}$

Budget of $n_{\max}$ function evaluations

for $n = n_{\text{init}}$ **to** $n_{\max} - 1$ **do**

  Update the GP: $f(\boldsymbol{x}) \mid \mathcal{D}_n \sim \mathsf{GP}(\hat{f}(\boldsymbol{x}), s(\boldsymbol{x}, \boldsymbol{x}'))$

  Compute the acquisition function: $a_n(\boldsymbol{x})$

  Solve the auxiliary optimization problem: $\boldsymbol{x}_{\text{next}} = \arg\max_{\boldsymbol{x} \in \mathcal{X}} a_n(\boldsymbol{x})$

  Query f at $\boldsymbol{x}_{\text{next}}$ to obtain y_{next}

  Augment data: $\mathcal{D}_{n+1} = \mathcal{D}_n \cup \{\boldsymbol{x}_{\text{next}}, y_{\text{next}}\}$

end for

Return:

Estimated minimum: $f_{\min} = \min(y_1, \ldots, y_{n_{\max}})$

Estimated point of minimum: $\boldsymbol{x}_{\min} = \arg\min(y_1, \ldots, y_{n_{\max}})$

auxiliary function which uses the available information in order to recommend the next query point $\boldsymbol{x}_{\text{next}}$, hence it is referred to as *acquisition function*. Optimization algorithms propose a sequence $\boldsymbol{x}_n$ of points that aim to converge to a global optimum $\boldsymbol{x}_{\text{global}}$. In order to propose such a sequence, BO algorithms start by evaluating the objective function f at an initial design: $\mathcal{D}_{n_{\text{init}}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n_{\text{init}}}$. Jones et al. (1998) recommend to use a space filling Latin hypercube design of $n_{\text{init}} = 10 \times d$ points, with d being the dimensionality of the input space. Then iterate until the maximum number of function evaluations $n_{\max}$ is reached: (1) obtain the predictive distribution of f given $\mathcal{D}_n$; (2) use the distribution of f given $\mathcal{D}_n$ to compute the auxiliary function $a_n(\boldsymbol{x})$; (3) solve the auxiliary optimization problem $\boldsymbol{x}_{\text{next}} = \arg\max a_n(\boldsymbol{x})$; (4) query f at the recommended point $\boldsymbol{x}_{\text{next}}$ and update the training data: $\mathcal{D}_{n+1} = \mathcal{D}_n \cup \{\boldsymbol{x}_{\text{next}}, y_{\text{next}}\}$. For a pseudocode-style algorithm, see Algorithm 6.1.

Different BO algorithms vary in the choice of the acquisition function. These can be grouped into three main categories: *optimistic*, *improvement-based* and

information-based (Shahriari et al., 2016). Many acquisition functions can be seen, according to the Bayesian decision-theoretic framework, as an expected utility arising from the evaluation of f at $\boldsymbol{x}$. Then, we usually select the point leading to the highest expected utility. All acquisition functions try to balance to a different extent the concepts of *exploitation* and *exploration*. The former indicates evaluating where the emulator predicts a low function value, while the latter means reducing our uncertainty about the model of f by evaluating at points of high predictive variance.

Optimistic policies (class 1) handle exploration and exploitation by being optimistic in the face of uncertainty, in the sense of considering the best case scenario for a given probability value. The approach of Cox and John (1997) was to consider a *statistical lower bound* on the minimum, $\text{LCB}(\boldsymbol{x}) = -\{\hat{f}(\boldsymbol{x}) - \kappa s(\boldsymbol{x})\}$, where the minus sign in front is needed as the acquisition function is maximized. This acquisition function is known as the lower confidence bound (LCB) policy. Here, κ is a parameter managing the trade-off between *exploitation* and *exploration*. When $\kappa = 0$, the focus is on pure exploitation, i.e. evaluating where the GP model predicts low function values. On the contrary, a high value of κ emphasizes exploration by inflating the model uncertainty, i.e. recommending to evaluate at points of high predictive uncertainty. For this acquisition function there are strong theoretical results on achieving the optimal regret derived by Srinivas et al. (2012).

The next group (class 2) of acquisition functions are *improvement-based*. Define the current best function value at iteration n to be $f_{\min} = \min(y_1, \ldots, y_n)$[1], and recall that $f(\boldsymbol{x}) \mid \mathcal{D}_n \sim \mathsf{N}(\hat{f}(\boldsymbol{x}), s^2(\boldsymbol{x}))$ from the marginalization property of GPs. By standardization, $z(\boldsymbol{x}) = \{f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\}/s(\boldsymbol{x})$ has a standard normal distribution. This class of functions is based on the random variable *Improvement:*

$$I(\boldsymbol{x}) = \max\{f_{\min} - f(\boldsymbol{x}), 0\}. \tag{6.2}$$

Intuitively, $I(\boldsymbol{x})$ assigns a reward of $f_{\min} - f(\boldsymbol{x})$ if $f(\boldsymbol{x}) < f_{\min}$, and zero otherwise. Kushner (1964) proposed to select the point that has the highest probability of improving upon the current best function value $f_{\min}$. This effectively corresponds to maximizing the probability of the event $\{I(\boldsymbol{x}) > 0\}$ or, equivalently, of $\{f(\boldsymbol{x}) < f_{\min}\}$.

[1]If the function values are corrupted by noise, $f_{\min} = \min \hat{f}(\boldsymbol{x})$.

Lemma 6.1. *The* Probability of Improvement *(PI) acquisition function is:*

$$\mathrm{PI}(\boldsymbol{x}) = \Phi\left(\frac{f_{\min} - \hat{f}(\boldsymbol{x})}{s(\boldsymbol{x})}\right). \tag{6.3}$$

Proof. From $f(\boldsymbol{x}) \mid \mathcal{D}_n \sim \mathsf{N}(\hat{f}(\boldsymbol{x}), s^2(\boldsymbol{x}))$ follows that:

$$\begin{aligned}
\mathrm{PI}(\boldsymbol{x}) &= \mathbb{P}\{I(\boldsymbol{x}) > 0\} \\
&= \mathbb{E}1_{\{f(\boldsymbol{x}) < f_{\min}\}} \\
&= \mathbb{P}\{f(\boldsymbol{x}) < f_{\min}\} \\
&= \mathbb{P}\left\{\frac{f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})}{s(\boldsymbol{x})} < \frac{f_{\min} - \hat{f}(\boldsymbol{x})}{s(\boldsymbol{x})}\right\} \\
&= \int_{-\infty}^{\frac{f_{\min} - \hat{f}(\boldsymbol{x})}{s(\boldsymbol{x})}} \phi(z)\mathrm{d}z \\
&= \Phi\left(\frac{f_{\min} - \hat{f}(\boldsymbol{x})}{s(\boldsymbol{x})}\right).
\end{aligned}$$

□

In the following, $\phi(x \mid \mu, \sigma^2)$ and $\Phi(x \mid \mu, \sigma^2)$ denote the probability density function (pdf) and the cumulative distribution function (cdf) of a $\mathsf{N}(\mu, \sigma^2)$ random variable. For brevity, when $\mu = 0$ and $\sigma^2 = 1$ we will simply write $\phi(x)$ and $\Phi(x)$. The PI acquisition function corresponds to the expectation of the utility $\mathfrak{u}(\boldsymbol{x}) = 1_{\{f(\boldsymbol{x}) < f_{\min}\}}$, which is $\mathfrak{u}(\boldsymbol{x}) = 1$ when $f(\boldsymbol{x}) < f_{\min}$ and 0 otherwise. In other words, the utility assigns a reward of 1 when we have an improvement, irrespective of the magnitude of this improvement, and 0 otherwise. It might seem naive to assign a reward always equal to 1 every time we improve on $f_{\min}$, irrespectively of the value. An acquisition function that accounts for the magnitude of the improvement is obtained by averaging over the utility $\mathfrak{u}(\boldsymbol{x}) = f_{\min} - f(\boldsymbol{x})$ when $f(\boldsymbol{x}) < f_{\min}$ and 0 otherwise, hence $\mathfrak{u}(\boldsymbol{x}) = I(\boldsymbol{x})$. Define $u = \{f_{\min} - \hat{f}(\boldsymbol{x})\}/s(\boldsymbol{x})$.

Lemma 6.2. *The* Expected Improvement *(EI) acquisition function (Mockus et al., 1978; Jones et al., 1998) corresponds to the expectation of the random variable $I(\boldsymbol{x})$ and is equal to:*

$$\begin{aligned}
\mathrm{EI}(\boldsymbol{x}) &= \mathbb{E}\{I(\boldsymbol{x})\} \\
&= \{f_{\min} - \hat{f}(\boldsymbol{x})\}\Phi(u) + s(\boldsymbol{x})\phi(u).
\end{aligned} \tag{6.4}$$

Proof. Using property (C.1) from Appendix C:

$$
\begin{aligned}
\mathrm{EI}(\boldsymbol{x}) &= \mathbb{E}[I(\boldsymbol{x})] \\
&= \mathbb{E}[\max\{f_{\min} - f(\boldsymbol{x}), 0\}] \\
&= \mathbb{E}\left[\{f_{\min} - f(\boldsymbol{x})\} 1_{\{f(\boldsymbol{x}) < f_{\min}\}}\right] \\
&= \int_{-\infty}^{\infty} \{f_{\min} - y\} 1_{\{y < f_{\min}\}} \phi(y \mid \hat{f}(\boldsymbol{x}), s^2(\boldsymbol{x})) \mathrm{d}y \\
&= \int_{-\infty}^{f_{\min}} \{f_{\min} - y\} \phi(y \mid \hat{f}(\boldsymbol{x}), s^2(\boldsymbol{x})) \mathrm{d}y \\
&= \int_{-\infty}^{u} \{f_{\min} - (\hat{f}(\boldsymbol{x}) + s(\boldsymbol{x})z)\} \phi(z) \mathrm{d}z \\
&= \int_{-\infty}^{u} \{f_{\min} - \hat{f}(\boldsymbol{x}) - s(\boldsymbol{x})z\} \phi(z) \mathrm{d}z \\
&= \{f_{\min} - \hat{f}(\boldsymbol{x})\} \int_{-\infty}^{u} \phi(z) \mathrm{d}z - s(\boldsymbol{x}) \int_{-\infty}^{u} z\phi(z) \mathrm{d}z \\
&= \{f_{\min} - \hat{f}(\boldsymbol{x})\} \Phi(u) + s(\boldsymbol{x}) \phi(u) \\
&= s(\boldsymbol{x}) \left\{u\Phi(u) + \phi(u)\right\}.
\end{aligned}
$$

□

This policy recommends to query at the point where we expect the highest improvement score over the current best function value. The EI is made up of two terms. The first term is increased by decreasing the predictive mean $\hat{f}(\boldsymbol{x})$, the second term is increased by increasing the predictive uncertainty $s(\boldsymbol{x})$. This shows how EI *automatically* balances *exploitation* and *exploration*.

Recent interest has focused on *information-based* acquisition functions (class 3). Here, the core idea is to query at points that can help us learn more about the location of the unknown minimum rather than points where we expect to obtain low function values. The main representatives of this class are Entropy Search (ES) (Hennig and Schuler, 2012), Predictive Entropy Search (PES) (Hernández-Lobato et al., 2014) and, more recently, Max-Value Entropy Search (MES) (Wang and Jegelka, 2017). Both ES and PES focus on the distribution of the argmin, $p(\boldsymbol{x}_{\text{global}} \mid \mathcal{D}_n)$, which is induced by the GP prior on f. These two policies recommend to query at the point $\boldsymbol{x}_{\text{next}}$ leading to the largest reduction in uncertainty about the distribution $p(\boldsymbol{x}_{\text{global}} \mid \mathcal{D}_n)$. This can be expressed as selecting the point $\{\boldsymbol{x}, y\}$ conveying the most information

about $\boldsymbol{x}_{\text{global}}$ in terms of the mutual information $\mathbb{I}(\{\boldsymbol{x}, y\}, \boldsymbol{x}_{\text{global}} \mid \mathcal{D}_n)$. The Entropy Search acquisition function is

$$\text{ES}(\boldsymbol{x}) = H[\boldsymbol{x}_{\text{global}} \mid \mathcal{D}_n] - \mathbb{E}\{H[\boldsymbol{x}_{\text{global}} \mid \mathcal{D}_n, \boldsymbol{x}, y]\},$$

where H is the entropy and the expectation is taken with respect to the density $p(y \mid \mathcal{D}_n, \boldsymbol{x})$. PES, instead, uses the symmetry of mutual information in order to obtain the equivalent formulation:

$$\text{PES}(\boldsymbol{x}) = H[y \mid \mathcal{D}_n, \boldsymbol{x}] - \mathbb{E}\{H[y \mid \mathcal{D}_n, \boldsymbol{x}, \boldsymbol{x}_{\text{global}}]\},$$

where the expectation is with respect to $p(\boldsymbol{x}_{\text{global}} \mid \mathcal{D}_n)$. This distribution is analytically intractable, and so is its entropy, hence calling for approximations based on a discretization of the input space, which incurs a loss of accuracy, and Monte Carlo sampling, which is computationally expensive. Furthermore, the point at which the global minimum is attained might not be unique. Instead of measuring the information about $\boldsymbol{x}_{\text{global}}$, which lies in a multidimensional space $\mathcal{X}$, Wang and Jegelka (2017) propose to focus on the simpler gain in information between y and the minimum value f_{global}, which lies in a one-dimensional space. The acquisition function hence becomes

$$\begin{aligned}\text{MES}(\boldsymbol{x}) &= \mathbb{I}(\{\boldsymbol{x}, y\}, f_{\text{global}} \mid \mathcal{D}_n) \\ &= H[y \mid \mathcal{D}_n, \boldsymbol{x}] - \mathbb{E}\{H[y \mid \mathcal{D}_n, \boldsymbol{x}, f_{\text{global}}]\},\end{aligned}$$

with expectation with respect to $p(f_{\text{global}} \mid \mathcal{D}_n)$. The expectation is approximated with Monte Carlo estimation by sampling a set of function minima. In summary the methods in this class involve (1) hyperparameters sampling for marginalization and (2) sampling global optima for entropy estimation. Step 2 substantially increases the computational cost of information-based acquisition functions, especially in the case of ES and PES, which sample in a multidimensional space.

For global convergence proofs of Bayesian optimization see the work of Bull (2011), which shows that under a fixed prior the Expected Improvement converges on the minimum of any function and also presents convergence rates. In practice, it is common to sequentially update the GP prior as new information arrives. However,

this might not converge in some cases, and Bull (2011) also demonstrates how to obtain estimators that reach the same convergence rates.

Li et al. (2016) show that, on a comprehensive benchmark including 117 datasets, Bayesian optimization is often outperformed by random search with twice as many iterations. Ahmed et al. (2016) also state that "for some specific problems BO does in fact handily beat random and we know that under certain smoothness assumptions BO can be exponentially faster than random, see Theorem 5 in Bull (2011)". To that end, Ahmed et al. (2016) present a first "harmless" BO algorithm that aims to always be no worse than random search. Furthermore, for situations where BO already outperforms random search, they show how to use gradient information in order to gain an even faster convergence.

6.4 Illustration

This section illustrates the Bayesian optimization algorithm with the EI acquisition function and the ARD Squared Exponential kernel, as commonly used in the literature (Jones et al., 1998; Santner et al., 2003).

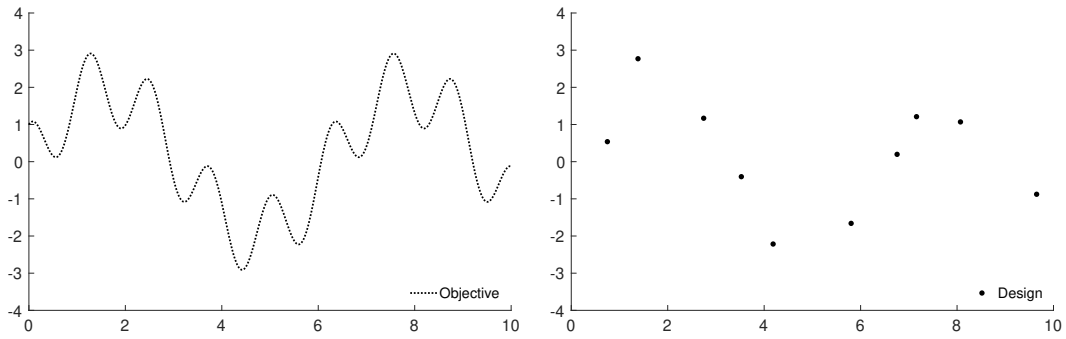
In this thesis, the maximization of the acquisition function is performed by evaluating it at 10^4 uniform random points in the input domain. The inputs are then ranked by their acquisition function values, and the 10 points having the highest score are found. A Nelder-Mead (Lagarias et al., 1998) local solver is run starting from each of these 10 points, until each solver reaches a relative tolerance on the function value of 10^{-3}. Among the 10 returned maximizers, the next evaluation point $\boldsymbol{x}_{\text{next}}$ is the one having the maximum acquisition function value, while the remaining 9 points are discarded.

Using Bayesian optimization, we minimize the "Cosine Sine" (CSF) objective function, defined as

$$f(x) = \cos(5x) + 2\sin(x), \tag{6.5}$$

over the compact space $\mathcal{X} = [0, 10]$ with a budget of $n_{\max} = 20$ function evaluations of which $n_{\text{init}} = 10$ are used for the initial design.

The objective function is shown in Figure 6.1a along with the derived initial



(a) Objective function **(b)** Initial design with $n_{\text{init}} = 10$

Figure 6.1: The objective function and the derived initial design.

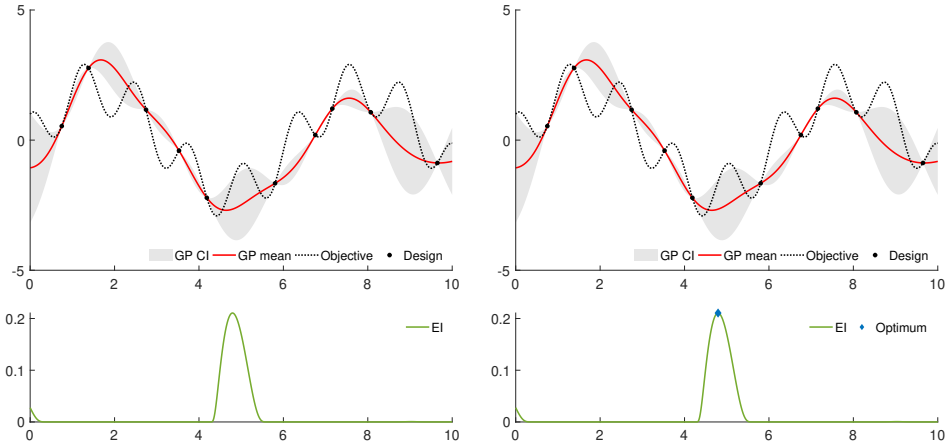
design comprising 10 points in Figure 6.1b. Given the initial design $\mathcal{D}_{n_{\text{init}}}$ and the budget of function evaluations $n_{\max}$, the BO algorithm proceeds by first fitting the GP emulator of f, shown in Figure 6.2a, given the available data (black dots). The red line represents the GP predictive mean, while the shaded grey area is the 95% confidence interval. The dotted black function is the objective function $f(x)$, which is usually not plottable due to the cost of each single pointwise evaluation. From the GP model of f we derive the EI acquisition function, shown as a green line in the bottom panel of Figure 6.2a. The acquisition function is then maximized to find the next query point x_{next}, shown in Figure 6.2b as a blue diamond. The costly objective function f is queried at the point x_{next} to obtain y_{next} and the training data are updated: $\mathcal{D}_{11} = \mathcal{D}_{10} \cup \{x_{\text{next}}, y_{\text{next}}\}$. The new GP model given $\mathcal{D}_{11}$ is shown in Figure 6.2c, along with the derived EI function and the next query point x_{next}. The algorithm proceeds along the same steps, giving rise to Figure 6.2d for 12 function evaluations, Figure 6.2e for 13 queries, and finally terminating at the budget $n_{\max} = 20$ as in Figure 6.2f. Note that in Figure 6.2c the EI policy recommends to evaluate at a point of high predictive variance, in order to improve the emulator.

Figure 6.3 shows the objective minimum trace, i.e. the incumbent minimum $f_{\min} = \min(y_1, \ldots, y_n)$ vs the iteration number n. Note that the true *global minimum* (shown as a dashed red line) is found in less than 20 function evaluations, even if the function has many local optima. On the contrary, traditional conjugate gradient methods usually get stuck in one of the local minima depending on the initial point and typically require a higher number of function evaluations. This issue is discussed

in more detail in Section 8.5, where the performance of BO is compared to standard global optimization algorithms.

6.5 Summary

This chapter presented a review of Bayesian optimization, an algorithm to minimize expensive-to-evaluate objective functions. The objective functions are typically black-box functions, i.e. not available in analytical form and without gradient information. After discussing the generic problem, I reviewed the main formulas from GPs, and defined the mean and covariance functions considered for Bayesian optimization. Then I discussed the generic BO algorithm, which relies on the choice of an acquisition function. Different acquisition functions define different point selection policies. I presented the three main classes of policies found in the literature: optimistic (class 1), improvement-based (class 2) and information-based (class 3). Finally, I illustrated BO on a simple 1D objective function, where the iterations can be plotted.



(a) GP emulator and derived EI **(b)** $n = 10$

(c) $n = 11$ **(d)** $n = 12$

(e) $n = 13$ **(f)** $n = 20$

Figure 6.2: Illustration of the BO algorithm using the EI acquisition function.

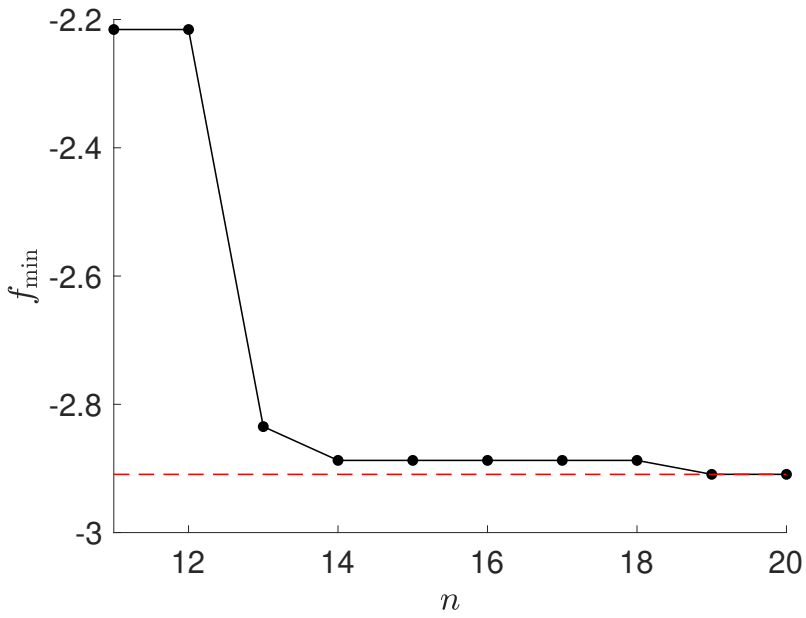


Figure 6.3: Objective minimum trace for the CSF function. The incumbent minimum $f_{\min}$ vs the iteration number n. The true global minimum f_{global} is shown as a dashed red line.

Chapter 7

Application to In Silico Medicine

Chapter 6 discussed Bayesian optimization, a method to optimize a costly objective function by using a limited number of function queries, hence a reduced computational time to solve the minimization problem. This chapter presents an application of BO to precision medicine, where the interest lies in the estimation of parameters of a partial differential equations (PDEs) model of the human pulmonary blood circulation system. Once inferred, these parameters can help clinicians in diagnosing a patient with pulmonary hypertension without going through the standard invasive procedure of right heart catheterization, which can lead to side effects and complications (e.g. severe pain, internal bleeding, thrombosis).

