Georgia Institute of Technology

Oral Qualifying Exam Report

Hyperkriging: Non-Markovian Multi-Fidelity Regression

Author:
Atticus Rex

Supervisor:
Elizabeth Qian, Ph.D.

Abstract

Multi-fidelity regression seeks to approximate an expensive or unknown "high-fidelity" function using high-fidelity data along with data collected from a set of computationally cheaper "low-fidelity" functions. Gaussian Process Regression is a Bayesian regression technique which models an unknown function as a multivariate Gaussian random variable. This enables the inference of a posterior distribution for the underlying function from known function evaluations at specific inputs. Existing multi-fidelity methods based on Gaussian Processes are discussed, including 1st-Order Autoregressive Cokriging (AR1), Nonlinear Autoregressive Gaussian Processes (NARGP), and Multi-Fidelity Deep Gaussian Processes (MF-DGP). We introduce Hyperkriging, a novel approach to multi-fidelity Gaussian Process regression which uses low-fidelity predictions as features in the high-fidelity regression. Documentation is provided on a "computational artifact" which consists of author-developed Gaussian Process and Multi-Fidelity libraries in Python. These libraries are used to illustrate the performance of single-fidelity Kriging, AR1, NARGP, and Hyperkriging on simple multi-fidelity test problems. The NARGP and Hyperkriging methods perform consistently across all four benchmark problems, while Kriging and AR1 methods perform worse when no linear mapping exists between fidelity levels.

April 3, 2025

Contents

Symbols & Notation

$\mathbf{x}$	Input vector in $\mathbb{R}^d$
$\mathbf{x}_*$	Unseen test input in $\mathbb{R}^d$
f	General function mapping a vector in $\mathbb{R}^d$ to a scalar QoI
f_{hi}	High-fidelity function mapping a vector in $\mathbb{R}^d$ to some scalar QoI.
f_ℓ	The ℓth low-fidelity function mapping a vector in $\mathbb{R}^d$ to a scalar QoI.
C_ℓ	The cost to evaluate the ℓth fidelity function.
N_ℓ	The number of observations available of the ℓth fidelity function.
$\mathcal{N}(\mu, \Sigma)$	Gaussian Normal probability distribution with mean μ and variance Σ
$\mathbb{E}[\cdot]$	Expected value of a random variable
$\mathbb{V}[\cdot]$	Variance of a random variable
$\mathbf{X}$	Input data-matrix in $\mathbb{R}^{d\times \text{\# of samples}}$ where each column is a sample of $\mathbf{x}$
$\mathbf{Y}$	Vector in $\mathbb{R}^{\text{\# of samples}}$ corresponding to function evaluations at each column of $\mathbf{X}$
$\mathbf{Z}_\ell$	Feature-matrix in $\mathbb{R}^{d+\ell-1\times N_\ell}$ containing $\mathbf{X}_\ell$ and low-fidelity functions evaluated at $\mathbf{X}_\ell$.
$\mathbf{Z}_\ell^*$	Vector in $\mathbb{R}^{d+\ell-1}$ containing input $\mathbf{x}_*$ and low-fidelity functions evaluated at $\mathbf{x}_*$.
$\lvert\cdot\rvert$	Matrix determinant
$\boldsymbol{\kappa}(\mathbf{x}, \mathbf{x}' \vert \theta)$	A kernel function parameterized by θ which maps $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$ to $\mathbb{R}$
$\mathbf{K}(\mathbf{X}, \mathbf{X})$	The kernel matrix with entries following $\mathbf{K}_{ij} = \boldsymbol{\kappa}(\mathbf{x}_i, \mathbf{x}_j)$
$\mathcal{GP}(\mathbf{X}, \mathbf{Y}, \boldsymbol{\kappa})$	A Gaussian Process parameterized by inputs $\mathbf{X}$, outputs $\mathbf{Y}$, and a kernel function $\boldsymbol{\kappa}$.

Chapter 1

Introduction

Many scientific and engineering methods depend on repeated physical experiments or evaluations of a computational model. This is known as **many-query** analysis, which naturally arises in optimization, uncertainty quantification, and inverse problems. A **surrogate** model is an approximation of a known system. Finding cheap, accurate surrogate models can enable previously infeasible many-query analysis. This report focuses on using **supervised machine learning** with Gaussian Processes to train surrogate models. As a motivating example, engineering design optimization seeks to find optimal system parameters to minimize a cost function. Iteratively adapting the design of an airfoil to minimize its coefficient of drag is an example of such a problem. In this scenario, the coefficient of drag is typically not readily available through an analytical solution and therefore must be obtained through experiments or computer models. If these models are expensive to evaluate, the optimization process may not converge to optimal results within time and computational constraints.

We consider such an experiment or simulation as a function which maps known system inputs to some unknown quantity of interest (QoI). The expensive or unknown function is referred to as the **high-fidelity** function and is considered to be the most accurate representation of the system. We define **low-fidelity** functions as cheaper approximations of the high-fidelity function. These low-fidelity functions usually come from either simplifying assumptions (e.g. switching from a 3-D spatial model to a 2-D or 1-D model, conditions like steady-state, linearizing about an equilibrium, etc.) or a lower spatiotemporal resolution (e.g. using a coarser mesh for finite-element simulations, using a larger timestep in dynamical system simulations, etc.). **Multi-fidelity** surrogate modeling combines high and low-fidelity data to train a model which is more accurate than any low-fidelity function, but is cheaper to evaluate than the high-fidelity function. In this report, we consider a non-intrusive setting in which we only have access to input-output data collected from each fidelity level. Because the high-fidelity function is expensive, we assume the number of high-fidelity training data points is small, perhaps containing fewer than ten points.

Gaussian Processes and Deep Neural Networks [1]–[4] are two common approaches used to approximate high-fidelity functions from multi-fidelity data. Both methods have universal function approximation properties and can therefore model complex input-output relationships. Gaussian Processes, specifically, yield closed-form expressions for the uncertainty associated with their predictions. However, these uncertainty estimates are only accurate if the underlying function can be modeled a Gaussian Process parameterized by a specific covariance kernel. Further, the computational cost to train a Gaussian Process grows cubically with the amount of training data. Many methods exist to speed up the training, parameter optimization and prediction processes. Greedy basis selection [5], Nystrom approximation [6], and partial Cholesky factorization [7] give low-rank approximations of the kernel matrix for more efficient storage and inversion of the kernel matrix. In the case of millions of training data points, functional stochastic gradient descent has been used to find an optimal surrogate without ever storing a kernel matrix [8]. Work exists using heuristic strategies to optimize the kernel parameters of Gaussian Processes, such as particle swarm optimization [9] and frequency domain analysis [10].

Under the assumption that the number of high-fidelity data points is small, Gaussian Processes are generally feasible for high-fidelity function approximation. For this reason and the closed-form uncertainty estimates, many existing multi-fidelity surrogate modeling approaches are built off of this framework. One such approach is that proposed by Kennedy & O'Hagan in [11], which uses a linear mapping between fidelity levels and corrects for discrepancies with a Gaussian Process. The entire model starts from the lowest fidelity level and predicts the output at the next fidelity level until the high-fidelity approximation is reached. This technique is referred to in this report as a "first-order autoregressive" (AR1) strategy. Since its publication, many modifications to the AR1 predictor emerged such as using autoregressive neural-networks instead of Gaussian Processes [1], hybrid sampling techniques [12], sample size optimization [13], projection-based model reduction [14], and non-uniform sampling [15]. These Gaussian Process methods and the AR1 estimator differ only in model class and their acquisition of training data—they all use the same input features and covariance kernels. However, as proposed by Perdikaris et al. in [16], novel multi-fidelity *kernels* began to emerge as effective tools to predict high-fidelity outputs. The construction of multi-fidelity kernels outperforms AR1-style estimators in the presence of nonlinear relationships between fidelity levels. Deep Gaussian Processes consist of multiple layers of Gaussian Processes, which can model complex functions through sequential, nonlinear transformations of the inputs [2]. In [17], [18], multi-fidelity modeling is framed as a Deep Gaussian Process where information propagates from the lowest fidelity level to the highest .

