DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Multifidelity Gaussian Processes for Uncertainty Quantification

Martin Klapacz

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Multifidelity Gaussian Processes for Uncertainty Quantification

Multifidelity Gauß Prozesse zur Quantifizierung von Unsicherheit

Author:	Martin Klapacz
Supervisor:	Prof. Dr. Hans-Joachim Bungartz
Advisor:	Kislaya Ravi, Dr. rer. nat. Tobias Neckel
Submission Date:	15.03.2021

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.03.2021 Martin Klapacz

Acknowledgments

I would like to express my sincere gratitude to my advisor Kislaya Ravi for his continuous support of my thesis, for his patience, his assistance and knowledge.

Abstract

Forward uncertainty quantification is used to obtain useful insight of many physical models. However, it is a challenge to get accurate results for a given amount of computational resources, when dealing with complex models. There are different methods to tackle such problems. In this work we combine two of such techniques namely, polynomial chaos expansion and multi-fidelity to create an efficient method to solve forward UQ problems. Firstly, we develop multi-fidelity Gaussian process regression models which fuse information from the low-fidelity model, derivative approximations of it and the parameter space. We also use a composite kernel which further improves the regression results. Thereafter, those models request adaptively new high-fidelity data points which improve the currently existing surrogate the best. Finally, we perform polynomial chaos expansion on Gaussian process surrogates to estimate the statistical moments of a quantity of interest. Combining the aforementioned methods leads to an efficient approach for solving forward UQ problems. In this work, we deal with only two level of fidelity. Our evaluation of the method shows considerable savings of computational effort.

Contents

1 Introduction

In many scientific research fields such as physical engineering and applied mathematics we often encounter a recurring kind of problem. Our goal is to study the properties of highly complex and stochastic systems. The insights obtained from these system are used to design and implement new technologies. In uncertainty quantification (UQ) we formalize this problem and calculate the statistical moments of such non-deterministic systems [30, 32]. However, we are limited in terms of available computational resources. Thus, UQ is a challenging problem that requires sophisticated methods to obtain knowledge about the quantity of interest.

Analysing such complex systems requires us to design appropriate models. In such problem settings we deal with trade-offs between precise but costly and cheap but inaccurate models. Multi-fidelity methods can be deployed in such problem settings. They fuse information obtained from cheap but inaccurate and accurate but costly models to enhance accuracy of the results. While ordinary single-fidelity approaches would use one of the available models for further treatment, more sophisticated multi-fidelity models try to learn cross-correlations between all available data sources using machine learning methods [18, 19, 20, 27]. In recent years multi-fidelity modelling has gained a lot of attention, as it has been successfully applied to various data-driven problems [11, 26, 28]. A detailed review of numerous aspects of multi-fidelity is presented in [25]. Other multi-fidelity approaches from numerical analysis are for example Richardson extrapolation [42], which takes different resolutions into account and significantly improves the rate of convergence. Multiresolution methods [4] and multiscale solvers deployed in computational physics [3] are used to numerically solve PDEs and make use of multi-fidelity modelling, as well. In this work we will use multi-fidelity modelling to solve forward UQ problems.

1.1 Motivation

Polynomial chaos expansion (PCE) is one way of performing UQ. It is an especially efficient way to calculate the statistical moments for computational expensive functions [30]. The number of function evaluation depends upon the complexity of the function and the number of stochastic parameters. Our goal is to minimize the number of costly evaluations and at the same time approximate the statistical moments accurately. One of the ways to achieve this is to adaptively choose the evaluation points [6, 9]. The flowchart of the process is shown in figure 1.1a.

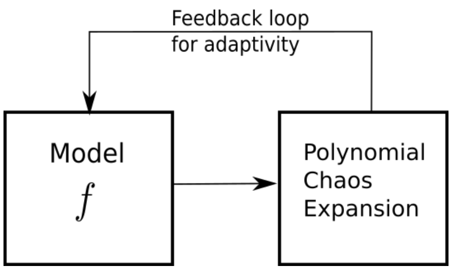
As described in the previous section, combining different fidelity models to reduce the overall computational cost has garnered significant attention in recent times. We can combine multi-fidelity and adaptation to efficiently perform polynomial chaos expansion. Figure 1.1b shows the general workflow of the algorithm. The details of this approach can be found in [23, 41]. The basic idea is to build a polynomial chaos expansion using a range of low-fidelity models and then use a high-fidelity model to determine and add the correction term. However, modelling highly non-linear correction relations requires us to evaluate a comparatively high number of evaluation points. Thus, we propose the use of Gaussian processes [29] to solve this issue.

Gaussian processes are stochastic processes where the posterior statistical moments are calculated using Bayesian statistics. There are many methods to build multi-fidelity Gaussian process (GP) regression models [27, 19, 18]. In this work we will build multi-fidelity Gaussian process surrogates and then perform PCE on it. The workflow of this approach is illustrated in figure 1.1c. Moreover, we will use a procedure in which we choose the function evaluation points adaptively in order to use the computational resources as efficiently as possible.

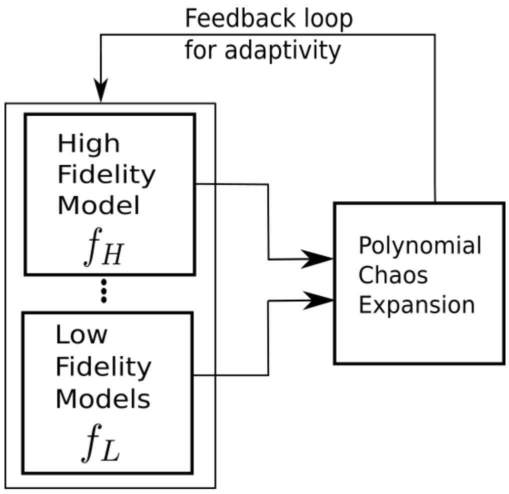
1.2 Outline

In the first part of this work we will introduce the basic theory, which is necessary for the methods described and tested in this work and which are covered in the second part. The last part illustrates test cases and performances of those methods.

Chapter 2 covers the theory behind polynomial chaos expansion. We will explain what kind of problem it is applied to and how it works. In chapter 3 we will describe Gaussian process regression. We will introduce Gaussian processes and go over to inference and optimization. Chapter 4 covers the multi-fidelity methods of this work. In the first section we introduce and formalize the idea behind multi-fidelity. In the following sections we will revisit existing multi-fidelity methods such as AR1 [18], NARGP [27] and Data Fusion GPs [20]. We start with AR1, as it is fundamental for NARGP. Data Fusion GPs is an extension of NARGP and will be explained after AR1 and NARGP. Next, we will introduce an own multi-fidelity model that combines adaptation and the before mentioned multi-fidelity methods. Section 4.6 gives an overview of the implementation of this work, which we used to obtain the results of this approach. In chapter 5 we visually illustrate the effects of adaptation on the model accuracy and compare it with randomly sampled training without adaptation. Afterwards, we test the accuracy of multi-fidelity models using adaptation in more specific problem settings which are likely occur in real-world problems. Finally, we perform PCE in combination with multi-fidelity modelling and compare it with direct GPC.



(a) Classical Approach



(b) Multifidelity polynomial chaos expansion

```
figures/mggp_pce_flowchart .png
```

(c) Proposed method (Combining multifidelity GP and PCE)

Figure 1.1: Workflow depicting different approached to efficiently perform polynomial chaos expansion

2 Polynomial Chaos Expansion

In real world applications we come across many physical systems where the parameters are stochastic. This causes the output to be stochastic too. Approximating the statistical moments of the output uncertainty falls under the category of Forward Uncertainty problems. Efficiently solving forward UQ problems is crucial for both theoretical and practical applications in scientific computing. In this chapter, we discuss a method which can be applied to such problems.

We start by providing a formal mathematical definition of forward UQ problems and state some existing methods to solve them in section 2.1. Thereafter, we explain polynomial chaos expansion (PCE) in detail in section 2.2. In the end, we briefly present sensitivity analysis in section 2.3, which is used to determine the relative contribution of each stochastic parameter to the total variance.

2.1 Forward Uncertainty Quantification problems

Firstly, we introduce the underlying problem setting of this chapter. Let us define a function $f(X, \Omega)$ expecting two types of input parameters. X is a vector of deterministic parameters and Ω is a vector of stochastic parameters. We assume the values $x \in X$ as well as the probability distribution of all the stochastic parameters $\omega \in \Omega$ to be given. Due to the stochastic parameters Ω the output of f is also stochastic, too. In the following, we call the required output Quantity of Interest (QoI). Our target is to find the the statistical moments of QoI. These kinds of problems are termed as Forward UQ problems. Figure 2.1 shows the structure of a generic forward UQ problem.

2.1.1 Methods for Forward UQ

There are several ways to solve a forward UQ problem. The three most commonly used methods are the following:

1. **Monte Carlo Method**: In Monte Carlo methods [15] we draw samples of the stochastic parameter Ω. We evaluate the model f for every sample. Each evaluation represents a sample from the output distribution. So, we can directly calculate the statistical moments of the QoI using the calculated function values.

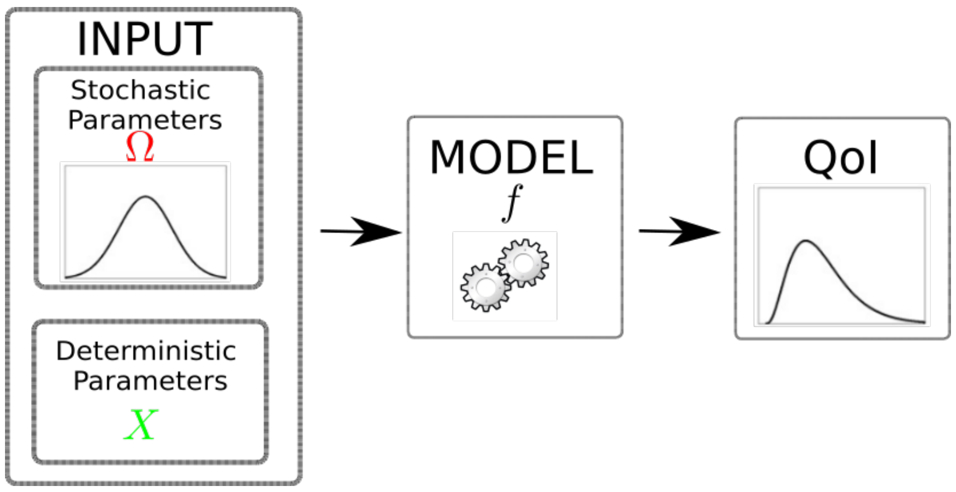


Figure 2.1: Forward UQ problem

This method is simple to implement. Moreover, the algorithm is embarrassingly parallelisable. However, this method also has some glaring disadvantages. The error term e is proportional to $\sigma/\sqrt{N}$, where N represents the sample size [22]. The square root error convergence rate is very slow. So, we need to evaluate f very often to get a small error. This leads to high and impractical computational requirements of the method. Moreover, high variance is also a commonly faced issue. To solve this, we use various methods like Quasi Monte Carlo method [15], Importance Sampling [35], etc.

2. **Polynomial Chaos Expansion**: As the name suggests, polynomial chaos expansion expresses the function as linear combination of polynomials [30, 40]. The first step of PCE is to separate the deterministic variable and the stochastic variable. Then, we span the model using polynomials. The family of the polynomials depends upon the distribution of the stochastic variable Ω. The deterministic parameters decides the constant term X. The details of this method will be discussed later in this chapter.

3. **Stochastic Galerkin Method**: This method is similar to the polynomial chaos expansion. However, it relies on the knowledge of the functional structure to determine the statistical moments [30].

2.2 Polynomial Chaos Expansion

The basic idea behind polynomial chaos expansion is to write the function f as a linear combination of polynomials [30]:

$$f(X,\Omega) = \sum_{m=0}^{\infty} \hat{f}(X)\phi_m(\Omega) \tag{2.1}$$

The polynomials ϕ_m depend only on the stochastic terms Ω. The coefficient terms $\hat{f}$ depend on the deterministic inputs X. We typically truncate the infinite expansion in equation 2.1 to M terms:

$$f(X,\Omega) \approx \sum_{m=0}^{M-1} \hat{f}(X)\phi_m(\Omega) \tag{2.2}$$

In order to perform PCE, we firstly need to choose the polynomials ϕ_m. Then, we need to calculate the coefficients $\hat{f}$. Thereafter, we need to select the truncation limit M. After finishing those three steps, we need to answer the question how to calculate the statistical moments of f using this expansion. In the next subsections, we will show how to perform those four steps.

2.2.1 Orthogonal polynomials

The first step is to choose the correct family of polynomials. As stated earlier, the polynomials only depend on the distribution of the stochastic input parameters. For the sake of simplicity, we start the description with the number of stochastic parameters being one (one dimensional stochastic space). Later, we generalise the stochastic space to multiple dimensions.

Let $\rho(\omega)$ be the probability density function defined on the stochastic input domain Ω. The polynomials are orthonormal with respect to the given probability density:

$$< \phi_i(\omega), \phi_j(\omega) >_{\rho(\omega)} = \delta_{ij} \tag{2.3}$$

where, $< \phi_i(\omega), \phi_j(\omega) >_{\rho(\omega)}$ is the inner product of the two polynomials $\phi_i(\omega)$ and $\phi_i(\omega)$ with respect to the density $\rho(\omega)$. The inner product is defined as follows:

$$< \phi_i(\omega), \phi_j(\omega) >_{\rho(\omega)} = \int_{supp(\rho)} \phi_i(\omega)\, \phi_j(\omega)\, \rho(\omega)\, d\omega$$

δ_{ij} is the Kronecker delta function:

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases}$$

Distribution	Density function	Polynomial basis	Support
Normal	$\frac{1}{\sqrt{2\pi}}e^{\frac{-x^2}{2}}$	Hermite $H_n(x)$	$[-\infty, \infty]$
Uniform	$\frac{1}{2}$	Legendre $P_n(x)$	$[-1, 1]$
Exponential	e^{-x}	Laguerre $L_n(x)$	$[0, \infty]$
Beta	$\frac{(1-x)^\alpha(1+x)^\beta}{2^{\alpha+\beta+1}B(\alpha+1,\beta+1)}$	Jacobi $P_n^{(\alpha,\beta)}(x)$	$[-1, 1]$

Table 2.1: List of the orthonormal polynomial basis based on the probability density function [23]

Table 2.1 shows the list of the orthonormal basis functions based on the probability density function. We present a more detailed description of the two most commonly used distributions namely standard uniform distribution and normal distribution.

Legendre Polynomials

Legendre polynomials [30] are orthogonal with respect to the uniform distribution $\mathcal{U}[-1, 1]$ with density function:

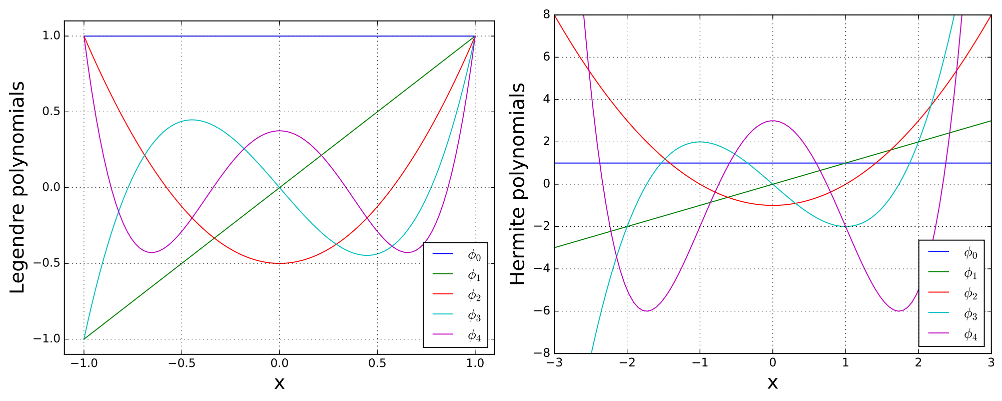
$$\rho(\omega) = \frac{1}{2}$$

The first five legendre Polynomials as shown in figure 2.2a are:

$$\begin{aligned}
\phi_0 &= 1 \\
\phi_1 &= \omega \\
\phi_2 &= \frac{1}{2}\left(3\omega^2 - 1\right) \\
\phi_3 &= \frac{1}{2}\left(5\omega^3 - 3\omega\right) \\
\phi_4 &= \frac{1}{8}\left(35\omega^4 - 30\omega^2 + 3\right)
\end{aligned} \tag{2.4}$$

However, Legendre polynomials are not normalised. So in order to be orthonormal and not just orthogonal, we need to divide by its norm. The analytical formula to calculate the norm of the Legendre polynomials is:

$$\int_{-1}^{1} \phi_i(\omega)\,\phi_i(\omega)\,\rho(\omega)\,d\omega = \frac{2}{2i+1} \tag{2.5}$$



(a) Legendre Polynomials

(b) Hermite Polynomials

Figure 2.2: Orthogonal polynomials

Hermite Polynomials

Hermite polynomials [30] are orthogonal with respect to the normal distribution $\mathcal{N}(0,1)$ with density function:

$$\rho(\omega) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$$

The first five Hermite polynomials as shown in figure 2.2b are:

$$\begin{aligned}
\phi_0 &= 1 \\
\phi_1 &= \omega \\
\phi_2 &= \omega^2 - 1 \\
\phi_3 &= \omega^3 - 3\omega \\
\phi_4 &= \omega^4 - 6\omega^2 + 3
\end{aligned} \tag{2.6}$$

However, the polynomials are again not normalised. This is why, we again need to divide by the corresponding norm. The analytical formulation of the Hermite polynomial norm is as follows:

$$\int_{-1}^{1} \phi_i(\omega)\, \phi_i(\omega)\, \rho(\omega)\, d\omega = i! \tag{2.7}$$

Multidimensional Stochastic domain

We assume the number of stochastic parameters to be d. Additionally, we assume the random vector Ω to be a vector of d independent random variables:

$$\Omega = (\omega_1, \omega_2, ..., \omega_d)$$

Let us consider a multi-index $m = (m_1, m_2, ..., m_d) \in \mathbb{N}_0^d$. We represent $\phi_{m_i}(\omega_i)$ as an orthonormal polynomial corresponding to stochastic variable ω_i with degree m_i. The multivariate polynomial can written as a product of univariate polynomials

$$\Phi_m(\Omega) = \prod_{i=1}^{d} \phi_{m_i}(\omega_i) \tag{2.8}$$

Let $\mathcal{M}$ represent set of all multi-indices. Now, the multidimensional polynomial chaos expansion can be written as

$$f(X, \Omega) = \sum_{m \in \mathcal{M}} \hat{f}(X) \Phi_m(\Omega) \tag{2.9}$$

2.2.2 Coefficients

The next step in polynomial chaos expansion is to calculate the coefficients $\hat{f}_k(X)$. In this work, we will discuss pseudo-spectral projection [39, 40, 30] which is used to calculate the coefficients. Please note that there are other ways to calculate the coefficients like Stochastic collocation [16].