Section 7.1 introduces the problem and outlines our goals, while Section 7.2 describes the considered PDE model of the human pulmonary circulation. Section 7.3 discusses how to extend the BO algorithm to handle simulation failures for some parameter settings, due to violations of the assumptions in the mathematical model. The BO algorithm with the EI acquisition function is then used in Section 7.4 to infer the parameters of the pulmonary circulation PDE model, with the ultimate goal to pave the way towards autonomous in silico diagnosis and prognosis.

Notes This chapter is adapted from: Noè, U., Chen, W., Filippone, M., Hill, N., and Husmeier, D. (2017). Inference in a Partial Differential Equations Model of Pulmonary Arterial and Venous Blood Circulation Using Statistical Emulation. In Bracciali, A., Caravagna, G., Gilbert, D., and Tagliaferri, R., editors, *Computational*

Intelligence Methods for Bioinformatics and Biostatistics. CIBB 2016. Lecture Notes in Computer Science, volume 10477, pages 184–198. Springer, Cham, Switzerland.

7.1 Motivation

Chronic pulmonary arterial hypertension (PH), i.e. high blood pressure in the pulmonary circulation, is often referred to as a "silent killer" and is a disease of the small pulmonary arteries. It can lead to irreversible changes in the pulmonary vascular structure and function, increased pulmonary vascular resistance, and right ventricle hypertrophy leading to right heart failure (Allen et al., 2014; Rosenkranz and Preston, 2015).

For diagnosis and ongoing treatment and assessment, clinicians measure blood flow and pressure within the pulmonary arteries. As opposed to blood pressure in the systemic circulation, measured using a sphygmomanometer, blood pressure in the pulmonary circulation can only be measured using invasive techniques such as right heart catheterization. Invasive techniques can lead to complications (internal bleeding, severe pain, thrombosis, etc.), for that reason, it is desirable to predict the blood pressure indirectly based on quantities that can be measured non-invasively. Furthermore, data about healthy patients are not available due to ethical reasons. This chapter uses a partial differential equations (PDEs) model of the pressure and flow wave propagation in the pulmonary circulation under normal physiological and pathological conditions, introduced by Qureshi et al. (2014) and also studied by Noè et al. (2017). The goal is to use the Bayesian optimization algorithm with the EI acquisition function, see (6.4), and the pulmonary circulation model cited above to infer indicators of pulmonary hypertension risk which could be used by clinicians to inform their diagnosis instead of taking invasive measurements.

The PDEs depend on various physiological parameters, related e.g. to blood vessel geometry, vessel stiffness and fluid dynamics. These parameters, which would give important insights into the status of a patient's pulmonary circulatory system, can typically not be measured in vivo and hence need to be inferred indirectly from the observed blood flow and pressure distributions. In principle, this is straightforward.

Under the assumption of a suitable noise model, the solutions of the PDEs define the likelihood of the data, and the parameters can then be inferred in a maximum likelihood sense. However, a closed-form solution of the maximum likelihood equations is not available, which calls for an iterative optimization procedure. Since a closed-form solution of the PDEs is not available either, each optimization step requires a numerical solution of the PDEs. This is computationally expensive, especially given that the likelihood function is typically multi-modal, and the optimization problem is NP-hard. Minimization of the residual sum of squares (negative log likelihood) hence calls for Bayesian optimization in order to reduce the computational costs of the inference. The estimated parameters of the PDEs will give clinicians insights into the patient-specific vessel structure that would not be obtainable in vivo such as vessel stiffnesses, a primary indicator of hypertension.

7.2 The Pulmonary Circulation Model

In the model of the pulmonary circulation by Qureshi et al. (2014), seven large arteries and four large veins are modelled explicitly, while the smaller vessels are represented by structured trees (Figure 7.1). A magnetic resonance imaging (MRI) based measurement of the right ventricular output provides the inlet flow for the system.

The large arteries and veins are modelled as tapered elastic tubes, and the geometries are based on measurements of proximal and distal radii and vessel lengths. The cross-sectional area averaged blood flow and pressure are predicted from a non-linear model based on the incompressible Navier–Stokes equations for a Newtonian fluid. The small arteries and veins are modelled as structured trees at each end of the terminal large arteries and veins to mimic the dynamics in the vascular beds. With a given parent vessel radius r_p, the daughter vessels are scaled linearly with radii $r_{d_1} = \alpha r_p$ and $r_{d_2} = \beta r_p$, where α and β are the scaling factors. The vessels bifurcate until the radius of each terminal vessel is smaller than a given minimum $r_{\min}$. The radius relation at bifurcations is:

$$r_p^\xi = r_{d_1}^\xi + r_{d_2}^\xi, \quad 2.33 \leq \xi \leq 3.0, \tag{7.1}$$

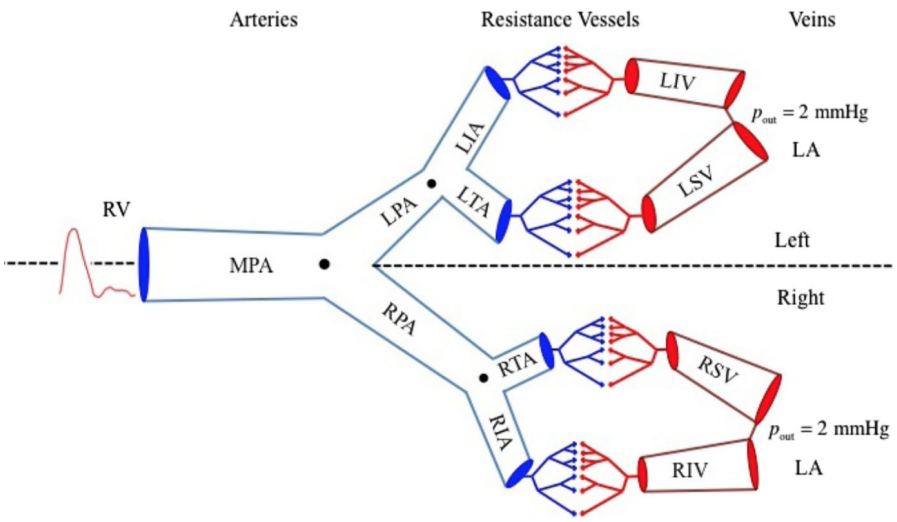


Figure 7.1: Schematic of the pulmonary circulation consisting of large arteries, arterioles, venules and large veins from Qureshi et al. (2014). Seven large arteries are considered in this model, i.e. the main pulmonary artery (MPA), the left (LPA) and right (RPA) pulmonary arteries, the left interlobular artery (LIA), the left trunk artery (LTA), the right interlobular artery (RIA), and the right trunk artery (RTA). The four terminal arteries LIA, LTA, RIA, and RTA are connected to four large veins, i.e. the left inferior vein (LIV), left superior vein (LSV), right inferior vein (RIV), and right superior vein (RSV), via structured trees of resistance vessels.

where the exponent $\xi = 2.33$ corresponds to laminar flow, $\xi = 3.0$ corresponds to turbulent flow (Olufsen, 1999), p represents the parent vessel, and d_1 and d_2 represent the daughter vessels. Given the area ratio $\eta = (r_{d_1}^2 + r_{d_2}^2)/r_p^2$ and the asymmetry ratio $\gamma = (r_{d_2}/r_{d_1})^2$, the scaling factors α and β satisfy $\alpha = (1 + \gamma^{\xi/2})^{-1/\xi}$ and $\beta = \alpha\sqrt{\gamma}$. The parameters, ξ, γ, $r_{\min}$ and a given root radius r_0, determine the size and density of the structured tree. The cross-sectional area averaged blood flow and pressure in these small arteries and veins are computed from the linearized incompressible axisymmetric Navier–Stokes equations (Qureshi et al., 2014).

For each large vessel, the pressure and flow are modelled as the solution of the one dimensional Navier–Stokes equation (Olufsen et al., 2012). It comprises two equations which ensure *conservation of volume and momentum,* and a third equation of state,

linking *pressure and cross-sectional area*. Let x denote the distance along a given vessel, t represent time, $p(x,t)$ the pressure, $q(x,t)$ the volumetric flow along any given vessel, $A(x,t)$ the corresponding cross-sectional area, ρ a constant representing the density of the blood, ν a constant representing kinematic viscosity, δ (constant) the boundary layer thickness, and $r(x,t)$ the radius of the given vessel. Conservation of volume and momentum is satisfied by:

$$\frac{\partial q}{\partial x}+\frac{\partial A}{\partial t}=0, \qquad \frac{\partial q}{\partial t}+\frac{\partial}{\partial x}\left(\frac{q^2}{A}\right)+\frac{A}{\rho}\frac{\partial p}{\partial x}=-\frac{2\pi\nu r}{\delta}\frac{q}{A}. \tag{7.2}$$

The constitutive law linking pressure and cross sectional area is given by:

$$p(x,t)-p_0=\frac{4}{3}\frac{Eh}{r_0}\left(1-\sqrt{\frac{A_0}{A}}\right), \tag{7.3}$$

where p_0 denotes the external pressure, E is Young's modulus, h the vessel wall thickness and r_0 the vessel radius when $p(x,t)=p_0$. The unstressed vessel area is obtained as $A_0=\pi r_0^2$. The term Eh/r_0 in (7.3) describes the elastic properties of a vessel's wall, and hence represents a parameter that controls the system compliance. This will be simply denoted by f_{L} in the large vessels and by f_{S} in the small vessels.

In the small vessels, similarly to the large ones, three equations determine the flow, pressure and area of each vessel in the structured tree. Olufsen et al. (2012) however, notice that in small vessels the nonlinear effects are small, effectively allowing for linearization of the constitutive equations. The full system of PDEs is presented in Qureshi et al. (2014), and its numerical solution, which depends on various physiological parameters, will henceforth be referred to as *simulation*. Figure 7.2 shows the simulated pressure (left) and flow (right) curves over time at three different locations in the main pulmonary artery (MPA).

Particular interest lies in the estimation of the parameter ξ, because low values are indicative of the clinically relevant problem of *vascular rarefaction*, which is a well-known finding in patients suffering from pulmonary hypertension, and represents the condition of having fewer blood vessels per tissue volume (Feihl et al., 2006). Estimation of ξ is performed in the range $2.33 \leq \xi \leq 3$, as given in (7.1). Other relevant parameters of interest for clinical diagnosis are the stiffness parameters in the large and small vessels, f_{L} and f_{S} respectively, with bounds $f_{\mathrm{L}} \in [1.33\times10^5, 5.33\times10^5]$

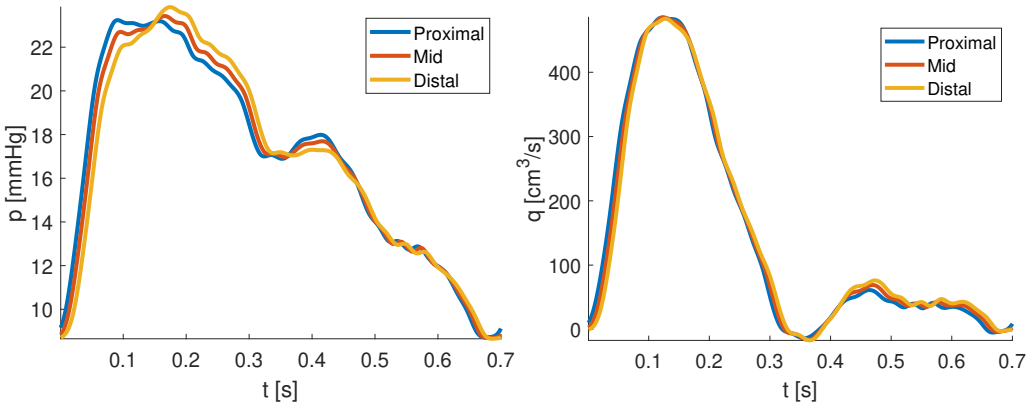


Figure 7.2: Simulated pressure (left) and flow (right) over time, at three different locations along the main pulmonary artery (MPA).

and $f_S \in [2.66 \times 10^4, 1.066 \times 10^5]$ as in Noè et al. (2017). The bounds have been obtained from joint discussions with clinicians and the mathematical model developers. Increased vessel stiffness is a major cause of pulmonary hypertension. During systole, a compliant artery expands to accommodate for the inflow, while it recoils during diastole to promote forward flow. As the capacity of an artery is limited, the pressure increases during systole and is partially maintained during diastole by the rebounding of the expanded arterial walls. When the stiffness is increased, the cushioning function of the vessel is compromised, leading to a higher systolic and a lower diastolic pressure. We focus on the estimation of the main clinically relevant parameters $\boldsymbol{q} = (f_L, f_S, \xi)$, while all remaining model parameters are fixed to biologically relevant values from the literature as in Qureshi et al. (2014).

7.3 Bayesian Optimization with Hidden Constraints

Mathematical models often rely on simplifying assumptions about the underlying system. When these assumptions do not hold, the model can return a failure instead of a numerical simulation. In the considered model of the human pulmonary circulation, for some specific settings of the PDE parameters this is indeed the case. However, the regions in the parameters space that lead to failures are unknown a priori. A problem is said to contain *hidden constraints* if a requested function value may turn out not to be obtainable. This is different from the case where an output value is

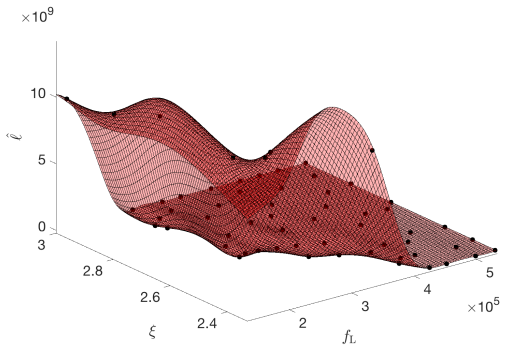
obtained but deemed not valid.

7.3.1 Assigning a High Objective Function Score

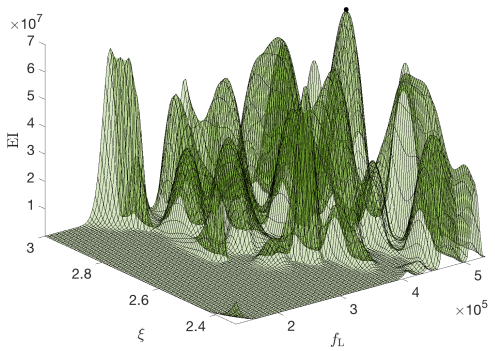
In principle, one could assign an arbitrary high objective function score to any failed simulation. In our case, the objective function corresponds to the loss or residual sum of squares $\ell(\boldsymbol{q})$, see (2.1), and a high loss indicates a low likelihood area. We can however show that handling hidden constraints using this naive method inherently misspecifies the GP surrogate of the RSS objective, leading to unrealistic and biased estimates of the parameters, as well as no improvement in the objective minimum trace, as the wrong emulator fails to suggest points which actually improve on the incumbent minimum from the initial design.

Consider, for example, assigning $\ell(\boldsymbol{q}) = 10^{10}$ to a failed simulation at $\boldsymbol{q}$. For visualization purposes, Figure 7.3a shows the training data (black dots) and the GP emulator, i.e. the predictive mean (in red), for the 2D parameters f_{L} and ξ. Figure 7.3b shows the EI acquisition function derived from the GP model, while Figure 7.3c shows the incumbent minimum trace vs n. The fact that the optimization algorithm fails to improve the estimate and the excessive multimodality of the derived EI function are direct consequences of a non-representative metamodel shown in Figure 7.3a. The flattened area of the GP and the non-improvement in the minimum trace are both pathologies due to the high-frequency oscillations induced by the high RSS values, leading to the kernel lengthscale being driven towards very small values. This produces a surrogate model, used to inform the optimization sequence, which is not a good representation of the true RSS. Compare instead Figure 7.3c to the trace in Figure 7.9e, which is obtained from a good model of the objective.

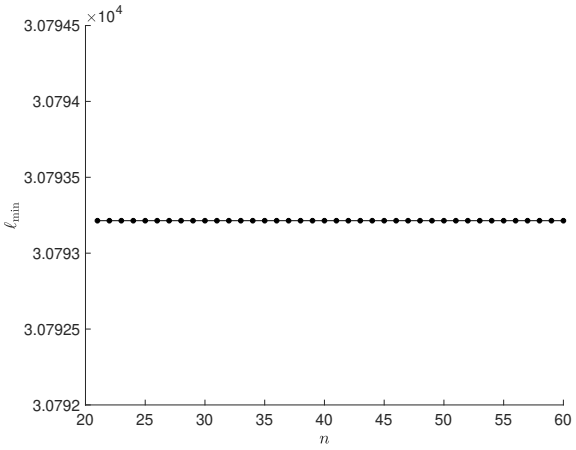
The problems mentioned above are even more evident when focusing just on the 1D estimation of the exponent ξ. Figures 7.4, 7.5 and 7.6 show the GP emulator and the derived EI acquisition function when the score for a failed simulation is $\ell(\boldsymbol{q}) = 10^5$, 10^{10} and 10^{20} respectively. Setting a priori the failure loss score is very hard. It could happen that the chosen value is lower than or very similar to the objective function minimum, see Figure 7.4a. The choice of the failure loss score also deeply affects the emulator quality and the multimodality of the derived acquisition function. By



(a) GP model of ℓ

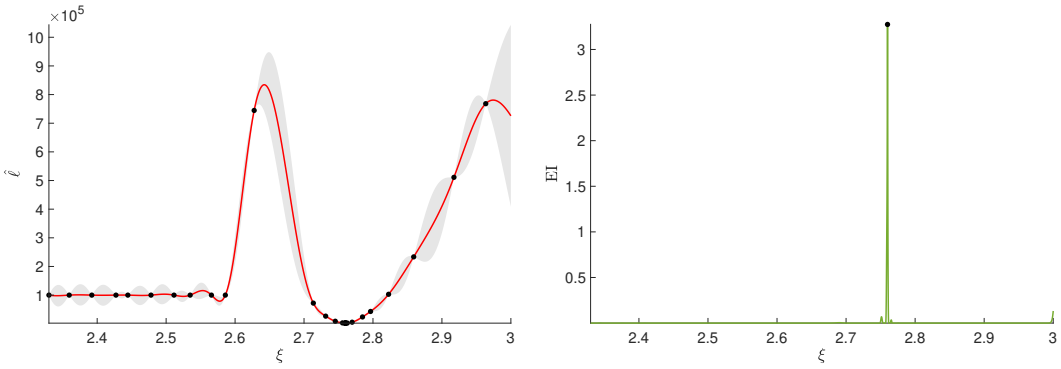


(b) Derived EI acquisition function



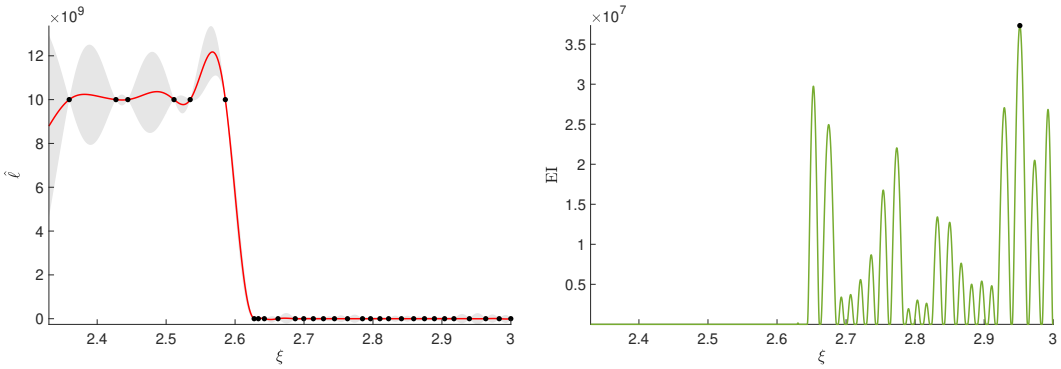
(c) Objective minimum trace

Figure 7.3: Assigning a high loss, $\ell(\boldsymbol{q}) = 10^{10}$, for a failed simulation at $\boldsymbol{q}$.



(a) GP model of ℓ **(b)** Derived EI acquisition function

Figure 7.4: Estimating ξ by assigning $\ell(\boldsymbol{q}) = 10^5$ for a failed simulation.



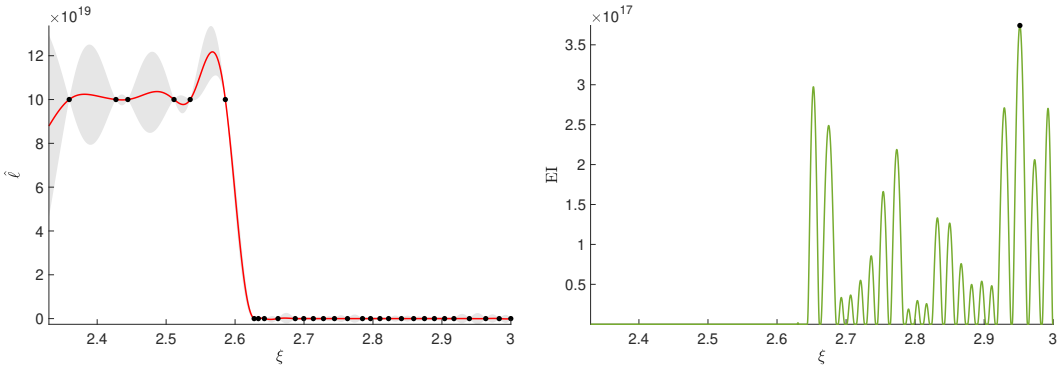
(a) GP model of ℓ **(b)** Derived EI acquisition function

Figure 7.5: Estimating ξ by assigning $\ell(\boldsymbol{q}) = 10^{10}$ for a failed simulation.

increasing the loss score for a failed simulation to 10^{10}, we find in Figure 7.5a a completely misspecified GP emulator, similarly to Figure 7.3a, with high frequency oscillations due to the kernel lengthscale being driven to small values. The derived EI acquisition function is highly multimodal, making the auxiliary optimization problem in Figure 7.5b challenging. This is also found by setting the error loss score to 10^{20} in Figure 7.6.

7.3.2 Building a Model of the Simulation Failures

In the case of unknown hidden constraints, i.e. unknown regions in the parameter domain leading to a simulation failure, the optimization of the function must be performed hand-in-hand with a sequential learning of the domain areas leading to



(a) GP model of ℓ **(b)** Derived EI acquisition function

Figure 7.6: Estimating ξ by assigning $\ell(\boldsymbol{q}) = 10^{20}$ for a failed simulation.

numerical failures. This requires a modification of Algorithm 6.1, as in Gelbart et al. (2014). The idea is that, along with each function evaluation $y_i = f(\boldsymbol{x}_i)$, we also keep track of the failure or success of the query in an auxiliary variable $h_i = h(\boldsymbol{x}_i)$. The convention I use for the binary variable $h(\boldsymbol{x})$ is to take the value 1 in case of a failure and -1 for a successful evaluation. Hence, we name $h \in \{-1, 1\}$ the failure indicator. The initial design will consist of triples $(\boldsymbol{x}_i, y_i, h_i)$ for $i = 1, \ldots, n_{\text{init}}$. The next step consists in obtaining two GP models:

(a) a GP model of the objective function, using the $(\boldsymbol{x}_i, y_i)$ pairs;

(b) a GP model of the failures, using the $(\boldsymbol{x}_i, h_i)$ pairs.

Model (a) represents a standard GP regression as described in Chapter 3, while model (b) requires predicting the posterior class probabilities of h for a new input $\boldsymbol{x}$, given a set of training data. For the GP classification with logit or probit link function, the class posterior probability is analytically intractable, see (6.76) in Bishop (2006). Different approximations have been proposed in order to predict binary-valued outcomes, for example iterative procedures like Expectation Propagation (EP), Laplace approximation (LA) or the simpler label regression (LR) approach.