These existing multi-fidelity methods are constructed using a "Markovian" property: the approximation of each fidelity level only uses information from the fidelity level immediately beneath it. This property implies that the lowest fidelity level can provide no more information about the high-fidelity function than the second highest fidelity function. This Markovian property is limiting because the lowest-fidelity functions may contain unique statistical information about the high-fidelity function which can be leveraged for more accurate prediction. The main contribution of this report is the removal of the Markovian property from the existing Gaussian Process estimators. We propose a novel approach which uses all low-fidelity functions as *features* to train a Gaussian Process predicting the output of the high-fidelity function, which we refer to as "Hyperkriging". We also remove the assumption that the low-fidelity functions are approximated with Gaussian Processes and generalize Hyperkriging to allow any regression model to predict the low-fidelity function outputs. This allows for greater flexibility in training the low-fidelity surrogate models, which are most expensive when the number of training data points is large. We also present a computational artifact which consists of a Gaussian Process Regression package using `jax` (a Python library for parallel array-based computation) and a Multi-Fidelity Toolbox package which implements existing multi-fidelity regression methods. We then use these libraries to compare the new method with existing methods on four synthetic test problems.

Chapter 2

Background & Existing Work

In this chapter, we define the problem of high-fidelity surrogate modeling, give theoretical background on Gaussian Process Regression, and outline three existing multi-fidelity methods.

2.1 Problem Statement

Denote $f_{\text{hi}} : \mathbb{R}^d \mapsto \mathbb{R}$ as the true high-fidelity function, which maps an input vector $\mathbf{x} \in \mathbb{R}^d$ to a scalar QoI. The goal of surrogate modeling is to produce some $\hat{f}_{\text{hi}}$ which optimally approximates f_{hi} over the domain of $\mathbf{x}$. We define a general Gaussian Process model as the conditional distribution of the model's prediction at an unseen test input $\mathbf{x}_*$ given training data $(\mathbf{X}, \mathbf{Y})$:

$$\mathcal{GP}(\mathbf{X}, \mathbf{Y}, \boldsymbol{\kappa}) = \mathcal{N}\left(\mathbb{E}\left[f(\mathbf{x}_*)|\mathbf{X}, \mathbf{Y}\right], \mathbb{V}\left[f(\mathbf{x}_*)|\mathbf{X}, \mathbf{Y}\right]\right) \tag{2.1}$$

where the expectation and variance are defined in Equations 2.9b and 2.9c. Specifically, we solve a supervised regression problem using a Gaussian Process of the following form:

$$\hat{f}_{\text{hi}} = \mathcal{GP}(\mathbf{Z}_{\text{hi}}, \mathbf{Y}_{\text{hi}}, \boldsymbol{\kappa}) \tag{2.2}$$

where $\mathbf{Z}_{\text{hi}} \in \mathbb{R}^{z \times N_{\text{hi}}}$ contains the input training data, $\mathbf{Y}_{\text{hi}} \in \mathbb{R}^{N_{\text{hi}}}$ contains the output training data, and $\boldsymbol{\kappa}$ is the kernel covariance function. Each column of $\mathbf{Z}_{\text{hi}}$ is the input features $\mathcal{Z}_{\text{hi}} : \mathbb{R}^d \mapsto \mathbb{R}^z$ evaluated at a specific input $\mathbf{x}_i$:

$$\mathbf{Z}_{\text{hi}} = \begin{bmatrix} | & & | \\ \mathcal{Z}_{\text{hi}}(\mathbf{x}_1) & \dots & \mathcal{Z}_{\text{hi}}(\mathbf{x}_{N_{\text{hi}}}) \\ | & & | \end{bmatrix} \tag{2.3}$$

The specific surrogate modeling problem addressed in this report answers the question:

"How can we choose the input features $\mathcal{Z}_{\text{hi}} : \mathbb{R}^d \mapsto \mathbb{R}^z$ and kernel covariance function $\boldsymbol{\kappa} : \mathbb{R}^z \times \mathbb{R}^z \mapsto \mathbb{R}$ such that $\hat{f}_{\text{hi}}$ optimally approximates f_{hi}?"

In the following sections, we show how existing single and multi-fidelity methods based on Gaussian Processes can be interpreted as specific choices of $\mathcal{Z}_{\text{hi}}$ and $\boldsymbol{\kappa}$ to solve the surrogate modeling problem.

2.2 Gaussian Process Regression

Summarized thoroughly in [19], Gaussian Process Regression (GPR)—sometimes referred to as Kriging—is a supervised machine learning technique. GPR seeks to approximate an unknown function

$f : \mathbb{R}^d \mapsto \mathbb{R}$ using only matched input-output evaluations of this function. Suppose we have N samples of an input variable, $\mathbf{x} \in \mathbb{R}^d$, concatenated into a data matrix, $\mathbf{X} \in \mathbb{R}^{d \times N}$:

$$\mathbf{X} = \begin{bmatrix} | & & | \\ \mathbf{x}_1 & \dots & \mathbf{x}_N \\ | & & | \end{bmatrix} \tag{2.4}$$

GPR assumes models f as a random variable following a prior multivariate normal distribution. This prior functional distribution is defined by:

$$f(\mathbf{X}) \sim \mathcal{N}\left(\mathbf{0}, \mathbf{K} + \sigma^2 \mathbf{I}\right) \tag{2.5}$$

where σ^2 is the variance of any Gaussian white noise in the observations and $\mathbf{K}$ is a covariance matrix whose entries are defined by some symmetric positive definite (SPD) covariance **kernel**, $\boldsymbol{\kappa} : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$:

$$\mathbf{K}_{ij} = \boldsymbol{\kappa}(\mathbf{x}_i, \mathbf{x}_j | \theta) \tag{2.6}$$

where the kernel function is parameterized by θ. The most common choice of SPD kernel is the Radial Basis Function (RBF) kernel, often referred to as the Gaussian or Squared Exponential kernel. The RBF kernel is defined as:

$$\boldsymbol{\kappa}(\mathbf{x}_i, \mathbf{x}_j | \theta) = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||_2^2}{2\theta^2}\right) \tag{2.7}$$

If the RBF kernel is used to define the functional distribution in 2.5, all functions contained in this distribution are infinitely differentiable [19]. The notation $\mathbf{K} = \boldsymbol{\kappa}(\mathbf{X}, \mathbf{X})$ will denote the $N \times N$ kernel matrix whose entries are evaluations of the kernel function, $\boldsymbol{\kappa}(\cdot, \cdot | \theta)$, at each pair of training inputs (see Equation 2.6). We then define $\mathbf{x}_*$ as some arbitrary test input at which we seek to approximate f. The distribution of $f(\mathbf{X})$ and $f(\mathbf{x}_*)$ is multivariate Gaussian defined by:

$$\begin{bmatrix} f(\mathbf{X}) \\ f(\mathbf{x}_*) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\kappa}(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I} & \boldsymbol{\kappa}(\mathbf{X}, \mathbf{x}_*) \\ \boldsymbol{\kappa}(\mathbf{x}_*, \mathbf{X}) & \boldsymbol{\kappa}(\mathbf{x}_*, \mathbf{x}_*) \end{bmatrix}\right) \tag{2.8}$$