In order to evaluate the coefficients, we exploit the orthonormality of the underlying basis functions. Our goal is to evaluate the coefficients $\hat{f}_k(x)$, which depend on the multi-index k. We project the left and right side of equation 2.9 to the polynomial Φ_k. Then, we make use of the orthonormality of the polynomials Φ_i to calculate the coefficient $\hat{f}_k(X)$. The calculation is summarised as following:

$$\begin{aligned}
f(X, \Omega) &= \sum_{m \in \mathcal{M}} \hat{f}(X) \Phi_m(\Omega) \\
< f(X, \Omega), \Phi_k(\Omega) >_\rho &=< \sum_{m \in \mathcal{M}} \hat{f}(X) \Phi_m(\Omega), \Phi_k(\Omega) >_\rho \\
< f(X, \Omega), \Phi_k(\Omega) >_\rho &= \sum_{m \in \mathcal{M}} \hat{f}(X) \underbrace{< \Phi_m(\Omega), \Phi_k(\Omega) >_\rho}_{\delta_{mk}} \\
< f(X, \Omega), \Phi_k(\Omega) >_\rho &= \hat{f}_k(X)
\end{aligned}$$

So we evaluate $\hat{f}_k(X)$ by projecting the model f on the corresponding orthonormal polynomial. However, we have to solve an integration problem which can be done using quadrature methods like Gaussian quadrature etc.

$$\begin{aligned}\hat{f}_k(X) &= < f(X,\Omega), \Phi_k(\Omega) >_\rho \\ &= \int f(X,\Omega)\Phi_k(\Omega)\rho(\Omega)d\Omega \\ &\approx \sum_{q_j \in Q, w_j \in W} f(X,q_j)\Phi(q_j)w_j\end{aligned}$$

where Q and W represent sets of quadrature points and weights, respectively.

2.2.3 Choice of multi-index set

The order of the polynomials in PCE depends on the model f. However, more complicated models require higher order polynomials. Making a suitable choice of the polynomial order is a very important step, which has a strong impact on the accuracy of the results. However, the higher the order of the polynomials, the more quadrature points need to be evaluated. This will be a challenge for computationally expensive problems. The process becomes even more difficult when we deal with multi-dimensional stochastic domains. Due to the curse of dimensionality, the number of quadrature points increases rapidly as the dimension of the problem increases. There exist various adaptive algorithms that helps us adaptively choose the multi-indices. The details of such algorithms are beyond the scope of this work. Interested reader can refer [6, 9] for details.

2.2.4 Statistical moments

Computing the stochastic moments of a QoI turns out to be surprisingly easy when working with polynomial chaos expansion. More precisely, we can use the coefficients directly to calculate the statistical moments. For the sake of simplicity we assume the stochastic domain to be one-dimensional. From subsection 2.2.1 we know that the order 0 polynomials are $\phi_0 = 1$. We calculate the expectation value $f(x,\omega)$ as follows:

$$\begin{aligned}\mathbb{E}[f(x,\omega)] &\approx \mathbb{E}\left[\sum_{m=0}^{M-1} \hat{f}_m(x)\,\phi_m(\omega)\right] \\ &= \sum_{m=0}^{M-1} \hat{f}_m(x)\,\mathbb{E}[1\cdot\phi_m(\omega)] \\ &= \sum_{m=0}^{M-1} \hat{f}_m(x)\,\underbrace{\mathbb{E}[\phi_0(\omega)\,\phi_m(\omega)]}_{=\delta_{0m}} = \hat{f}_0(x)\end{aligned}$$

The expectation value is the first coefficient. This concept can be generalized to multidimensional stochastic problems, too. In those cases the expectation value will be equal to the coefficient with zero multi-index:

$$\mathbb{E}[f(X,\Omega)] = \hat{f}_{m_0} \tag{2.10}$$

with $m_0 = \underbrace{(0,\dots,0)}_{d \text{ times}}$.

We derive the variance of f similarly. Again, we start with one-dimensional stochastic problems:

$$\begin{aligned}
\mathbb{V}[f(t,\omega)] &= \mathbb{E}\left[(f(t,\omega) - \mathbb{E}[f(t,\omega)])^2\right] \\
&\approx \mathbb{E}\left[\left(\sum_{n=0}^{N-1} \hat{f}_n(t)\,\phi_n(\omega) - \hat{f}_0(x)\right)^2\right] \\
&= \mathbb{E}\left[\left(\sum_{n=1}^{N-1} \hat{f}_n(t)\,\phi_n(\omega)\right)^2\right] \\
&= \sum_{n=1}^{N-1} \hat{f}_n^2(t)\,\underbrace{\mathbb{E}\left[\phi_n(\omega)^2\right]}_{=1} = \sum_{n=1}^{N-1} \hat{f}_n^2(t)
\end{aligned}$$

We can see that the variance of f is the squared sum of all coefficients which do not corresponding to the zero order polynomial ϕ_0. Of course, this can be extended to multi-dimensional settings:

$$\mathbb{V}[f(X,\Omega)] = \left(\sum_{m\in\mathcal{M}} \hat{f}_m^2\right) - \hat{f}_{m_0}^2 \tag{2.11}$$

where $m_0 = \underbrace{(0,\dots,0)}_{d \text{ times}}$ and $\mathcal{M}$ represents the set of multi-indices representing the order of polynomials.

2.3 Sensitivity Analysis

The goal of sensitivity analysis is determine the relative contribution of a stochastic parameter to the total variance. We will give a brief explanation of the global Sobol sensitivity index [**Sobol2001global**]. Suppose, we have d stochastic input parameters and we want to calculate the Sobol index of the $j-$th parameter. $\mathcal{M}$ is the multi-index set which represents the order of the individual parameters of all polynomials in PCE. Let us define $\mathcal{M}_j$ as following:

$$\mathcal{M}_j = \{m \in \mathcal{M} | m_j \neq 0\} \tag{2.12}$$

where m_j is the j-th element of the multi-index. The global Sobol sensitivity index is defined as:

$$\mathcal{S}_j = \frac{\sum_{m \in \mathcal{M}_j} \hat{f}_m}{\sum_{m \in \mathcal{M}} \hat{f}_m} \tag{2.13}$$

$\mathcal{S}_j$ represents the contribution of the $j-$th stachastic parameter to the total variance. In order to reduce the complexity of the model, we can compute the Sobol indezes of all stochastic parameters and neglect inputs with a very small contribution.

3 Gaussian Process Regresssion

In this chapter, we provide a basic description of Gaussian progress regression. We mainly focus on the subtopics, which are important for the following chapters. We start with some probability theory in section 3.1. This lays the foundations for the following Gaussian processes. Afterwards, we will discuss posterior predictions using Gaussian processes in section 3.2 and section 3.3. Finally, we will describe some approaches to influence and improve a GP's prediction performance.

3.1 Multivariate Gaussian distributions

Gaussian distributions, also called normal distributions, are the basic building blocks of Gaussian process regression and therefore fundamental for the methods described in this paper. A multivariate Gaussian distributed random variable $X \sim \mathcal{N}(\mu, \Sigma)$ is fully defined by its mean vector $\mu \in \mathbb{R}^d$ and its covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$. Its probability density function is defined as:

$$f_X(x) = \frac{1}{\sqrt{(2\pi)^p det(\Sigma)}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right). \tag{3.1}$$

μ and Σ are defined as follows:

$$\mu = \mathbb{E}[X] = \underset{x \in \mathbb{R}^d}{\arg\max} f_X(x) \tag{3.2}$$

$$\Sigma = cov[X] = \mathbb{E}[(X-\mu)(X-\mu)] \tag{3.3}$$

Gaussian distributed random variables come with a few useful characteristics [2]. Their mean is equal to their expectation value and to their mode. For a multivariate Gaussian distribution $X = (X_1, \ldots, X_d)$, the correlation between X_i and X_j is determined by Σ_{ij}.

High positive values of Σ_{ij} implicate strong correlations between X_i and X_j, whilst small values implicate weak correlations. Conditional distribution and marginalization of Gaussian distributions are relevant for this work. Therefore, before starting with the regression part, those two concepts are introduced in the following.

3.1.1 Marginalization

Assuming a multivariate Gaussian distribution is given and its mean and components can be organized like this:

$$\begin{pmatrix} X_1 \\ X_2 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}\right) \tag{3.4}$$

As the covariance matrix Σ is symmetric, Σ_{11} and Σ_{22} are symmetric as well. Additionally, Σ_{21} and Σ_{21} are reflections along the diagonal from each other. This is why $\Sigma_{21} = \Sigma_{21}^T$ holds. The marginalization property of Gaussian distributions states that all subvectors of the samples from equation 3.4 are Gaussian distributed as well [29]. Therefore X_1 and X_2 have the following distributions:

$$\begin{aligned} X_1 &\sim \mathcal{N}\left(\mu_1, \Sigma_{11}\right) \\ X_2 &\sim \mathcal{N}\left(\mu_2, \Sigma_{22}\right) \end{aligned} \tag{3.5}$$

3.1.2 Conditional distribution

Let us consider two random variables X and Y are jointly Gaussian. According to [2], their conditional distribution $X|Y$ will be Gaussian as well. Therefore, the conditional distribution $X_1|X_2 \sim \mathcal{N}\left(\mu_{1|2}, \Sigma_{1|2}\right)$ from equation 3.4 is Gaussian as well defined by the following parameters:

$$\begin{aligned} \mu_{1|2} &= \mu_1 - \Sigma_{12}\Sigma_{22}^{-1}\left(x_2 - \mu_2\right) \\ \Sigma_{1|2} &= \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} \end{aligned} \tag{3.6}$$

3.2 Gaussian Processes

Formally, Gaussian processes are defined as collections of random variables, any finite number of which have a joint Gaussian distribution [29]. The size of this collection or the collection in general is neither fixed nor predefined. Instead, Gaussian processes should be rather understood as distributions over functions $f(x)$ which can be evaluated at positions $x_1, \ldots, x_n$ [2]. Those evaluations can be arranged to a vector and correspond to random variables which are jointly Gaussian distributed. In order to be well defined, this multivariate Gaussian distribution needs well defined statistics, i.e. mean vector and covariance matrix. However, as the number of random variables is not fixed and may be infinite, Gaussian processes are typically defined as

$$f(x) \sim \mathcal{GP}\big(m(x), k(x, x')\big) \tag{3.7}$$

where $m(x)$ is the mean function and $k(x, x')$ specifies the kernel function. Equivalently to multivariate normal distributions the following identities hold [2]:

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

The kernel function $k(x, x')$ is a means of expressing a certain notion of correlation between the function values $f(x)$ and $f(x')$. This correlation becomes stronger with increasing similarity between x and x' [2]. The radial basis kernel, or RBF kernel, is the most popular kernel function and is defined as

$$k(x, x') = \sigma^2 \exp\left(-\frac{-\|x - x'\|^2}{2l}\right) \tag{3.8}$$

with $\sigma, l \in \mathbb{R}$ being free parameters which we currently assume to be given. For a given set of values $x_1, \ldots, x_n$ we can build a so called Gram matrix $K \in \mathbb{R}^{\times}$ whose entries are defined as $K_{ij} = k(x_i, x_j)$. We can define a Gaussian process with mean function $m(x) = 0$ and RBF kernel 3.8 function to create a Gaussian $X \sim \mathcal{N}(0, K)$ from which we can generate random samples. If we have chosen a sufficiently high number of input points $x_1, \ldots, x_n$, we can plot a few of these randomly generated high dimensional sample vectors like ordinary functions and see, that Gaussian distributions indeed can be seen as distributions over functions.

Figure 3.1 illustrates a number of random samples. The smoothness and behavior of samples depend upon the kernel function. The radial basis functions generates samples $f \in C^\infty$ that are infinitely differentiable [29]. Other kernel functions generates random samples with other properties. For details on other type of kernel function refer to [29].

3.3 Inference

In this section we explain how to use Gaussian processes for regression. The regression workflow strongly resembles Bayesian Inference [2], where we formulate a prior to incorporate our domain specific knowledge and belief about the behavior of the model. Then we use the empirical data to update the prior. This results in the posterior model which is a compromise between our prior believes and the empirical data.

Equivalently, in GP regression the prior model is a Gaussian Process defined as $\mathcal{GP}(0, k(x, x'))$. For the sake of symmetry the mean function is typically defined as $m(x) = 0$. The choice of a proper kernel function $k(x, x')$ is a crucial design decision

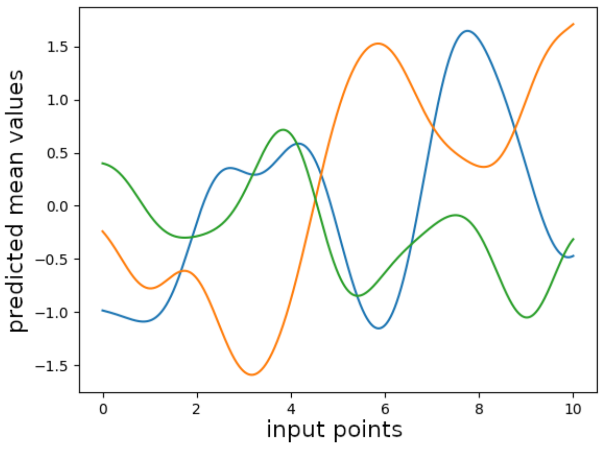


Figure 3.1: Sample vectors $Y \in \mathbb{R}^{1000}$ plotted as function curves and generated from a Gaussian process with RBF kernel and input points $x_1, \ldots, x_{1000} \in [0, 10]$

and offers a great opportunity for the developer to directly influence the shape of the regression curve. In Gaussian process regression training the model with the data set $D = \{(x_i, y_i) \,|i = 1, \ldots, N\}$ is equivalent to updating the prior with the empirical data in Bayesian statistics. First of all, we formulate the joint distribution of the training and test target values f and f_* using the training and test inputs X and X_* [29]:

$$\begin{pmatrix} f \\ f_* \end{pmatrix} \sim \mathcal{N}\left(0, \begin{pmatrix} k(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{pmatrix}\right) \tag{3.9}$$

$K(X, X)$ or $K(X_*, X_*)$ are the Gram matrizes of the training input values X and test input values X_*, respectively. $K(X, X_*)$ consists of the covariances between X and X_*. As the distribution in equation 3.9 is Gaussian, we apply the conditional property introduced in subsection 3.1.2 to formulate the predictive posterior distribution

$$f_* | X_*, X, f \sim \mathcal{N}\left(\mu_{pred}, \Sigma_{pred}\right) \tag{3.10}$$

with parameters

$$\mu_{pred} = K(X_*, X)K(X, X)^{-1} f \tag{3.11}$$

$$\Sigma_{pred} = K(X_*, X_*) - K(X_*, X)K(X, X)^{-1}K(X, X_*) \tag{3.12}$$

where we condition the joint Gaussian prior distribution on the training data [29]. We get a multivariate Gaussian distribution where the dimensionality is equal to the number of test inputs. This can be interpreted as single variate normal distributions with mean μ_i^{pred} and variance Σ_{ii}^{pred} for each test input point X_i. As stated in equation 3.2, for each test point the most probable target value is the respective mean. An advantage of Gaussian process regression is the fact that each target value prediction comes with its uncertainty, i.e. the respective variance. The higher the variance value, the less confident is the GP about its mean prediction target value. Exactly like the prior model mentioned previously, the trained GP is nothing but a Gaussian and can therefore generate sample functions conditioned on the training points.