Following the experiments and recommendations of Kuss (2006), which suggest that label regression works surprisingly well in practice and with a lower error rate than the competing methods in high dimensions, we apply LR in order to build a model of the simulation failures, ignoring the binary nature of the variable h in

favour of a quicker runtime and a closed-form posterior. Any approximation involving extra iterative procedures could cause the modelling and point selection time to be higher than simply evaluating the objective function. In other words, we would spend more time in modelling the function rather than evaluating it. While this can be arguably acceptable for really expensive simulators, it would not be computationally optimal for the presented application. A discussion of the computational costs of the pulmonary circulation model under study can be found in Section 7.4.

Denote the failure indicator model as $h(\boldsymbol{x}) \sim \mathsf{GP}(\hat{h}(\boldsymbol{x}), s_h(\boldsymbol{x}, \boldsymbol{x}'))$, obtained by applying the formulas summarized in Section 6.2 to the $\{(\boldsymbol{x}_i, h_i)\}$ data. By the marginalization property of GPs, at point $\boldsymbol{x}$ the random variable $h(\boldsymbol{x})$ follows a $\mathsf{N}(\hat{h}(\boldsymbol{x}), s_h^2(\boldsymbol{x}))$ distribution. As failures are labelled as 1 and successful evaluations as -1, by taking the probability of the Gaussian random variable being less than 0 we obtain an indication of the probability of a successful evaluation (no failure):

$$\mathbb{P}\{h(\boldsymbol{x}) = -1\} = \Phi(0 \mid \hat{h}(\boldsymbol{x}), s_h^2(\boldsymbol{x})). \tag{7.4}$$

This probability can then be used to weight the score that any acquisition function assigns to a point in the domain as follows (Gelbart et al., 2014):

$$\begin{aligned} a_n^*(\boldsymbol{x}) &= a_n(\boldsymbol{x}) \times \mathbb{P}\{h(\boldsymbol{x}) = -1\} \\ &= a_n(\boldsymbol{x}) \times \Phi(0 \mid \hat{h}(\boldsymbol{x}), s_h^2(\boldsymbol{x})). \end{aligned} \tag{7.5}$$

We refer to $a_n^*(\boldsymbol{x})$ as the hidden-constraints-weighted acquisition function. The algorithm then proceeds normally by choosing the next query point as the point maximizing $a_n^*(\boldsymbol{x})$. A pseudocode summary can be found in Algorithm 7.1. From (7.5) we see that both models (a) and (b) are required in order to compute the hidden-constraints-weighted acquisition function at each iteration of the BO algorithm.

At termination, the learned failure GP model can be used to obtain insights into the regions in the parameter domain leading to failure. For plotting purposes, in Figure 7.7 we show the probability of a successful evaluation (no failure), $\mathbb{P}\{h(\boldsymbol{x}) = -1\} = \Phi(0 \mid \hat{h}(\boldsymbol{x}), s_h^2(\boldsymbol{x}))$, for the 2D parameter space $\boldsymbol{q} = (f_{\mathrm{L}}, \xi)$. The figure shows how it is possible to have regions of failure inside a larger area of successful simulations.

Algorithm 7.1 Bayesian optimization with hidden constraints.

Inputs:

Initial design and corresponding failure labels: $\mathcal{D}_{n_{\text{init}}} = \{(\boldsymbol{x}_i, y_i, h_i)\}_{i=1}^{n_{\text{init}}}$

Budget of $n_{\max}$ function evaluations

for $n = n_{\text{init}}$ **to** $n_{\max} - 1$ **do**

  Update the objective GP: $f(\boldsymbol{x}) \mid \mathcal{D}_n \sim \mathsf{GP}(\hat{f}(\boldsymbol{x}), s(\boldsymbol{x}, \boldsymbol{x}'))$

  Update the failure GP: $h(\boldsymbol{x}) \mid \mathcal{D}_n \sim \mathsf{GP}(\hat{h}(\boldsymbol{x}), s_h(\boldsymbol{x}, \boldsymbol{x}'))$

  Compute the acquisition function: $a_n^*(\boldsymbol{x}) = a_n(\boldsymbol{x}) \times \Phi(0 \mid \hat{h}(\boldsymbol{x}), s_h^2(\boldsymbol{x}))$

  Solve the auxiliary optimization problem: $\boldsymbol{x}_{\text{next}} = \arg\max_{\boldsymbol{x} \in \mathcal{X}} a_n^*(\boldsymbol{x})$

  Query f at $\boldsymbol{x}_{\text{next}}$ to obtain y_{next} and h_{next}

  Augment data: $\mathcal{D}_{n+1} = \mathcal{D}_n \cup \{\boldsymbol{x}_{\text{next}}, y_{\text{next}}, h_{\text{next}}\}$

end for

Return:

Estimated minimum: $f_{\min} = \min(y_1, \ldots, y_{n_{\max}})$

Estimated point of minimum: $\boldsymbol{x}_{\min} = \arg\min(y_1, \ldots, y_{n_{\max}})$

7.4 Estimation of the Model Parameters

In the computational model of the pulmonary circulation, denoted by m, a forward simulation for fixed parameters takes around 23 seconds of CPU time[1]. The data collected by clinicians typically include pressure and flow measurements only from the midpoints of the 11 large vessels. In light of this, we refer to a *simulation* as the 22-dimensional vector $\boldsymbol{y} = \mathsf{m}(\boldsymbol{q})$ containing pressure and flow measurements at the midpoint location of each of the 7 large arteries and 4 large veins, for a given vector of PDE parameters $\boldsymbol{q}$. Given the costs of a single function evaluation, parameter estimation comes at substantial computational demands as standard global optimization algorithms require a large number of forward simulations. Motivated by real-time decision making, we want to reduce the computational time required to

[1]On a Dell Precision R7610 workstation with dual 10core Intel Xeon CPU, hyper-threading and 32GB RAM.

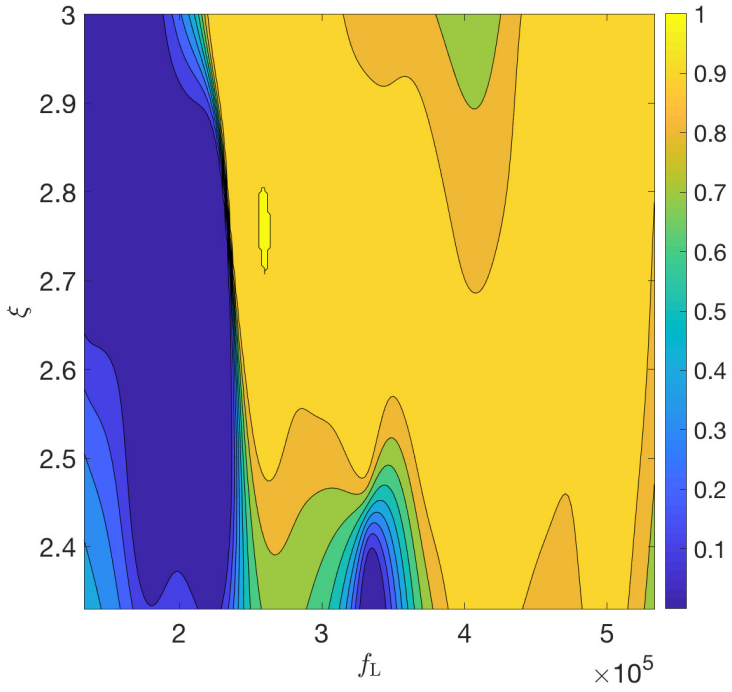


Figure 7.7: Learned probability of a successful simulation in the PDEs model of the human pulmonary circulation. A score near 1 indicates a high chance of a successful simulation, while a score near 0 indicates a high probability of failure.

estimate the PDE parameters by keeping the number of function evaluations as low as possible. To do so, we tackle this problem by using Bayesian optimization with hidden constraints and the EI acquisition function (6.4).

Let $\boldsymbol{q} = (f_{\mathrm{L}}, f_{\mathrm{S}}, \xi)$ denote the three parameters of relevance for the diagnosis of pulmonary hypertension. The simulated pressure and flow data $\boldsymbol{y}^{\mathrm{obs}}$ are obtained by a forward simulation of the computational model at the vector of parameters $\boldsymbol{q}^* = (2.6 \times 10^5, 5 \times 10^4, 2.76)$, assumed to be the underlying truth, and the data are then corrupted by i.i.d. additive Gaussian noise with a signal-to-noise ratio (SNR) of 10db. Pretending that the true parameter vector $\boldsymbol{q}^*$ is unknown, interest lies in its estimation from the noisy observations $\boldsymbol{y}^{\mathrm{obs}}$. This is to present a proof-of-concept study carried out on simulated data, for which it is possible to assess the inference as the gold-standard is known, but with the objective to ultimately apply it to real data and move it into the clinic. For i.i.d. additive Gaussian noise, the residual sum of

squares is proportional to the negative log likelihood. Hence, the objective function considered in this study is the squared L2 loss, or residual sum of squares, between a simulation $\mathsf{m}(\boldsymbol{q})$ and the data $\boldsymbol{y}^{\text{obs}}$:

$$\ell(\boldsymbol{q}) = \|\mathsf{m}(\boldsymbol{q}) - \boldsymbol{y}^{\text{obs}}\|^2. \tag{7.6}$$

Since each evaluation of the squared loss ℓ involves an expensive forward simulation from the PDE model $\mathsf{m}(\boldsymbol{q})$, also each evaluation of ℓ will be expensive. We estimate the parameters $\boldsymbol{q} = (f_{\text{L}}, f_{\text{S}}, \xi)$ by minimizing the squared loss using Bayesian optimization with hidden constraints and the EI acquisition function, with the following notational correspondence in Algorithm 7.1:

- Objective function: $f(\boldsymbol{x}) \equiv \ell(\boldsymbol{q})$;
- Input: $\boldsymbol{x}_i \equiv \boldsymbol{q}_i$;
- Output: $y_i \equiv \ell_i$.

Following Snoek et al. (2012) we use the ARD Matérn 5/2 kernel. We first consider the 1D estimation of ξ, then the 2D problem of jointly estimating f_{L} and ξ, and finally the 3D problem $f_{\text{L}}, f_{\text{S}}, \xi$, using the bounds discussed in Section 7.2. This allows us to visualize the GP emulator of the loss, the acquisition function and the failure GP model. For the 1D estimation problem we set a priori the maximum budget of function evaluations to $n_{\max} = 30$. For the 2D and 3D problems, instead, we set it to $n_{\max} = 60$. The optimization of ℓ is repeated five times using different random number seeds.

7.5 Results

Table 7.1 reports a summary of the results, calculated over five independent design instantiations having different random number generator seeds. The first column shows the problem dimensionality, while the second column shows the parameters that have been inferred simultaneously. The 3$^{\text{rd}}$ column contains the underlying truth for the parameters. In the fourth column we find the estimated parameters: the average and the standard error over the five runs. The final column reports

Dim	Parameters	Truth	Estimate		Budget
d	$\boldsymbol{q}$	$\boldsymbol{q}^*$	Mean	Std. Err.	$n_{\max}$
1	ξ	2.76	2.7601	< 0.0001	30
2	f_{L}	2.6×10^5	2.5989×10^5	98	60
	ξ	2.76	2.76	< 0.0001	
3	f_{L}	2.6×10^5	2.6178×10^5	948	60
	f_{S}	50000	49737	78	
	ξ	2.76	2.7597	0.0016	

Table 7.1: Inference results for the pulmonary circulation model. Averages and standard errors over five design instantiations with different random number generator seeds.

the total number of RSS evaluations allowed, i.e. the a priori budget of function evaluations and time allocated to the numerical experiment. With the settings shown in the last column, we required a solution in at most 12 minutes for the 1D problem, and approximately 23 minutes for the 2D and 3D problems.

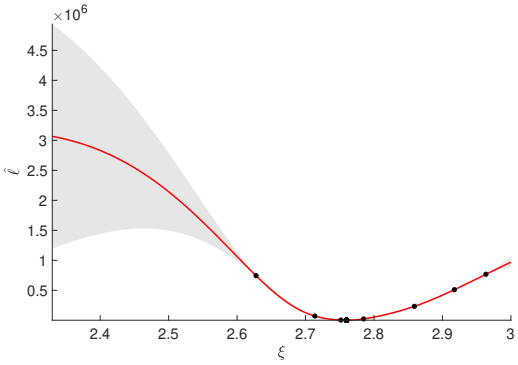
In the 1D inference problem we find a good estimate of ξ, with a very high confidence in the inferred value. Figure 7.8a shows how, by handling the errors correctly, the RSS function has a quadratic-like shape. Figure 7.8b shows the hidden-constraints-weighted EI acquisition function. Figure 7.8c shows the simulation failure GP model that is used to derive the probability of a successful simulation in Figure 7.8d according to (7.4), introduced above. Figure 7.8e shows the incumbent minimum trace vs the iteration number. We see a fast convergence in less than 20 function evaluations, where the value is not changing for the next iterations, but just improving the last decimals. Similar considerations can be done for Figure 7.9 and Figure 7.10. It is worth remarking that the plotted GP models refer to one of the five repetitions, while Table 7.1 presents an overall summary of the five independent runs. In the 2D case, the estimation of ξ is still accurate, and we also obtain a good estimate of the stiffness in the large vessels, with a small standard error in just 60 function evaluations. The full 3D estimation is still accurate for the exponent ξ, and satisfactory for the remaining two parameters.

It is worth comparing in particular the GP model of ℓ obtained by properly handling hidden constraints, Figure 7.9a, and the one obtained by assigning a high loss score at failed simulations, Figure 7.3a. At the final iteration, we can equivalently plot the learned probability of no failure (Figure 7.9d) as a contour plot in order to obtain insight into the regions in the parameter domain which violate the assumptions of the mathematical model (Figure 7.7).

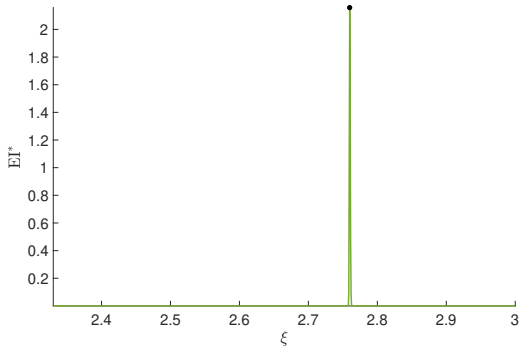
In this chapter, we estimated the GP hyperparameters by maximizing the log marginal likelihood, as discussed in Section 3.10. Osborne et al. (2009), instead, marginalize the hyperparameters using Bayesian Monte Carlo (Rasmussen and Ghahramani, 2003). Without integrating them out, we underestimate the uncertainty. However, this does not seem to be critical because of the low-dimensionality of the parameter space and the inspection of the results shows that BO works well for this task. Marginalization of the hyperparameters or estimation by maximizing the log marginal likelihood only affects the exploration-exploitation trade-off. The current results show that the method considered, while potentially underestimating the uncertainty, is still adequate. Furthermore, our goal is building real-time decision support systems, and hyperparameter marginalization using the Bayesian Monte Carlo method would add another layer of computational complexity to every iteration of the BO algorithm. For this reason, we prefer the simpler optimization approach.

7.6 Summary

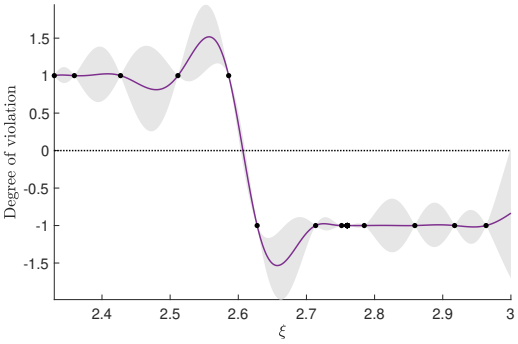
In this chapter I studied how to perform inference in a computationally expensive and novel model of the combined arterial and venous pulmonary blood circulation. The parameters of interest are f_{L}, f_{S} and ξ. The exponent ξ governs the vessel parent-to-daughter radius relation (7.1), with low values indicating vascular problems of clinical interest. As ξ increases, the number of vessels in the structured tree will also increase; similarly, as it decreases, the number of vessels will also decrease, simulating the vascular rarefaction clinical condition. The stiffness parameters in large, f_{L}, and small vessels, f_{S}, are also of particular interest because stiffening of these vessels is a primary cause of pulmonary arterial hypertension which leads to



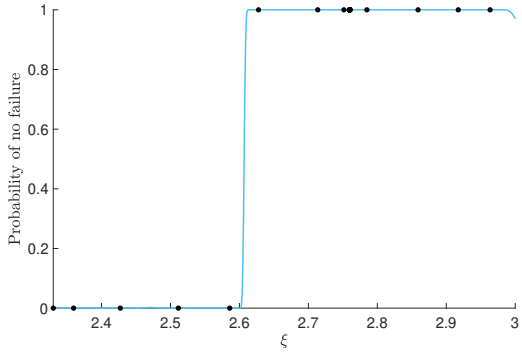
(a) GP emulator of ℓ



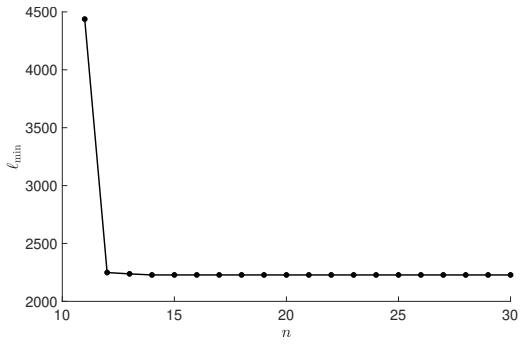
(b) Derived EI acquisition function



(c) GP model of the failure indicators

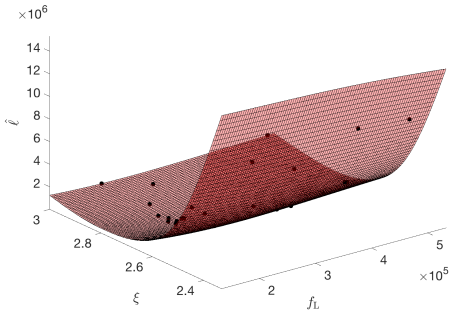


(d) Derived probability of no failure

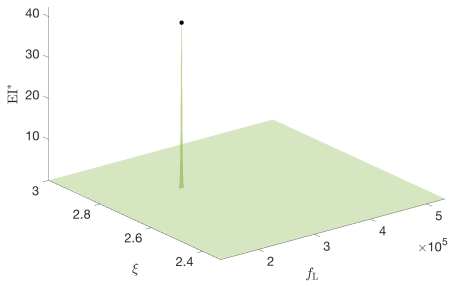


(e) Objective minimum trace

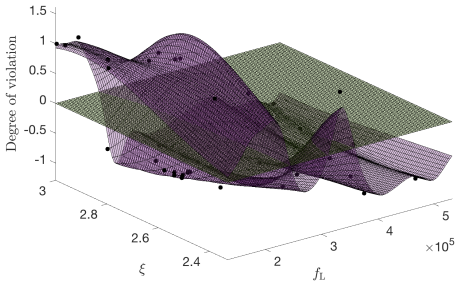
Figure 7.8: Estimation of the exponent ξ. Figures for one of the five repetitions.



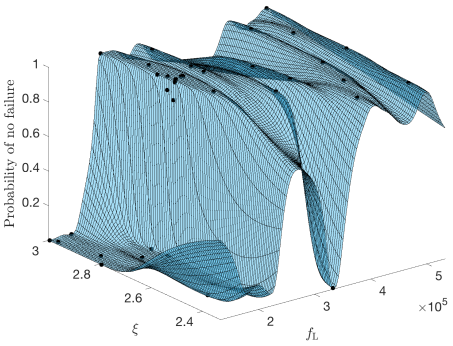
(a) GP model of ℓ



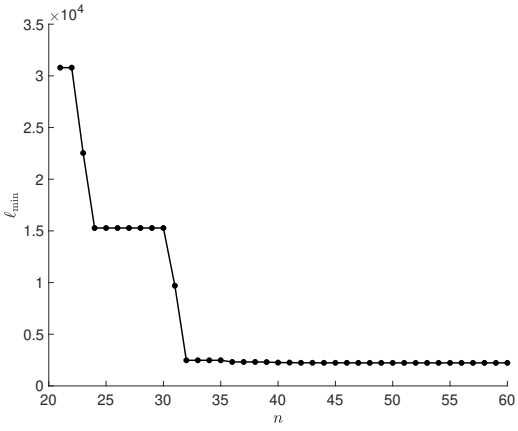
(b) Derived EI acquisition function



(c) GP model of the failure indicators



(d) Derived probability of no failure



(e) Objective minimum trace

Figure 7.9: Estimation of the stiffness parameter in the large vessels f_{L} and the exponent ξ. Figures for one of the five repetitions.

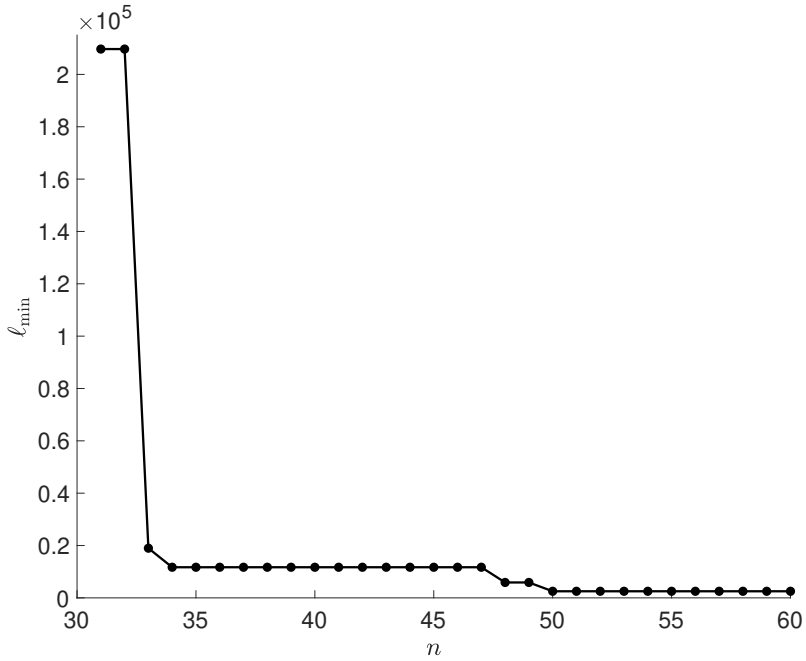


Figure 7.10: Estimation of the 3 parameters of primary clinical interest. Objective minimum trace for one of the five repetitions.

right heart failure.

In previous studies with state-of-the-art non-emulation-based global optimization solvers, like Genetic Algorithms or Global Search methods (Ugray et al., 2007), we found that the number of required function evaluations was between 10^3 and 10^4 for 1D problems, and even exceeded 10^4 function evaluations for simple 2D or 3D scenarios. Given that the computational costs of a single forward simulation are about 23 seconds of CPU time, the total computational costs would be in the order of 13 hours for 1D inference tasks and could reach two and a half days for 3D inferential problems. The results in Figures 7.8e, 7.9e and 7.10, show that the considered emulation-based approach achieves a substantial reduction in the number of forward simulations, effectively converging to a very good estimate of the parameters in less than the allowed budget of function evaluations (last column in Table 7.1), while spending the remaining iterations refining the last decimals. In the 1D scenario it reached convergence in less than 11 minutes, while for the 2D and 3D scenario in about 23 minutes or less. This corresponds to a total reduction of the computational complexity by two orders of magnitude. Two and a half days is not a suitable time

period for real-time clinical decision support systems. A period of roughly 20 minutes would mean that the patient could get a diagnosis from the clinician, informed by the inferred indicators of pulmonary hypertension, after a short wait, while the clinician examines the case.