If f varies according to 2.8, the posterior distribution of $f(\mathbf{x}_*)$ given a set of training data, $\{\mathbf{X}, \mathbf{Y}\}$, is multivariate normal defined by:

$$P(f(\mathbf{x}_*) | \mathbf{X}, \mathbf{Y}) = \mathcal{N}\left(\mathbb{E}\left[f(\mathbf{x}_*) | \mathbf{X}, \mathbf{Y}\right], \mathbb{V}\left[f(\mathbf{x}_*) | \mathbf{X}, \mathbf{Y}\right]\right) \tag{2.9a}$$

$$\mathbb{E}\left[f(\mathbf{x}_*) | \mathbf{X}, \mathbf{Y}\right] = \boldsymbol{\kappa}(\mathbf{x}_*, \mathbf{X}) \left(\boldsymbol{\kappa}(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}\right)^{-1} \mathbf{Y} \tag{2.9b}$$

$$\mathbb{V}\left[f(\mathbf{x}_*) | \mathbf{X}, \mathbf{Y}\right] = \boldsymbol{\kappa}(\mathbf{x}_*, \mathbf{x}_*) - \boldsymbol{\kappa}(\mathbf{x}_*, \mathbf{X}) \left(\boldsymbol{\kappa}(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}\right)^{-1} \boldsymbol{\kappa}(\mathbf{X}, \mathbf{x}_*) \tag{2.9c}$$

This estimator holds for any SPD kernel, $\boldsymbol{\kappa}(\cdot, \cdot | \theta)$, which defines a prior functional distribution. If we assume $\boldsymbol{\kappa}$ is parameterized by θ, the most common method to choose optimal kernel parameters and the noise variance, σ^2, is by maximizing the log-likelihood of sampling the training-data:

$$\ell(\theta,\sigma^2) = \log(P(\mathbf{Y}|\mathbf{X},\theta,\sigma^2)) \tag{2.10a}$$
$$= -\frac{1}{2}\left[\mathbf{Y}^\top\left(\mathbf{K}(\theta)+\sigma^2\mathbf{I}\right)^{-1}\mathbf{Y} + \log\left(|\mathbf{K}(\theta)+\sigma^2\mathbf{I}|\right) + \log(2\pi)\right] \tag{2.10b}$$

We can achieve optimal kernel parameters by maximizing this log-likelihood-function through gradient-based or heuristic optimization with respect to θ and σ^2. This optimization problem amounts to the following simplified minimization problem:

$$\theta^*,\sigma^* = \arg\min_{\theta,\sigma^2} \quad \mathbf{Y}^\top\left(\mathbf{K}(\theta)+\sigma^2\mathbf{I}\right)^{-1}\mathbf{Y} + \log\left(|\mathbf{K}(\theta)+\sigma^2\mathbf{I}|\right) \tag{2.11}$$

The optimization process involves the formation and inversion of a new kernel matrix, $\mathbf{K}(\theta)$, for each parameter update. Further, the problem is generally non-convex.

GPR can be broken up into a kernel matrix formation step, a training step and a prediction step. The formation and training steps are both offline processes that only need to be performed once. The prediction step is an online process which needs to be performed for each new input we wish to evaluate the model on. In the following bullets, we consider the cost to make *one* online prediction at some input $\mathbf{x}_*$. The cost to make M predictions is simply M times this cost. Suppose we have N input-output training datapoints of f. The algorithm proceeds as follows:

- **Formation & Training:** We first must form the kernel matrix, a non-trivial step. If evaluating the kernel has time-complexity $O(d)$ (where d is the input dimension), the total time-complexity to form the kernel matrix required for training the model is $\mathcal{O}(N^2d)$. The training step involves solving a linear system of size N:

  $$\boldsymbol{\alpha} = \left(\mathbf{K}+\sigma^2\mathbf{I}\right)^{-1}\mathbf{Y} \tag{2.12}$$

  The most efficient way to solve this system is with Cholesky Decomposition since $\left(\boldsymbol{\kappa}(\mathbf{X},\mathbf{X})+\sigma^2\mathbf{I}\right)$ is SPD. With this approach, we decompose $\mathbf{K}$ into the product of a lower-triangular matrix:

  $$\mathbf{L}\mathbf{L}^\top\boldsymbol{\alpha} = \mathbf{Y} \tag{2.13}$$

  Once the matrix $\mathbf{L}$ is formed, forward and backward substitution are used to efficiently solve for $\boldsymbol{\alpha}$. The formation of this Cholesky decomposition is $\mathcal{O}(N^3)$ and the forward/ backward solve is $\mathcal{O}(N^2)$. Note that this $\boldsymbol{\alpha}$ only needs to be computed and stored once (assuming optimal parameters).

- **Parameter Optimization** Let us assume the kernel parameter optimization as a result of gradient descent steps of the loss function in 2.10b takes G iterations to converge. Then, the kernel parameter optimization process will incur a complexity of $\mathcal{O}(GN^3)$.

- **Prediction:** The prediction step involves computing the following:

  $$\hat{f}(\mathbf{x}_*) = \boldsymbol{\kappa}(\mathbf{x}_*,\mathbf{X})\boldsymbol{\alpha} \tag{2.14}$$

  This is a dot-product which has total time-complexity $\mathcal{O}(Nd)$ per input, $\mathbf{x}_*$.

In summary, the Cholesky Decomposition of $\boldsymbol{\kappa}(\mathbf{X},\mathbf{X})+\sigma^2\mathbf{I}$ which runs in $\mathcal{O}(N^3)$ dominates the computational cost as N becomes large. The storage of complexity of GPR is quadratic: storing the training kernel matrix, $\mathbf{K}$, requires $\mathcal{O}(N^2)$ space which can also be limiting for large amounts of training data. However, once the model is trained, we only need $\mathcal{O}(N)$ space for storing $\boldsymbol{\alpha}$ and $\mathcal{O}(N)$ operations for online predictions.

2.3 Existing Multi-Fidelity Strategies

2.3.1 AR1: First-Order Autoregressive Gaussian Processes

A landmark example of an autoregressive multi-fidelity approach is that proposed by Kennedy & O'Hagan in [11]. The low-fidelity functions will be denoted $f_\ell : \mathbb{R}^d \mapsto \mathbb{R}$ where $\ell = 1, \ldots, K$ specifies the level of fidelity. A higher ℓ means a higher fidelity-level i.e. the function more closely resembles f_{hi}. The AR1 method approximates f_ℓ as follows:

$$f_\ell(\mathbf{x}) = \rho \cdot f_{\ell-1}(\mathbf{x}) + \delta(\mathbf{x}) \tag{2.15}$$

where ρ is a constant scaling $f_{\ell-1}$ to f_ℓ and encoding their correlation. $\delta(\mathbf{x})$ is the difference function to account for any variation not captured by ρ. In [11], $\delta(\mathbf{x})$ is a realization of a Gaussian Process:

$$\delta(\mathbf{x}) = \mathcal{GP}\left(\mathbf{X}_\ell, [\mathbf{Y}_\ell - \mathbf{Y}_{\ell-1}], \boldsymbol{\kappa}_\ell\right) \tag{2.16}$$

where $\mathbf{X}_\ell$ is the training inputs available at fidelity level ℓ and $\boldsymbol{\kappa}_\ell$ is the kernel covariance function at this fidelity level. The parameters ρ and θ (kernel parameters) are chosen via Maximum Likelihood Estimation in a manner similar to that described in Equation 2.11. LeGratiet & Garnier in [20] propose an efficient recursive strategy in which instead of the prior $f_{\ell-1}(\mathbf{x})$ informing the estimation of $f_\ell(\mathbf{x})$, we instead use the *posterior* result of GPR to obtain $f_{\ell-1}(\mathbf{x})$. This induces a more computationally efficient model, but requires the use of a specific nested sampling approach.