From equation 3.11 and 3.12 we can observe that in order to calculate the predicted posterior mean and variance, we need to compute the inverse of the covariance matrix K. This is typically done using Cholesky decomposition [33]. The complexity of cholesky decomposition is cubic with the number of data points $\mathcal{O}(N^3)$. This cubic complexity is the main issue against Gaussian process for big data application. However, in literature there exists many methods to reduce the complexity of Gaussian processes [1].

3.4 Kernel algebra

A function $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is a valid kernel function if and only if the corresponding Gram matrix K is positive semi-definite [2]. This is the case, if k can be written as $k(x, x') = \phi(x)^T \phi(x')$ and $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^n$ is a basis function we know from ordinary regression tasks [2]. A useful characteristic of valid kernel functions is that there is a set of simple operations defined on them which form a closed algebra. Meaning, we can combine different kernel function to get new ones. For valid kernels $k_1(x, x')$ and $k_2(x, x')$ and $c > 0$ the following composite kernels will be valid as well [2]:

$$\begin{aligned} k(x, x') &= ck_1(x, x') \\ k(x, x') &= k_1(x, x') + k_2(x, x') \\ k(x, x') &= k_1(x, x')k_2(x, x') \end{aligned} \tag{3.13}$$

Later in this work, we will intensively use a composite kernel function which can be created using the rules in equation 3.13.

3.5 Hyperparameter Optimization

The behavior of a GP regressor will strongly depend on the chosen kernel function. In section 3.3 we have assumed that the hyperparameters in the kernel functions are given. However, we should choose values for those hyperparameters that improve the prediction performance of the model. This is done by maximizing the log-likelihood of the training data with respect to the kernel hyperparameters θ. The log likelihood is given as:

$$\ln p(y|\theta) = -\frac{1}{2}\ln\det(\Sigma) - \frac{1}{2}y^T\Sigma^{-1}y - \frac{N}{2}\ln(2\pi) \tag{3.14}$$

There are multiple gradient based optimization algorithms in literature which solve this problem [38]. They use partial derivatives of equation 3.14 to maximize the log-likelihood [2]. Its partial derivatives are defined as follows:

$$\frac{\partial}{\partial\theta_i}\ln p(x|\theta) = -\frac{1}{2}Tr\left(\Sigma^{-1}\frac{\partial\Sigma}{\partial\theta_i}\right) + \frac{1}{2}x^T\Sigma^{-1}\frac{\partial\Sigma}{\partial\theta_i}\Sigma^{-1}x. \tag{3.15}$$

Generally, the log probability is not convex. Therefore, it is advised to repeat this procedure multiple times with different initialization values to increase the probability of finding the global maximum.

3.6 Automatic Relevance Determination (ARD)

In a multidimensional setting, we can use ARD to improve the model performance. We modify the kernel function by assigning a dedicated hyperparameter to each input dimension [2]. The standard RBF kernel from equation 3.8 would result in

$$k_{ARD}^{(d)}(x,x') = \sigma^2\exp\left(-\sum_{i=1}^{d}\frac{w_i(x_i - x_i')^2}{2l}\right) \tag{3.16}$$

with parameters $\theta = (\sigma, l, w_1, \dots, w_d)$. Hyperparameter optimization will then initialize all parameters θ with values that maximize the log-likelihood of the training data X. It is likely that some ARD weights w_i get relatively high values while others almost diminish. An explanation of this phenomenon is that often predictions mostly depend on a smaller subset of input entries, meaning some input dimensions are less relevant for the prediction than others [2]. In equation 3.16, each w_i assigns a relevance value to the i-th dimension of x. In order to reduce the complexity of the hyperparameter optimization problem we can neglect dimensions with extremely small ARD weights. Figure 3.2 illustrates the optimization run of a three dimensional kernel with ARD weights w_1 (red), w_2 (blue) and w_3 (green). With increasing number of optimization steps the blue weight slowly diminishes. Thus, we could reduce the dimensionality of the problem by ignoring the blue input without really reducing the model's accuracy.

We can also extend kernel 3.16 with e.g. ARD weighed multiplicative and constant components in order to express other underlying trends, if they exist:

$$k_{ARD2}^{(d)}(x,x') = k_{ARD}^{(d)}(x,x') + \sum_{i=1}^{d} w_{i+d}x_i x_i' + w_{2d+1} \tag{3.17}$$

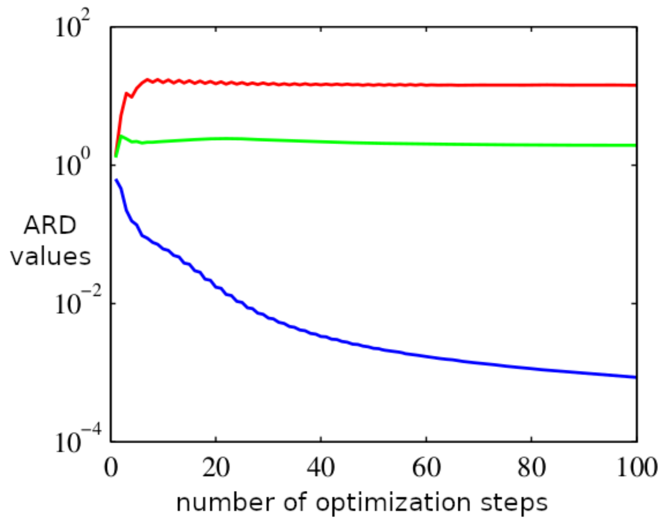


Figure 3.2: ARD weight evolution during optimization [2]

However, each new ARD weight increments the dimensionality of the hyperparameter optimization problem. Due to the curse of dimensionality, with linearly growing dimensionality the space growths exponentially. Therefore, we are more prone to finding an unsatisfactory local optimium, while the optimization process becomes more expensive. This trade-off between flexibility and computational cost has to be considered when making use of the ARD framework.

4 Multi-fidelity Gaussian Processes

In this chapter we make the step from single-fidelity to multi-fidelity problem settings. Firstly, we explain and formalize the term *Multi-fidelity*. Then we introduce a range of multi-fidelity algorithms, which are based on Gaussian Process Regression described in chapter 3. All those algorithms are extensions of each other, meaning we start with the simplest and end with more sophisticated ones. In the last section we extensively present a new approach, which combines some features of the methods described before.

4.1 Multifidelity

In many different scientific fields measuring real world data is fundamental for design and decision making. When working with different measure devices or computer simulations, data comes with a certain quality and quantity describing the underlying phenomenon. In situations like this people usually have to make a trade-off between working with few and precise data called high-fidelity or plentiful but inaccurate low-fidelity data. In this chapter we are going to discuss multi-fidelity algorithms, which can be used in this setting to take multiple data sets of different fidelities into consideration. Our goal is to leverage our model performance with high accuracy from the high-fidelity data and the data-richness from the low-fidelity data. In this section we will introduce a formalization of this problem setting, which we will use in the following sections.

4.1.1 Multifidelity Models

Let us define multi-fidelity models [25] as functions $f : \mathcal{Z} \to \mathcal{Y}$ mapping an input $z \in \mathcal{Z} \subseteq \mathbb{R}^d$ to an output $y \in \mathcal{Y} \subseteq \mathbb{R}$. Each of those model evaluations comes with a certain workload $c \in \mathbb{R}^+$. We distinguish between the high-fidelity model $f_{hi} : \mathcal{Z} \to \mathcal{Y}$ with evaulation workload $c_{hi} \in \mathbb{R}^+$ and low-fidelity models $f_{lo}^{(i)} : f : \mathcal{Z} \to \mathcal{Y}$ with evaluation workloads $c_{lo}^{(i)} \in \mathbb{R}^+$. Evaluating f_{hi} is much more expensive than evaluating $f_{lo}^{(i)}$:

$$\forall i = 0, \ldots, N-1 : c_{lo}^{(i)} < c_{hi}$$

However, while we consider the high-fidelity model to be a comparitively accurate approximation of the underlying function f_{exact}, i.e. $f_{exact} \approx f_h$, all $f_{lo}^{(i)}$ return bad approximations of f_{exact}. In theory, they can be understood as versions of f_{exact}, that have been corrupted by a transformation function

$$f_{lo}^{(i)} = h_{trans}^{(i)} \circ f_{exact}.$$

The more complex the underlying transformation function $h_{trans}^{(i)}$, the worse is the approximation performance of $f_{lo}^{(i)}$. This means, a low-fidelity model f_{lo}^{1} with an underlying complex nonlinear transformation function such as $h_{trans}^{(1)}(x) = 2x^2 + \sin(x)$ will give worse approximations of f_{exact} than a low-fidelity model with a simple linear transformation such as $h_{trans}^{(2)}(x) = 1.2x$. We can easily simulate different low-fidelity functions using such transformations with different complexities. Figure 4.1 illustrates

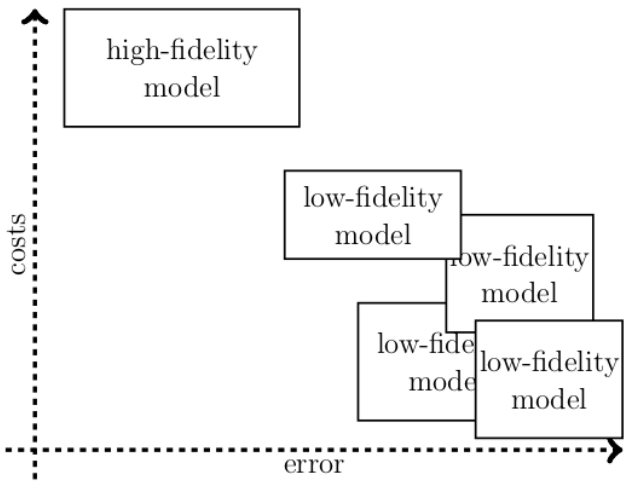


Figure 4.1: Illustration of a trade-off between low- and high-fidelity models [25].

an example multi-fidelity setting. We have multiple models, each being a compromise between cost and error. The most expensive but precise model is the high-fidelity model. In addition to that, we have a cluster of low-fidelity models in an area of high error and low costs.

4.1.2 Multifidelity Model Management Strategies

A model management strategy typically specifies how to combine and apply different fidelity levels [25]. Components of algorithms discussed in the following sections can be categorized as implementations of such management strategies.

Adaptation

Adaptation iteratively updates a low-fidelity model using input-output-pairs of a high-fidelity model [25]. While the high-fidelity model must provide a sufficient accuracy, the low-fidelity model must be able to improve its own performance using input-output pairs of the high-fidelity model. The low-fidelity models, which are derived by the input-output pairs of high-fidelity models are also called data-fit models [25]. Gaussian process regression, discussed in chapter 3, and interpolation in general [25] are examples of data-fit models. After such an adaptation process, the cheap low-fidelity model turns into a more precise model, which can be used to make more accurate and still cheap input evaluations.

Fusion

The second model management strategy is Information fusion. Some multi-fidelity algorithms expect us to merge, concatenate or combine information from different fidelity sources and to pass it as an input [25]. Information fusion plays an important part in multi-fidelity models.

4.2 Autoregressive Schemes (AR1)

In this section we introduce the first and most basic multi-fidelity algorithm introduced by Kennedy and O'Hagan in 2000 [18]. It forms the basis for the multi-fidelity algorithms, which follow in the next sections. An ordinary Gaussian process discussed in chapter 3 expects a single data set and is not able to take multiple qualities of different data sets into consideration. Nonetheless, various multi-fidelity methods are based on Gaussian processes. In this section, we assume to have a range of data sets $D_1, \dots, D_s$ of increasing quality and decreasing sizes $n_1 > \cdots > n_s$. Each D_t is a set of input output pairs (x, y) with $x \in X_t \subseteq \mathbb{R}^d$ being a d-dimensional input point and $y \in Y_t \subseteq \mathbb{R}$ being a corresponding one-dimensional target value.

The recursive GP regression framework introduced by Kennedy and O'Hagan [18] is a simple multi-fidelity method whose key idea can be expressed in the following equation:

$$f_t(x) = \rho_{t-1} f_{t-1}(x) + \delta_t(x) \tag{4.1}$$

$f_t(x)$ is a prediction function representing on D_t. $\delta_t(x)$ is an independent GP with own mean μ_t and covariance function $k_t(x, x')$. Equation 4.1 gives rise to a linear interconnection between all fidelity levels with constant regression parameter $\rho_{t-1} \in \mathbb{R}$ that specifies the strength of the correlation between the fidelity levels. For a given value $f_{t-1}(x)$ we can infer nothing more about f_t, if we have additional information in the form of $f_{t-1}(x')$ with $x \neq x'$. This characteristic is known as the Markov property [18]. Replacing the GP priors with their respective predictive posterior distributions f_t^* results in an inference scheme consisting of a number of chained standard GP regression problems [18]. In order to compute a prediction, we have to pass the input to the GP of lowest fidelity $f_1(x)$ which is a standard Gaussian process trained on D_1 and propagate the results through the entire recursive chain built by equation 4.1 from lowest to highest fidelity level. The last GP's prediction will be the prediction of the overall model. This method transforms the prediction task into s standard GP prediction problems. [19] introduces an improvement of this method. The constant in equation ρ_{t-1} is replaced by a space-dependent scaling factor

$$\rho_{t-1}(x) = \frac{Cov(f_t(x), f_{t-1}(x))}{var(f_{t-1}(x))}.$$

In a multi-fidelity setting, autoregressive approaches should be used instead of standard GPs as they make use of multiple data sets with different qualities. However, [27] shows that the autoregressive scheme in [18] is less flexible than other more sophisticated multi-fidelity algorithms. It is unable to learn more complex cross-correlations with non-linearities. Hence, we describe more sophisticated methods in the next sections.

4.3 Nonlinear autoregressive multi-fidelity GPs (NARGP)

As we want to work with more complex fidelity cross-correlations, we now discuss Nonlinear Information Fusion Gaussian processes (NARGP). NARGP can be seen as a generalisation of the autoregressive scheme, where we replace equation 4.1 by

$$f_t(x) = g_t(x, f_{t-1}^*(x)). \tag{4.2}$$

The constant ρ and the Gaussian process δ_t are implicitly included in g_t [27]. $g_t : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}$ is a deterministic mapping that jointly maps model inputs and output from the underlying fidelity level to the output of the current fidelity level t. g can be trained by maximizing the marginal likelihood. We can get an intuitive understanding of what g actually is by taking a look at figure 4.2. For the sake of simplicity we assume that we have only two fidelity levels $f_l : \mathbb{R} \to \mathbb{R}$ specifying the low-fidelity model and $f_h : \mathbb{R} \to \mathbb{R}$ specifying the high-fidelity model. Both models expect one-dimensional inputs, i.e. $d = 1$.

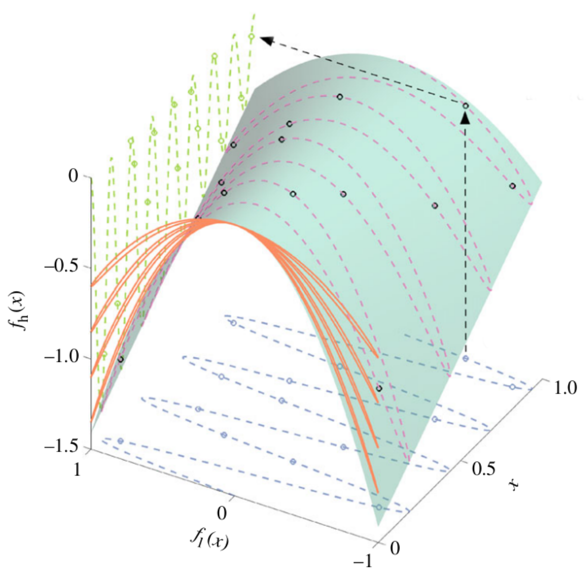


Figure 4.2: Graphical representation [27] of the manifold function g (green) in equation 4.2

We try to learn a manifold mapping g whose input dimension is the input dimension of $f_l + 1$. Therefore, we can visualize g as a 3D-manifold which is defined on the entire input space $\mathbb{R}^2$. Its specific shape, is spanned by the high-fidelity training points which live in the manifold. From a visual point of view, we can break down the prediction procedure in two steps [20]. Firstly, we map the input $x \in \mathbb{R}$ on its augmented version $(x, f_l(x))$. Then we map the augmented version of x to the manifold. The height of the manifold at this point will be the approximations of $f_h(x)$. g is therefore the mean prediction function of a Gaussian process trained on the augmented data set $D_* = \{(x, f_l(x))\,|x \in D\}$. In practical applications the mean predict function f_l can either be defined as a closed function or by data [20]. We can use a closed low-fidelity function directly for augmentation. If only low-fidelity data is available, we train a Gaussian process on that data and use its prediction function as f_l. This consideration can be generalized to $s > 2$ fidelity sets, where the procedure visualized in figure 4.2 is repeated $s - 1$ times.