Chapter 8

On a New Improvement-Based Acquisition Function for Bayesian Optimization

Chapter 6 reviewed Bayesian optimization (BO), a popular algorithm for solving challenging optimization tasks. It is designed for problems where the objective function is expensive to evaluate, perhaps not available in exact form, without gradient information and possibly returning noisy values. Different versions of the algorithm vary in the choice of the acquisition function, which recommends the point to query the objective at next. Initially, researchers focused on improvement-based acquisitions, while recently the attention has shifted to more computationally expensive information-theoretical measures. This chapter presents two major contributions to the literature. First, I propose a new improvement-based acquisition function that recommends query points where the improvement is expected to be high with high confidence. The proposed algorithm is evaluated on a large set of benchmark functions from the global optimization literature, where it turns out to perform at least as well as current state-of-the-art acquisition functions, and often better. This suggests that it is a powerful default choice for BO. The novel policy is then compared to widely used global optimization solvers in order to confirm that BO methods reduce the computational costs of the optimization by keeping the number of function evaluations small. The second main contribution represents an application

of the novel acquisition function to precision medicine, where the interest lies in the estimation of parameters of the partial differential equations model of the human pulmonary blood circulation system introduced in Chapter 7.

In Section 8.1 I derive the variance of the improvement quantifier and use it to define a new acquisition function. It improves the literature ones by accounting for another layer of uncertainty in the problem, namely the uncertainty in the improvement random variable. Section 8.2 presents a visual comparison of the novel policy and the widely used EI acquisition function. Sections 8.3 and 8.4 compare the newly introduced acquisition function with state-of-the-art acquisition functions from the Bayesian optimization literature on a large set of test functions for global optimization. Section 8.5 confirms that Bayesian optimization reduces the number of function evaluations required to reach the global optimum to a certain tolerance level, compared to standard global optimization algorithms. The proposed algorithm is then used in Section 8.6 to infer the parameters of the partial differential equations (PDEs) model of the human pulmonary blood circulation system introduced in Chapter 7, with the ultimate goal of real-time in silico diagnosis and prognosis.

Notes This chapter is adapted from: Noè, U. and Husmeier, D. (2018). On a New Improvement-Based Acquisition Function for Bayesian Optimization. *eprint arXiv:1808.06918*.

8.1 Scaled Expected Improvement

In Chapter 6 it was shown that the widely used *Expected Improvement* (EI) acquisition function automatically balances *exploitation* and *exploration*. What it does not account for, however, is the uncertainty in the improvement value $I(\boldsymbol{x})$. This might not be "orthogonal" to the uncertainty in the model of f, but it nevertheless represents an important source of information about our belief in the quality of a candidate point $\boldsymbol{x}$. Ideally, to avoid unnecessary expensive function evaluations, we hope to evaluate at points where, on average, the improvement is expected to be high, with high confidence. This is to avoid expensive queries at points where the improvement

is high, but the variability of this value is also high, effectively meaning that the improvement score $I(\boldsymbol{x})$ could be low, and thus we would evaluate at a sub-optimal point $\boldsymbol{x}$. In order to reach such a goal, we derive the variance of the improvement quantifier $I(\boldsymbol{x})$.

Lemma 8.1. *The variance of the random variable Improvement is:*

$$\mathbb{V}[I(\boldsymbol{x})] = s^2(\boldsymbol{x})\{(u^2+1)\Phi(u) + u\phi(u)\} - \{\mathrm{EI}(\boldsymbol{x})\}^2, \tag{8.1}$$

where, again, $u = \{f_{\min} - \hat{f}(\boldsymbol{x})\}/s(\boldsymbol{x})$.

Proof. Using properties (C.1) and (C.2) of Gaussian pdfs from Appendix C:

$$\begin{aligned}
\mathbb{V}[I(\boldsymbol{x})] &= \mathbb{E}[I^2(\boldsymbol{x})] - \{\mathbb{E}[I(\boldsymbol{x})]\}^2 \\
&= \mathbb{E}[\max\{f_{\min} - f(\boldsymbol{x}), 0\}^2] - \{\mathrm{EI}(\boldsymbol{x})\}^2 \\
&= \int_{-\infty}^{f_{\min}} \{f_{\min} - y\}^2 \phi(y \mid \hat{f}(\boldsymbol{x}), s^2(\boldsymbol{x}))\mathrm{d}y - \{\mathrm{EI}(\boldsymbol{x})\}^2 \\
&= \int_{-\infty}^{u} \{f_{\min} - \hat{f}(\boldsymbol{x}) - s(\boldsymbol{x})z\}^2 \phi(z)\mathrm{d}z - \{\mathrm{EI}(\boldsymbol{x})\}^2 \\
&= \int_{-\infty}^{u} \{[f_{\min} - \hat{f}(\boldsymbol{x})]^2 + z^2 s^2(\boldsymbol{x}) \\
&\qquad - 2zs(\boldsymbol{x})[f_{\min} - \hat{f}(\boldsymbol{x})]\}\phi(z)\mathrm{d}z - \{\mathrm{EI}(\boldsymbol{x})\}^2 \\
&= \{f_{\min} - \hat{f}(\boldsymbol{x})\}^2 \int_{-\infty}^{u} \phi(z)\mathrm{d}z \\
&\qquad + s^2(\boldsymbol{x}) \int_{-\infty}^{u} z^2 \phi(z)\mathrm{d}z \\
&\qquad - 2s(\boldsymbol{x})\{f_{\min} - \hat{f}(\boldsymbol{x})\} \int_{-\infty}^{u} z\phi(z)\mathrm{d}z - \{\mathrm{EI}(\boldsymbol{x})\}^2 \\
&= \{f_{\min} - \hat{f}(\boldsymbol{x})\}^2 \Phi(u) + 2s(\boldsymbol{x})\{f_{\min} - \hat{f}(\boldsymbol{x})\}\phi(u) \\
&\qquad + s^2(\boldsymbol{x}) \int_{-\infty}^{u} (z^2 - 1)\phi(z)\mathrm{d}z \\
&\qquad + s^2(\boldsymbol{x}) \int_{-\infty}^{u} \phi(z)\mathrm{d}z - \{\mathrm{EI}(\boldsymbol{x})\}^2 \\
&= \{f_{\min} - \hat{f}(\boldsymbol{x})\}^2 \Phi(u) + 2s(\boldsymbol{x})\{f_{\min} - \hat{f}(\boldsymbol{x})\}\phi(u) \\
&\qquad - s^2(\boldsymbol{x})u\phi(u) + s^2(\boldsymbol{x})\Phi(u) - \{\mathrm{EI}(\boldsymbol{x})\}^2.
\end{aligned}$$

From the definition of $u = \{f_{\min} - \hat{f}(\boldsymbol{x})\}/s(\boldsymbol{x})$, we obtain:

$$\begin{aligned}\mathbb{V}[I(\boldsymbol{x})] &= [\{f_{\min} - \hat{f}(\boldsymbol{x})\}^2 + s^2(\boldsymbol{x})]\Phi(u) \\ &\quad + s(\boldsymbol{x})\{f_{\min} - \hat{f}(\boldsymbol{x})\}\phi(u) - \{\mathrm{EI}(\boldsymbol{x})\}^2 \\ &= s^2(\boldsymbol{x})\{(u^2+1)\Phi(u) + u\phi(u)\} - \{\mathrm{EI}(\boldsymbol{x})\}^2.\end{aligned}$$

□

I now define a new acquisition function called the *Scaled Expected Improvement* (ScaledEI).

Definition 8.1. The *Scaled Expected Improvement* (ScaledEI) acquisition function is defined as the expectation of $I(\boldsymbol{x})$ divided by the standard deviation of $I(\boldsymbol{x})$:

$$\mathrm{ScaledEI}(\boldsymbol{x}) = \mathbb{E}[I(\boldsymbol{x})]/\{\mathbb{V}[I(\boldsymbol{x})]\}^{1/2}. \tag{8.2}$$

Selecting the next query point by maximizing this acquisition function corresponds to selecting query points where the improvement score is expected to be high with high confidence.

For every $\boldsymbol{x}$ in the domain $\mathcal{X}$, we have a random variable $I(\boldsymbol{x})$. The Scaled Expected Improvement, $\mathbb{E}[I(\boldsymbol{x})]/\{\mathbb{V}[I(\boldsymbol{x})]\}^{1/2}$, effectively corresponds to the *mean per unit of variance* and *is a dimensionless quantity.* This is a desirable feature for an acquisition function and is also shared by the Probability of Improvement, but not by the Expected Improvement.

One may wonder if the proposed division of the Expected Improvement by its standard deviation is accounting for the uncertainty twice. As is seen from (6.4), the expression of the Expected Improvement $\mathbb{E}[I(\boldsymbol{x})]$ already includes the variance of the interpolant $f(\boldsymbol{x})$ at the argument $\boldsymbol{x}$, coming from the distribution defined by the Gaussian process. So why do we have to account for the variance again? To clarify this issue, note the difference between the variance of the interpolant $f(\boldsymbol{x})$ and the variance of the Improvement random variable $I(\boldsymbol{x})$. The Gaussian process defines a distribution over functions $f(\boldsymbol{x})$. Since $I(\boldsymbol{x})$ is a transformation of $f(\boldsymbol{x})$, this induces a distribution over $I(\boldsymbol{x})$. However, the acquisition function reduces this distribution to its expectation value, $\mathbb{E}[I(\boldsymbol{x})]$. In doing so, we lose the information

about the dispersion of $I(\boldsymbol{x})$. Recall that the expectation value is the first moment of the distribution, whereas uncertainty quantification requires the knowledge of higher-order moments.

For comparison, consider a random variable Z with a normal distribution, $p(z) = \mathsf{N}(z \mid 0, \sigma^2)$. If we know σ^2, we know everything about the variable's distribution[1]. Now consider the transformation $f(z) = z^2$. Since $\mathbb{E}[f(Z)] = \sigma^2$, knowing σ^2 implies that we know the function's expectation value. However, we know nothing else. In particular, we have no explicit knowledge of the transformed random variable's dispersion despite the fact that we know the original random variable's variance.

As a second example, consider a random variable $Y(x)$ drawn from a product of normal distributions, $p(y) \propto \mathsf{N}_1(y \mid \mu_1(x), \sigma_1^2(x))\mathsf{N}_2(y \mid \mu_2(x), \sigma_2^2(x))$. This so-called "product of experts" model is widely used in control theory as a "fuzzy" logical AND operation. It is straightforward to show that the expectation of $Y(x)$ is given by

$$\mathbb{E}[Y(x)] = \frac{\mu_1(x)\sigma_1^2(x) + \mu_2(x)\sigma_2^2(x)}{\sigma_1^2(x) + \sigma_2^2(x)}.$$

So, the expectation of $Y(x)$ depends on the variances $\sigma_1^2(x)$ and $\sigma_2^2(x)$ of both components of the product, i.e. it takes the uncertainty of both "experts" into account. However, this does *not* quantify the uncertainty of $Y(x)$ itself. We emphasize again that uncertainty quantification requires knowledge of higher-order moments of a distribution.

In the same vein, the expression for the Expected Improvement $\mathbb{E}[I(\boldsymbol{x})]$ depends on the uncertainty of the interpolant at $\boldsymbol{x}$, as quantified by the Gaussian process. However, this does *not* quantify the uncertainty of the Improvement random variable $I(\boldsymbol{x})$ itself.

For that reason, we have generalized the established improvement-based acquisition function and explicitly derived an expression for the variance of $I(\boldsymbol{x})$, to take both the expectation of $I(\boldsymbol{x})$, $\mathbb{E}[I(\boldsymbol{x})]$, and its standard deviation, $\sqrt{\mathbb{V}[I(\boldsymbol{x})]}$, into account. It is this standard deviation that we need to quantify the uncertainty of $I(\boldsymbol{x})$ at the lowest possible order.

[1]The notation $\mathsf{N}(x \mid \mu, \sigma^2)$ denotes the probability density function of a $\mathsf{N}(\mu, \sigma^2)$ random variable evaluated at x.

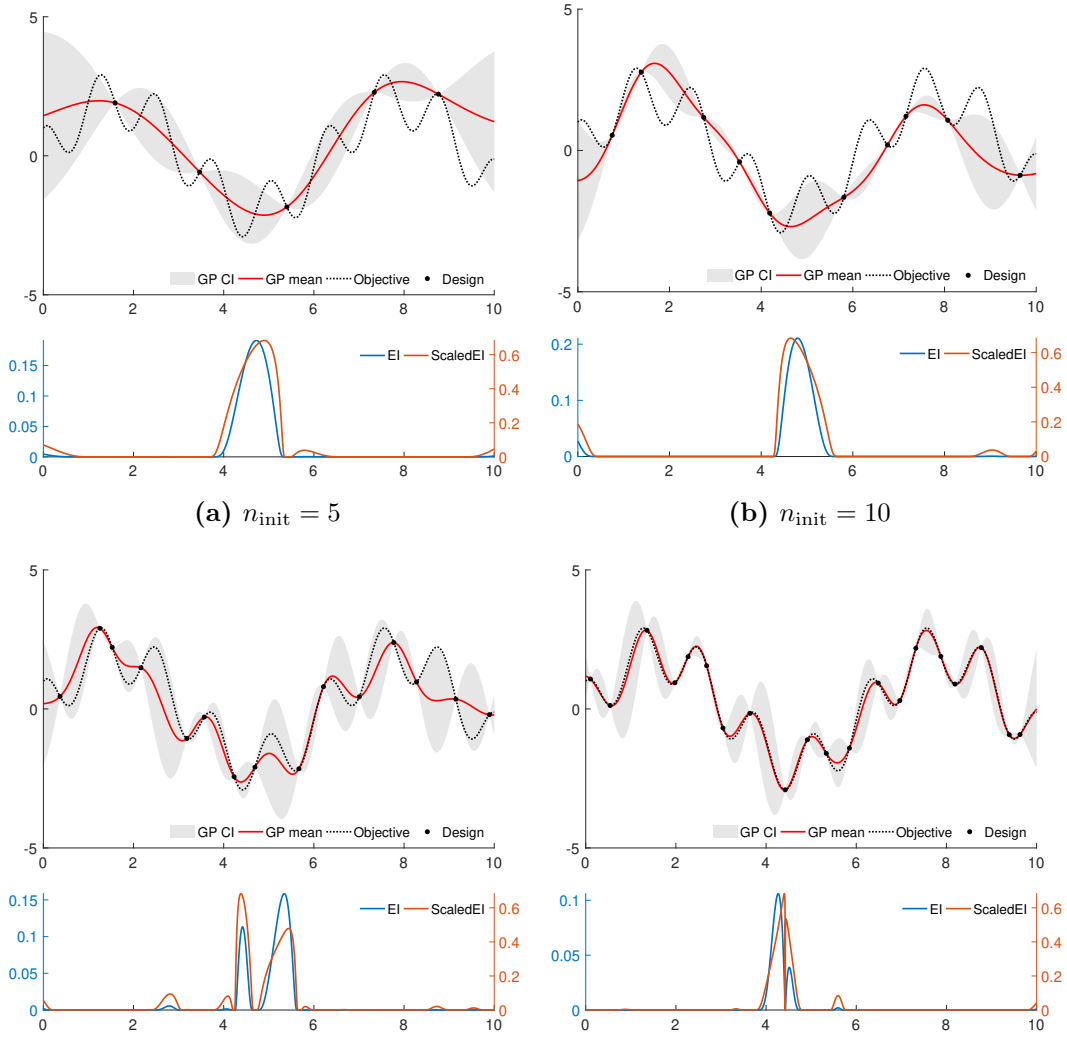
8.2 Visual Comparison

It is interesting to visually compare the two different acquisition functions EI and ScaledEI, in order to see which areas of the domain each policy emphasizes. Consider again optimizing the "Cosine Sine" (CSF) function $f(x) = \cos(5x) + 2\sin(x)$ over $\mathcal{X} = [0, 10]$, using BO with the ARD Squared Exponential kernel. Figure 8.1 shows the EI (blue) and the ScaledEI (orange) policies, for different initial designs. Figure 8.1a, using $n_{\text{init}} = 5$, shows that ScaledEI is slightly more conservative than EI, proposing a point nearer to the incumbent minimum. A similar behaviour for ScaledEI is found in Figure 8.1b, while still allowing for sufficient exploration. In Figure 8.1c, however, we notice how the EI policy puts more emphasis on exploration, rather than improving the decimals of the incumbent minimum. ScaledEI suggests a much better point, while still giving enough weight to exploration (see the second mode). Finally, Figure 8.1d shows how ScaledEI correctly focuses on refining the already found minimum, while EI suggests to step away from the best location found.

Figure 8.2 shows the iterations of the BO algorithm using the ScaledEI acquisition function, which can be directly compared to Figure 6.2 obtained for the EI policy. Notice how in this scenario ScaledEI does not evaluate at the domain boundaries, since the relative score given to the left boundary is smaller than EI, see Figure 6.2c. Figure 8.3, similarly to Figure 6.3, shows the corresponding objective minimum trace vs n. It shows how the ScaledEI algorithm finds the global minimum already at $n = 13$, while for the EI this happens at $n = 19$ only.

8.3 Benchmark Study

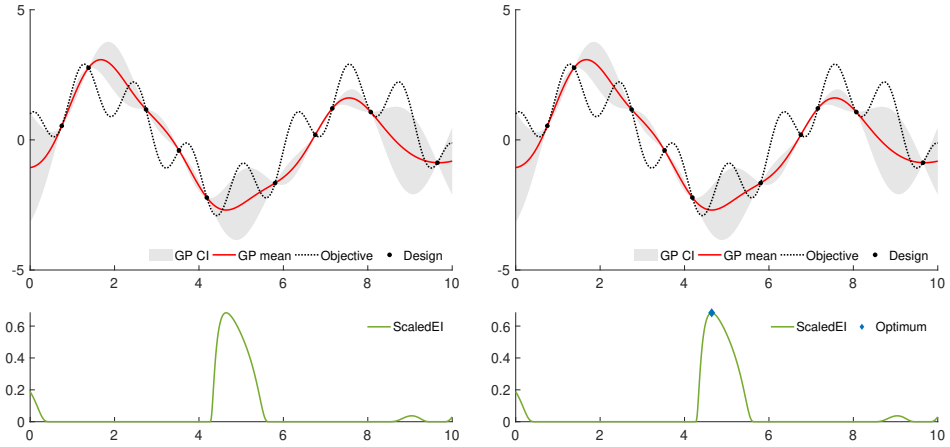
We test the performance of the acquisition function introduced in (8.2) and the ones from the BO literature (summarized in Section 6.3) on an extensive test set of objective functions taken from the global optimization literature. The set of test functions for global optimization is summarized in Table 8.1. This comprehensive test set includes benchmark functions found in leading global optimization articles such as Jones et al. (1993) and Huyer and Neumaier (1999), with the addition of the 1D Cosine Sine (CSF) test function, which we defined as $f(x) = \cos(5x) + 2\sin(x)$.



(a) $n_{\text{init}} = 5$ **(b)** $n_{\text{init}} = 10$

(c) $n_{\text{init}} = 15$ **(d)** $n_{\text{init}} = 20$

Figure 8.1: Visual comparison of EI and ScaledEI.



(a) GP emulator and derived ScaledEI **(b)** $n = 10$

(c) $n = 11$ **(d)** $n = 12$

(e) $n = 13$ **(f)** $n = 20$

Figure 8.2: Illustration of the BO algorithm using the ScaledEI acquisition function.

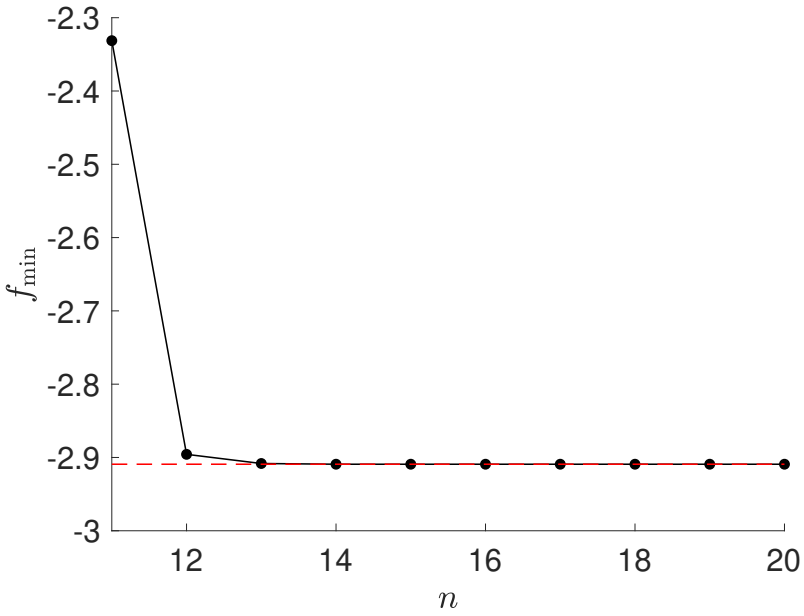


Figure 8.3: Objective minimum trace for the CSF function using the ScaledEI policy. The incumbent minimum $f_{\min}$ vs the iteration number n. The true global minimum f_{global} is shown as a dashed red line.

These optimization problems are challenging due to the presence of multiple local minima, the sharp variation in the y-axis, and symmetries with the presence of multiple points at which the global minimum is attained. Figure 8.4 shows a plot of the 1D test function CSF and the contours of the 2D objective functions along with the global optima.

Let f_{global} denote the globally optimal function value known from the literature and denote by $f_{\min}$ the best function value at iteration n. To check for convergence to the global minimum f_{global} we report the $\log_{10}$ distance, defined as follows:

$$\log_{10} \text{distance} = \log_{10} |f_{\min} - f_{\text{global}}|,$$

where the dependency of the distance on n comes through $f_{\min}$.

The established acquisition functions that we compared with the proposed new acquisition function, ScaledEI, are LCB, representing the optimistic policies (class 1); PI and EI from the improvement-based ones (class 2) and MES representing the information-theoretic measures (class 3), which has been shown to outperform ES and PES; see Wang and Jegelka (2017). We also include as benchmarks two naive approaches: RND($\boldsymbol{x}$) and MN($\boldsymbol{x}$). The first corresponds to random search

Table 8.1: Key characteristics of the test functions.

Test function	Abbrev.	Number of dimensions	Number of local minima	Number of global minima
Cosine Sine	CSF	1	8	1
Rosenbrock	ROS	2	1	1
Branin RCOS	BRA	2	3	3
Goldstein and Price	GPR	2	4	1
Six-Hump Camel	CAM	2	6	2
Two-Dimensional Shubert	SHU	2	760	18
Hartman 3	HM3	3	4	1
Shekel 5	SH5	4	5	1
Shekel 7	SH7	4	7	1
Shekel 10	SH10	4	10	1
Hartman 6	HM6	6	4	1
Rastrigin	RAS	10	11^{10}	1

(Bergstra et al., 2011; Bergstra and Bengio, 2012), which proposes a point from a uniform distribution within the bounded domain $\mathcal{X}$. The second corresponds to iteratively maximizing the negative GP predictive mean, $\text{MN}(\boldsymbol{x}) = -\hat{f}(\boldsymbol{x})$, and is the extreme case of focusing only on *exploitation* while ignoring uncertainty. The GP model uses the ARD Squared Exponential kernel and a constant mean function. The model hyperparameters are estimated at each iteration by maximizing the log marginal likelihood using the Quasi-Newton method as described in Section 3.10, while maximization of the acquisition function was discussed in Section 6.4. For all experiments we set a priori the maximum budget of function evaluations to be $n_{\max} = 1000$, and an upper bound on the computational runtime for the computer cluster[2] of 2 weeks. The maximum budget is usually set by the analyst, considering the unit price for a single evaluation. It could be an actual monetary price, if the experiment involves materials and trained staff, or computational resources. The

[2]The computer cluster used in this work includes eight CentOS 7 machines with 24 cores and 32GB RAM each.

code for the acquisition function MES has been cloned as of July 2017 from Wang and Jegelka (2017) first author's GitHub repository[3]. Since July 2017 the authors have not made significant changes to the core functionality apart from some documentation updates. For the experiments, the settings have been kept the same as in Wang and Jegelka (2017) in terms of GP mean and kernel choice. For optimal performance and a fair comparison with the other methods, the GP hyperparameters were updated at every iteration instead of their default choice of every 10 iterations. Most importantly, between the two versions of MES presented by the authors (MES-R and MES-G), we chose the version that in their paper was shown to perform best, namely the MES-G acquisition function with f_{global} sampled from the approximate Gumbel distribution. The choice was also motivated by the statement in Wang and Jegelka (2017) that MES-R is better for problems with only a few local optima, while MES-G works better in highly multimodal problems as more exploration is needed. As the set of test functions used is characterized by high multimodality and the presence of multiple points at which the global minimum is attained, we present results for the MES-G policy, which will be simply denoted as MES. The number of f_{global} sampled was set to 100 as in the experiments of Wang and Jegelka (2017). For the LCB acquisition function, representing the class of optimistic policies, we set $\kappa = 2$ as commonly used, see for example Turner and Rasmussen (2012). We ran Bayesian optimization using each of the acquisition functions on every benchmark function, and we repeated each optimization five times using different random number seeds for the initial design.