2.3.2 NARGP: Nonlinear Autoregressive Gaussian Processes

Improvements to the AR1 estimator include that proposed by Perdikaris et al. in [16]. This approach generalizes the realizations of ρ and $\delta(\mathbf{x})$ into one general Gaussian Process at each level, ℓ:

$$f_\ell(\mathbf{x}) = g_\ell(\mathbf{x}, f_{\ell-1}(\mathbf{x})) \tag{2.17}$$

Note that Equation 2.15 is an instance of this formulation. We also note that this expression takes into account both the input vector, $\mathbf{x}$, and the next lowest fidelity function, $f_{\ell-1}$. To leverage both quantities, the NARGP method introduces a kernel covariance function that takes this information into account:

$$\boldsymbol{\kappa}_\ell(\mathbf{x}, \mathbf{x}') = \boldsymbol{\kappa}(\mathbf{x}, \mathbf{x}'|\theta_p) \cdot \boldsymbol{\kappa}(f_{\ell-1}(\mathbf{x}), f_{\ell-1}(\mathbf{x}')|\theta_f) + \boldsymbol{\kappa}(\mathbf{x}, \mathbf{x}'|\theta_\delta) \tag{2.18}$$

This new kernel is the combination of three individual kernels: a scaling kernel $\boldsymbol{\kappa}(\mathbf{x}, \mathbf{x}'|\theta_p)$, a low-fidelity kernel $\boldsymbol{\kappa}(f_{\ell-1}(\mathbf{x}), f_{\ell-1}(\mathbf{x}')|\theta_f)$, and a baseline kernel $\boldsymbol{\kappa}(\mathbf{x}, \mathbf{x}'|\theta_\delta)$. Each of these kernels is parameterized by θ_p, θ_f, and θ_δ respectively. We can interpret the high-fidelity surrogate model as the following:

$$\hat{f}_{\text{hi}} = \mathcal{GP}\left(\mathbf{Z}_{\text{hi}}, \mathbf{Y}_{\text{hi}}, \boldsymbol{\kappa}_{\text{hi}}\right) \tag{2.19}$$

where the features used to form $\mathbf{Z}_{\text{hi}}$ are defined by:

$$\mathcal{Z}_{\text{hi}}(\mathbf{x}) = \begin{bmatrix} \mathbf{x} \\ f_K(\mathbf{x}) \end{bmatrix} \tag{2.20}$$

And $\kappa_{\text{hi}} : \mathbb{R}^{d+1} \times \mathbb{R}^{d+1} \mapsto \mathbb{R}$ is defined in Equation 2.18. From this formulation we see that the contributions of the NARGP method are twofold: a set of features to train $\hat{f}_{\text{hi}}$ and a unique kernel formulation.

2.3.3 MF-DGP: Multi-Fidelity Deep Gaussian Processes

The approach detailed by Cutajar et al. in [18] shows how multi-fidelity modeling can be interpreted as a Deep Gaussian Process. This approach is similar to that proposed by Perdkiaris in [16], however in [18], the authors propose a more complex multi-fidelity kernel:

$$\kappa_{\text{mf}} = \kappa_x(\mathbf{x}_i, \mathbf{x}_j|\theta_x) \left[\sigma_\ell^2 f_{\ell-1}(\mathbf{x}_i)^\top f_{\ell-1}(\mathbf{x}_j) + \kappa_f(f_{\ell-1}(\mathbf{x}_i), f_{\ell-1}(\mathbf{x}_j)|\theta_f)\right] + \kappa_\delta(\mathbf{x}_i, \mathbf{x}_j|\theta_{\ell,\delta}) \tag{2.21}$$

This kernel was shown to be competitive with NARGP for synthetic multi-fidelity test problems. In this approach, the unique contribution is a novel kernel design using the same features as Equation 2.20.

2.4 Limitations of Existing Methods

Early multi-fidelity methods assumed only two levels of fidelity—the existence of only one high and one low-fidelity function [11]. Subsequent methods extended this two-level approach to include more than one low-fidelity function. This extension naturally produced the Markovian property which implies that $f_{\ell-1}(\mathbf{x})$ provides all the necessary information to accurately predict the output of $f_\ell(\mathbf{x})$. While lower-fidelity functions may not be as strongly correlated with f_ℓ, they may still contain unique statistical information about f_ℓ that can be leveraged for more accurate prediction. This is the main limitation this report seeks to address: performing a regression at each fidelity level using all lower-fidelity function evaluations as auxiliary features.

Chapter 3

Method Overview

In this chapter, we define the input features and multi-fidelity kernel of the Hyperkriging model, outline its offline training and online prediction algorithms, provide a detailed explanation of the method, and analyze the computational cost of the algorithm.

3.1 The Hyperkriging Model

The main result in this report is an approach which removes the Markovian construction of the AR1, NARGP and MF-DGP estimators. Recall from the problem statement that we are concerned with training a Gaussian Process according to Equation 2.2. Hyperkriging approximates f_{hi} with features containing the input $\mathbf{x}$ and all low-fidelity function evaluations at $\mathbf{x}$:

$$\mathcal{Z}_{\text{hi}}(\mathbf{x}) = \begin{bmatrix} \mathbf{x} \\ f_1(\mathbf{x}) \\ \vdots \\ f_K(\mathbf{x}) \end{bmatrix} : \mathbb{R}^d \mapsto \mathbb{R}^{d+K} \tag{3.1}$$

A multi-fidelity kernel is formed as individual kernel evaluations at these features:

$$\boldsymbol{\kappa}(\mathcal{Z}_{\text{hi}}(\mathbf{x}), \mathcal{Z}_{\text{hi}}(\mathbf{x}')|\theta) = \boldsymbol{\kappa}_x(\mathbf{x}, \mathbf{x}'|\theta_x) \prod_{\ell=1}^{K} \boldsymbol{\kappa}_f(f_\ell(\mathbf{x}), f_\ell(\mathbf{x}')|\theta_\ell) \tag{3.2}$$

This kernel incorporates multi-fidelity information into the high-fidelity regression. The training and prediction of the Hyperkriging requires some structural assumptions about the training data. We assume the functions are ordered by computational cost:

$$\text{Cost of low-fidelity evaluations} = \{C_\ell\}_{\ell=1}^{K} \quad \text{where: } C_1 < \cdots < C_K \ll C_{\text{hi}} \tag{3.3}$$

Because we assume the low-fidelity functions are ordered according to increasing computational cost, we also assume that as a direct result of this ordering, the cheaper functions have more observations available. Hence, the sample sizes follow the ordering:

$$\text{Low-fidelity sample sizes} = \{N_1, \ldots, N_K\} \quad \text{where: } N_1 > \cdots > N_K \gg N_{\text{hi}} \tag{3.4}$$

where N_{hi} is the number of high-fidelity evaluations. Further, like many existing multi-fidelity methods, Hyperkriging adopts the *nested* sampling approach described in [20]. This means the input samples for one level of fidelity are a subset of the samples for the lower-level of fidelity. If we denote the input training data as $\mathbf{X}_\ell$, where ℓ is the fidelity-level, then $\mathbf{X}_{\text{hi}} \subset \mathbf{X}_K \subset \cdots \subset \mathbf{X}_1$. This induces zero error in the training process. This will be the *only* assumption we make about the ordering of f_ℓ. In many existing methods, the low-fidelity functions are assumed to be in order of decreasing accuracy, but this requirement is not necessary for the proposed Hyperkriging method.

Lastly, the existing methods discussed in Chapter 2 require that all low-fidelity surrogate models are Gaussian Processes. However, GPR becomes computationally infeasible when the amount of training data is large. Hence, in the proposed method we relax the requirement that the low-fidelity surrogate models must be Gaussian Processes to make the training and prediction process more computationally flexible. The only requirement of Hyperkriging is that the high-fidelity surrogate is a GPR model, but the low-fidelity regression models, denoted g_ℓ, can be any regression model. The offline training and online prediction for Hyperkriging is described in **Algorithm 1** and **Algorithm 2**, respectively.