4.3.1 Composite NARGP kernel

Fusing input points and posterior predictions from two completely different spaces gives raise to the question, if a standard RBF kernel is still a reasonable choice for a covariance function. A special composite kernel [27] better incorporates the autoregressive nature in equation 4.1:

$$k_{NARGP}(x,x') = k_1\left(x,x';\theta_1\right) k_2\left(f^*_{t-1}(x), f^*_{t-1}(x');\theta_2\right) + k_3\left(x,x';\theta_3\right) \tag{4.3}$$

The kernel combination rules listed in 3.13 are used to build this composite kernel. We will refer to this kernel as NARGP kernel. The subkernels k_1, k_2 and k_3 are RBF kernels with ARD weights:

$$k_1(x,x';\theta_1) = \sigma_1^2 \exp\left(-\frac{\sum_{i=1}^d w_{1i}(x_j - x'_j)^2}{2l_1}\right) \text{with } \theta_1 = (\sigma_1, w_{11}, \dots, w_{1d}, l_1) \tag{4.4}$$

$$k_2(x,x';\theta_2) = \sigma_2^2 \exp\left(\frac{w_2\left(f_{t-1}(x) - f_{t-1}(x')\right)^2}{2l_2}\right) \text{with } \theta_2 = (\sigma_2, w_2, l_2) \tag{4.5}$$

$$k_3(x,x';\theta_3) = \sigma_3^2 \exp\left(-\frac{\sum_{i=1}^d w_{3i}(x_j - x'_j)^2}{2l_3}\right) \text{with } \theta_3 = (\sigma_3, w_31, \dots, w_3d, l_3) \tag{4.6}$$

Notice that kernel k_2 expects a 1-dimensional input, while k_1 and k_3 expect d-dimensional inputs. k_2 is responsible for taking the low-fidelity prediction into account. Each subkernel has its own hyperparameter vector. $\theta = (\theta_1, \theta_2, \theta_3)$ is therefore the hyperparameter vector of k_{NARGP}. At this point we could also choose different kernel classes for k_1, k_2 and k_3. We could also use a mixture of different kernel classes. Again, this is a point where the developer has a lot of freedom to influence the behavior of the model. In this work we will only use RBF kernels for k_1, k_2, k_3.

4.3.2 NARGP workflow

Equipped with basic knowledge of how NARGP works, we summarize the workflow of a NARGP application [27] with multiple fidelity levels:

First data set

For the lowest data set D_1 we train a GP with kernel $k_{NARGP}(x,x';\theta_1)$ and optimize the kernel parameters by maximizing the log-likelihood using equation 3.14. Training to model scales with $\mathcal{O}(n_1^3)$ where n_1 is the size of the first data set, which is greater than the rest of the data sets [27].

Subsequent data sets

For all subsequent data sets $D_2, \dots, D_s$ we create GPs $f_i(x)$ and train them on augmented data sets $D_i^* = \{((x, f_{i-1}(x); y(x)) | (x,y) \in D_i\}$. So each GP f_i, with $i > 1$ will be trained using input data of dimension $d+1$ which is augmented using the prediction of the previous GP. Unlike the first GP, all subsequent GPs have therefore input dimensionality of $d+1$. Again, each training phase is followed by an optimization phase as described in section 3.5, using a gradient based optimizer with multiple restarts [38]. Each GP treatment comes again with a cubic complexity of $\mathcal{O}(n_i^3)$, but as the data set size n_i decreases, the workload of each GP treatment becomes smaller.

Prediction

At this points all GPs are trained and optimized. For a given test set X^* we make predictions by passing X^* to the first GP, whose result will be given to the next GP and used to augment X^*. The result of each inner GP is propagated through the entire chain of models. The result of the highest fidelity layer is the prediction of the NARGP model.

4.4 Multi-fidelity Data Fusion GPs (DFGP)

The NARGP framework is more powerful than the autoregressive scheme, because it is able to find nonlinear cross-correlations between data sets. However, we can further improve the regression process by indirectly using the derivative information of low fidelity models. In this section, we introduce NARGP linked with with data-driven manifold embeddings [20], which is a modification of the NARGP framework described in section 4.3. The goal of this approach is to learn more complicated non-linearities. To keep things simple, we assume to have have only one low-fidelity f_l and one high-fidelity model f_h. We generalizes the approximation manifold function in equation 4.2 to

$$f_h(t) = g(t, f_l(t), f_l^{(1)}(x), \dots, f_l^{(K)}(x)) \tag{4.7}$$

by taking the first K derivatives into account. Of course, this requires f_h to be at least K times differentiable. Another requirement assumed in [20] is:

$$\| \frac{\partial}{\partial x_i} g(x_1, \dots, x_{K+2}) \|_{L^\infty} \leq \epsilon \in \mathbb{R} \tag{4.8}$$

Each entry in each partial derivative of g is bounded by a small constant. We should ask ourselves if there exists a function that the model can learn using its derivatives and if those derivatives are helpful at all. Firstly, we have to understand the underlying idea. We can write each real and infinitely differentiable function f as its corresponding

Taylor series [12] defined as

$$f(x) = \sum_{k=0}^{\infty} \frac{(x-x_0)^k}{k!} f^{(k)}(x).$$

We can formulate f_h evaluated at a neighborhood point $t + \tau \in \mathbb{R}^d$ in a similar way [20]:

$$f_h(t+\tau) = f_h(t) + \frac{\partial}{\partial t} f_h(t)\tau + \mathcal{O}\left(\| \frac{\partial^2}{\partial t^2} f_h \| \right)$$

As we assume that the first K derivatives exist, we can hope that g takes the form a truncated version of f_h's Taylor series which should be a reasonable approximation which we can use. Equation 4.8 makes sure that the truncations don't lead to a too high error. But if we wanted to directly learn the truncated Taylor series of f_h, we would need multiple of its derivatives. However, the closed form derivatives of f_h are not available. A workaround of this problem are finite difference stencils used to approximate a functions derivatives [5]. For example, we can approximate the first two derivatives of a function $f : \mathbb{R} \to \mathbb{R}$ as [5]:

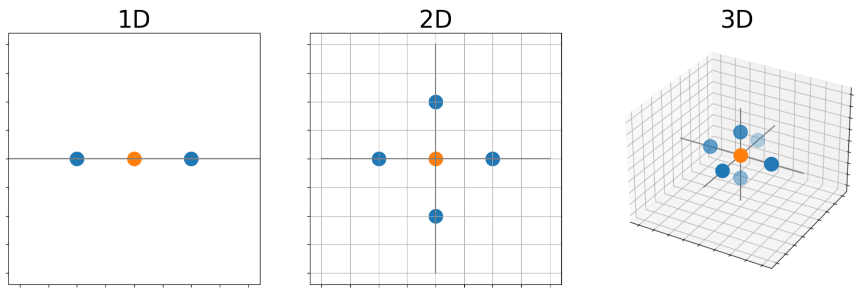
$$\frac{\partial}{\partial x} f(x) \approx \frac{f(x+\tau) - f(x)}{\tau}$$
$$\frac{\partial^2}{\partial^2 x} f(x) \approx \frac{f(x+\tau) - 2f(x) + f(x-\tau)}{\tau^2}$$

More generally, we approximate the m-th derivatives of f using the m-th finite difference stencil $\Delta_m f(x)$, which we can again express in terms of evaluations of f at delayed locations $x - k\tau$ with $|k| < m$ [20]. However, due to the multi-fidelity setting we cannot afford to request an arbitrary amount of new f_h evaluations. This is where f_l comes into play. We replace the delayed f_h evaluations in the truncated Taylor series approximation by evaluations of f_l and make the whole approximation computationally tractable. Therefore, the Gaussian process f_h tries to learn a mapping g by finding a truncated Taylor series with finite difference stencils in order to approximate f_h. Using finite differences approximation instead of derivatives changes equation 4.7 to a more implicit form:

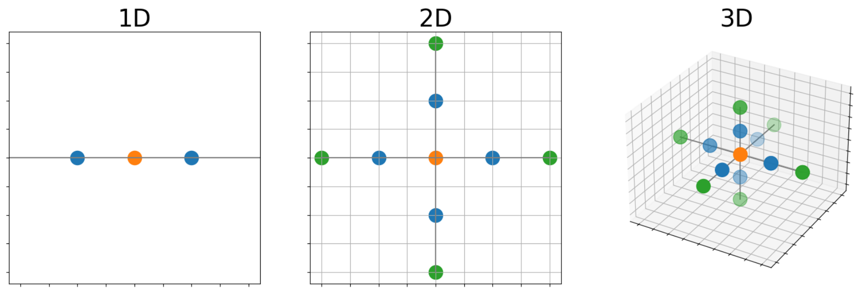
$$f_h(t) = g(t, f_l(t), f_l(t-\tau) \ldots, f_l(t-K\tau)). \tag{4.9}$$

Whenever we deploy this approach, we have to make the following design decisions.

1. The number of derivatives to include: The higher the estimated complexity of the underlying cross-correlation between the high- and low data sets, the more derivatives are necessary to find a good approximation [20].



(a) Augmentation with one derivative



(b) Augmentation with two derivatives

Figure 4.3: Visualisation of a possible delay pattern: $t \in \mathbb{R}^d$ is denoted by the orange point, the blue points are neighbour points of t with distance τ, green points with distance 2τ.

2. The delay pattern of the parameters passed to g.

While [20] uses negative delays $(x, x-\tau, \ldots, x-K\tau)$, we can also work with 'forward-and-backward' shifts such as

$$f_h(t) = g(t, f_l(t-2\tau), f_l(t-\tau), f_l(t), f_l(t+\tau), f_l(t+2\tau)) \tag{4.10}$$

with $K = 2$ and $d = 1$. Especially, when working in a higher dimensional input space of $d > 1$ with $f_h : \mathbb{R}^d \to \mathbb{R}$ it is important to find a appropriate delay pattern. To get a better understanding of how shifting with multiple delays and multiple dimensions works we illustrate forward-and-backward-delays used in equation 4.10 in figure 4.3. All points except the orange point in the center are neighbour points $t - e_i\tau$ of t with $e_i \in \mathbb{R}^d$ being the i-th canonical unit vector defined by

$$e_i^{(j)} = \begin{cases} 1, i = j \\ 0, otherwise \end{cases}$$

As shown in equation 4.10, we pass the neighbour points to the low-fidelity function and fuse the corresponding results with t. Figure 4.3 also shows that with increasing number of derivatives and dimensions, the number of delayed neighbour points dramatically increases. In this example, the number of delays is specified by $\mathcal{O}(2dn)$ with d being the input dimension and n being the number of derivatives. The delay pattern from [20] used in 4.9 would grow with $\mathcal{O}(dn)$ as the delays go towards one direction. This gives raise to an interesting conflict: With increasing number of derivatives more complex functions can be learned. However when using ARD, additional augmentation entries increase the number of parameters which we have to optimize in hyperparameter optimization. As described in section 3.6, this makes finding the optimal hyperparameter vector more difficult. Therefore, too many new augmentation entries can make the model less accurate.

DFGP Workflow

The DFGP workflow is almost identical to standard NARGP, which we have already described in subsection 4.3.2. The most important difference is that more parameters are passed to the mapping g. Therefore firstly, we need to specify the following configurations:

- Distance to the surrounding neighbour points $\tau > 0$
- Number of derivatives n
- Delay pattern

Due to these additional configurations, the augmented version D_t^* of a input set D_t is typically more complex and has more additional entries than in NARGP.

4.5 Data Fusion Gaussian processes with NARGP kernel and Adaptation

In this work we are going to introduce and test a new method which we refer to as Data Fusion Gaussian processes with composite NARGP kernel (DFGPC). For a single delay term τ with second order derivative stencil, the covariance kernel looks as following:

$$
\begin{aligned}
k_{DFGPC}(x,x') &= k_1(k_2+k_3+k_4)+k_5 \\
k_1(x,x';\theta_1) &= \sigma_1^2 \exp\left(-\frac{\sum_{i=1}^{d} w_{1i}(x_j-x_j')^2}{2l_1}\right) \\
k_2(x,x';\theta_2) &= \sigma_2^2 \exp\left(\frac{w_2\left(f_{t-1}(x)-f_{t-1}(x')\right)^2}{2l_2}\right) \\
k_3(x,x';\theta_3) &= \sigma_2^2 \exp\left(\frac{w_3\left(f_{t-1}(x-\tau)-f_{t-1}(x'-\tau)\right)^2}{2l_3}\right) \\
k_4(x,x';\theta_4) &= \sigma_2^2 \exp\left(\frac{w_4\left(f_{t-1}(x+\tau)-f_{t-1}(x'+\tau)\right)^2}{2l_4}\right) \\
k_5(x,x';\theta_3) &= \sigma_3^2 \exp\left(-\frac{\sum_{i=1}^{d} w_{3i}(x_j-x_j')^2}{2l_3}\right)
\end{aligned}
\tag{4.11}
$$

It is simply a DFGP model (described in chapter 4.4), but with an internal high-fidelity level which uses the composite kernel instead of a RBF kernel. Additionally, we deploy an implementation of the model management strategy called adaptation from subsection 4.1.2. For the sake of simplicity, we assume to have only two fidelity levels, the high- and the low-fidelity data set.

4.5.1 Adaptation optimization

Additionally, we use adaptation described in subsection 4.1.2 to choose evaluation points. Instead of training the model on a random set of high-fidelity points and then using it for predictions, we firstly train the model on an extremely small data set of m_0 points. The model must still be highly underfitted after the first training run. The next step is to apply adaption. As described in subsection 4.1.2 we update a low-fidelity model with input-output pairs from a high-fidelity model. This underfitted model will correspond to the low-fidelity model. We enter an adaptation loop with n iterations, where in each step the low-fidelity model chooses a new high-fidelity input point $x \in \mathcal{X} \subseteq \mathbb{R}^d$ from the input domain. For this chosen point it requests the corresponding accurate target value $f_{high}(x) \in \mathbb{R}$. After calling the high-fidelity model, the new input-output pair $(x, f_{high}(x))$ will be merged into the existing high-fidelity training data set. We now retrain the low-fidelity model, taking the new input-output pair into account. During a procedure like this with n steps, we must train the model on data sets of sizes $m_0, m_0 + 1, \ldots, m_0 + n$. Of course, this is way more computationally expensive than directly using a random data set of appropriate size. But on the other side, this procedure makes sure that the model gets a much better training data set.