8.4 Benchmark Results

This section empirically shows that the proposed acquisition function performs as well as or better than the state-of-the-art methods reviewed in Section 6.3, using an extensive set of benchmark problems from the global optimization literature (Jones et al., 1993; Huyer and Neumaier, 1999). Figure 8.5 shows the $\log_{10}$ distance in function space to the global optimum as a function of the objective function

[3] `https://github.com/zi-w/Max-value-Entropy-Search`

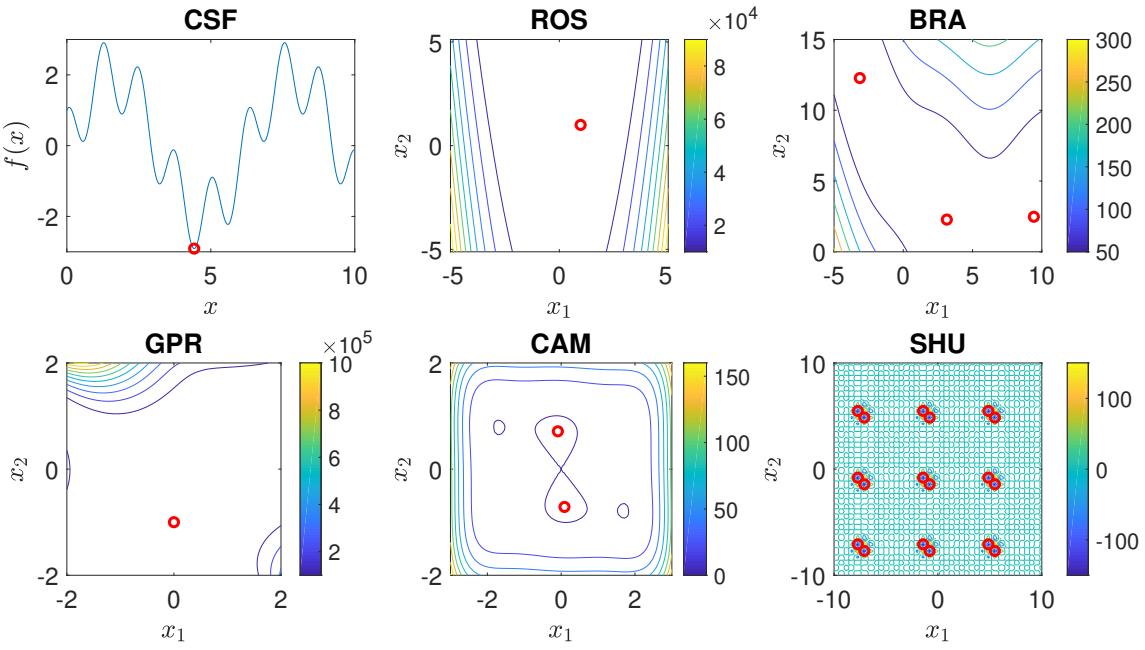


Figure 8.4: Plot of the 1D function CSF and contours of the 2D test functions. The red circles represent the global optima.

evaluations n. Each trace represents the average $\log_{10}$ distance of a given algorithm over the 5 different initial design instantiations, and the error bars show the standard error of the mean for the whole spectrum of iteration numbers. For two of the problems, CAM and SHU, the Max-Value Entropy Search method did not reach the maximum budget $n_{\max}$ in the allowed computational time (2 weeks)[4]. In these cases the $\log_{10}$ distance in function space is shown up to the latest iteration.

The results in Figure 8.5 show that the RND policy is consistently outperformed by the improvement-based acquisition functions. In the 1D scenario (CSF) RND is the worst method, followed by MN, which has huge variation in the results. The best methods include improvement-based policies and MES, but there does not seem to be a significant difference between them. For most of the 2D functions (ROS, GPR, CAM, SHU) it appears that improvement-based policies outperform the information-theoretic one. In three of these functions the proposed ScaledEI

[4]This is based on using the software implementation accompanying the paper by Wang and Jegelka (2017). Note that for the two problems where MES did not converge, the $\log_{10}$ distance to the optimum is still very far from the other competing algorithms. This suggests that, even if we were to add more iterations, the results would still be not satisfactory.

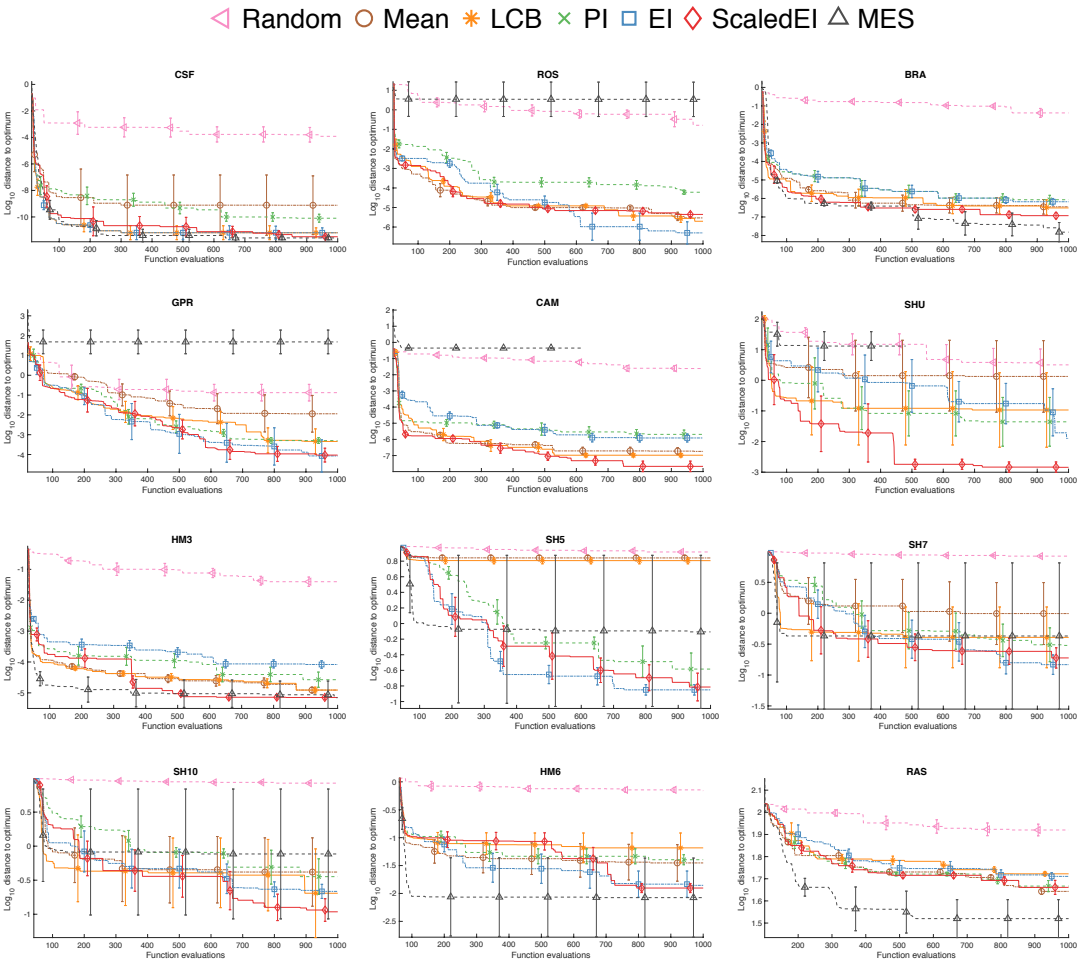


Figure 8.5: Comparison of the $\log_{10}$ distances (in function space) to the global optimum. Each panel represents a given benchmark function. The traces show the average distance over the five design instantiations for the whole spectrum of iterations, while the error bars show plus or minus one standard error of the mean.

Table 8.2: Statistical test for the significance in the mean difference of the final $\log_{10}$ distances. The ScaledEI acquisition function was tested against all remaining acquisition functions using a paired t-test with significance level 0.05. Codes: 0 indicates a non significant difference and 1 (-1) indicates that ScaledEI performed better (worse), i.e. it has a significantly lower (higher) average distance.

	ScaledEI vs					
Test function	RND	MN	LCB	PI	EI	MES
CSF	1	0	0	1	0	0
ROS	1	0	0	1	0	1
BRA	1	0	0	0	1	0
GPR	1	0	0	0	0	1
CAM	1	0	0	1	1	1
SHU	1	0	0	0	0	1
HM3	1	0	0	0	1	0
SH5	1	1	1	0	0	0
SH7	1	0	0	0	0	0
SH10	1	0	0	0	0	0
HM6	1	0	0	1	0	0
RAS	1	0	0	0	0	0
Same	0%	92%	92%	67%	75%	67%
Better	100%	8%	8%	33%	25%	33%
Worse	0%	0%	0%	0%	0%	0%

outperforms all others. In the BRA function, MES seems to be the best, but by a small margin compared to the evident gap in the other 2D scenarios. In the 3D problem (HM3) EI is outperformed by the other methods, except for RND. However, the proposed method, ScaledEI, is the best, emphasizing the power of the proposed adjustment. Even in the 4D, 6D and 10D test functions the ScaledEI policy appears to be one of the most competitive methods, while its information-based competitor MES suffers from highly variable results.

In order to facilitate the interpretation of the results, we test in Table 8.2 the significance of the difference in means of the final $\log_{10}$ distances[5] for ScaledEI vs each of the remaining acquisition functions, using a paired t-test[6] with significance level 0.05. We remark that the choice of performing a t-test on the log distances at the final iteration is only for summary purposes, and Figure 8.5 shows the full spectrum of performance scores for all function evaluations ranging from $n = 100$ to 1000. In most cases, the log distance curves are fairly flat between 300 and 1000, so the table presented here is representative of the majority of the choices of n. In general we would not get the true optimum at a high degree of accuracy with only 200 function evaluations. Nevertheless, we have carried out the statistical hypothesis tests for $n = 600$ and $n = 200$ as well, and the results can be found in Tables 8.3 and 8.4. A score of 0 indicates that the null hypothesis of equal average $\log_{10}$ distance is *not* rejected. Both 1 and -1 indicate the rejection of the null hypothesis, where a score of 1 indicates that the proposed method, ScaledEI, achieves a significant improvement, while a score of -1 shows that ScaledEI is significantly worse.

The proposed acquisition function ScaledEI consistently outperforms the naive RND policy. Compared with the established acquisition functions, ScaledEI nearly always achieves equal (67-92%) or significantly better (8-33%) performance, without

[5]These are the distances at the last iteration, where either the pre-defined budget or the maximum CPU time was exceeded.

[6]One could also use the nonparametric Wilcoxon signed-rank test. However, the test is based on only five data points. We believe that any inference based on five points only is hopeless without any kind of structural assumption. The t-test, being parametric, can provide more statistical power to detect a significant difference. Furthermore, the t-test is a mere summary of Figure 8.5, which contains the full spectrum of results and can be inspected for more information.

being significantly outperformed by the competitors. One third of the times, ScaledEI is significantly better than PI and the information-theoretic competitor MES. This is corroborated in Figure 8.5, where for the benchmark functions ROS, GPR, CAM, SHU the information-theoretic policy does not come close to the minimum. Then follow EI and LCB, which are outperformed 25% and 8% of the times, and MN, which is, surprisingly, outperformed only 8% of the times. This finding provides reassurance for conservative BO strategies, as in Wang et al. (2013b). However, the summary table based on hypothesis tests incurs a loss of information. From the t-test it is not evident that the MN policy suffers from huge variations in the results (see Figure 8.5). The fact that for some random number seeds MN performs well is due to the initial design generating a point near the global minimum by chance. Then, by emphasizing exploitation only, this point will be fine-tuned to the global optimum. ScaledEI never appears to be significantly worse than its competitors, and in particular the widely applied EI method. Our results suggest that the proposed acquisition function, ScaledEI, which combines high expected improvement with high confidence in the improvement being high, performs as well as or better than state-of-the-art acquisition functions. This makes ScaledEI a good default choice for standard BO applications.

As already mentioned above, we would not expect the algorithms to fine-tune the returned optimum in only $n = 200$ steps. However, for representational completeness we have carried out the statistical hypothesis tests for $n = 600$ and $n = 200$ nevertheless. These are shown in Tables 8.3 and 8.4 respectively.

Table 8.3 shows that at 600 iterations the results are consistent with Table 8.2, which uses the full budget of function evaluations. ScaledEI is always significantly better than RND, but only 8% of the times better than MN. Again, this is due to the random generation of an initial design point near a global minimum by chance, which is fine-tuned by focusing on exploitation only. However, this approach carries substantial variations in the results, and the success of the MN method depends entirely on the initial design choice, making it a non-optimal policy. For the remaining methods, ScaledEI is significantly better than each competitor 17-42% of the times, and in the remaining cases the methods are not significantly different. One third of

the times it is better than the widely used EI method and the information-theoretic MES. Furthermore, ScaledEI is never significantly outperformed by its competitors, including MES. We finally report in Table 8.4 the same kind of test but stopping at $n = 200$ iterations only. In this scenario, ScaledEI performed significantly better than PI and EI, 17% and 42% of the times respectively, while in the remaining cases the two were not significantly different. Comparing ScaledEI and the information theoretic strategy (MES), 50% of the time the two are not significantly different, but 33% ScaledEI is performing significantly better, and it is outperformed by MES in two benchmark functions only: HM3 and RAS.

As remarked in Section 6, different BO algorithms vary in the choice of the acquisition function. The proposed one, ScaledEI, was tested against literature methods on a set of 12 benchmark functions having different functional characteristics. According to the no-free-lunch theorem, we do not expect to see one method consistently outperforming all the remaining algorithms on all benchmark functions and for any arbitrary choice of function evaluations n. However, our results, shown in Figure 8.5 and Tables 8.2 to 8.4, suggest that ScaledEI tends to perform as well as or better than the alternative methods, and hence constitutes a powerful default choice for Bayesian optimization.

Table 8.3: Statistical test for the significance in the mean difference of the $\log_{10}$ distances at $n = 600$. The ScaledEI acquisition function was tested against all remaining acquisition functions using a paired t-test with significance level 0.05. Codes: 0 indicates a non significant difference and 1 (-1) indicates that ScaledEI performed better (worse), i.e. it has a significantly lower (higher) average distance.

	ScaledEI vs					
Test function	RND	MN	LCB	PI	EI	MES
CSF	1	0	0	1	0	0
ROS	1	0	0	1	0	1
BRA	1	0	0	1	1	0
GPR	1	0	0	0	0	1
CAM	1	0	0	1	1	1
SHU	1	0	0	0	1	1
HM3	1	0	0	1	1	0
SH5	1	1	1	0	0	0
SH7	1	0	0	0	0	0
SH10	1	0	0	0	0	0
HM6	1	0	0	0	0	0
RAS	1	0	1	0	0	0
Same	0%	92%	83%	58%	67%	67%
Better	100%	8%	17%	42%	33%	33%
Worse	0%	0%	0%	0%	0%	0%

Table 8.4: Statistical test for the significance in the mean difference of the $\log_{10}$ distances at $n = 200$. The ScaledEI acquisition function was tested against all remaining acquisition functions using a paired t-test with significance level 0.05. Codes: 0 indicates a non significant difference and 1 (-1) indicates that ScaledEI performed better (worse), i.e. it has a significantly lower (higher) average distance.

	ScaledEI vs					
Test function	RND	MN	LCB	PI	EI	MES
CSF	1	0	0	0	0	0
ROS	1	0	0	1	1	1
BRA	1	0	0	1	1	0
GPR	0	0	0	0	0	1
CAM	1	0	0	0	1	1
SHU	0	0	0	0	1	1
HM3	1	0	0	0	1	-1
SH5	1	0	0	0	0	0
SH7	1	0	0	0	0	0
SH10	1	0	0	0	0	0
HM6	1	0	0	0	0	0
RAS	1	0	0	0	0	-1
Same	17%	100%	100%	83%	58%	50%
Better	83%	0%	0%	17%	42%	33%
Worse	0%	0%	0%	0%	0%	17%

8.5 Comparative Study with Standard Global Optimization Solvers

The goal of BO is to reduce the computational costs required to optimize an expensive-to-evaluate function f, by reducing the number of function queries. This section corroborates the claim by presenting a proof-of-concept study recording the number of function evaluations required to reach a $\log_{10}$ distance to the true global optimum equal to -6. In this experiment we used the objective function CSF, defined in (6.5) and shown in the top left panel of Figure 8.4, and compared ScaledEI with a range of algorithms widely used, for example, by applied mathematicians and engineers:

1. Genetic Algorithm (Goldberg, 1989; Conn et al., 1991, 1997);
2. Global Search (Ugray et al., 2007);
3. Simulated Annealing (Ingber, 1996);
4. Particle Swarm (Mezura-Montes and Coello Coello, 2011; Pedersen, 2010);
5. Multi Start (10 random starting points) (Ugray et al., 2007; Glover, 1998);
6. Pattern Search (Audet and Dennis, 2002; Abramson et al., 2009).

We use the implementation found in MATLAB's Global Optimization Toolbox[7], with the default automatic settings for each algorithm. Each optimization was repeated 15 times, using different random number generator seeds.

Figure 8.6 shows, for each optimization algorithm, the average number of function evaluations (over the 15 random number seeds) required to reach a $\log_{10}$ distance of -6, while the error bars show plus or minus one standard error of the mean.

The Genetic Algorithm, requiring between 10^3 and 10^4 function evaluations, is the least suitable algorithm for expensive objective functions. Then follow Global Search, which requires in the order of 10^3 evaluations, Simulated Annealing and Particle Swarm, both requiring between 10^2 and 10^3 evaluations. The Multi Start and Pattern Search solvers rank as the most efficient ones, in terms of the number

[7] `https://uk.mathworks.com/products/global-optimization.html`

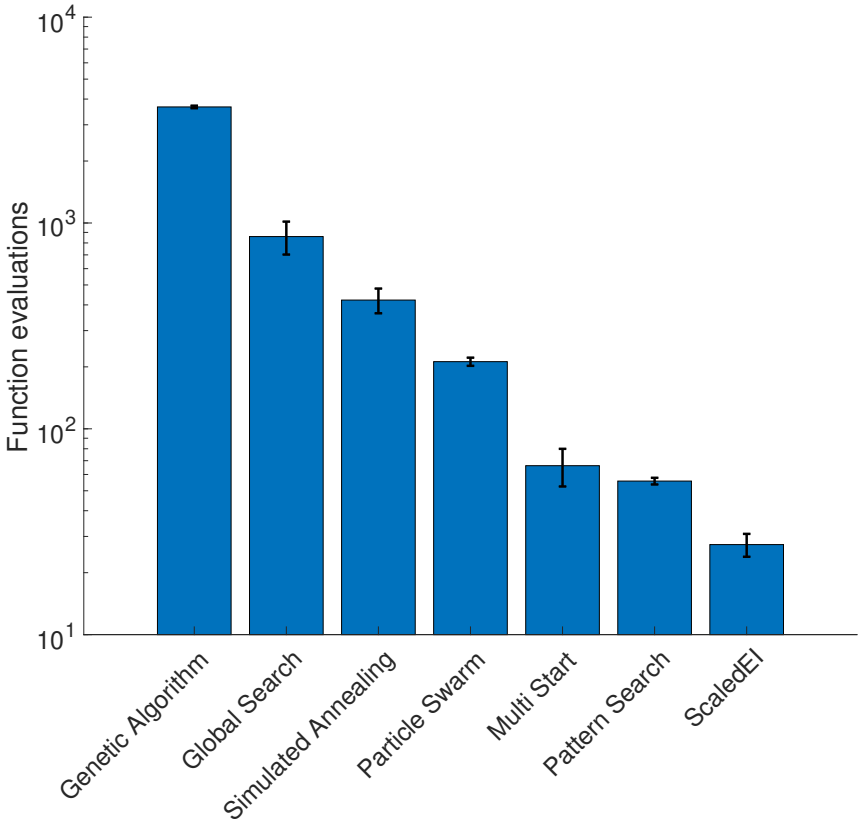


Figure 8.6: Number of function evaluations required to reach a $\log_{10}$ distance to the true global optimum (in function space) equal to -6.

of function evaluations, but they are clearly outperformed by ScaledEI. This comes as no surprise as, by construction, BO algorithms use all the information available from previous function evaluations to internally maintain a surrogate model of the objective function and infer its geometric properties in order to recommend the next candidate point.

In summary, Bayesian optimization reduces the computational costs required to optimize (6.5) by 2 orders of magnitude compared to the Genetic Algorithm, and more than 1 order of magnitude compared to Global Search, Simulated Annealing and Particle Swarm methods. This makes non-Bayesian optimization methods not suitable for clinical decision support systems or personalized medicine, where the unit cost of a single function query for soft tissue biomechanical models is usually high.

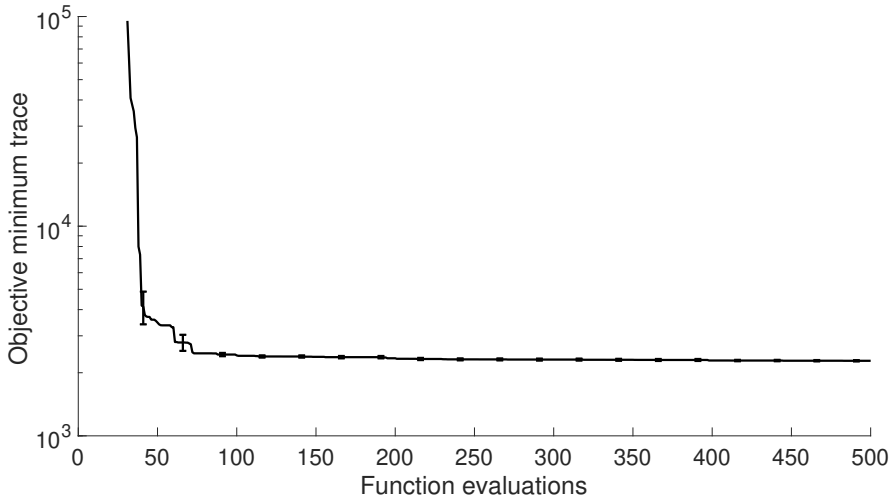


Figure 8.7: Optimization of the squared L2 loss for the pulmonary circulation model using the ScaledEI algorithm. The plot shows the minimum observed squared L2 loss trace, averaged over the 15 Latin hypercube design instantiations, plus or minus one standard error, as function of the number of function evaluations.