Algorithm 1 Hyperkriging Offline Training

```
1: Inputs:
     • High and low-fidelity nested training data, $(\mathbf{X}_{\text{hi}}, \mathbf{Y}_{\text{hi}})$ and $\{(\mathbf{X}_\ell, \mathbf{Y}_\ell)\}_{\ell=1}^K$
     • Kernel function $\boldsymbol{\kappa} : \mathbb{R}^{d+K} \times \mathbb{R}^{d+K} \mapsto \mathbb{R}$
2: Outputs: Trained high and low-fidelity regression models: $(\boldsymbol{\alpha}, \theta^*)$ and $\{g_\ell\}_{\ell=1}^K$
3: Set $\mathbf{Z}_1 = \mathbf{X}_1$
4: for $\ell = 1 : K$ do
5:     Train model $g_\ell$ using training data $(\mathbf{Z}_\ell, \mathbf{Y}_\ell)$
6:     Form the inputs for the next level, $\mathbf{Z}_{\ell+1}^\top = \begin{bmatrix} \mathbf{X}_\ell^\top & f_1(\mathbf{X}_\ell)^\top & \dots & f_{\ell-1}(\mathbf{X}_\ell)^\top \end{bmatrix}$
7:     (Since $\mathbf{X}_{\ell+1} \subset \mathbf{X}_\ell$, then $f_\ell(\mathbf{X}_{\ell+1}) \subset \mathbf{Y}_\ell$.)
8: end for
9: Set $\mathbf{Z}_{\text{hi}} = \mathbf{Z}_{K+1}$
10: Train a Gaussian Process model by minimizing:
        $$\theta^* = \arg\min_\theta \quad \mathbf{Y}_{\text{hi}}^\top \mathbf{K}(\theta)^{-1} \mathbf{Y}_{\text{hi}} + \log(|\mathbf{K}(\theta)|) \qquad (3.5)$$
        $$\text{where} \quad \mathbf{K}(\theta) = \boldsymbol{\kappa}(\mathbf{Z}_{\text{hi}}, \mathbf{Z}_{\text{hi}} | \theta) + \sigma^2 \mathbf{I} \qquad (3.6)$$
11: Once $\theta^*$ is found, solve for $\boldsymbol{\alpha}$:
        $$\boldsymbol{\alpha} = \left(\boldsymbol{\kappa}(\mathbf{Z}_{\text{hi}}, \mathbf{Z}_{\text{hi}} | \theta^*) + \sigma^2 \mathbf{I}\right)^{-1} \mathbf{Y}_{\text{hi}} \qquad (3.7)$$
12: return $(\boldsymbol{\alpha}, \theta^*)$ and trained regression models $\{g_\ell\}_{\ell=1}^K$
```

Algorithm 2 Hyperkriging Online Prediction

```
 1: Inputs:
      • High-fidelity Gaussian Process parameters $(\boldsymbol{\alpha}, \theta^*)$
      • Kernel function $\boldsymbol{\kappa} : \mathbb{R}^{d+K} \times \mathbb{R}^{d+K} \mapsto \mathbb{R}$
      • Trained low-fidelity regression models $\{g_\ell\}_{\ell=1}^{K}$
      • Queried test input $\mathbf{x}_* \in \mathbb{R}^d$
 2: Outputs: Mean, $\mu$, and Variance, $\Sigma$, of Hyperkriging estimate of $f_{\text{hi}}(\mathbf{x}_*)$
 3: Set $\mathbf{Z}_1^* = \mathbf{x}_*$
 4: for $\ell = 1 : K$ do
 5:     Compute $g_\ell(\mathbf{Z}_\ell^*)$.
 6:     Form the inputs for the next fidelity level:
            $$\left(\mathbf{Z}_{\ell+1}^*\right)^\top \approx \begin{bmatrix} \mathbf{x}_*^\top & g_1(\mathbf{x}_*)^\top & \dots & g_\ell(\mathbf{Z}_\ell^*)^\top \end{bmatrix} \tag{3.8}$$
 7: end for
 8: Set $\mathbf{Z}_{\text{hi}}^* = \mathbf{Z}_{K+1}^*$
 9: Use $\boldsymbol{\alpha}$ and $\theta^*$ in the following kernel evaluations:
10: $\mu = \boldsymbol{\kappa}(\mathbf{Z}_{\text{hi}}^*, \mathbf{Z}_{\text{hi}})\boldsymbol{\alpha}$
11: $\Sigma = \boldsymbol{\kappa}(\mathbf{Z}_{\text{hi}}^*, \mathbf{Z}_{\text{hi}}^*) - \boldsymbol{\kappa}(\mathbf{Z}_{\text{hi}}^*, \mathbf{Z}_{\text{hi}}) \left(\boldsymbol{\kappa}(\mathbf{Z}_{\text{hi}}, \mathbf{Z}_{\text{hi}}) + \sigma^2\mathbf{I}\right)^{-1} \boldsymbol{\kappa}(\mathbf{Z}_{\text{hi}}, \mathbf{Z}_{\text{hi}}^*)$
12: return $(\mu, \Sigma)$
```

3.2 Detailed Explanation

We define $\mathbf{Z}_{\text{hi}} \in \mathbb{R}^{(d+K)\times N_{\text{hi}}}$ as $\mathcal{Z}_{\text{hi}}$ evaluated at the high-fidelity training data $\mathbf{X}_{\text{hi}}$ and $\mathbf{Z}_{\text{hi}}^* \in \mathbb{R}^{d+K}$ as $\mathcal{Z}_{\text{hi}}$ evaluated on some test input $\mathbf{x}_*$:

$$\mathbf{Z}_{\text{hi}} = \begin{bmatrix} \mathbf{X}_{\text{hi}} \\ f_1(\mathbf{X}_{\text{hi}}) \\ \vdots \\ f_K(\mathbf{X}_{\text{hi}}) \end{bmatrix}, \qquad \mathbf{Z}_{\text{hi}}^* = \begin{bmatrix} \mathbf{x}_* \\ f_1(\mathbf{x}_*) \\ \vdots \\ f_K(\mathbf{x}_*) \end{bmatrix} \tag{3.9}$$

We also introduced a covariance kernel which fuses the information from the input-variable $\mathbf{x}$ and the low-fidelity functions (see Equation 3.2):

$$\text{Cov}\left[f_{\text{hi}}(\mathbf{x}), f_{\text{hi}}(\mathbf{x}')\right] = \boldsymbol{\kappa}(\mathcal{Z}_{\text{hi}}(\mathbf{x}), \mathcal{Z}_{\text{hi}}(\mathbf{x}')) \tag{3.10}$$

This new kernel takes into account both the input point, $\mathbf{x}$, and the low-fidelity functions evaluated at $\mathbf{x}$. Intuitively, we hope the low-fidelity functions contain unique statistical information about f_{hi} not present in the input $\mathbf{x}$. Because engineering problems rarely have more than a dozen low-fidelity functions available, we assume $\mathcal{Z}_{\text{hi}}(\mathbf{x}_*) \in \mathbb{R}^{K+d}$ is sufficiently low-dimensional such that GPR does not break down due to the geometry of high dimensions (i.e. the curse of dimensionality). With this composite multi-fidelity kernel, we now describe an approach to accurately estimate f_{hi} at some arbitrary new input, $\mathbf{x}_*$. Recall that $\mathbf{X}_{\text{hi}} \in \mathbb{R}^{d \times N_{\text{hi}}}$ is the matrix of inputs for which we have empirical evaluations of f_{hi}. Denote $f_{\text{hi}}(\mathbf{X}_{\text{hi}}) = \mathbf{Y}_{\text{hi}} \in \mathbb{R}^{N_{\text{hi}}}$ as the vector of these outputs. $\mathbf{X}_{\text{hi}}$ and $\mathbf{Y}_{\text{hi}}$ form our high-fidelity training data, from which we can form a GPR estimator:

$$\mathcal{GP}\left(\mathbf{X}_{\text{hi}}, \mathbf{Y}_{\text{hi}}, \boldsymbol{\kappa}\right) \tag{3.11}$$