Choosing the next data point

During the adaptation loop, the model has to choose the next input point, whose respective target value should be requested. So far, we have left out this point. We deploy an information based approach. The method is inspired from single fidelity Bayesian optimisation methods [21]. More precisely, we refer to the max-value entropy search (MES) [36]. In this approach the next evaluation point corresponds to the point of maximum information gain. The gain of information is the difference between the entropy at a point of interest before and after the function evaluation at this point. Let us represent $f_* = f(x)$ as the exact prediction of an input x of a GP surrogate. $D_n = \{x_1, x_2, ..., x_n\}$ is a set of n evaluation points and x_{n+1} is the $(n+1) - th$ evaluation point. The information gain at x_{n+1} due to the function evaluation at this point is:

$$\mathcal{I}(x_{n+1}, f_*|D_n) = H(f_*|D_n) - H(f_*|D_n, x_{n+1}) \tag{4.12}$$

where $H(f_*|D_n)$ is the conditional entropy. We compute the next evaluation point by solving the optimisation problem:

$$x_{n+1} = \underset{x_{n+1} \in \mathcal{X}}{\arg\max}\, \mathcal{I}(x_{n+1}, f_*|D_n) \tag{4.13}$$

In equation 4.12, we can see that the information gain depends upon the conditional entropy. We know from equation 3.10 that the predicted value f_* is Gaussian distributed with posterior mean μ_p and variance σ_p defined in equation 3.11 and 3.12, respectively.

$$p(f_*|D_n) = \frac{1}{\sqrt{2\pi{\sigma_p}^2}} \exp\left(\frac{-(f_* - \mu_p)^2}{2{\sigma_p}^2}\right)$$

We simplify the conditional entropy as:

$$\begin{aligned}
H(f_*|D_n) &= -\int_{-\infty}^{\infty} p(f_*|D_n)\log p(f_*|D_n)df_* \\
&= -\int_{-\infty}^{\infty} p(f_*|D_n)\log\left(\frac{1}{\sqrt{2\pi{\sigma_p}^2}}\exp\left(\frac{-(f_*-\mu_p)^2}{2{\sigma_p}^2}\right)\right)df_* \\
&= -\log\left(\frac{1}{\sqrt{2\pi{\sigma_p}^2}}\right)\underbrace{\int_{-\infty}^{\infty} p(f_*|D_n)df_*}_{=1} + \frac{1}{2{\sigma_p}^2}\underbrace{\int_{-\infty}^{\infty}(f_*-\mu_p)^2 p(f_*|D_n)df_*}_{={\sigma_p}^2} \\
&= \frac{1}{2}\log\left(2\pi{\sigma_p}^2\right) + \frac{{\sigma_p}^2\log(e)}{2{\sigma_p}^2} \\
&= \frac{1}{2}\log\left(2\pi e{\sigma_p}^2\right) \\
&= \log\left(\sigma_p\sqrt{2\pi e}\right)
\end{aligned} \tag{4.14}$$

We use the above derived expression of entropy of Gaussian distribution for further derivation. When we evaluate the function at a point, the posterior standard deviation at that point is equal to the observation noise σ_{noise}. Using the result from equation 4.14, we can write:

$$H(f_*|D_n, x_{n+1}) = \log\left(\sigma_{noise}\sqrt{2\pi e}\right) \tag{4.15}$$

Substituting equation 4.14 and 4.15 in equation 4.12:

$$\mathcal{I}(x_{n+1}, f_*|D_n) = \log\left(\sigma_p\sqrt{2\pi e}\right) - \log\left(\sigma_{noise}\sqrt{2\pi e}\right) = \log\sigma_p - \log\sigma_{noise} \tag{4.16}$$

The second term of equation 4.16 $\log\sigma_{noise}$ is constant. So, optimising the information gain is equivalent to optimising the first term $\log\sigma_p$. Moreover, logarithm is a monotonic function. Therefore, optimising $\log\sigma_p$ is equal to optimising σ_p. Hence, we can simplify the optimisation problem in equation 4.13 to

$$x_{n+1} = \underset{x_{n+1}\in\mathcal{X}}{\arg\max}\,\sigma_p = \underset{x_{n+1}\in\mathcal{X}}{\arg\min} -\sigma_p \tag{4.17}$$

However, we need to specify a bounded search domain $\mathcal{X} \in \mathbb{R}^d$ where we search for the global optimum so that the optimization problem in equation 4.17 is well defined. In simple words, our goal is to find the input point with maximum posterior variance. We evaluate the high-fidelity function at that point, which will modify the covariance kernel k after training and optimization. This changes the variances at all points. Again, we find the point with highest variance and repeat the process.

Therefore, searching for the next evaluation point turns out to be an optimisation problem defined (equation 4.17), that is non-convex and that has multiple critical points. Therefore, obtaining the global minimum is a very challenging problem. There are different approaches mentioned in literature to solve this problem like gradient based methods [38], Covariance matrix adaptation evolution strategy (CMA-ES) [24], DIRECT [17] etc. In this work we only use the DIRECT algorithm.

DIRECT-l algorithm

The implementation of this work uses the DIRECT-l global optimization algorithm [17]. It is applicable to following optimization problems:

$$\begin{aligned} &\underset{x}{\operatorname{argmin}} && f(x) \\ &\text{s.t.} && i = 1, \dots, m : x_i^{(L)} \leq x_i \leq x_i^{(U)} \end{aligned} \tag{4.18}$$

Locally-based DIRECT (DIRECT-l) is based on the standard DIRECT optimization algorithm, which tends to suffer from a slow refinement of the current optimal result. DIRECT-l improves this weaknesses [17]. Formulation 4.18 makes no specific assumptions about $f(x)$. This complies with the adaptation setting described in subsection 4.1.2 where the high-fidelity model is also treated as a black-box. Hence, we do not need any gradients or special information of $f(x)$. This is an important characteristic, because we have no further information about the uncertainty function $\sigma(x)$. It is therefore not guaranteed to be differentiable. Moreover, DIRECT-l is deterministic [17]. So, unlike hyperparameter optimization (described in section) there is no need for multiple restarts.

A drawback of DIRECT-l is that it tends to be less successful in higher dimensional problem [17]. However, this weakness is no big deal for our purposes, as in this work we mainly focus on problems up to four dimensions.

4.6 Implementation

Now, we describe the implementation of this work, which is written in Python 3.8.5. We use GPy (version 1.9.9) [14] to create basic GP regression models. It is a Gaussian process framework written in Python from the Sheffield machine learning group. We use Chaospy (version 4.2.3) [10], which is a Python package for uncertainty quantification using polynomial chaos expansion. The link for the Github repository of the implementation is `https://github.com/MartinKlapacz/multifidelity-datafusion-GPs.git`. Figure 4.4 illustrates the structure of the implementation. To keep things simple the illustrated classes contain only the most important methods and attributes.

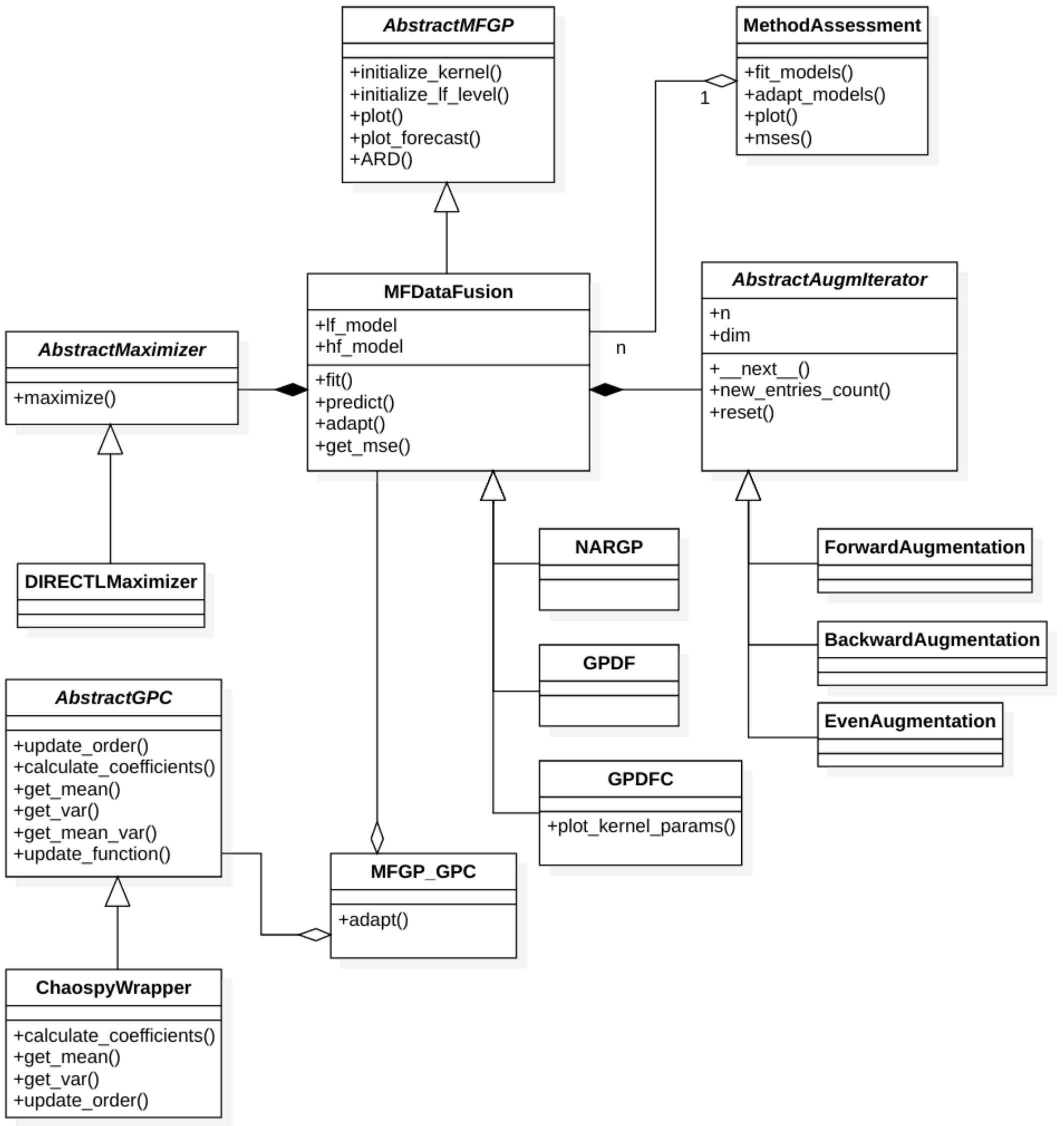


Figure 4.4: UML class diagram of the implementation

4.6.1 MultifidelityDataFusion

The `MultifidelityDataFusion` class is the core of the implementation. It is a highly parameterized class that can be used to build own multi-fidelity Data Fusion models. From a higher level perspective a `MultifidelityDataFusion` object expects the following information:

- Input dimensionality and input bounds
- The exact ground-truth function, which will serve as the high-fidelity model
- A specification of the low-fidelity model
- Information about derivatives to include
- The high-fidelity kernel
- The number of adaptation steps.

In this implementation we will only consider multi-fidelity settings of two fidelity levels. Hence, each `MultifidelityDataFusion` object has an internal low-fidelity and high-fidelity layer.

Data-driven and function-driven low-fidelity model

Depending on the multi-fidelity problem, the low-fidelity model is available in the form of data (input-output-pairs) or in the form of a closed function. Therefore, there is a data-driven and a function-driven way of specifying the low-fidelity model of a `DFGP` object. If we pass the data to the constructor, a `GPy.models.GPRegresssion` object is created, trained and optimized using this data. From now on its mean prediction function will represent the low-fidelity model. However, if it is available as a closed function, we can pass the function to the constructor, where it is saved and used for further computations.

Training and prediction

The method `fit` trains the model using the high-fidelity data and expects only the input values. The corresponding target values are computed using high-fidelity function which has been passed to the model as a parameter before. Internally, it creates (and optimizes) a `GPy.models.GPRegresssion` model trained on an augmented version of the passed training data. It uses the low-fidelity function for augmentation. This model will serve as the high-fidelity model. We can make predictions using the `predict` method, which augments the input data and calls the prediction function of the internal high-fidelity model. We always optimize the models by calling `GPy.models.GPRegression.optimize`, which internally uses BFGS optimization [38].

4.6.2 AbstractMFGF

`AbstractMFGP` is an abstract superclass of `MultifidelityDataFusion`. `MultifidelityDataFusion` contains most of the methods, which the user will directly use. `AbstractMFGP` contains private helper methods, which are important for plotting and internal computation.

4.6.3 Models

A normal user would quickly feel overwhelmed when directly working with `MultifidelityDataFusion`. Instead, they should take a look at the subclasses of `MultifidelityDataFusion` as they implement concrete multi-fidelity algorithms described in this paper (using `MultifidelityDataFusion`). New multi-fidelity models can be added in the future by simply inheriting from `MultifidelityDataFusion`. What follows is a description the implemented multi-fidelity algorithm.

`NARGP` class

`NARGP` is a subclass of `MultifidelityDataFusion` and provides an implementation of the NARGP algorithm, which is described in chapter 4.3. It uses the composite NARGP kernel and no derivatives. Therefore, initializing an object of this class requires the low- and high-fidelity specification and the input dimension. The composite NARGP kernel plays an important part in this method. GPy provides a kernel algebra implementation, which makes combining different kernel functions very easy. We use this GPy feature to implement the NARGP kernel in equation 4.4.

`GPDF` class

`GPDF` provides an implementation of Data Fusion Gaussian processes described in chapter 4.4. As described in chapter 4.4, Data Fusion GPs take a specified number of derivatives into account. Therefore, when initializing a `GPDF` object, the number of derivatives and the step length must be passed to the constructor, as well. The internal high-fidelity model of a `GPDF` object uses a GPy RBF kernel.

`GPDFC` **class**

`GPDFC` is similar to `GPDF`. Unlike `GPDF`, the internal high-fidelity model of `GPDFC` uses a composite NARGP kernel. It also provides plotting methods for the NARGP kernel hyperparameters.

4.6.4 Augmentation Iterators

Throughout most of the algorithms in this work, augmentation plays an important part. High-fidelity data is fused with a number low-fidelity predictions in order to finally pass it to the high-fidelity model. While NARGP has a simple augmentation mapping $x \mapsto (x, f_l(x))$, Data Fusion GPs can have more complex mappings (as illustrated in figure 4.3), where dimensionality and number of derivatives must be taken into consideration. These dynamic augmentation mappings are implemented using Python iterator classes. The number sequences they generate are used as the prefactors of τ, which are added to x to get the neighbour points, where we evaluate the low-fidelity function. The following table shows the iterator classes and the augmentation pattern which they implement.

Class	Numbersequence	Augmentation mapping
`Backward Augmentation`	$0, -1, \dots, -K$	$x, f_l(x), \dots, f_l(x - K\tau)$
`Forward Augmentation`	$0, 1, \dots, K$	$x, f_l(x), \dots, f_l(x + K\tau)$
`Even Augmentation`	$0, -1, 1, \dots, -K, K$	$x, f_l(x), f_l(x-\tau), f_l(x+\tau), \dots,$ $f_l(x - K\tau), f_l(x + K\tau)$

Table 4.1: Augmentation classes and their corresponding number sequence

All iterator classes must implement the abstract class `AbstractAugmIterator`. As we deploy the Strategy Pattern, we can easily add new augmentation patterns to the implementation.

4.6.5 Acquisition optimizers

In each iteration of the adaptation process, the model has to choose and acquire a new high-fidelity point. Section 4.5.1 provides a detailed description of the decision making process. We use an optimization algorithm to find this input value. Again, we deploy the Strategy Pattern to provide an extendable set of optimization strategies. Each optimization class must implement the abstract class `AbstractMaximizer`.

Currently, we only provide one optimization method. However, new subclasses of `AbstractMaximizer` can easily be added to create new maximization strategies in the future.

`DIRECTLMaximizer` **class**

This maximizer class uses a Python package called DIRECT (version 1.0.1). It is a wrapper for the Fortran implementation of the DIRECT algorithm written by Joerg. M. Gablonsky [13]. It provides an implementations of the DIRECT and DIRECT-l optimization algorithms. In most cases we use DIRECT-l, which we described in chapter 4.5.1. Some plots in section 5.3 were created using the Python package scipydirect (version 1.3), which implements the DIRECT algorithm, as the DIRECT packaged caused some errors.

`MethodAssessment` **class**

`MethodAssessment` can be used to compare multiple multi-fidelity models. It trains and adapts multiple models on the same input data. The performance of these models can be displayed and compared on a shared mean squared error plot.

4.6.6 General polynomial chaos

In this work combine multi-fidelity Gaussian processes with general polynomial chaos expansion. Therefore, this implementation also provides basic functionality that enables the usage of GPC. `ChaospyWrapper` is a wrapper of the Chaospy [10] functionality. We can use it to compute the statistical moments of a computationally complex function such as the mean prediction function of multi-fidelity model. This is how we combine multi-fidelity models and polynomial chaos expansion. `MFGP_GPC` is similar to `MethodAssessment`. Its purpose is to assess and display PCE performances using a multi-fidelity model with different numbers of adapted data points. It expects a multi-fidelity model and a GPC object and computes the accuracies of the GPC object combined with differently adapted versions of the model.

5 Results

The previous chapter provided a detailed description of different multi-fidelity models. In this chapter we test the aforementioned methods and illustrate their performances. The results and plots are obtained using the implementation described in section 4.6. For the sake of simplicity we only work with two fidelity levels. Additionally, all deployed models use a internal function-driven low-fidelity model. We therefore assume that the low-fidelity level is infinitely cheaper than the high-fidelity model.

5.1 Uncertainty development during Adaptation

In this section we visually illustrate the development of the uncertainty curve during the adaptation process. We start with a data set of 8 uniformly distributed high-fidelity points $x \sim \mathcal{U}[0,1]$ and pass it to a NARGP model. For simplicity we choose the low-fidelity level to be function-driven and therefore assume that low-fidelity evaluations are infinitely cheap. For illustrative purposes we only show two adaptation steps, which will result in a NARGP model trained on 10 high-fidelity points. We use a multi-fidelity setting from [27], which is defined as:

$$\begin{aligned} f_l(t) &= \sin(8\pi t) \\ f_h(t) &= (t - \sqrt{2}) \times f_l(t)^2 \end{aligned}$$

Figure 5.1 illustrates the development of the mean curve (left column) and variance curve (right column) during the adaptation steps. The left column shows the exact curves (dotted blue) and the predicted mean curves (green). The width of the light-green margins around the mean is two times the variance at the current position and therefore proportional to the variance curves in the right column. The right column plots the variance or uncertainty curves. All subplots have the same x-axis. The first row shows the model curves before adaptation. Each subsequent row shows the current model curves after performing one adaptation step. The red crosses in the right column denote a new high-fidelity point which will be acquired in the current adaptation step.