8.6 Application to the Pulmonary Circulation Model

This section applies the novel ScaledEI Bayesian optimization algorithm introduced in (8.2) to the problem of estimating the parameters of the human pulmonary circulation PDE model described in Chapter 7. The goal is to infer indicators of pulmonary hypertension risk for clinical decision support systems, without the need for invasive measurements.

In this experiment we set a priori the maximum budget of function evaluations to $n_{\max} = 500$. We repeated the optimization of ℓ using fifteen different random number seeds. Let $\ell_{\min} = \min(\ell_1, \ldots, \ell_n)$ denote the minimum observed objective (current best function value) at iteration n.

Figure 8.7 shows the average objective minimum trace $\ell_{\min}$, over the fifteen designs, as function of the number of function evaluations n ($n = 1, \ldots, n_{\max}$). The error bars represent plus or minus one standard error of the mean. When the maximum budget of function evaluations is exceeded, the BO algorithm stops by returning the estimated objective minimum, $\ell_{\min}$, and the point $\boldsymbol{q}_{\min}$ at which the minimum is

Table 8.5: The PDE parameters underlying the simulated data (Truth) and the estimated parameters (Estimate) using BO with the ScaledEI acquisition function. Mean and standard error over the 15 design instantiations.

	Truth	Estimate ($n = 500$)		Estimate ($n = 100$)	
		Mean	Std. Err.	Mean	Std. Err.
f_{L}	2.6×10^5	2.6005×10^5	189	2.599×10^5	287
f_{S}	50000	50003	35	50038	56
ξ	2.76	2.7603	0.0002	2.76	0.0004

attained. The vector $\boldsymbol{q}_{\min}$ represents the estimate of the true but unknown $\boldsymbol{q}^*$.

Table 8.5 shows the average and standard error, over the 15 repetitions, of the estimated point of minimum $\boldsymbol{q}_{\min}$ at iteration $n = n_{\max}$, next to the truth $\boldsymbol{q}^*$ used to generated the data. Considering that the data were corrupted by noise, the estimation (Mean) is accurate, and with reasonably small uncertainty (Std. Err.) given the scale of each parameter. Furthermore, each element of the true parameter vector $\boldsymbol{q}^*$ lies inside the 95% confidence interval obtained as the Mean plus or minus two Std. Err. in Table 8.5. We notice that the curve in Figure 8.7 is approximately flat after 100 function evaluations. For this reason, we could have effectively stopped at approximately between 100 and 200 iterations, without a substantial loss in accuracy, while reducing the overall computational time from 3 hours to less than 1 hour. In Table 8.5 we also report the estimated parameters stopping at $n = 100$ function evaluations only, where one run of the optimization takes approximately 30 minutes, compared to the 3 hours required for $n = 500$. These timings can be used for in-clinic decision support systems of practical relevance. However, for a standard optimization algorithm requiring, for example, 10^4 function evaluations, one run of the algorithm would have taken approximately 3 days, making traditional algorithms not suitable for in-clinic applications. We remark that if everybody had access to a high-performance computer (HPC), then emulation would not be needed at all. However, the whole field of emulation was born because not everybody has access to an HPC, perhaps due to not enough funding to maintain one. This can also be seen as a matter of prioritizing computer power and time. Is it better to use the

Table 8.6: The PDE parameters underlying the simulated data (Truth) and the estimated parameters (Estimate) using BO with the EI acquisition function. Mean and standard error over the 15 design instantiations.

	Truth	Estimate ($n = 500$)		Estimate ($n = 100$)	
		Mean	Std. Err.	Mean	Std. Err.
f_{L}	2.6×10^5	2.6004×10^5	236	2.6082×10^5	595
f_{S}	50000	50024	27	49983	81
ξ	2.76	2.7604	0.0001	2.7595	0.0007

high-performance computer cluster for this task or another one? One could free up space in the HPC cluster by relegating one task to the emulation approach.

We also report in Table 8.6 similar results, but obtained using the EI acquisition function. In most cases, using ScaledEI we get a lower standard error than the estimates obtained using EI. However, in one case we get a higher standard error, but a lower bias.

8.7 Summary

In this chapter I proposed a new acquisition function for Bayesian optimization (BO) which falls into the class of improvement-based policies (class 2), summarized in Chapter 6. It is based on a random variable, called *Improvement,* defined in (6.2), which quantifies the improvement on the incumbent optimum. I discussed that the established *Expected Improvement* acquisition function (6.4) does not account for the uncertainty in the *Improvement* random variable, which conveys information about our confidence in its value. To overcome this problem I derived the variance of *Improvement* in (8.1) and used it to define a new acquisition function, referred to as ScaledEI, which is the ratio of the *Expected Improvement* and the standard deviation of *Improvement*, see (8.2). The proposed acquisition function accounts for another source of uncertainty, and the scaling factor plays a role in both exploitational and explorational moves. By selecting the point that maximizes the ScaledEI policy we effectively select a point for which we expect, on average, a high degree of

improvement at high confidence.

I evaluated the performance of the proposed acquisition function on an extensive set of benchmark problems from the global optimization literature. The test suite includes problems of different dimensionality, varying from 1D to 10D, having multiple local minima and, additionally, symmetries corresponding to multiple equivalent global minima.

The performance was evaluated in terms of the $\log_{10}$ distance (in function space) to the global optimum. The results indicate that ScaledEI tends to perform as well as or better than the representative set of state-of-the-art methods from the BO literature included in our study. This suggests that by adopting a new search strategy that explicitly combines the expected improvement with its estimated uncertainty, we obtain a better trade-off between *exploration* and *exploitation*. The result is a new competitive search strategy that does not only compare favourably with other improvement-based alternatives (class 2), but also with optimistic (class 1) and information-theoretic (class 3) strategies.

Next, I presented a proof-of-concept study that confirms the reduction in the number of function evaluations required to optimize the CSF function. The proposed ScaledEI algorithm was compared to a set of widely used global optimization solvers, by reporting the number of function evaluations required to reach a given tolerance on the f value. The plot in Figure 8.6 confirms that Bayesian optimization with the proposed ScaledEI acquisition function has indeed the lowest number of function evaluations, since it uses a surrogate model of the objective function to inform the next evaluation.

Finally, I used the proposed ScaledEI algorithm for the proof-of-concept study based on a PDE fluid dynamics model of the human pulmonary circulation presented in Chapter 7. This is potentially relevant to precision medicine and non-invasive real-time diagnosis. The aim was to use the PDE model in order to give clinicians three clear indicators of pulmonary hypertension, without going through the invasive procedure of right heart catheterization. The three indicators (large vessels stiffness, small vessels stiffness and density of the structured tree, representing vascular rarefaction) are derived from the parameters of the constitutive equations of the soft

tissues, which give pathophysiological insights that are very difficult to obtain in vivo. I showed how to estimate the three parameters using the proposed ScaledEI method, introduced in Section 8.1. In particular, the estimates were obtained in a time frame that is suitable for in-clinic diagnosis and prognosis. As seen from Figure 8.6, this goal would be more challenging to achieve with conventional non-Bayesian optimization routines. Hence, the combination of the new ScaledEI method with the fledgling fluid dynamics model of the human pulmonary blood circulation system is an important first stepping stone on the pathway to an autonomous in silico clinical decision support system.

Chapter 9

Conclusions

This thesis focused on how to accelerate parameter estimation in expensive computational models using the concept of emulation, with the ultimate goal of real-time decision making and personalized diagnosis. Traditional likelihood-based estimation methods typically require running an iterative optimization procedure to maximize the log likelihood function. However, each likelihood evaluation involves a query to the computationally-expensive simulator, effectively meaning that the total time required to obtain an estimate is equal to the number of iterations times the unit cost of a single simulation.

To speed up the inference, we considered the concept of emulation, reviewed in Chapter 2. It entails replacing a computationally expensive function by a surrogate model, i.e. a statistical approximation of it based on a set of training runs. I considered two emulation targets: the simulator output and the inferential loss function. Chapter 3 reviewed the type of statistical model commonly used in the emulation literature: the Gaussian process. The approximate unbiasedness of emulating the loss was studied in Chapter 4 for a nonstandard variant of the Lotka-Volterra model of prey-predator interactions. Chapter 5 instead compared and contrasted the two emulation targets (output vs loss) in a computational model of the left ventricle. In that application we found that emulation methods lead to a reduction in the computational costs of the inference by 3 orders of magnitude, highlighting the strength of this approach.

Instead of keeping the emulator fixed, Chapter 6 described the Bayesian opti-

mization algorithm, which iteratively updates an emulator of the loss function using an adaptive strategy. This method is recommended in scenarios where the simulator is not considered to be a "stable release", but is going through some revisions and changes, and it is known that a new version is going to be released soon. It was discussed how simulating many training points, even if using massive parallelization on a computer cluster, would be computationally not optimal, since the emulator is model-specific and it would be obsolete as soon as the incumbent simulator version is made available.

Bayesian optimization relies on the choice of the adaptive strategy (the acquisition function). Section 6.3 reviewed the commonly used policies from the literature. The acquisition functions were grouped into three main classes: optimistic, improvement-based and information-based. Chapter 7 presented an application where BO with the EI policy was used to infer indicators of pulmonary hypertension risk from a PDE model of the human pulmonary blood circulation. The estimates were obtained in a time frame suitable for in-clinic decision support systems, not exceeding approximately 20 minutes. On the contrary, standard global optimization solvers which do not rely on emulation would need two and a half days.

Chapter 8 focused on a limitation of improvement-based policies, namely that they do not account for the uncertainty in the random variable *Improvement*, defined in (6.2). The Expected Improvement policy recommends points where on average the improvement is high. However, it does not account for the fact that the random variable Improvement also has variability. A point having a high expected improvement but high variability would effectively be suboptimal. If we were to query at that point, we would be evaluating an expensive simulator at a point where the improvement has a high chance of being low. In order to address this issue I derived the Variance of Improvement (VI) in (8.1) and used it to define a new acquisition function that I called ScaledEI, see (8.2). This policy recommends query points where the improvement is on average high, with high confidence. ScaledEI was tested in Section 8.3 against the three main classes of policies from the BO literature, where it turned out to be a powerful default choice for the BO algorithm. It performs at least as well as, or better, than its competitors on a wide range of benchmark

functions for global optimization having different dimensionality, sharp variation in the y-axis, multiple local minima and symmetries corresponding to multiple inputs at which the global minimum is attained. Next, the novel algorithm was compared to standard global optimization solvers, to make sure that ScaledEI leads to a reduction in the required number of function evaluations, as Bayesian optimization algorithms should do. Finally, the ScaledEI algorithm was used to infer indicators of pulmonary hypertension risk using the PDE model of the human pulmonary circulation presented in Chapter 7. The estimates obtained using the novel ScaledEI acquisition function are compared to those from the EI policy, where it turns out that the estimates from ScaledEI are more precise.

Appendix A

Detecting Convergence in Numerical Optimization Algorithms

Given a real-valued function $f(\boldsymbol{x})$ having domain $\mathcal{X} \subseteq \mathbb{R}^d$, the goal is to find a minimizer $\boldsymbol{x}_{\text{global}}$ such that

$$f(\boldsymbol{x}_{\text{global}}) = \min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x}).$$

Iterative methods start with an initial guess $\boldsymbol{x}^0$ and produce a sequence of points $\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots$ that aim to get closer to the minimizer $\boldsymbol{x}_{\text{global}}$. It is not a requirement of these methods that the point $\boldsymbol{x}^{n+1}$ must be better than $\boldsymbol{x}^n$. Hence, the best value should be considered as $\hat{\boldsymbol{x}} = \arg\min(f(\boldsymbol{x}^0), f(\boldsymbol{x}^1), f(\boldsymbol{x}^2), \ldots)$.

Most iterative optimization algorithms stop and return a point $\hat{\boldsymbol{x}}$ when at least one of the following stopping criteria is met:

- Upper bound on the number of iterations exceeded:

$$n > n_{\max};$$

- Relative step difference smaller than tolerance (convergence in domain):

$$\max_{i=1,\ldots,d} \left| \frac{x_i^{n+1} - x_i^n}{x_i^n} \right| < x_{\text{tol}};$$

- Relative function value difference smaller than tolerance (convergence in range):

$$\left| \frac{f(\boldsymbol{x}^{n+1}) - f(\boldsymbol{x}^n)}{f(\boldsymbol{x}^n)} \right| < f_{\text{tol}};$$

- If f is differentiable, maximum norm of the gradient less than tolerance:

$$\max_{i=1,\ldots,d} \left| \frac{\partial}{\partial x_i} f(\boldsymbol{x}) \right| \Bigg|_{\boldsymbol{x}=\boldsymbol{x}^n} < \nabla_{\text{tol}}.$$

Appendix B

Derivatives of Linear Combinations of Kernels

The estimation problem in (4.8) is a constrained optimization of the GP posterior mean. The GP formulation has three major advantages: the posterior mean, its gradient with respect to the input, and the Hessian are all available in closed form. Optimization solvers should exploit this fact and avoid numerical approximations of the gradient or the Hessian matrix.

In this chapter I derive analytical formulas for the gradient and the Hessian of the predictive mean for two classes of kernels: the ARD Squared Exponential and the Periodic kernel.

B.1 Supervised Learning

Let us briefly recall what a supervised regression problems consists in. The goal is to infer a mapping $f(\cdot)$ from a vector of input variables $\boldsymbol{x}$ to an output y in light of training data $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^{N}$. The function output at a training input might be different from the corresponding training target due to measurement error or others sources of noise. Given an unseen test input $\boldsymbol{x}_{N+1}$, we aim to get a prediction $\hat{f}(\boldsymbol{x}_{N+1})$, plus confidence intervals, for the corresponding output $f(\boldsymbol{x}_{N+1})$. The Bayesian framework involves specifying a prior $p(f)$ over the hypothetical functions that might have generated the data and a likelihood $p(\mathcal{D} \mid f)$ that gives the plausibility

with which each function f could be the generative model of the observed data. In this case a widely used prior over the input-output mappings f is the Gaussian process, reviewed in Chapter 3.

B.2 The Predictive Mean

Consider the supervised regression problem for a dataset $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$. Assuming a zero-mean GP prior, the posterior distribution of the latent function $p(f \mid \mathcal{D})$ is obtained by conditioning on the training data and is again a GP:

$$f(\cdot) \mid \mathcal{D} \sim \mathsf{GP}(\hat{f}(\cdot), s(\cdot,\cdot)).$$

Define $\boldsymbol{k}(\boldsymbol{x}_{N+1}) = (k(\boldsymbol{x}_1, \boldsymbol{x}_{N+1}), \ldots, k(\boldsymbol{x}_N, \boldsymbol{x}_{N+1}))$ and $\boldsymbol{K} = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1}^N$. The analytical expression for the mean prediction $\hat{f}(\boldsymbol{x}_{N+1}) = \mathbb{E}\left(f(\boldsymbol{x}_{N+1}) \mid \mathcal{D}\right)$ of the value of the function $f(\cdot)$ at an unseen point $\boldsymbol{x}_{N+1} \in \mathcal{X}$ is:

$$\begin{aligned}
\hat{f}(\boldsymbol{x}_{N+1}) &= \boldsymbol{k}(\boldsymbol{x}_{N+1})^\top \left[\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right]^{-1} \boldsymbol{y} \qquad \text{(B.1)} \\
&= \sum_{n=1}^N h_n(\boldsymbol{x}_{N+1}) y_n \qquad (\hat{f} \text{ as a linear predictor}) \\
&= \sum_{n=1}^N a_n k(\boldsymbol{x}_n, \boldsymbol{x}_{N+1}) \qquad (\hat{f} \text{ as a linear combination of kernel functions})
\end{aligned}$$

where $\boldsymbol{a} = \left[\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right]^{-1} \boldsymbol{y} \in \mathbb{R}^N$ and $\boldsymbol{h}(\boldsymbol{x}_{N+1}) = \left[\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right]^{-1} \boldsymbol{k}(\boldsymbol{x}_{N+1})$ is a vector function (called weight function) which specifies the weights to apply to targets $\boldsymbol{y}$. The posterior covariance function has the form $s(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) - \boldsymbol{k}(\boldsymbol{x})^\top \left[\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right]^{-1} \boldsymbol{k}(\tilde{\boldsymbol{x}})$ for all $\boldsymbol{x}, \tilde{\boldsymbol{x}} \in \mathcal{X}$.

B.3 The ARD Squared Exponential Kernel

This section considers the Squared Exponential (SE) kernel with Automatic Relevance Determination (ARD), discussed in Section 3.6. The covariance function has the following analytical expression for $\boldsymbol{x} = (x_1, \ldots, x_D)$, $\tilde{\boldsymbol{x}} = (\tilde{x}_1, \ldots, \tilde{x}_D) \in \mathcal{X} \subseteq \mathbb{R}^D$:

$$k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \sigma_f^2 \exp\left\{-\frac{1}{2} \sum_{d=1}^D \frac{(x_d - \tilde{x}_d)^2}{\lambda_d^2}\right\}. \qquad \text{(B.2)}$$

If we define the $D \times D$ diagonal matrix

$$\boldsymbol{L} = \text{diag}(\lambda_1^{-2}, \lambda_2^{-2}, \ldots, \lambda_D^{-2}) = \begin{pmatrix} \lambda_1^{-2} & 0 & \cdots & 0 \\ 0 & \lambda_2^{-2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_D^{-2} \end{pmatrix},$$

we can rewrite equation (B.2) in the compact matrix form

$$k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \sigma_f^2 \exp\left\{-\frac{1}{2}(\boldsymbol{x} - \tilde{\boldsymbol{x}})^\top \boldsymbol{L}(\boldsymbol{x} - \tilde{\boldsymbol{x}})\right\}. \tag{B.3}$$

B.3.1 The Predictive Mean Using the SE Kernel

If we substitute in (B.1) the analytical expression of the SE kernel (B.3), we get:

$$\begin{aligned} \hat{f}(\tilde{\boldsymbol{x}}) &= \sum_{n=1}^{N} a_n k(\boldsymbol{x}_n, \tilde{\boldsymbol{x}}) \\ &= \sigma_f^2 \sum_{n=1}^{N} a_n \exp\left\{-\frac{1}{2}(\boldsymbol{x}_n - \tilde{\boldsymbol{x}})^\top \boldsymbol{L}(\boldsymbol{x}_n - \tilde{\boldsymbol{x}})\right\}, \end{aligned} \tag{B.4}$$

where again $\boldsymbol{x}_n = (x_{n1}, x_{n2}, \ldots, x_{nD}) \in \mathbb{R}^D$ for $n = 1, \ldots, N$.

B.3.2 Gradient of the Predictive Mean

The gradient of the predictive mean (B.4) can be easily derived using Einstein's notation where a double index means a summation over its possible range. To get an analytical expression for the gradient we first focus on the quadratic form that appears in the exponential:

$$\begin{aligned} \frac{\partial}{\partial \tilde{x}_h}(x_{nk} - \tilde{x}_k)[\boldsymbol{L}]_{kl}(x_{nl} - \tilde{x}_l) &= -\delta_{hk}[\boldsymbol{L}]_{kl}(x_{nl} - \tilde{x}_l) - \delta_{lh}[\boldsymbol{L}]_{kl}(x_{nk} - \tilde{x}_k) \\ &= -[\boldsymbol{L}]_{hl}(x_{nl} - \tilde{x}_l) - [\boldsymbol{L}]_{kh}(x_{nk} - \tilde{x}_k) \\ &= -[\boldsymbol{L}]_{hl}(x_{nl} - \tilde{x}_l) - [\boldsymbol{L}]_{hk}(x_{nk} - \tilde{x}_k) \qquad \text{as } \boldsymbol{L} = \boldsymbol{L}^\top \\ &= -\left[\boldsymbol{L}(\boldsymbol{x}_n - \tilde{\boldsymbol{x}})\right]_h - \left[\boldsymbol{L}(\boldsymbol{x}_n - \tilde{\boldsymbol{x}})\right]_h \\ &= -\left[2\boldsymbol{L}(\boldsymbol{x}_n - \tilde{\boldsymbol{x}})\right]_h. \end{aligned} \tag{B.5}$$

So,

$$\nabla_{\tilde{\boldsymbol{x}}} \left\{ (\boldsymbol{x}_n - \tilde{\boldsymbol{x}})^\top \boldsymbol{L} (\boldsymbol{x}_n - \tilde{\boldsymbol{x}}) \right\} = -2\boldsymbol{L}(\boldsymbol{x}_n - \tilde{\boldsymbol{x}}). \tag{B.6}$$

This can be used to easily obtain the gradient of $\hat{f}(\tilde{\boldsymbol{x}})$ as in the following:

$$\begin{aligned} \nabla_{\tilde{\boldsymbol{x}}} \left\{ \hat{f}(\tilde{\boldsymbol{x}}) \right\} &= -\sigma_f^2 \sum_{n=1}^{N} a_n 2\boldsymbol{L}(\boldsymbol{x}_n - \tilde{\boldsymbol{x}}) \left(-\frac{1}{2} \right) \exp \left\{ -\frac{1}{2} (\boldsymbol{x}_n - \tilde{\boldsymbol{x}})^\top \boldsymbol{L} (\boldsymbol{x}_n - \tilde{\boldsymbol{x}}) \right\} \\ &= \sigma_f^2 \sum_{n=1}^{N} a_n \boldsymbol{L}(\boldsymbol{x}_n - \tilde{\boldsymbol{x}}) \exp \left\{ -\frac{1}{2} (\boldsymbol{x}_n - \tilde{\boldsymbol{x}})^\top \boldsymbol{L} (\boldsymbol{x}_n - \tilde{\boldsymbol{x}}) \right\} \\ &= \sigma_f^2 \boldsymbol{L} \sum_{n=1}^{N} (\boldsymbol{x}_n - \tilde{\boldsymbol{x}}) a_n \exp \left\{ -\frac{1}{2} (\boldsymbol{x}_n - \tilde{\boldsymbol{x}})^\top \boldsymbol{L} (\boldsymbol{x}_n - \tilde{\boldsymbol{x}}) \right\}. \end{aligned} \tag{B.7}$$

If we define the matrix $\boldsymbol{Z} \in \mathbb{R}^{N \times D}$ as $\boldsymbol{Z}^\top = [\boldsymbol{x}_1 - \tilde{\boldsymbol{x}}, \ldots, \boldsymbol{x}_N - \tilde{\boldsymbol{x}}]$, the vector $\boldsymbol{b}$ with n^{th} element $b_n = \exp \left\{ -\frac{1}{2} (\boldsymbol{x}_n - \tilde{\boldsymbol{x}})^\top \boldsymbol{L} (\boldsymbol{x}_n - \tilde{\boldsymbol{x}}) \right\}$ and the vector $\boldsymbol{c}$ with n^{th} element $c_n = a_n \times b_n$, we can now rewrite equation (B.7) more concisely as

$$\nabla_{\tilde{\boldsymbol{x}}} \left\{ \hat{f}(\tilde{\boldsymbol{x}}) \right\} = \sigma_f^2 \boldsymbol{L} \boldsymbol{Z}^\top \boldsymbol{c}. \tag{B.8}$$

B.3.3 Hessian of the Predictive Mean

By taking again the gradient wrt $\tilde{\boldsymbol{x}}$ of (B.7) and recalling (B.6) we obtain:

$$\begin{aligned} \boldsymbol{H}_{\tilde{\boldsymbol{x}}} = -\sigma_f^2 \boldsymbol{L} \sum_{n=1}^{N} a_n \exp \left\{ -\frac{1}{2} (\boldsymbol{x}_n - \tilde{\boldsymbol{x}})^\top \boldsymbol{L} (\boldsymbol{x}_n - \tilde{\boldsymbol{x}}) \right\} + \\ \sigma_f^2 \sum_{n=1}^{N} a_n \boldsymbol{L} (\boldsymbol{x}_n - \tilde{\boldsymbol{x}}) (\boldsymbol{x}_n - \tilde{\boldsymbol{x}})^\top \boldsymbol{L} \exp \left\{ -\frac{1}{2} (\boldsymbol{x}_n - \tilde{\boldsymbol{x}})^\top \boldsymbol{L} (\boldsymbol{x}_n - \tilde{\boldsymbol{x}}) \right\}. \end{aligned} \tag{B.9}$$