Because the Hyperkriging kernel, $\boldsymbol{\kappa}$, involves both the input, $\mathbf{x}$, and the low-fidelity functions evaluated at $\mathbf{x}$, this kernel must be evaluated at the training inputs *and* the testing inputs. This is because we need to form the following two kernel matrices:

$$\boldsymbol{\kappa}\left(\mathbf{Z}_{\text{hi}}, \mathbf{Z}_{\text{hi}}\right) \tag{3.12}$$

$$\boldsymbol{\kappa}\left(\mathbf{Z}^*_{\text{hi}}, \mathbf{Z}_{\text{hi}}\right) \tag{3.13}$$

Due to the nested nature of our training data, we have access to explicit evaluations of the low-fidelity functions at the training inputs, $\mathbf{X}_{\text{hi}}$. However, we do *not* have low-fidelity function evaluations at the test input, $\mathbf{x}_*$. Therefore, we will need to *approximate* $\mathbf{Z}^*_{\text{hi}}$ to form the kernel matrix in 3.13. Realizing the need to accurately estimate the low-fidelity function evaluations at $\mathbf{x}_*$, we see that we arrive at identical multi-fidelity problems with fewer low-fidelity functions. Our most uncertain low-fidelity function is f_K as it has the lowest sample size. Thus, we can leverage f_1 through f_{K-1} to provide as accurate an approximation of $f_K(\mathbf{x}_*)$ as possible. But we still need access to $f_{K-1}(\mathbf{x}_*)$, which is unknown. If we continue this logic down to the base case, we end up needing to approximate $f_1(\mathbf{x}_*)$. Because f_1 has the most available samples (since it is the cheapest model), we can produce a reliable single-fidelity approximation of this function. In actual experimental settings, it may be computationally feasible to simply compute $f_1(\mathbf{x}_*)$ directly, which would base the model in truth instead of an approximation. The algorithm unfolds as follows:

1. **Base-case:** Starting from the lowest fidelity-level, f_1 (which we assume has the most available training data), we need evaluations of f_1 at the desired test input $\mathbf{x}_*$.

   (a) If f_1 is available and sufficiently inexpensive, one option is to compute $f_1(\mathbf{x}_*)$ explicitly. This is desirable as it introduces zero uncertainty into our model.

   (b) If f_1 is unavailable or too expensive, we *approximate* $f_1(\mathbf{x}_*)$ using GPR or another regression model like a simple neural network or K-Nearest Neighbors regression if N_1 is too large for Cholesky decomposition. If we choose to model f_1 with a Gaussian Process, we have:

   $$f_1(\mathbf{x}_*) \approx g_1(\mathbf{x}_*) \tag{3.14}$$

   $$g_1 = \mathcal{GP}\left(\mathbf{X}_1, \mathbf{Y}_1, \boldsymbol{\kappa}_1\right) \tag{3.15}$$

   In the base-case, we have no lower-fidelity functions to leverage, which means our kernel takes the canonical form:

   $$\boldsymbol{\kappa}_1 = \boldsymbol{\kappa}(\mathbf{x}_i, \mathbf{x}_j | \theta_1) \tag{3.16}$$

   For smooth functions, a Gaussian/Radial Basis Function/Squared Exponential kernel optimized via the MLE approach in 2.10b typically gives good results.

2. **Iteration until high-fidelity is reached:** Once we have computed the base-case, we perform the following algorithm at each level, ℓ, increasing until the highest-fidelity function is reached:

   (a) At fidelity-level, ℓ, we seek to approximate $f_\ell(\mathbf{x}_*)$. We have the following training data available: the training inputs $\mathbf{X}_\ell \in \mathbb{R}^{d \times N_\ell}$ and training outputs $\mathbf{Y}_\ell \in \mathbb{R}^{N_\ell}$. We form $\mathbf{Z}_\ell$, the features $\mathcal{Z}_\ell$ evaluated at the training inputs:

$$\mathbf{Z}_\ell = \begin{bmatrix} \mathbf{X}_\ell \\ f_1(\mathbf{X}_\ell) \\ \vdots \\ f_{\ell-1}(\mathbf{X}_\ell) \end{bmatrix} \tag{3.17}$$

(b) With these extra features, we can approximate f_ℓ at the test-inputs using GPR:

$$f_\ell(\mathbf{x}_*) \approx g_\ell(\mathbf{x}_*) \tag{3.18}$$
$$g_\ell = \mathcal{GP}\left(\mathbf{X}_\ell, \mathbf{Y}_\ell, \boldsymbol{\kappa}_\ell\right) \tag{3.19}$$

where $\boldsymbol{\kappa}_\ell$ is the multi-fidelity kernel described in 3.2—$\boldsymbol{\kappa}_\ell(\mathcal{Z}_\ell(\mathbf{x}), \mathcal{Z}_\ell(\mathbf{x}')|\theta_\ell)$.

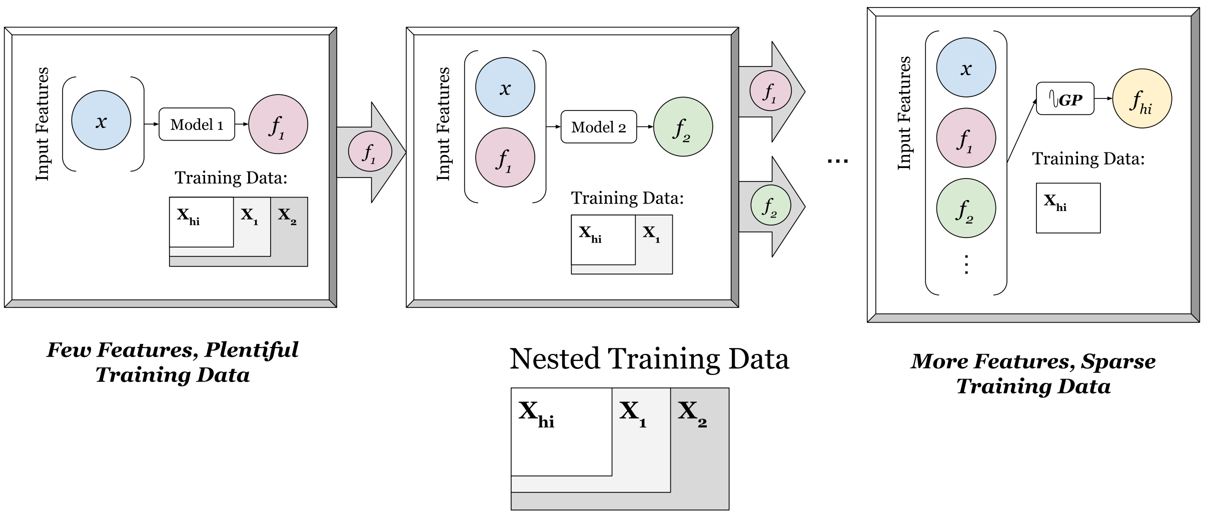

FIGURE 3.1: A conceptual diagram of the algorithm outlined in 3.1. Each model represents a regression from the inputs at the start of the arrows to the output at the end of the arrows. The algorithm starts at the left and moves rightward until it arrives at an estimate of f_{hi}. The algorithm begins with less informative features, but the most training data and ends with more features but the lowest amount of training data.