As described in section 4.5.1, the goal in each adaptation step is to find the input point that leads to maximum information gain. Our strategy in this work is to find the input $x \in \mathcal{X}$ that maximizes the current uncertainty curve. Our assumption is that this

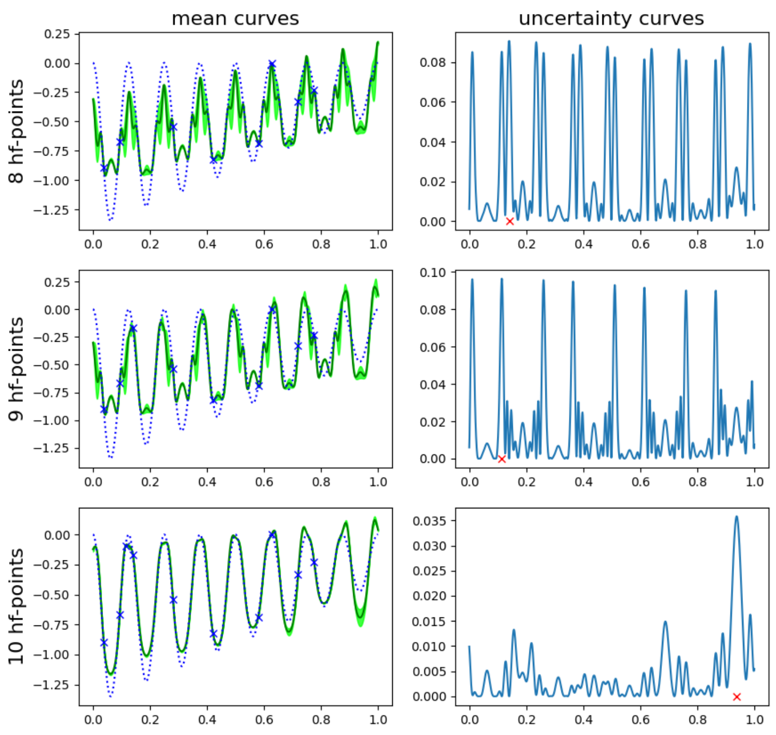


Figure 5.1: Snapshot of the adaptation process: Prediction (green) and exact curve (dotted blue) in the left column, variance/uncertainty curve (blue) and the new adaptation point (red cross) in the right column

reduces the model entropy and therefore increases the accuracy of the mean prediction. Figure 5.1 illustrates how each adaptation step slightly flattens the uncertainty curves in the right column, while the similarity between the mean prediction curve and the exact curve increases. Especially the first two uncertainty curves have a lot of very similar local maxima. Nonetheless, DIRECT-l always finds the highest peak in the illustrated steps. When comparing the mean prediction curve at 8 and at 10 high-fidelity points we conclude that adding those two points has significantly reduced the difference between the prediction curve and the exact curve.

5.2 Comparing Adaptation and random sampling

Equipped with an formal and visual understanding of the adaptation process we move on to our second experiment. We illustrate the additional accuracy, which we gain when using an adapted data set instead of a randomly generated one. The multi-fidelity setting is defined as following:

$$f_h(x, y) = sin(2x_0)^2 + cos(2x_1)$$
$$f_l(x, y) = 1.5 f_h(x, y)^2 + 3$$

The low-fidelity model is the result of a quadratic transformation followed by a linear transformation applied to the high-fidelity model. In this section we focus on the effect of adaptation. This is why we apply it to four differently configured models and show that adaptation improves them independently of their underlying configuration:

1. A standard NARGP model
2. A Data Fusion GP with RBF kernel and one derivative (DFGP1)
3. A Data Fusion GP with composite kernel and one derivative (DFGPC1)
4. A Data Fusion GP with composite kernel and two derivatives (DFGPC2)

All models, except the NARGP model, use backward-shifting:

$$x \longmapsto (x, f_l(x), f_l(x-\tau), \dots, f_l(x-K\tau)) \tag{5.1}$$

Firstly, we train each model with five randomly selected high-fidelity data points $x \sim \mathcal{U}[0, 1]$. Next, each model adapts 20 data points. We calculate and plot the mean squared error values after each adaptation steps.

Additionally, we train equal models on data sets of 5, 10, 15, 20 and 25 uniformly distributed high-fidelity points $x \sim \mathcal{U}[0, 1]$. For each data set size we make 10 runs and average and plot the means squared error values. All subplots in figure 5.2 show that the adapted model performs better. Their mse curves drop significantly quicker than the averaged mse curves of the models with random training data. A random data set is likely to have clusters of points stacked together. This leads to an ineffective utilization of resources, because each point is prone to contributing only a small amount of information to the data set. However, each data point still costs one high-fidelity model evaluation. On the other hand, in each adaptation step the next chosen point maximizes the information gain at the current state of the data set. This leads to an optimal utilization of resources as we make sure that we get the maximum amount

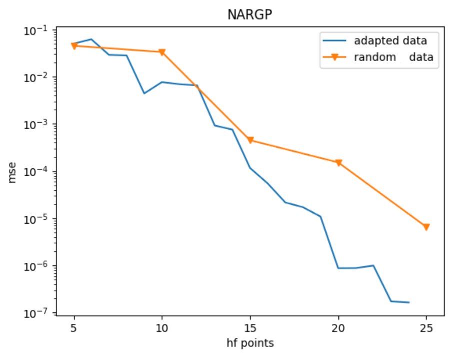
model attributes				mean squared error	
name	kernel	number of delays	τ	random	adapted
NARGP	Composite kernel	-	-	2.305×10^{-5}	3.362×10^{7}
DFGP1	RBF kernel	1	0.01	7.804×10^{-3}	2.201×10^{-3}
DFGPC1	Composite kernel	1	0.01	1.260×10^{-5}	2.251×10^{-7}
DFGPC2	Composite kernel	2	0.01	1.263×10^{-5}	3.253×10^{-7}

Table 5.1: Comparison of mean square error between random vs adaptive selection of evaluation points

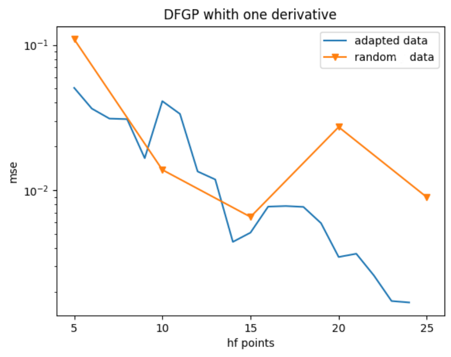
of information for one high-fidelity evaluation. Finally, an adapted data set has less redundancy and a higher amount of information than a random data set. This leads to a better model accuracy.

In each subplot, we also observe that before reaching 15 high-fidelity points, adaptation and random sampling lead to similar accuracies. After reaching 15 high-fidelity points the adapted version starts to outperform the randomly sampled version. The bigger the size of the randomly sampled data set, the the higher is the number of points with close neighbors, which reduces their amount of new information contributed to the data set. This is why adaptation will only lead to an accuracy advantage (compared to random sampling), if the existing data set has reached a certain minimum density of points.

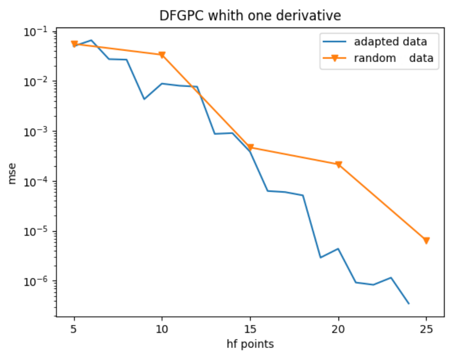
Table 5.1 lists the models configurations and the corresponding mean squared errors after 25 high-fidelity points. But we should also take the downside of this method into consideration. Adaption of 20 data points requires a training run on a small data set of five points followed by 20 training runs on data sets with $5, 6, \ldots, 25$ points. This tremendously increases the computational training costs. This again leads to a trade-off we must consider.



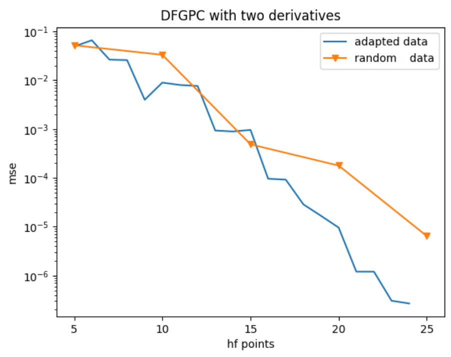
(a) NARGP performances



(b) DFGP1 performances



(c) DFGPC1 performances



(d) DFGPC2 performances

Figure 5.2: Comparison of models with random and with adapted training data: mse curve of adapted models (blue), averaged mse curves of models with random training data

5.3 Comparison of different methods

In the last section we have seen how adapted training data sets positively affect the model performance. In this section, we compare different methods on a few chosen problems. The problem settings [20] are defined as:

1. Setting with phase shifted oscillations
2. Setting with different periodicities
3. Discontinuous curves
4. Product of sinusoidal functions with 2 and 4 input dimensions.

We make the assumption that low fidelity functions are infinitely cheaper than the high fidelity functions. The models are trained on random high-fidelity data sets of $5, 10, 15, 20$ and 25 points. For each model and data set size, we plot the mean squared error values against the number of high fidelity evaluations. Training and testing data is sampled from a uniform distribution $\mathcal{U}[0, 1]$. The following table lists the configurations of the deployed models:

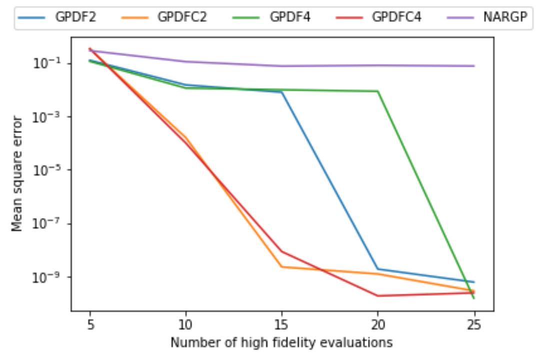
Name	τ	Number of delay points	Kernel
DFGP2	0.01	2	RBF
DFGPC2	0.01	2	Composite
DFGP4	0.01	4	RBF
DFGPC4	0.01	4	Composite
NARGP	-	-	Composite

Phase shifted oscillations

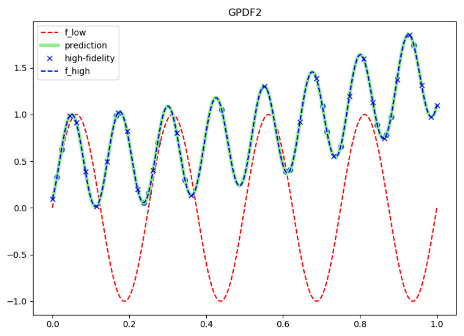
When working with real world data of different resolutions, fidelity levels often have similar oscillations but different phases [20]. An example multi-fidelity setting with this characteristic is:

$$f_l(t) = sin(8\pi t) \tag{5.2}$$
$$f_h(t) = t^2 + sin\left(8\pi t + \frac{\pi}{10}\right) \tag{5.3}$$

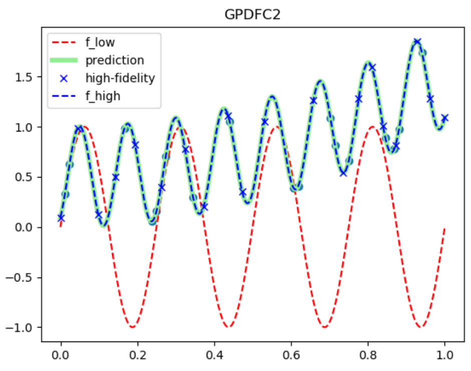
Figure 5.3 shows the results of fitting adaptive multi-fidelity GP on the aforementioned model. We observe from Figure 5.3a that NARGP has the highest amount of



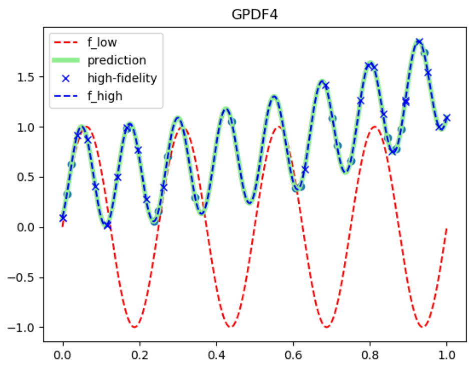
(a) Mean square error vs number of high fidelity evaluations



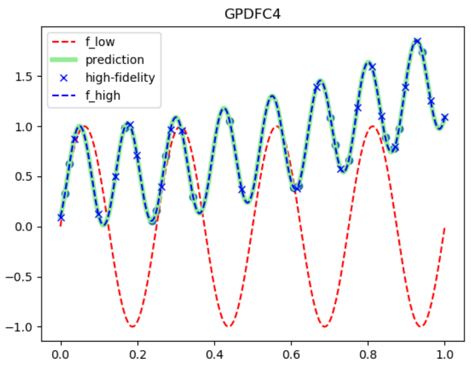
(b) GPDF2



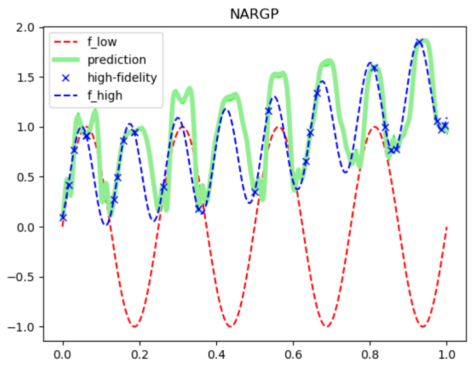
(c) GPDFC2



(d) GPDF4



(e) GPDFC4



(f) NARGP

Figure 5.3: Phase Shifted Oscillation

error. To understand this we perform the Taylor series expansion of the high-fidelity function:

$$\begin{aligned} f_h(t) &= t^2 + sin\left(8\pi t + \frac{\pi}{10}\right) \\ &= t^2 + sin\left(8\pi t\right) + \frac{\pi}{10}\frac{dsin\left(8\pi t\right)}{dt} + \epsilon \\ &= t^2 + f_l(t) + \frac{\pi}{10}\frac{df_l}{dt} + \epsilon \end{aligned}$$

Here, ϵ is the error value. We can see that the derivative of the low-fidelity function is present in the expansion of the high fidelity function. Since NARGP does not use any derivative terms, its performance quite poor compared to other methods.

Different Periodicities

Lets continue with an example where the high and low fidelity models are sinusoidal functions but with different periodicities [20]:

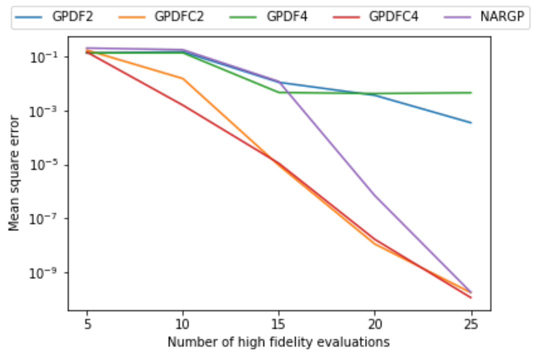
$$\begin{aligned} f_l(t) &= sin(6\sqrt{2}\pi t) \\ f_h(t) &= sin\left(8\pi t + \frac{\pi}{10}\right) \end{aligned} \tag{5.4}$$

In this example, the cross-correlation between the fidelities are characterized not only by a slight phase shift of $\frac{\pi}{10}$, but also by different periods. Figure 5.4 shows the results of applying adaptive multi-fidelity GPs to the aforementioned setting. Figure 5.4a shows that methods with composite kernels have a much lower mean square error than modelds with RBF kernels. Also, GPDFC2 and GPDFC4 have a lower error than NARGP. We therefore recommend to use GPDFC models when working with cross-correlations which include different periodicities. In order to understand why the covariance kernel leads to a better performance, we should take a more precise look at the high-fidelity function in equation 5.4. Firstly, we rewrite f_h using trigonometric addition rules:

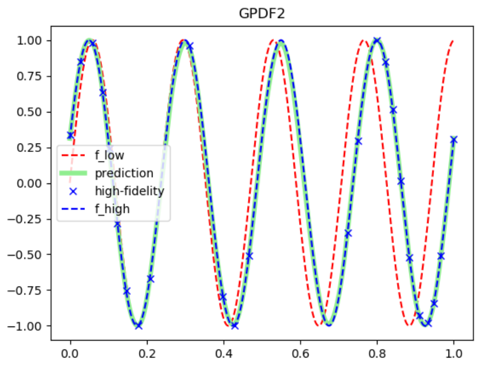
$$f_h(t) = \sin\left(8\pi t\right)\cos\left(\frac{\pi}{10}\right) + \cos\left(8\pi t\right)\sin\left(\frac{\pi}{10}\right) \tag{5.5}$$

Now, we can rewrite $\sin(8\pi t)$ and $\sin(8\pi t)$ in terms of f_l and derivatives of f_l.