Let $\boldsymbol{U}^\top = [a_1 b_1 (\boldsymbol{x}_1 - \tilde{\boldsymbol{x}}), \ldots, a_N b_N (\boldsymbol{x}_N - \tilde{\boldsymbol{x}})]$. Then we can write the Hessian at $\tilde{\boldsymbol{x}}$ more concisely as

$$\boldsymbol{H}_{\tilde{\boldsymbol{x}}} = -\sigma_f^2 \boldsymbol{L} \boldsymbol{a}^\top \boldsymbol{b} + \sigma_f^2 \boldsymbol{L} \boldsymbol{Z}^\top \boldsymbol{U} \boldsymbol{L}.$$

B.4 The Periodic Kernel

If the data present hints of periodicity, the Periodic kernel is a better choice than the ARD Squared Exponential. For $\boldsymbol{x}, \tilde{\boldsymbol{x}} \in \mathcal{X} \subset \mathbb{R}^D$ it has functional form (Vanhatalo

et al., 2012, 2013):

$$k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \sigma_f^2 \exp\left\{ -\sum_{d=1}^{D} \frac{2\sin^2(\pi/\gamma(x_d - \tilde{x}_d))}{\lambda_d^2} \right\}$$

with γ controlling the inverse length of the periodicity and λ_d, as in the SE kernel, controlling the correlation decay in dimension d. With this particular choice of kernel function the posterior mean (B.1) of the GP has the following functional form:

$$\begin{aligned} \hat{f}(\tilde{\boldsymbol{x}}) &= \sum_{n=1}^{N} a_n k(\boldsymbol{x}_n, \tilde{\boldsymbol{x}}) \\ &= \sigma_f^2 \sum_{n=1}^{N} a_n \exp\left\{ -\sum_{d=1}^{D} \frac{2\sin^2(\pi/\gamma(x_{nd} - \tilde{x}_d))}{\lambda_d^2} \right\}. \end{aligned} \tag{B.10}$$

B.4.1 Gradient of the Predictive Mean

Let us focus on the following derivative first:

$$\begin{aligned} \frac{\partial}{\partial \tilde{x}_h} \left\{ -\sum_{d=1}^{D} \frac{2\sin^2(\pi/\gamma(x_{nd} - \tilde{x}_d))}{\lambda_d^2} \right\} &= -\sum_{d=1}^{D} \frac{\partial}{\partial \tilde{x}_h} \left\{ \frac{2\sin^2(\pi/\gamma(x_{nd} - \tilde{x}_d))}{\lambda_d^2} \right\} \\ &= -\sum_{d=1}^{D} \frac{1}{\lambda_d^2} \times 2 \times \frac{\partial}{\partial \tilde{x}_h} \left\{ \sin^2\left(\frac{\pi}{\gamma}(x_{nd} - \tilde{x}_d) \right) \right\} \\ &= -\sum_{d=1}^{D} \frac{2}{\lambda_d^2} \times 2\sin\left(\frac{\pi}{\gamma}(x_{nd} - \tilde{x}_d) \right) \times \\ &\quad \cos\left(\frac{\pi}{\gamma}(x_{nd} - \tilde{x}_d) \right) \left(-\frac{\pi}{\gamma}\delta_{dh} \right) \\ &= \sum_{d=1}^{D} \frac{2}{\lambda_d^2} \frac{\pi}{\gamma} \sin\left\{ 2\frac{\pi}{\gamma}(x_{nd} - \tilde{x}_d) \right\} \delta_{dh} \\ &= 2\frac{\pi}{\gamma} \sum_{d=1}^{D} \frac{1}{\lambda_d^2} \sin\left\{ 2\frac{\pi}{\gamma}(x_{nd} - \tilde{x}_d) \right\} \delta_{dh} \\ &= 2\frac{\pi}{\gamma} \frac{1}{\lambda_h^2} \sin\left\{ 2\frac{\pi}{\gamma}(x_{nh} - \tilde{x}_h) \right\}. \end{aligned} \tag{B.11}$$

Using the previous result we can derive

$$\frac{\partial}{\partial \tilde{x}_h} \hat{f}(\tilde{\boldsymbol{x}}) = \sigma_f^2 \sum_{n=1}^{N} a_n \frac{\pi}{\gamma} \frac{2}{\lambda_h^2} \sin\left\{ 2\frac{\pi}{\gamma}(x_{nh} - \tilde{x}_h) \right\} \exp\left\{ -\sum_{d=1}^{D} \frac{2\sin^2(\pi/\gamma(x_{nd} - \tilde{x}_d))}{\lambda_d^2} \right\}.$$

B.4.2 Hessian of the Predictive Mean

The Hessian of $\hat{f}(\cdot)$ at $\tilde{\boldsymbol{x}}$ has generic hk^{th} element equal to

$$
\begin{aligned}
[\boldsymbol{H}_{\tilde{\boldsymbol{x}}}]_{hk} = & \\
& \sigma_f^2 \sum_{n=1}^{N} a_n \frac{2}{\lambda_h^2} \frac{\pi}{\gamma} (-\delta_{hk}) \frac{2\pi}{\gamma} \cos\left\{ 2\frac{\pi}{\gamma}(x_{nh} - \tilde{x}_h) \right\} \exp\left\{ -\sum_{d=1}^{D} \frac{2\sin^2(\pi/\gamma(x_{nd} - \tilde{x}_d))}{\lambda_d^2} \right\} \\
& + \sigma_f^2 \sum_{n=1}^{N} a_n \left[\frac{2}{\lambda_h^2} \frac{\pi}{\gamma} \sin\left\{ 2\frac{\pi}{\gamma}(x_{nh} - \tilde{x}_h) \right\} \right] \left[\frac{2}{\lambda_k^2} \frac{\pi}{\gamma} \sin\left\{ 2\frac{\pi}{\gamma}(x_{nk} - \tilde{x}_k) \right\} \right] \times \\
& \exp\left\{ -\sum_{d=1}^{D} \frac{2\sin^2(\pi/\gamma(x_{nd} - \tilde{x}_d))}{\lambda_d^2} \right\} \\
& = \sigma_f^2 \sum_{n=1}^{N} a_n \frac{4}{\lambda_h^2} \frac{\pi^2}{\gamma^2} \times \\
& \left[-\delta_{hk} \cos\left\{ 2\frac{\pi}{\gamma}(x_{nh} - \tilde{x}_h) \right\} + \frac{1}{\lambda_k^2} \sin\left\{ 2\frac{\pi}{\gamma}(x_{nh} - \tilde{x}_h) \right\} \sin\left\{ 2\frac{\pi}{\gamma}(x_{nk} - \tilde{x}_k) \right\} \right] \times \\
& \exp\left\{ -\sum_{d=1}^{D} \frac{2\sin^2(\pi/\gamma(x_{nd} - \tilde{x}_d))}{\lambda_d^2} \right\}.
\end{aligned}
$$

Appendix C

Derivatives of the Gaussian Density

Let $\phi(z) = (\sqrt{2\pi})^{-1} \exp(-z^2/2)$ be the probability density function (pdf) of a $\mathsf{N}(0,1)$ random variable. Then,

$$\begin{aligned} \phi'(z) &= \frac{\mathrm{d}}{\mathrm{d}z}\phi(z) \\ &= \phi(z) \times \left(-\frac{1}{2} \times 2z\right) \\ &= -z\phi(z). \end{aligned} \tag{C.1}$$

The second derivative of the standard Gaussian pdf is:

$$\begin{aligned} \phi''(z) &= \frac{\mathrm{d}}{\mathrm{d}z}\phi'(z) \\ &= \frac{\mathrm{d}}{\mathrm{d}z}\left\{-z\phi(z)\right\} \\ &= -\phi(z) + (-z)(-z\phi(z)) \\ &= (z^2 - 1)\phi(z). \end{aligned} \tag{C.2}$$

Appendix D

ScaledEI with the ARD Matérn 5/2 Kernel

This section presents similar results to Section 8.4, but using a Gaussian process with the ARD Matérn 5/2 kernel. The only exception is represented by the MES policy, whose code, provided by Wang and Jegelka (2017), only allows for the ARD Squared Exponential kernel. Figure D.1 shows the full spectrum of $\log_{10}$ distances (in function space) to the global optimum, for all function evaluations ranging from $n = 100$ to 1000.

Tables D.1, D.2 and D.3 test the significance of the difference in means of the $\log_{10}$ distances at $n = 1000, 600, 200$ respectively, for ScaledEI vs each of the remaining acquisition functions, using a paired t-test with significance level 0.05. Table D.1 shows that ScaledEI always outperforms the simple RND policy, and often performs as well as (50-83%) or significantly better (17-50%) than the competing algorithms, with only one exception. ScaledEI is half of the time better than PI, followed by the information theoretic competitor MES, where ScaledEI performs significantly better 33% of the times. Then follow LCB and EI, both outperformed 25% of the times and finally the conservative MN policy, outperformed only 17% of the times. Due to the information loss inherent in reducing an entire graph to a single number, the huge variations in the results of MN get lost, but they are clear in Figure D.1. By chance a point in the initial Latin hypercube design can be near the global optimum, and this will be fine-tuned because of the excessive exploitative strategy. ScaledEI

appears to be worse than EI in only one test function. By inspecting in Figure D.1 the trace of the $\log_{10}$ distance for that function, SH10, we see that both algorithms are among the best performing methods, and the difference, even if significant, is marginal in absolute terms. Similar conclusions can be obtained from Tables D.2 and D.3.

We now compare Table D.1, obtained using the ARD Matérn 5/2 kernel and Table 8.2, for the ARD Squared Exponential. ScaledEI performed always significantly better than the RND policy. Using the Squared Exponential kernel, ScaledEI is only once significantly better than MN while this happens twice using the Matérn kernel. For the Squared Exponential kernel, the ScaledEI method has a significantly better performance in only one of the benchmark functions, compared to LCB, while in the Matérn one this happens three times. Using the Matérn kernel, half of the time ScaledEI is better than PI, while using the infinitely-differentiable kernel, this happens only one third of the times. The comparison with EI is of particular interest. The column of t-test results are the same, apart from one function: SH10. Using the Squared Exponential kernel, there is not significant difference between ScaledEI and EI, while for the Matérn one the conclusion is that ScaledEI performed significantly worse. For both kernels, ScaledEI is significantly better than EI 25% of the times. We recall that the code of MES is only available for the Squared Exponential kernel. ScaledEI using either a Squared Exponential or Matérn 5/2 kernel is 33% of the times significantly better then MES with Squared Exponential kernel, and the remaining 67% of the times they are not significantly different.

Overall, 81% of the hypothesis test labels in Tables 8.2 to 8.4 versus Tables D.1 to D.3 agree between the two kernels, while in 19% of the cases they are different. This suggests that the two kernels lead to similar results.

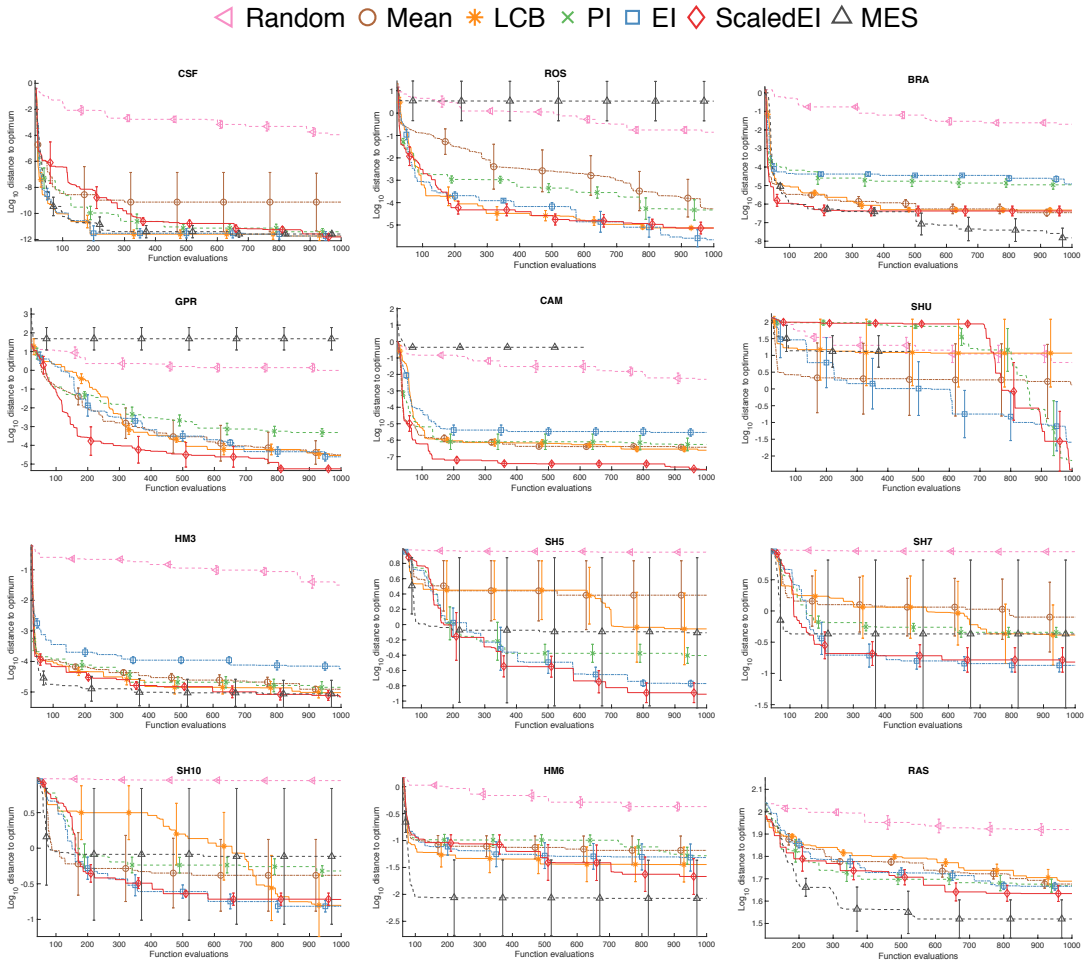


Figure D.1: Comparison of the $\log_{10}$ distances (in function space) to the global optimum. Each panel represents a given benchmark function. The traces show the average distance over the five design instantiations for the whole spectrum of iterations, while the error bars show plus or minus one standard error of the mean. These results use the ARD Matérn 5/2 kernel.

Table D.1: Statistical test for the significance in the mean difference of the final $\log_{10}$ distances. The ScaledEI acquisition function was tested against all remaining acquisition functions using a paired t-test with significance level 0.05. Codes: 0 indicates a non significant difference and 1 (-1) indicates that ScaledEI performed better (worse), i.e. it has a significantly lower (higher) average distance. These results use the ARD Matérn 5/2 kernel.

	ScaledEI vs					
Test function	RND	MN	LCB	PI	EI	MES
CSF	1	0	0	0	0	0
ROS	1	0	0	0	0	1
BRA	1	0	0	1	1	0
GPR	1	0	1	1	0	1
CAM	1	1	1	1	1	1
SHU	1	0	1	0	0	1
HM3	1	0	0	0	1	0
SH5	1	1	0	1	0	0
SH7	1	0	0	1	0	0
SH10	1	0	0	1	-1	0
HM6	1	0	0	0	0	0
RAS	1	0	0	0	0	0
Same	0%	83%	75%	50%	67%	67%
Better	100%	17%	25%	50%	25%	33%
Worse	0%	0%	0%	0%	8%	0%

Table D.2: Statistical test for the significance in the mean difference of the $\log_{10}$ distances at $n = 600$. The ScaledEI acquisition function was tested against all remaining acquisition functions using a paired t-test with significance level 0.05. Codes: 0 indicates a non significant difference and 1 (-1) indicates that ScaledEI performed better (worse), i.e. it has a significantly lower (higher) average distance. These results use the ARD Matérn 5/2 kernel.

	ScaledEI vs					
Test function	RND	MN	LCB	PI	EI	MES
CSF	1	0	0	0	0	0
ROS	1	0	0	1	0	1
BRA	1	0	0	1	1	0
GPR	1	0	0	0	0	1
CAM	1	1	1	1	1	1
SHU	-1	0	0	0	0	0
HM3	1	0	0	0	1	0
SH5	1	0	0	0	0	0
SH7	1	0	0	0	0	0
SH10	1	0	0	1	0	0
HM6	1	0	0	0	0	0
RAS	1	0	0	0	0	0
Same	0%	92%	92%	67%	75%	75%
Better	92%	8%	8%	33%	25%	25%
Worse	8%	0%	0%	0%	0%	0%

Table D.3: Statistical test for the significance in the mean difference of the $\log_{10}$ distances at $n = 200$. The ScaledEI acquisition function was tested against all remaining acquisition functions using a paired t-test with significance level 0.05. Codes: 0 indicates a non significant difference and 1 (-1) indicates that ScaledEI performed better (worse), i.e. it has a significantly lower (higher) average distance. These results use the ARD Matérn 5/2 kernel.

	ScaledEI vs					
Test function	RND	MN	LCB	PI	EI	MES
CSF	1	0	-1	0	-1	-1
ROS	1	1	0	1	0	1
BRA	1	1	1	1	1	0
GPR	1	0	1	0	0	1
CAM	1	1	1	1	1	1
SHU	0	0	0	0	0	0
HM3	1	0	0	0	1	0
SH5	1	0	0	0	0	0
SH7	1	0	0	0	0	0
SH10	1	0	0	0	0	0
HM6	1	0	0	0	0	0
RAS	1	0	0	0	0	0
Same	8%	75%	67%	75%	67%	67%
Better	92%	25%	25%	25%	25%	25%
Worse	0%	0%	8%	0%	8%	8%

Appendix E

Reproducing Kernel Hilbert Spaces

This chapter summarizes the link between Gaussian processes (GPs) and reproducing kernel Hilbert spaces (RKHS). The presentation is based on lecture notes by Sayan Mukherjee[1] and Chapter 6 of Rasmussen and Williams (2006). More details can be found in Wahba (1990). The standard regression setting applies: we have n i.i.d. data $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ from a joint distribution $P_{\boldsymbol{x},y}$ and our goal is to build an accurate predictive model $y = \hat{f}(\boldsymbol{x})$. In Chapter 3 we followed the Bayesian recipe: starting from a GP prior over a functional space, we obtained a posterior GP given data $\mathcal{D}$. The predictive mean represents the predictor $\hat{f}(\cdot)$ and we also have a measure of the predictive uncertainty given by the posterior GP variance.

A related approach is given by *regularization*. In this framework, the prior assumptions on the underlying function are specified in terms of a penalization term that prefers simpler (smooth) functions, which is considered along with a data fit term. Without any regularization, we would end up overfitting the (noisy) data. At the same time, very simple models might not fit the data at all. The tradeoff between the two terms is controlled by a regularization parameter $\lambda > 0$. The estimation problem is then reduced to the following minimization over the space of candidate functions $\mathcal{H}$:

$$\hat{f} = \arg\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(f(\boldsymbol{x}_i), y_i) + \lambda \|f\|_{\mathcal{H}}^2, \tag{E.1}$$

where $L(\cdot, \cdot)$ is a loss function measuring the cost we incur when predicting y_i by

[1] `https://www2.stat.duke.edu/~sayan/561/2015/stat_ml.pdf`

$f(\boldsymbol{x}_i)$, and $\|\cdot\|_{\mathcal{H}}$ is the norm of the space $\mathcal{H}$.

We are now left with characterizing what is a suitable space of functions for the task at hand and, to do so, we need a few definitions.

Hilbert spaces. A Hilbert space $\mathcal{H}$ is an infinite-dimensional linear space of functions, which is complete, separable, and has an inner product $\langle\cdot,\cdot\rangle_{\mathcal{H}}$. In the following, the norm is considered to be $\|f\|_{\mathcal{H}} = \sqrt{\langle f, f\rangle_{\mathcal{H}}}$.

An example is the space $L_2[a,b]$ of square-integrable functions on the interval $[a,b]$, with the inner product

$$\langle f, g\rangle = \int_a^b f(x)g(x)\mathrm{d}x.$$

This space, however, has the issue that it also includes functions which can take any arbitrary value at a finite number of points. In order to estimate smooth predictive models, this is not a desirable feature. We therefore need to consider functional spaces with elements that are "better behaved": reproducing kernel Hilbert spaces.

Evaluation functionals. An *evaluation functional* over $\mathcal{H}$ is a linear functional $\mathcal{F}_{\boldsymbol{x}} : \mathcal{H} \to \mathbb{R}$ returning the pointwise evaluation of each function at $\boldsymbol{x} \in \mathcal{X}$:

$$\mathcal{F}_{\boldsymbol{x}}[f] = f(\boldsymbol{x}) \qquad \text{for all } f \in \mathcal{H}.$$

Reproducing kernel Hilbert spaces. A *reproducing kernel Hilbert space* (RKHS) is a Hilbert space $\mathcal{H}$ with bounded evaluation functionals, i.e. there exists an $M > 0$ such that

$$|\mathcal{F}_{\boldsymbol{x}}[f]| = |f(\boldsymbol{x})| \leq M\|f\|_{\mathcal{H}} \qquad \text{for all } f \in \mathcal{H}.$$

Note that the evaluation functional in the $L_2[a,b]$ space (which is not a RKHS) is the delta function

$$\delta(x) = \begin{cases} +\infty & x = 0 \\ 0 & x \neq 0, \end{cases}$$

since $f(x) = \int_a^b f(u)\delta(u-x)\mathrm{d}u$. Furthermore, the delta function is not in $L_2[a,b]$. On the contrary, for a RKHS $\mathcal{H}$, the bounded evaluation functionals are also elements of $\mathcal{H}$.

Kernel functions. A *kernel* is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that is:

1. symmetric, i.e. $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = k(\boldsymbol{x}_j, \boldsymbol{x}_i)$;
2. positive definite, i.e. $\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0$ for any $n \in \mathbb{N}$, indices $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathcal{X}$ and $a_1, \ldots, a_n \in \mathbb{R}$.

Relationship between RKHS and kernels. Let $\mathcal{H}$ be a Hilbert space of real-valued functions on some domain $\mathcal{X}$. It can be proved that $\mathcal{H}$ is a RKHS if and only if there exists a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that:

1. for every $\boldsymbol{x} \in \mathcal{X}$, the function $k(\boldsymbol{x}, \cdot) \in \mathcal{H}$;
2. for every $f \in \mathcal{H}$, $\langle f(\cdot), k(\boldsymbol{x}, \cdot) \rangle_{\mathcal{H}} = f(\boldsymbol{x})$.

The last property is called the *reproducing property* of k. The function k is the *reproducing kernel* of $\mathcal{H}$, and it is unique and positive definite. Because both functions $k(\boldsymbol{x}, \cdot)$ and $k(\boldsymbol{x}', \cdot)$ are in $\mathcal{H}$, we further have that $\langle k(\boldsymbol{x}, \cdot), k(\boldsymbol{x}', \cdot) \rangle_{\mathcal{H}} = k(\boldsymbol{x}, \boldsymbol{x}')$, from which the name *reproducing kernel* arises.

Moore–Aronszajn theorem. To every positive definite function $k(\cdot, \cdot)$ on $\mathcal{X} \times \mathcal{X}$ corresponds a reproducing kernel Hilbert space (RKHS) and vice versa (Aronszajn, 1950).

This characterization is important because it lets us define a RKHS directly from a reproducing kernel, rather than trying to derive the kernel from the definition of the function space. Furthermore, for every positive definite $k(\cdot, \cdot)$ on $\mathcal{X} \times \mathcal{X}$ there exists a zero-mean Gaussian process having $k(\cdot, \cdot)$ as covariance function, see Wahba (1990, 1999).