3.3 Algorithmic Cost Analysis

The Hyperkriging procedure defined in 3.1 forms a surrogate model for $f_{\text{hi}}(\mathbf{x}_*)$ when information about f_{hi} is scarce. It is often the case that the low-fidelity functions have so much available data that GPR is more expensive to train than other regression models. When this is the case, using alternative surrogate models like linear regression, K-Nearest Neighbors, or a simple neural network for the low-fidelity functions can lower offline training and online prediction cost. For this reason, we will offer algorithmic analysis in terms of general regression models for the low-fidelity functions, and a GPR for the high-fidelity so we obtain the analytical uncertainty estimates. In the following analysis, we approximate time-complexity in FLOPs and space complexity in the amount of floating point numbers stored.

Again, we only stipulate that the high-fidelity regression is a Gaussian Process—at each lower fidelity level, ℓ, we use a general regression model g_ℓ with an associated $C^{(\ell)}_{\text{train}}$ and $C^{(\ell)}_{\text{predict}}$ (the cost of offline training and online prediction, respectively). The computational cost of GPR can be extended to the low-fidelity models if it is used.

3.3.1 Data Acquisition

Multi-fidelity analysis hinges on the idea of having one expensive function and at least one computationally cheaper approximation available. Therefore, we must consider the computational cost to *obtain* the training data at each fidelity level ℓ. Recall that the cost to evaluate f_ℓ is given by C_ℓ. Hence, the total cost to acquire the training data is:

$$C_{\text{acquisition}} = N_{\text{hi}} C_{\text{hi}} + \sum_{\ell=1}^{K} N_\ell C_\ell \tag{3.20}$$

3.3.2 Offline Model Training

High-Fidelity Parameter Optimization

Because the high-fidelity observations are scarce, parameter optimization at the highest fidelity is computationally feasible and significantly improves the generalization of the overall surrogate model. This requires optimizing the loss function described in Equation 2.11. The minimization is usually accomplished via gradient-based optimization (e.g. ADAM, SGD, etc.) where the gradient with respect to the kernel-parameters is:

$$\frac{\partial \mathcal{L}}{\partial \theta_k} = \mathbf{Y}_{\text{hi}}^\top \mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta_k} \mathbf{K}^{-1} \mathbf{Y}_{\text{hi}} + \text{Tr}\left(\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta_k}\right) \tag{3.21}$$

Computing this gradient involves inverting the kernel matrix, and computing the derivative of the kernel-matrix with respect to all parameters (which in this case is $K + d$). The computation of the gradients involves the following steps:

- Forming the updated kernel matrix which takes $\mathcal{O}((K+d)N_{\text{hi}}^2)$ operations.
- Cholesky Decomposition $\mathbf{K} = \mathbf{L}\mathbf{L}^\top$ which incurs $\frac{1}{3}N_{\text{hi}}^3 + \mathcal{O}(N_{\text{hi}}^2)$ operations.
- Solving $\mathbf{K}^{-1}\mathbf{Y}_{\text{hi}}$ with forward and backward substitution, which incurs $\mathcal{O}(N_{\text{hi}}^2)$ operations.
- Differentiating $\mathbf{K}$ which we assume can be performed analytically and incurs no significant increase in computational cost. For a Gaussian Kernel, $\left[\frac{\partial K}{\partial \theta_k}\right]_{ij} = -\frac{1}{2}(\mathbf{x}_{i,k} - \mathbf{x}_{j,k})^2 \boldsymbol{\kappa}(\mathbf{x}_i, \mathbf{x}_j|\theta)$. In most cases, this incurs an additional $\mathcal{O}(N_{\text{hi}}^2)$ operations.
- Solving $\mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \theta_k}$ which can be accomplished via forward and backward substitution for each column of the matrix takes $\mathcal{O}(N_{\text{hi}}^3)$ operations.
- Multiplying the terms in the gradient takes $\mathcal{O}(N_{\text{hi}}^2)$ operations.

Hence, one parameter update step takes:

$$C_{\text{step}} = \mathcal{O}\left((K+d)N_{\text{hi}}^3 + (K+d)N_{\text{hi}}^2\right) \tag{3.22}$$

Computing G parameter update steps means the time-complexity of parameter optimization is:

$$C_{\text{optim}} = \mathcal{O}\left(G(K+d)N_{\text{hi}}^3 + G(K+d)N_{\text{hi}}^2\right) \tag{3.23}$$

Total Training Cost

Once these processes are executed, we have the following total cost to train the model:

$$C_{\text{total}} = C_{\text{optim}} + \sum_{\ell=1}^{K} C_{\text{train}}^{(\ell)} + C_{\text{acquisition}} \tag{3.24}$$

$$= \mathcal{O}\left(G\left[K+d\right] N_{\text{hi}}^3 + G\left[K+d\right] N_{\text{hi}}^2 + N_{\text{hi}} C_{\text{hi}} + \sum_{\ell=1}^{K} \left[C_{\text{train}}^{(\ell)} + N_\ell C_\ell \right] \right) \tag{3.25}$$

The total cost to train has a cubic time complexity in the number of high-fidelity training data points and dependent on the offline training and acquisition costs of the K low-fidelity models. The time-complexity is also dependent on the number of parameter optimization steps taken. The cubic complexity of N_{hi} dominates in the high-fidelity Gaussian Process training, though this complexity is agnostic of the training cost of the low-fidelity regressions.

3.3.3 Online Prediction

The cost of online prediction after the model has been trained is cheaper than the offline training process. Consider some queried input $\mathbf{x}_*$. For each low-fidelity function, we define a model prediction cost of $C_{\text{predict}}^{(\ell)}$ incurring a total online prediction cost of:

$$C_{\text{prediction}} = \mathcal{O}\left(N_{\text{hi}}(d+K) + \sum_{\ell=1}^{K} C_{\text{predict}}^{(\ell)} \right) \tag{3.26}$$

If we have M test inputs, the online prediction cost of M test inputs is $M \cdot C_{\text{prediction}}$.

Chapter 4

Computational Artifact

In this chapter, we provide documentation for the computational artifact requirement of the oral qualifying exam. My computational artifact consists of a Gaussian Process package and a Multi-Fidelity regression package.

4.1 Artifact Overview

1. Gaussian Process Regression Package

   Main Functionality

   - `GaussianProcess()` - creates a `GaussianProcess` object with the constructor.
   - `GaussianProcess.fit()` - trains a `GaussianProcess` object.
   - `GaussianProcess.predict()` - makes online predictions with a `GaussianProcess` object.
   - `GaussianProcess.optimize_hyperparams()` - optimizes the kernel parameters of a `GaussianProcess` object.

   Helper Functions:

   - `rbf_kernel()` - radial basis function kernel with diagonal covariance matrix.
   - `K()` - efficient kernel matrix formation using `jax.vmap()`.

2. Multi-Fidelity Toolbox Package

   - `MultiFidelityRegressor()` - creates an instance of a multi-fidelity regressor object.
   - `Kriging.fit()` - fits a kriging regression to the high-fidelity data.
   - `Kriging.predict()` - makes online predictions using a fitted kriging model.
   - `AR1.fit(), NARGP.fit(), PFSGP.fit()` - fits the multi-fidelity models to the entire multi-fidelity dataset.
   - `AR1.predict(), NARGP.predict(), PFSGP.predict()` - makes online predictions using the multi-fidelity models.

4.2 Gaussian Process Regression Package

For both my own understanding and the purposes of my research, I created my own package for Gaussian Process Regression using `jax`, documentation linked here. This is an optimized, array-based computational module developed by Google for Python. `jax` includes features like optimized matrix and vector operations for CPU and GPU architectures, as well as useful functionality for machine learning like automatic differentiation. The code I developed loosely mimics the style of the popular `scikit-learn` regression models. This package is agnostic of kernel choice, making it compatible with custom, user-defined kernel functions.

4.2.1 `GaussianProcess()` Constructor

Used to create an instance of a Gaussian Process.