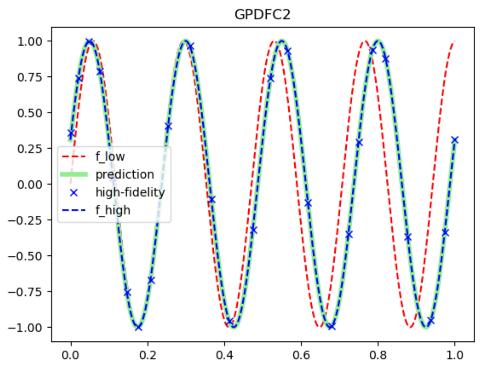
$$\begin{aligned} \sin(8\pi t) &= \sin\left(6\sqrt{2}\pi t + \left(8\pi - 6\sqrt{2}\pi\right)t\right) \\ &= \sin\left(at - bt\right) \\ &= \sin\left(at\right)\cos\left(bt\right) - \cos\left(at\right)\sin\left(bt\right) \\ &= \cos\left(bt\right)f_l(t) - \sin\left(bt\right)f_t^{(1)}(t) \end{aligned} \tag{5.6}$$



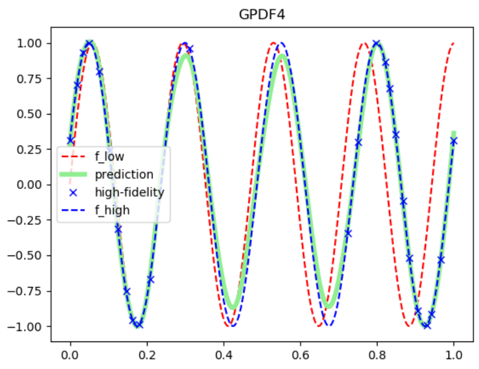
(a) Mean square error vs number of high fidelity evaluations



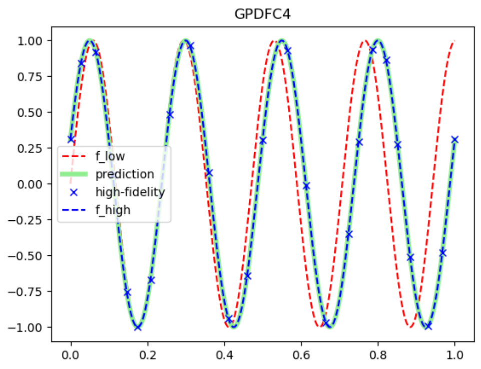
(b) GPDF2



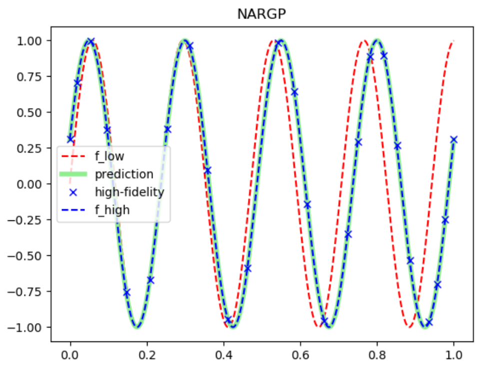
(c) GPDFC2



(d) GPDF4



(e) GPDFC4



(f) NARGP

Figure 5.4: Different Periodicity

where $a = 6\sqrt{2}\pi$ and $b = 6\sqrt{2} - 8\pi$. We can also rewrite $\cos(8\pi t)$:

$$\begin{aligned}\cos(8\pi t) &= \cos\left(6\sqrt{2}\pi t + \left(8\pi - 6\sqrt{2}\pi\right) t\right)\\ &= \cos(at - bt)\\ &= \cos(at)\cos(bt) + \sin(at)\sin(bt)\\ &= \cos(bt) f_t^{(1)}(t) + \sin(bt) f_l(t)\end{aligned} \tag{5.7}$$

Substituting equation 5.6 and 5.7 in equation 5.5 shows that $f_h(x)$ can indeed be represented in terms of f_l and $f_l^{(1)}$:

$$\begin{aligned}f_h(t) = &\left(\cos\left(\frac{\pi}{10}\right)\cos(bt) + \sin\left(\frac{\pi}{10}\right)\sin(bt)\right) f_l(t)\\ &+ \left(\sin\left(\frac{\pi}{10}\right)\cos(bt) - \cos\left(\frac{\pi}{10}\right)\sin(bt)\right) f_t^{(1)}(t)\end{aligned} \tag{5.8}$$

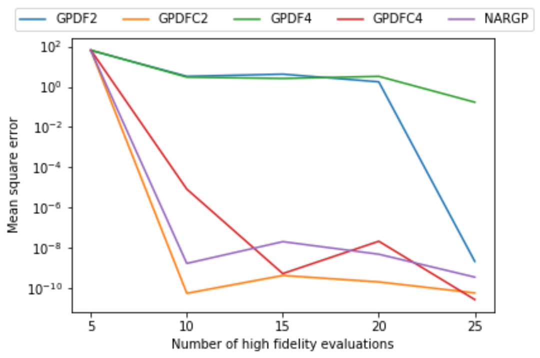
Models which are applied to this multi-fidelity setting must find a way to approximating f_h. The derived form of f_h consists of the following components: The prefactors, $f_l(t)$ and $f_t^{(1)}(t)$. The NARGP kernel assigns its three subkernels to one of those components and is therefore more successful at learning the underlying cross-correlation than the more limited RBF kernel. This is the reason why the models with composite kernels outperform the models with only RBF kernel.

Discontinuity

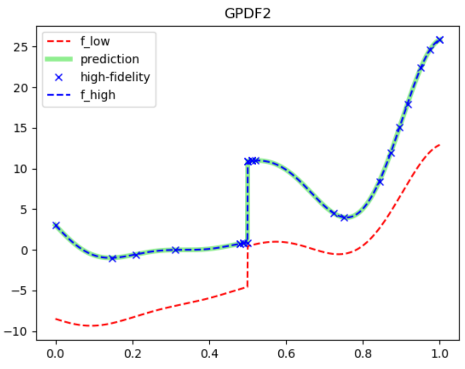
Now, we take a look at an example with a discontinuity [20]:

$$\begin{aligned}f_l(t) &= \begin{cases}0.5(6t-2)^2 sin(12t-4) + 10(t-0.5) - 5, & \text{for } t \leq 0.5\\ 0.5(6t-2)^2 sin(12t-4) + 10(t-0.5), & \text{for } 0.5 < t\end{cases}\\ f_h(t) &= 2f_l(t) - 20t + 20\end{aligned} \tag{5.9}$$

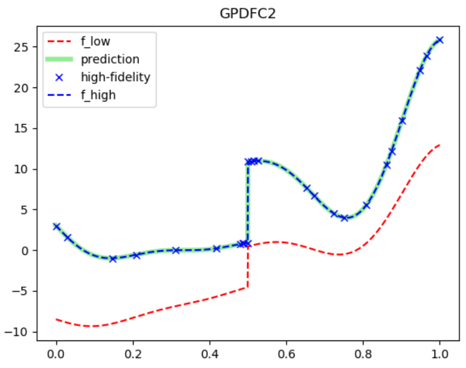
Figure 5.5 shows the results of applying adaptive multi-fidelity GP to the discontinuous setting. In Figure 5.5a we can observe that all models have low errors except GPDF4. Including more derivatives in the computation seems to reduce the accuracy of the model predictions. Figure 5.5d helps us understand this phenomenon. It illustrates the green prediction curve of a GPDFC model with four derivatives. The model is fitted with 25 high-fidelity data points which are marked with the blue crosses. 20 of them are adapted and five are randomly sampled points $x \in [0, 1]$. In this example, 25 high-fidelity points are enough to make the prediction curve almost identical to the exact curve. However, we are interested in the locations of the high-fidelity points. We can see that most of the points are very close to the discontinuity at $t = 0.5$. During



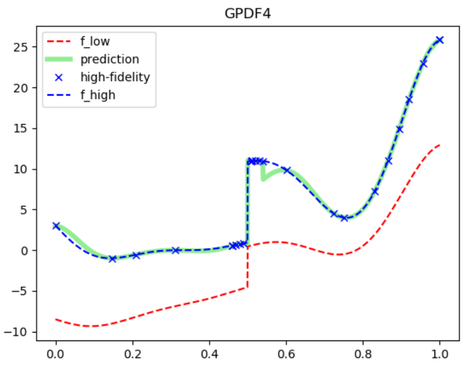
(a) Mean square error vs number of high fidelity evaluations



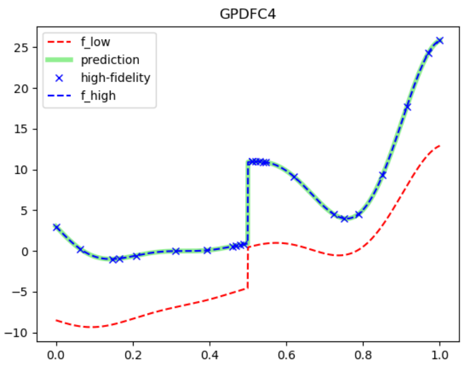
(b) GPDF2



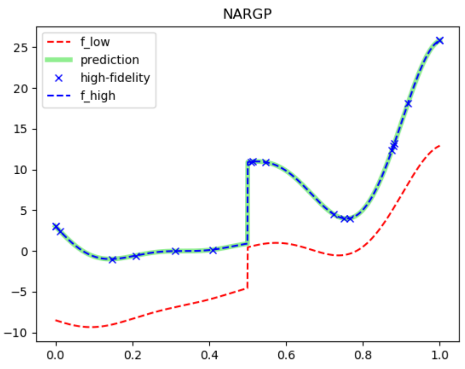
(c) GPDFC2



(d) GPDF4



(e) GPDFC4



(f) NARGP

Figure 5.5: Discontinuous

adaptation, the highest uncertainty (and therefore maximum information gain) can be found at areas where the curve has sudden and unexpected behaviors. This is the case at discontinuous locations, such as $t = 0.5$. This is why the model keeps focusing on this area during adaptation.

However, a intensive concentration of points near the discontinuity also causes this strange behavior. The function is not differentiable at this discontinuity. However due to adaptation, a large proportion of the high-fidelity data is located directly at this non-differentiable point. Data Fusion GPs use approximations of the first K derivatives at high-fidelity data points. Computing (multiple) derivatives at areas which are not differentiable does not make sense. This disturbs the approximation of f_h and therefore negatively affects the model performance. With increasing number of included derivatives this negative effect grows. However, this effect is not observed in GPDFC4. The reason for that might be the weight corresponding to delay terms convergences to a very low value during hyperparameter optimisation (ARD). However, this does not happen when using GPDF4, as the RBF kernel is not flexible enough to adapt to this situation. Likelihood is a multi-modal function. Which is why the optimiser might get lost in a unsatisfactory local minimum despite of multiple restarts.

Product of sinusoidal function with different input dimensions

In this section we focus on different input dimensionalities. We consider problems with two and four input dimensions. The two dimensional problem is defined as following:

$$\begin{aligned} f_h(x) &= \sin(2.2\pi x_1)\sin(\pi x_2) \\ f_l(x) &= 2f_h(x) - 1.2\left(\sin\left(\frac{pi}{10}x_1\right) + \sin\left(\frac{pi}{10}x_2\right)\right) \end{aligned} \tag{5.10}$$

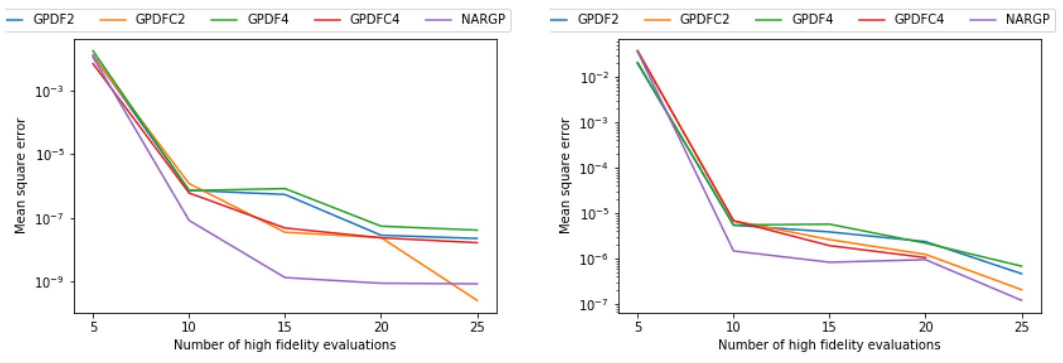
The four dimensional problem is as:

$$\begin{aligned} f_h(x) &= \sin(\pi x_1)\sin(\pi x_2)\sin(\pi x_3)\sin(\pi x_4) \\ f_l(x) &= 2f_h(x) - 0.25\left(\sin\left(\frac{pi}{10}x_1\right) + \sin\left(\frac{pi}{10}x_2\right) + \left(\frac{pi}{10}x_3\right) + \left(\frac{pi}{10}x_4\right)\right) \end{aligned} \tag{5.11}$$

Figure 5.6 shows that all the methods work fine in high dimensional cases, as well.

5.4 Uncertainty Quantification using multi-fidelity Gaussian processes

Since we have tested the accuracies of multi-fidelity models in various problem settings with different configurations, we can finally move on to our final experiment. In this



(a) Mean square error evolution for 2d function (b) Mean square error evolution for 4d function

Figure 5.6: Product of sinusoidal functions

section we combine polynomial chaos and multi-fidelity GP models. As described in chapter 2 we can use polynomial chaos to create approximations of complex and costly functions. Moreover, approximating the statistical moments of a QoI turned out to be very easy when working with PCE. As described in subsection 2.2.4, we need evaluations of the QoI to compute the coefficients $\hat{f}_k(x)$. This is the standard approach when using GPC. However, in this approach we train a multi-fidelity model on the QoI and pass its mean prediction function to the polynomial chaos. This prediction function will be used to compute the coefficients $\hat{f}_k(x)$. In this experiment we compare both approaches in terms of computational costs and relative error when estimating mean and variance of the QoI. We assume that low-fidelity evaluations are infinitely cheap compared to high-fidelity evaluations, which is why the computational costs are proportional to the number of high-fidelity evaluations. Our goal is to check if the computed statistical moments using the mean prediction function instead of the QoI provide good estimations of the exact solutions. Moreover, we will illustrate how the relative error of each GPC run develops for several adaptation runs. Our final goal is to find out how much computational effort we can save by using GPC linked with multi-fidelity GPs instead of direct GPC.

We will consider three multi-fidelity settings with two, three and four input dimensions. The analytical mean $\mu(a)$ and variance $\sigma(a)$ depend upon a sequence

Name	τ	Number of delay points	Kernel
NARGP	-	-	composite
GPDF	0.001	2	RBF
GPDFC	0.001	2	composite

Table 5.2: Configuations of the deployed multi-fidelity GPs

$a = (a_0, \ldots, a_{d-1})$ of parameters and are always defined as follows:

$$\mu = \prod_{i=0}^{d-1} \left(1 - \frac{\cos(a_i)}{a_i}\right) \tag{5.12}$$

$$\sigma = t_1(a) + t_2(a) + \mu(a)^2 \tag{5.13}$$

$$t_1 = \prod_{i=0}^{d-1} \left(\frac{1}{2} - \frac{\sin(2a_i)}{4a_i}\right) \tag{5.14}$$

$$t_2 = 2\mu(a)(-1)^{d-1} \prod_{i=0}^{n-1} \frac{\cos(a_i) - 1}{a_i} \tag{5.15}$$

In each example the high-fidelity function is defined as follows:

$$f_h(x) = 5 + \prod_{i=0}^{d-1} \sin(a_i x_i)$$

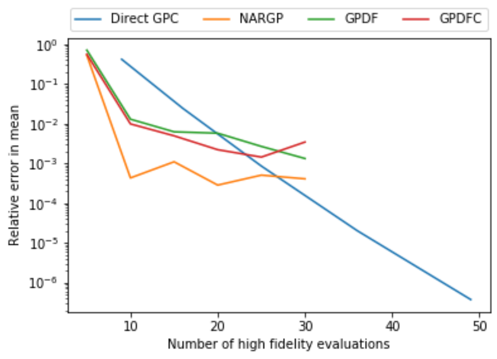
Training and testing input points are uniformly distributed and d-dimensional with entries $x_i \sim \mathcal{U}[0, 1]$. As described in table 2.1, we use polynomials from the Legendre family. In addition to direct GPC, we deploy GPC with different multi-fidelity models. Their configurations are listed in table 5.2.