Representer theorem. The representer theorem (Kimeldorf and Wahba, 1971) shows that the minimizer $\hat{f}(\cdot)$ of (E.1) over the RKHS $\mathcal{H}$ has the form:

$$\hat{f}(\boldsymbol{x}) = \sum_{i=1}^{n} a_i k(\boldsymbol{x}_i, \boldsymbol{x}),$$

for some $(a_1, \ldots, a_n) \in \mathbb{R}^n$. In other words, the solution to the regularization problem is a linear combination of the reproducing kernel evaluated at the training inputs.

This means that the optimization over the infinite dimensional space effectively boils down to a minimization over $\mathbb{R}^n$. We immediately notice the correspondence between the Gaussian process posterior mean derived in (3.27), which can also be written as a linear combination of kernels:

$$\begin{aligned}\hat{f}(\boldsymbol{x}) &= \boldsymbol{k}(\boldsymbol{x})^\top \left[\boldsymbol{K} + \sigma^2 \boldsymbol{I}\right]^{-1} \boldsymbol{y} \\ &= \boldsymbol{k}(\boldsymbol{x})^\top \boldsymbol{a},\end{aligned}$$

where $\boldsymbol{k}(\boldsymbol{x}) = (k(\boldsymbol{x}_1, \boldsymbol{x}), \ldots, k(\boldsymbol{x}_n, \boldsymbol{x}))$ and $\boldsymbol{a} = [\boldsymbol{K} + \sigma^2 \boldsymbol{I}]^{-1} \boldsymbol{y}$. The same result can be obtained by solving the regularization problem for a squared error term:

$$\hat{f} = \arg\min_{f \in \mathcal{H}} \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - f(\boldsymbol{x}_i))^2 + \frac{1}{2} \|f\|_{\mathcal{H}}^2,$$

see Section 6.2.2 in Rasmussen and Williams (2006). However, unlike the Bayesian approach, the regularization approach does not return an estimate of the predictive uncertainty, nor it specifies a method to compute the log marginal likelihood, a useful quantity for selecting the kernel hyperparameters or for model comparison.

Bibliography

Abramson, M. A., Audet, C., Dennis, J. E., and Digabel, S. L. (2009). OrthoMADS: A Deterministic MADS Instance with Orthogonal Directions. *SIAM Journal on Optimization*.

Ahmed, M. O., Shahriari, B., and Schmidt, M. (2016). Do we need "harmless" Bayesian optimization and "first-order" Bayesian optimization? *NIPS BayesOpt*.

Allen, R. P., Schelegle, E. S., and Bennett, S. H. (2014). Diverse forms of pulmonary hypertension remodel the arterial tree to a high shear phenotype. *American journal of physiology. Heart and circulatory physiology*, 307(3):H405–17.

Álvarez, M., Luengo, D., and Lawrence, N. D. (2009). Latent Force Models. In van Dyk, D. and Welling, M., editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 9–16, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA. PMLR.

Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404.

Audet, C. and Dennis, J. E. (2002). Analysis of Generalized Pattern Searches. *SIAM Journal on Optimization*.

Bergstra, J. and Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for Hyper-

Parameter Optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems (NIPS)*, pages 2546–2554.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer, New York, NY.

Bowman, A. W. and Azzalini, A. (1997). *Applied smoothing techniques for data analysis: the kernel approach with S-Plus illustrations*. Clarendon Press.

Branch, M. A., Coleman, T. F., and Li, Y. (1999). A Subspace, Interior, and Conjugate Gradient Method for Large-Scale Bound-Constrained Minimization Problems. *SIAM Journal on Scientific Computing*, 21(1):1–23.

Bull, A. D. (2011). Convergence Rates of Efficient Global Optimization Algorithms. *Journal of Machine Learning Research*, 12(Oct):2879–2904.

Byrd, R. H., Gilbert, J. C., and Nocedal, J. (2000). A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89(1):149–185.

Byrd, R. H., Schnabel, R. B., and Shultz, G. A. (1988). Approximate solution of the trust region problem by minimization over two-dimensional subspaces. *Mathematical Programming*, 40(1-3):247–263.

Calandra, R., Seyfarth, A., Peters, J., and Deisenroth, M. P. (2016). Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76(1-2):5–23.

Candès, E. and Wakin, M. (2008). An Introduction To Compressive Sampling. *IEEE Signal Processing Magazine*, 25(2):21–30.

Chabiniok, R., Wang, V. Y., Hadjicharalambous, M., Asner, L., Lee, J., Sermesant, M., Kuhl, E., Young, A. A., Moireau, P., Nash, M. P., et al. (2016). Multiphysics and multiscale modelling, data–model fusion and integration of organ physiology in the clinic: ventricular cardiac mechanics. *Interface focus*, 6(2):20150083.

Chib, S. and Jeliazkov, I. (2001). Marginal Likelihood From the Metropolis–Hastings Output. *Journal of the American Statistical Association*, 96(453):270–281.

Cohen, J. E. (2004). Mathematics Is Biology's Next Microscope, Only Better; Biology Is Mathematics' Next Physics, Only Better. *PLoS Biology*, 2(12):e439.

Conn, A. R., Gould, N., and Toint, P. (1997). A globally convergent Lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds. *Mathematics of Computation*.

Conn, A. R., Gould, N. I. M., and Toint, P. (1991). A Globally Convergent Augmented Lagrangian Algorithm for Optimization with General Constraints and Simple Bounds. *SIAM Journal on Numerical Analysis*, 28(2):545–572.

Conti, S., Gosling, J. P., Oakley, J. E., and O'Hagan, A. (2009). Gaussian process emulation of dynamic computer codes. *Biometrika*, 96(3):663–676.

Conti, S. and O'Hagan, A. (2010). Bayesian emulation of complex multi-output and dynamic computer models. *Journal of Statistical Planning and Inference*, 140(3):640–651.

Cox, D. D. and John, S. (1997). SDO: A Statistical Method for Global Optimization. In Alexandrov, N. M. and Hussaini, M. Y., editors, *Multidisciplinary Design Optimization: State of the Art*, pages 315–329.

Davidson, J. (2000). *Econometric theory*. Blackwell Publishers.

Davies, V., Noè, U., Lazarus, A., Gao, H., Macdonald, B., Berry, C., Luo, X., and Husmeier, D. (2018). Fast Parameter Inference in a Computational Model of the Left Ventricle Using Emulation. *In submission*.

Dokos, S., Smaill, B. H., Young, A. A., and LeGrice, I. J. (2002). Shear properties of passive ventricular myocardium. *American Journal of Physiology-Heart and Circulatory Physiology*, 283(6):H2650–H2659.

Fang, K., Li, R., and Sudjianto, A. (2006). *Design and modeling for computer experiments*. Chapman & Hall/CRC.

Feihl, F., Liaudet, L., Waeber, B., and Levy, B. I. (2006). Hypertension: A Disease of the Microcirculation? *Hypertension*, 48(6):1012–1017.

Friel, N. and Pettitt, A. N. (2005). Marginal likelihood estimation via power posteriors. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 70:589–607.

Gal, Y. and Turner, R. (2015). Improving the Gaussian Process Sparse Spectrum Approximation by Representing Uncertainty in Frequency Inputs. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 655–664, Lille, France. PMLR.

Gao, H., Aderhold, A., Mangion, K., Luo, X., Husmeier, D., and Berry, C. (2017). Changes and classification in myocardial contractile function in the left ventricle following acute myocardial infarction. *Journal of The Royal Society Interface*, 14(132):20170203.

Gao, H., Li, W. G., Cai, L., Berry, C., and Luo, X. Y. (2015). Parameter estimation in a Holzapfel–Ogden law for healthy myocardium. *Journal of Engineering Mathematics*, 95(1):231–248.

Garnett, R., Osborne, M. A., and Roberts, S. J. (2010). Bayesian optimization for sensor set selection. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 209–219, New York, New York, USA. ACM Press.

Gelbart, M. A., Snoek, J., and Adams, R. P. (2014). Bayesian optimization with unknown constraints. In *UAI'14 Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, pages 250–259, Quebec City, Quebec, Canada. AUAI Press.

Gentle, J. E. (2009). *Computational Statistics*. Statistics and Computing. Springer, New York, NY.

Givens, G. H. and Hoeting, J. A. (2012). *Computational Statistics*. John Wiley & Sons, Inc.

Glover, F. (1998). A template for scatter search and path relinking. In Hao, J.-K., Lutton, E., Ronald, E., Schoenauer, M., and Snyers, D., editors, *Artificial Evolution. AE 1997. Lecture Notes in Computer Science*, volume 1363, pages 1–51. Springer, Berlin, Heidelberg.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Longman Publishing Co., Inc.

Gramacy, R. B. and Apley, D. W. (2015). Local Gaussian Process Approximation for Large Computer Experiments. *Journal of Computational and Graphical Statistics*, 24(2):561–578.

Grzegorczyk, M., Aderhold, A., and Husmeier, D. (2017). Targeting Bayes factors with direct-path non-equilibrium thermodynamic integration. *Computational Statistics*, 32(2):717–761.

Guccione, J. M., McCulloch, A. D., Waldman, L., et al. (1991). Passive material properties of intact ventricular myocardium determined from a cylindrical model. *J Biomech Eng*, 113(1):42–55.

Hennig, P. and Schuler, C. J. (2012). Entropy Search for Information-Efficient Global Optimization. *Journal of Machine Learning Research*, 13(Jun):1809–1837.

Hensman, J., Durrande, N., and Solin, A. (2018). Variational Fourier Features for Gaussian Processes. *Journal of Machine Learning Research*, 18(151):1–52.

Hernández-Lobato, J. M., Hoffman, M. W., and Ghahramani, Z. (2014). Predictive Entropy Search for Efficient Global Optimization of Black-box Functions. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS)*, pages 918–926, Montreal, Canada. MIT Press.

Holzapfel, G. A. and Ogden, R. W. (2009). Constitutive modelling of passive myocardium: a structurally based framework for material characterization. *Philo-*

sophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 367(1902):3445–3475.

Huang, Y. (2016). *Multivariate Adaptive Regression Splines Based Emulation of the Heart Kinematics*. Msc thesis, University of Glasgow.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential Model-Based Optimization for General Algorithm Configuration. In Coello, C. A. C., editor, *Learning and Intelligent Optimization. LION 2011. Lecture Notes in Computer Science*, volume 6683, pages 507–523. Springer, Berlin, Heidelberg.

Huyer, W. and Neumaier, A. (1999). Global Optimization by Multilevel Coordinate Search. *Journal of Global Optimization*, 14(4):331–355.

Iacus, S. M. (2008). *Simulation and Inference for Stochastic Differential Equations*, volume 1 of *Springer Series in Statistics*. Springer, New York, NY.

Ingber, L. (1996). Adaptive simulated annealing (ASA): Lessons learned. *Control and Cybernetics*.

Jones, D. R., Perttunen, C. D., and Stuckman, B. E. (1993). Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181.

Jones, D. R., Schonlau, M., and Welch, W. J. (1998). Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization*, 13(4):455–492.

Kammann, E. and Wand, M. P. (2003). Geoadditive models. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 52(1):1–18.

Kennedy, M. C. and O'Hagan, A. (2001). Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3):425–464.

Kimeldorf, G. and Wahba, G. (1971). Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33(1):82–95.

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., and Leyton-Brown, K. (2017). Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 18(25):1–5.

Kushner, H. J. (1964). A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise. *Journal of Basic Engineering*, 86(1):97–106.

Kuss, M. (2006). *Gaussian Process Models for Robust Regression, Classification, and Reinforcement Learning*. PhD thesis, Technische Universität Darmstadt, Darmstadt, Germany.

Lagarias, J. C., Reeds, J. A., Wright, M. H., and Wright, P. E. (1998). Convergence Properties of the Nelder–Mead Simplex Method in Low Dimensions. *SIAM Journal on Optimization*, 9(1):112–147.

Lazarus, A., Husmeier, D., and Papamarkou, T. (2018). Multiphase MCMC Sampling for Parameter Inference in Nonlinear Ordinary Differential Equations. In Storkey, A. and Perez-Cruz, F., editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 1252–1260, Playa Blanca, Lanzarote, Canary Islands. PMLR.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2016). Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *eprint arXiv:1603.06560*.

Lizotte, D., Wang, T., Bowling, M., and Schuurmans, D. (2007). Automatic gait optimization with Gaussian process regression. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 944–949.

Locatelli, M. and Schoen, F. (2013). *Global optimization: theory, algorithms, and applications*. Society for Industrial and Applied Mathematics (SIAM).

Meng, X.-L. and Wong, W. H. (1996). Simulating ratios of normalizing constants via a simple identity: a theoretical exploration. *Statistica Sinica*, 6:831–860.

Mezura-Montes, E. and Coello Coello, C. A. (2011). Constraint-handling in nature-inspired numerical optimization: Past, present and future. *Swarm and Evolutionary Computation*, 1(4):173–194.

Mingari Scarpello, G. and Ritelli, D. (2003). A New Method for the Explicit Integration of Lotka-Volterra Equations. *Divulgaciones Matemáticas*, 11.

Mockus, J. (1975). On Bayesian methods for seeking the extremum. In Marchuk, G. I., editor, *Optimization Techniques IFIP Technical Conference. Optimization Techniques 1974. Lecture Notes in Computer Science*, volume 27, pages 400–404. Springer, Berlin, Heidelberg.

Mockus, J. (1977). On Bayesian Methods for Seeking the Extremum and their Application. In *IFIP Congress*, pages 195–200.

Mockus, J. (1989). *Bayesian Approach to Global Optimization*, volume 37 of *Mathematics and Its Applications*. Springer, Dordrecht, Netherlands.

Mockus, J., Tiesis, V., and Zilinskas, A. (1978). *The application of Bayesian methods for seeking the extremum*, volume 2. North-Holland.

Nash, J. C. (1990). *Compact numerical methods for computers: linear algebra and function minimisation*. CRC press.

Nikou, A., Dorsey, S. M., McGarvey, J. R., Gorman, J. H., Burdick, J. A., Pilla, J. J., Gorman, R. C., and Wenk, J. F. (2016). Computational modeling of healthy myocardium in diastole. *Annals of biomedical engineering*, 44(4):980–992.

Noè, U., Chen, W., Filippone, M., Hill, N., and Husmeier, D. (2017). Inference in a Partial Differential Equations Model of Pulmonary Arterial and Venous Blood Circulation Using Statistical Emulation. In Bracciali, A., Caravagna, G., Gilbert, D., and Tagliaferri, R., editors, *Computational Intelligence Methods for Bioinformatics and Biostatistics. CIBB 2016. Lecture Notes in Computer Science*, volume 10477, pages 184–198. Springer, Cham, Switzerland.

Noè, U., Filippone, M., and Husmeier, D. (2015). Emulation of ODEs with Gaussian processes. In *Proceedings of the 30th International Workshop on Statistical Modelling*, pages 191–194.

Noè, U. and Husmeier, D. (2018). On a New Improvement-Based Acquisition Function for Bayesian Optimization. *eprint arXiv:1808.06918*.

O'Hagan, A. (2006). Bayesian analysis of computer code outputs: A tutorial. *Reliability Engineering & System Safety*, 91(10-11):1290–1300.

Olufsen, M. S. (1999). Structured tree outflow condition for blood flow in larger systemic arteries. *American Journal of Physiology-Heart and Circulatory Physiology*, 276(1):H257–H268.

Olufsen, M. S., Hill, N. A., Vaughan, G. D. A., Sainsbury, C., and Johnson, M. (2012). Rarefaction and blood pressure in systemic and pulmonary arteries. *Journal of fluid mechanics*, 705:280–305.

Osborne, M. a., Garnett, R., and Roberts, S. J. (2009). Gaussian Processes for Global Optimization. *Proceedings of the 3rd Learning and Intelligent OptimizatioN Conference (LION 3)*, pages 1–15.

Pasetto, M. E., Husmeier, D., Noè, U., and Luati, A. (2017a). Statistical Inference in the Duffing System with the Unscented Kalman Filter. In *Proceedings of the 32nd International Workshop on Statistical Modelling*, pages 119–122, Groningen.

Pasetto, M. E., Noè, U., Luati, A., and Husmeier, D. (2017b). Inference with Unscented Kalman Filter and Optimization of Sigma Points. In Petrucci, A. and Verde, R., editors, *SIS 2017. Statistics and Data Science: new challenges, new generations. Proceedings of the Conference of the Italian Statistical Society*, pages 767–772, Florence. Firenze University Press.

Pedersen, M. E. H. (2010). Good Parameters for Particle Swarm Optimization. *Technical Report HL1001*.

Qureshi, M. U., Vaughan, G. D. A., Sainsbury, C., Johnson, M., Peskin, C. S., Olufsen, M. S., and Hill, N. A. (2014). Numerical simulation of blood flow and pressure drop in the pulmonary arterial and venous circulation. *Biomechanics and Modeling in Mechanobiology*, 13(5):1137–1154.

Rasmussen, C. E. (2004). Gaussian Processes in Machine Learning. In Bousquet, O., von Luxburg, U., and Rätsch, G., editors, *Advanced Lectures on Machine Learning. ML 2003. Lecture Notes in Computer Science*, volume 3176, pages 63–71. Springer, Berlin, Heidelberg.

Rasmussen, C. E. and Ghahramani, Z. (2003). Bayesian Monte Carlo. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*, pages 489–496, Cambridge, MA, USA. MIT Press.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian processes for machine learning*. MIT Press.

Remme, E. W., Hunter, P. J., Smiseth, O., Stevens, C., Rabben, S. I., Skulstad, H., and Angelsen, B. (2004). Development of an in vivo method for determining material properties of passive myocardium. *Journal of biomechanics*, 37(5):669–678.

Roberts, S. and Everson, R. (2001). *Independent component analysis: principles and practice*. Cambridge University Press.

Roberts, S., Osborne, M., Ebden, M., Reece, S., Gibson, N., and Aigrain, S. (2012). Gaussian processes for time-series modelling. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1984):20110550–20110550.

Rosenkranz, S. and Preston, I. R. (2015). Right heart catheterisation: best practice and pitfalls in pulmonary hypertension. *European Respiratory Review*, 24(138):642–652.

Santner, T. J., Williams, B. J., and Notz, W. I. (2003). *The Design and Analysis of Computer Experiments*. Springer Series in Statistics. Springer, New York, NY.

Särkkä, S., Álvarez, M. A., and Lawrence, N. D. (2017). Gaussian Process Latent Force Models for Learning and Stochastic Control of Physical Systems. *eprint arXiv:1709.05409*.

Särkkä, S. and Solin, A. (2018). *Applied Stochastic Differential Equations*. Cambridge University Press.

Sermesant, M., Moireau, P., Camara, O., Sainte-Marie, J., Andriantsimiavona, R., Cimrman, R., Hill, D. L., Chapelle, D., and Razavi, R. (2006). Cardiac function estimation from mri using a heart model and data assimilation: advances and difficulties. *Medical Image Analysis*, 10(4):642–656.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. (2016). Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1):148–175.

Smith, N., de Vecchi, A., McCormick, M.and Nordsletten, D., Camara, O., Frangi, A., Delingette, H., Sermesant, M., Relan, J., Ayache, N., et al. (2011). euheart: personalized and integrated cardiac care using patient-specific cardiovascular modelling. *Interface focus*, pages 349–364.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS)*, pages 2951–2959. Curran Associates, Inc.

Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. W. (2012). Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting. *IEEE Transactions on Information Theory*, 58(5):3250–3265.

Sun, K., Stander, N., Jhun, C.-S., Zhang, Z., Suzuki, T., Wang, G.-Y., Saeed, M., Wallace, A. W., Tseng, E. E., Baker, A. J., et al. (2009). A computationally efficient formal optimization of regional myocardial contractility in a sheep with left ventricular aneurysm. *Journal of biomechanical engineering*, 131(11):111001.

Titsias, M. (2009). Variational Learning of Inducing Variables in Sparse Gaussian Processes. In van Dyk, D. and Welling, M., editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 567–574, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA. PMLR.

Turner, R. and Rasmussen, C. E. (2012). Model based learning of sigma points in unscented Kalman filtering. *Neurocomputing*, 80:47–53.

Ugray, Z., Lasdon, L., Plummer, J., Glover, F., Kelly, J., and Martí, R. (2007). Scatter Search and Local NLP Solvers: A Multistart Framework for Global Optimization. *INFORMS Journal on Computing*, 19(3):328–340.

van der Vaart, A. W. and van Zanten, J. H. (2009). Adaptive Bayesian estimation using a Gaussian random field with inverse Gamma bandwidth. *The Annals of Statistics*, 37(5B):2655–2675.

Vanhatalo, J., Riihimäki, J., Hartikainen, J., Jylänki, P., Tolvanen, V., and Vehtari, A. (2012). Bayesian Modeling with Gaussian Processes using the GPstuff Toolbox. *eprint arXiv:1206.5754*.

Vanhatalo, J., Riihimäki, J., Hartikainen, J., Jylänki, P., Tolvanen, V., and Vehtari, A. (2013). GPstuff: Bayesian Modeling with Gaussian Processes. *Journal of Machine Learning Research*, 14(Apr):1175–1179.

Wahba, G. (1990). *Spline models for observational data*. Society for Industrial and Applied Mathematics (SIAM).

Wahba, G. (1999). *Support Vector Machines, Reproducing Kernel Hilbert Spaces and the Randomized GACV*. MIT Press, Cambridge, MA.

Wang, H. M., Gao, H., Luo, X. Y., Berry, C., Griffith, B. E., Ogden, R. W., and Wang, T. J. (2013a). Structure-based finite strain modelling of the human left ventricle in diastole. *International Journal for Numerical Methods in Biomedical Engineering*, 29(1):83–103.

Wang, H. M., Luo, X. Y., Gao, H., Ogden, R. W., Griffith, B. E., Berry, C., and Wang, T. J. (2014). A modified Holzapfel-Ogden law for a residually stressed finite strain model of the human left ventricle in diastole. *Biomechanics and Modeling in Mechanobiology*, 13(1):99–113.

Wang, V., Nielsen, P., and Nash, M. (2015). Image-based predictive modeling of heart mechanics. *Annual review of biomedical engineering*, 17:351–383.

Wang, Z. and Jegelka, S. (2017). Max-value Entropy Search for Efficient Bayesian Optimization. In *Proceedings of the 34th International Conference on Machine Learning (PMLR)*, pages 3627–3635.

Wang, Z., Mohamed, S., and Freitas, N. (2013b). Adaptive Hamiltonian and Riemann Manifold Monte Carlo Samplers. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning (PMLR)*, volume 28, pages 1462–1470, Atlanta, Georgia, USA. PMLR.

Watanabe, S. (2010). Asymptotic Equivalence of Bayes Cross Validation and Widely Applicable Information Criterion in Singular Learning Theory. *Journal of Machine Learning Research*, 11(Dec):3571–3594.

Watanabe, S. (2013). A Widely Applicable Bayesian Information Criterion. *Journal of Machine Learning Research*, 14(Mar):867–897.

Widmaier, E. P., Hershel, R., and Strang, K. T. (2016). *Vander's Human Physiology: The Mechanisms of Body Function*. McGraw-Hill Education, 14 edition.

Wilkinson, R. D. (2014). Accelerating ABC methods using Gaussian processes. *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*.

Wood, S. N. (2017). *Generalized additive models: an introduction with R*. CRC press.

Xi, J., Lamata, P., Lee, J., Moireau, P., Chapelle, D., and Smith, N. (2011). Myocardial transversely isotropic material parameter estimation from in-silico measurements based on a reduced-order unscented kalman filter. *Journal of the mechanical behavior of biomedical materials*, 4(7):1090–1102.

Xi, J., Shi, W., Rueckert, D., Razavi, R., Smith, N. P., and Lamata, P. (2014). Understanding the need of ventricular pressure for the estimation of diastolic biomarkers. *Biomechanics and modeling in mechanobiology*, 13(4):747–757.