5.4.1 2D problem setting

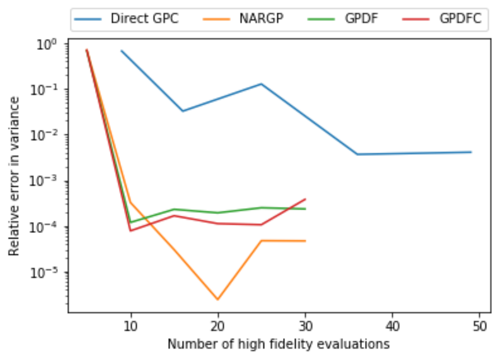
We start with a two dimensional example. The parameter sequence a and the low-fidelity function are defined as follows:

$$\begin{aligned} a &= (2.2\pi, \pi) \\ f_l(x) &= f_h(x) - 1.2\left(\sin\left(\frac{\pi}{10}x_0\right) + \sin\left(\frac{\pi}{10}x_1\right)\right) \end{aligned} \tag{5.16}$$

Figure 5.7 displays the relative mean and variance estimation errors of the aforementioned GPC approaches for different numbers of high-fidelity evaluations. In both subfigures we can see that for few high-fidelity points the error curves of the multi-fidelity approaches drop much quicker than direct GPC. After reaching 10 evaluation points the accuracy of the multi-fidelity approaches stays on the same level. In figure 5.7a NARGP-GPC with 10 high-fidelity evaluations leads to the same accuracy as direct GPC with 30 high-fidelity evaluations. In figure 5.7b the accuracies of the multi-fidelity approaches are even better. 10 high-fidelity evaluations return variance estimations with relatives errors of approximately 10^{-4}. Direct GPC, however, reaches far worse accuracies of approximately 0.05 at the costs of 50 high-fidelity points. Using a multi-fidelity approach instead of direct GPC leads therefore to much better results at lower costs.



(a) Mean square error evolution for 2d setting



(b) Mean square error evolution for 2d setting

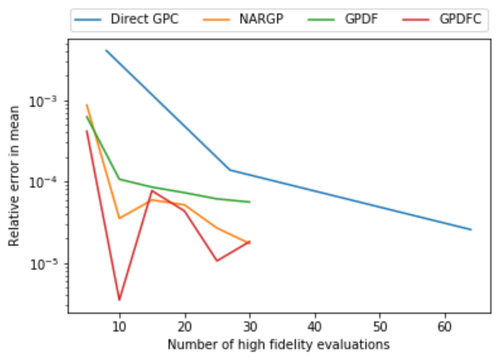
Figure 5.7: Evolution of the relative error in mean (a) and variance (b) with respect to the number of high fidelity function

5.4.2 3D problem setting

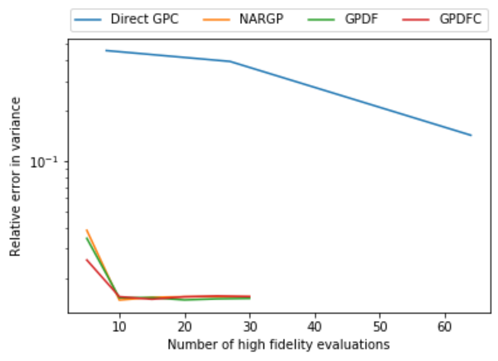
Now, we extend the previous setting to three dimensions. The multi-fidelity setting is defined as follows:

$$\begin{aligned} a &= (3\pi, 2\pi, \pi) \\ f_l(x) &= f_h(x) - \frac{1}{4}\left(\sin\left(\frac{\pi}{10}x_1\right) + \sin\left(\frac{\pi}{20}x_2\right) + \sin\left(\frac{3\pi}{20}x_3\right)\right) \end{aligned} \tag{5.17}$$

Figure 5.8 illustrates a similar situation as figure 5.7. In both subfigures all multi-fidelity error curves drop quickly to a low level, while the accuracy of direct GPC improves very slowly. According to subfigure 5.8a, direct GPC needs 60 high-fidelity points to reach the same accuracy as NARGP-GPC reaches with 10 high-fidelity points. However, it is still tremendously outperformed by GPDFC-GPC with 10 high-fidelity. Using one of both multi-fidelity approaches would safe at least 50 high-fidelity evaluations. When estimating the variace in figure 5.8a, the difference between both approaches is even higher.



(a) Mean square error evolution for 3d setting



(b) Mean square error evolution for 3d setting

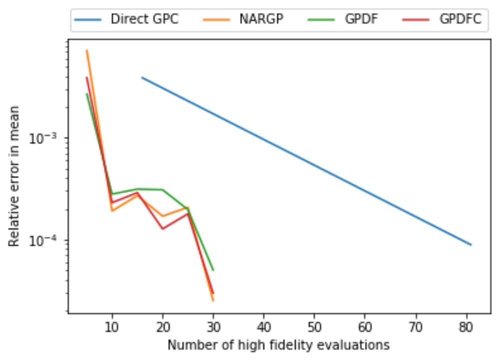
Figure 5.8: Evolution of the relative error in mean (a) and variance (b) with respect to the number of high fidelity function

5.4.3 4D problem setting

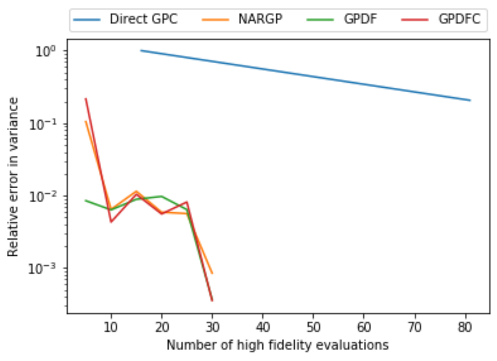
In this example we work with 4 dimensions. The multi-fidelity setting is defined as follows:

$$a = (\pi, \pi, \pi, \pi)$$
$$f_l(x) = f_h(x) - \frac{1}{4}\left(\sin\left(\frac{\pi}{10}x_1\right) + \sin\left(\frac{\pi}{20}x_2)\right) + \sin\left(\frac{3}{20}\pi x_3\right) + \sin\left(\frac{\pi}{5}x_4\right)\right) \tag{5.18}$$

Again, the high-fidelity approach leads to much better estimations of the statistical moments as illustrated in figure 5.9. According to figure 5.9a, a multi-fidelity GPC with 30 high-fidelity points leads to a relative error of approximately 10^{-4}, which is equal to standard GPC with 80 high-fidelity points. When estimating the variance, the computational costs which are safed when using multi-fidelity GPC is even higher.



(a) Mean square error evolution for 4d setting



(b) Mean square error evolution for 4d setting

Figure 5.9: Evolution of the relative error in mean and variance with respect to the number of high fidelity function

5.4.4 Conclusion

In all examples GPC linked with multi-fidelity models led to a high accuracy gain at significantly lower computational costs. In real world forward UQ problems, where one high-fidelity evaluation can take up to one day of computation, it is crucial to make as few high-fidelity evaluations as possible. Using this approach we can save a lot of time, as it reaches similar accuracies with much less computational effort. Therefore, GPC linked with multi-fidelity models is capable of turning previously intractable UQ problems into tractable ones.

6 Conclusion

In this work, we have given an overview of Gaussian processes and polynomial chaos expansion. We described recent multi-fidelity regression algorithms and introduced an own approach that combines several of their features. We also deployed a multi-fidelity model management strategy called adaptation. As shown in chapter 5.2, it is applicable to all multi-fidelity GP models and leads to a significant increase of the model accuracy. However, this accuracy gain comes at the price of running extra optimisation algorithms. We assume that we apply our method in computationally intensive models. So, the gain by decreasing the number of evaluation points overshadows other overheads. Finally, we have introduced GPC linked with multi-fidelity GPs, which combines multi-fidelity GPs and polynomial chaos expansion. We have tested it in differently dimensional problems with varying multi-fidelity models. In all cases GPC linked with multi-fidelity GPs led to significantly better estimations with less computational costs than standard GPC. According to our results, it can be used in forward UQ problems as a substitute for standard GPC to save a tremendous amount of computational resources. Throughout this work we have intensively tested multi-fidelity models in various situations and learned that there is one specific trade-off determining their accuracy. The more complex the model is, the more difficult are the cross-correlations it is capable of learning. As expected, we observe that the model needs more evaluations to get a decent accuracy as the dimensionality of the problem increases. Moreover, as we increase the dimensionality, the number of hyperparameters to be optimised also increases. This may sometimes reduce the accuracy of the model.

6.1 Future Works

At this point the methods described in this paper and this research area in general still provide a lot of future possibilities. Different modifications of adaptation could be worth studying. *Probability of Improvement*, *Expected Improvement* and *GP Upper Confidence Bound* are possible acquisition functions described in [31]. Deploying other acquisition functions and comparing their effect on the adaptation process would be very interesting. All models which we deployed made use of the RBF kernel class and combinations of it. In this work we mainly focused on multi-fidelity and less on kernel functions. Using the multi-fidelity models in combination with more diverse kernel classes such as the Matérn family or the rational quadratic kernel [29] could have a high potential and may be studied in further research. Moreover, all methods in this work were tested on models with two fidelity levels. Deploying and testing them in settings with a higher number of fidelity levels could lead to interesting insights, too. We also assume that our low fidelity functions are very cheap. However, in real world cases this might not always be correct. In such cases, we also build a surrogate for the low-fidelity model. This could be another direction in which we could take in future. There are numerous global optimization algorithms [37], which can be used instead of DIRECT to find the next evaluation point. Other well suited optimization methods may improve the runtime performance of adaptation. In this work we relied on multi-fidelity GPs from [20, 27]. But there are also completely different ways of performing multi-fidelity using Gaussian process. We can use deep Gaussian processes [8, 34, 7] to build multi-fidelity models, as well. In such models, each layer represents one fidelity. These more advanced models are less prone to overfitting as observed in case of scalable Gaussian processes [1]. Sparse grid multi-fidelity [23] is another way of implementing the multi-fidelity. However, the effects of adaptivity on such models are yet to be studied. The goal of this work was to show the effectiveness of the presented methods. The aim of future works could be to improve the efficiency of the aforementioned methods. One could create a parallelized version of adaptation which returns multiple evaluation points in each step. In general, evaluating multiple input points could be done in parallel. One of the existing methods of multifidelity GPC is using sparse grids [23]. It would be interesting to compare the two methods and test them in different scenarios.

Indeed, there are still a lot of open questions and possibilities in this research field. We are convinced that the promising results of this work are just the tip of the iceberg and that further research will lead to fascinating insights.

List of Figures

List of Tables

Bibliography

[1] M. Bauer, M. van der Wilk, and C. E. Rasmussen. "Understanding probabilistic sparse Gaussian process approximations." In: *Advances in neural information processing systems*. 2016, pp. 1533–1541.

[2] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[3] A. Brandt. "Multiscale solvers and systematic upscaling in computational physics." In: *Computer Physics Communications* 169.1-3 (2005), pp. 438–441.

[4] A. Brandt. "Multiscale scientific computation: Review 2001." In: *Multiscale and multiresolution methods* (2002), pp. 3–95.

[5] H.-J. Bungartz and M. Schäfer. "Modelling, simulation." In: *Optimization* 53 (2014), p. 53.

[6] P. R. Conrad and Y. M. Marzouk. "Adaptive Smolyak pseudospectral approximations." In: *SIAM Journal on Scientific Computing* 35.6 (2013), A2643–A2670.

[7] K. Cutajar, M. Pullin, A. Damianou, N. Lawrence, and J. González. "Deep gaussian processes for multi-fidelity modeling." In: *arXiv preprint arXiv:1903.07320* (2019).

[8] A. Damianou and N. D. Lawrence. "Deep gaussian processes." In: *Artificial intelligence and statistics*. PMLR. 2013, pp. 207–215.

[9] I.-G. Farcaş, T. Görler, H.-J. Bungartz, F. Jenko, and T. Neckel. "Sensitivity-driven adaptive sparse stochastic approximations in plasma microinstability analysis." In: *Journal of Computational Physics* 410 (2020), p. 109394.

[10] J. Feinberg and H. P. Langtangen. "Chaospy: An open source tool for designing methods of uncertainty quantification." In: *Journal of Computational Science* 11 (2015), pp. 46–57.

[11] A. Forrester, A. Sobester, and A. Keane. *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.

[12] J. Fosso-Tande. *Applications of Taylor series*. Aug. 2013.

[13] J. M. Gablonsky et al. "Modifications of the DIRECT Algorithm." In: (2001).

[14] GPy. *GPy: A Gaussian process framework in python*. `http://github.com/SheffieldML/GPy`. since 2012.

[15] J. Hammersley. *Monte carlo methods*. Springer Science & Business Media, 2013.

[16] S. S. Isukapalli. "Uncertainty analysis of transport-transformation models." In: (1999).

[17] D. R. Jones and J. R. Martins. "The DIRECT algorithm: 25 years Later." In: *Journal of Global Optimization* (2020), pp. 1–46.

[18] M. C. Kennedy and A. O'Hagan. "Predicting the output from a complex computer code when fast approximations are available." In: *Biometrika* 87.1 (2000), pp. 1–13.

[19] L. Le Gratiet. "Multi-fidelity Gaussian process regression for computer experiments." PhD thesis. Université Paris-Diderot-Paris VII, 2013.

[20] S. Lee, F. Dietrich, G. E. Karniadakis, and I. G. Kevrekidis. "Linking Gaussian process regression with data-driven manifold embeddings for nonlinear data fusion." In: *Interface focus* 9.3 (2019), p. 20180083.

[21] J. Mockus. *Bayesian approach to global optimization: theory and applications*. Vol. 37. Springer Science & Business Media, 2012.

[22] C. Z. Mooney. *Monte carlo simulation*. 116. Sage, 1997.

[23] L. W.-T. Ng and M. Eldred. "Multifidelity uncertainty quantification using non-intrusive polynomial chaos and stochastic collocation." In: *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference 20th AIAA/ASME/AHS Adaptive Structures Conference 14th AIAA*. 2012, p. 1852.

[24] J. Ocenasek, S. Kern, N. Hansen, and P. Koumoutsakos. "A mixed Bayesian optimization algorithm with variance adaptation." In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2004, pp. 352–361.

[25] B. Peherstorfer, K. Willcox, and M. Gunzburger. "Survey of multifidelity methods in uncertainty propagation, inference, and optimization." In: *Siam Review* 60.3 (2018), pp. 550–591.

[26] P. Perdikaris and G. Karniadakis. "Model inversion via multi-fidelity Bayesian optimization: A new paradigm for parameter estimation in haemodynamics, and beyond." In: *Journal of The Royal Society Interface* 13 (May 2016), p. 20151107. DOI: `10.1098/rsif.2015.1107`.

[27] P. Perdikaris, M. Raissi, A. Damianou, N. D. Lawrence, and G. E. Karniadakis. "Nonlinear information fusion algorithms for data-efficient multi-fidelity modelling." In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 473.2198 (2017), p. 20160751.

[28] P. Perdikaris, D. Venturi, and G. E. Karniadakis. "Multifidelity information fusion algorithms for high-dimensional systems and massive data sets." In: *SIAM Journal on Scientific Computing* 38.4 (2016), B521–B538.

[29] C. E. Rasmussen. "Gaussian processes in machine learning." In: *Summer School on Machine Learning*. Springer. 2003, pp. 63–71.

[30] R. C. Smith. *Uncertainty quantification: theory, implementation, and applications*. Vol. 12. Siam, 2013.

[31] J. Snoek, H. Larochelle, and R. P. Adams. "Practical bayesian optimization of machine learning algorithms." In: *arXiv preprint arXiv:1206.2944* (2012).

[32] C. Soize. *Uncertainty quantification*. Springer, 2017.

[33] G. Strang and K. Borre. *Linear algebra, geodesy, and GPS*. Siam, 1997.

[34] M. Titsias and M. Lázaro-Gredilla. "Doubly stochastic variational Bayes for non-conjugate inference." In: *International conference on machine learning*. PMLR. 2014, pp. 1971–1979.

[35] S. T. Tokdar and R. E. Kass. "Importance sampling: a review." In: *Wiley Interdisciplinary Reviews: Computational Statistics* 2.1 (2010), pp. 54–60.

[36] Z. Wang and S. Jegelka. "Max-value entropy search for efficient Bayesian optimization." In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3627–3635.

[37] T. Weise. "Global optimization algorithms-theory and application." In: *Self-Published Thomas Weise* (2009).

[38] J. Wu, M. Poloczek, A. G. Wilson, and P. I. Frazier. "Bayesian optimization with gradients." In: *arXiv preprint arXiv:1703.04389* (2017).

[39] D. Xiu. "Efficient collocational approach for parametric uncertainty analysis." In: *Communications in computational physics* 2.2 (2007), pp. 293–309.

[40] D. Xiu. *Numerical methods for stochastic computations: a spectral method approach*. Princeton university press, 2010.

[41] X. Zhu, A. Narayan, and D. Xiu. "Computational aspects of stochastic collocation with multifidelity models." In: *SIAM/ASA Journal on Uncertainty Quantification* 2.1 (2014), pp. 444–463.

[42] Z. Zlatev, I. Dimov, I. Faragó, and Á. Havasi. *Richardson extrapolation: practical aspects and applications*. Vol. 2. Walter de Gruyter GmbH & Co KG, 2017.