Arguments:

- *None*

Keyword Arguments:

- `kernel_func` (default = `rbf_kernel`): *function*. The user may pass a custom kernel function to the constructor if the user does not wish to use the provided RBF kernel function. See below for the structure of custom kernel functions.
- `double_precision` (default = `False`): *boolean*. Specifies whether the user wishes to use double-precision (only available on specific CPU models). This generally makes the computation more stable.

Example Usage:

```
model = GaussianProcess(sigma_guess, double_precision=True, rcond=1e-12)
```

4.2.2 `GaussianProcess.fit()`

Used to train the model on the training data, for fixed `kernel_params` provided as input. It accomplishes the offline training process by solving:

$$\boldsymbol{\alpha} = (\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{Y}$$

Arguments:

- `X`: *array-like*. An 2d array where each column is a specific vector input to the model.
- `Y`: *array-like*. A 1d array containing the scalar outputs corresponding to each input in `X`.
- `kernel_params`: *array-like*. the kernel parameters of the kernel function which can be passed in as a `numpy` array. The dimension of this array will depend on the dimension of the input.

Keyword Arguments:

- `noise_var` (default = 0.0): *float*. A scalar denoting the variance of any Gaussian white noise in the outputs, `Y`.

Example Usage:

```
model.fit(X, Y, kernel _params, noise_var = 1.0e-5)
```

4.2.3 `GaussianProcess.predict()`

For online predictions once the model has been trained. Note that this function can only be called once $\boldsymbol{\alpha}$ has been computed. The online predictions at some new input, $\mathbf{x}_{\text{new}}$ are created using:

$$\hat{f}(\mathbf{x}_{\text{new}}) = \sum_{i=1}^{N} \alpha_i \cdot \boldsymbol{\kappa}(\mathbf{x}_{\text{new}}, \mathbf{x}_i; \theta)$$

Arguments:

- `Xtest`: *array-like.* An 2d array where each column is a specific vector input to the model at which the user would like to evaluate the model.

Keyword Arguments:

- `include_std` (default=`True`): *bool.* Specifies whether or not to return the analytical variance associated with the GP's predictions.

Returns:

- (*array-like*) The model's predictions for `Xtest`.
- (*array-like*) The variance associated with each prediction at `Xtest` (will only be returned if `include_std=True`).

Example Usage:

```
Yhat = model.predict(Xtest, include_std = True)
```

4.2.4 `GaussianProcess.optimize_kernel_params()`

For optimizing the kernel parameters. It accomplishes this using the ADAM algorithm with adaptive step-sizes. It minimizes the following loss-function:

$$\mathcal{L}(\theta) = \mathbf{Y}^{\top}\mathbf{K}(\theta)^{-1}\mathbf{Y} + \log(|\mathbf{K}(\theta)|)$$

Arguments:

- `kernel_param_guess`: *array-like.* The user must input a starting estimate for the kernel-parameters to initialize the optimization.

Keyword Arguments:

- `lr` (default = `1e-2`): *float.* The initial learning rate of the optimization. If this value is set too high, it may cause numerical instability in the solution.
- `tol` (default = `1e-8`): *float.* The tolerance for early termination of the gradient-descent step i.e. if the objective doesn't change by more than `tol` amount, we terminate the minimization.
- `max_iter` (default = 10000): *float.* The maximum allowed number of gradient-descent steps.
- `verbose` (default = `True`): If this is set to true, the script outputs the loss-function value at each step of the optimization. This is useful for debugging and assessing convergence rates, but if set to `False` the optimization is more efficient.

Example Usage:

```
model.optimize_hyperparams(sigma_guess, lr=1e-2, tol = 1e-8, max_iter = 10000, verbose=True)
```

4.2.5 `rbf_kernel()`

Radial Basis Function/Squared Exponential/Gaussian kernel with a diagonal covariance matrix:

$$\kappa(\mathbf{x}, \mathbf{x}'; \theta) = \exp\left(-(\mathbf{x} - \mathbf{x}')^\top \mathbf{\Sigma}^{-1}(\mathbf{x} - \mathbf{x}')\right)$$

where $\mathbf{\Sigma} = \text{diag}(\theta)$. This kernel is provided as the default in the constructor of this package. Passing in custom kernels to the GaussianProcess() constructor will only be compatble with the rest of the package if it uses this argument structure.

Arguments:

- `x1, x2`: *array-like*. 1-dimensional arrays of kernel inputs.
- `kernel_params`: *array-like*. The vector θ of kernel parameters which are used to form $\mathbf{\Sigma}$.

Returns:

- (*float*) The scalar RBF kernel evaluated at the two inputs.

Example Usage:

```
kernel_eval = rbf_kernel(x1, x2, np.array([0.1, 0.1]))
```

4.2.6 `K()`

Used to efficiently form a kernel matrix by using the `jax.vmap()` function to efficiently call the `kernel_func` on each entry of the matrix.

Arguments:

- `X1, X2`: *array-like*. 2-dimensional arrays of kernel inputs. These matrices should have the same number of rows as this is the dimension of the input vectors.
- `kernel_params`: *array-like*. The vector θ of kernel parameters which are used to form $\mathbf{\Sigma}$.

Returns:

- (*array-like*) the kernel-matrix where the ijth entry is the kernel function evaluated at column i of `X1` and column j of `X2`.

Example Usage:

```
Ktrain = K(Xtrain, Xtrain, rbf_kernel, np.array([0.1, 0.1]))
```

4.3 Multi-Fidelity Toolbox Package

I also developed a `multifitoolbox` package which implements the following four methods:

- **Kriging**: Kriging/Gaussian Process Regression performed on the high-fidelity data without taking any low-fidelity information into consideration. Used as a baseline to assess performance gains from the multi-fidelity algorithms.
- **AR1**: The first-order autoregressive method described in section 2.3.1.
- **NARGP**: The Nonlinear Autoregressive Gaussian Process method described in section 2.3.2.
- **Hyperkriging**: The approach proposed in this report.

4.3.1 `MultiFidelityRegressor()` - Constructor

This is the constructor syntax compatible with any child class of the `MultiFidelityRegressor` object. The child classes are `Kriging`, `AR1`, `NARGP`, and `PFSGP`. These child-classes are constructed exactly the same as the parent class.

Arguments:

- `K`: *int*. The number of levels of fidelity.
- `Ns`: *list/tuple*. A list of the number of datapoints of each fidelity-level.
- `data_dict`: *dict*. A dictionary where the keys are the fidelity-levels and the values are dictionaries with keys 'X' and 'Y' for the input and output data at that fidelity-level.

Example Usage:

```
   model = Kriging(2, [10, 250], {0:{'X':Xtrain_0, 'Y':Ytrain_0}, 1:{'X':Xtrain_1, 'Y':Ytrain_1
}})
```

4.3.2 `Kriging.fit()`

For training a Kriging model using only the the high-fidelity data.

Keyword Arguments:

- `sigma_guess` (default = `None`): *array-like*. An array for the initial guess for the kernel kernel parameters.
- `lr` (default = `1e-6`): The learning rate for the optimization algorithm.
- `max_iter` (default = `500`): The maximum number of iterations for the optimization algorithm.

Example Usage:

```
   model.fit(sigma_guess = np.array([1.0, 1.0]), lr= 1e-3, max_iter = 2500)
```

4.3.3 `Kriging.predict()`

For making test predictions using a trained Kriging model.

Arguments:

- `Xtest`: *array-like*. The input data for which to make predictions.

Returns:

- `Yhat`: *array-like*. The predicted output data.

Example Usage:

```
Yhat = model.predict(Xtest)
```

4.3.4 `AR1.fit(), NARGP.fit(), PFSGP.fit()`

The `.fit()` and `.predict()` commands work the exact same for the these three multi-fidelity models.

For training a multi-fidelity model to all the data across fidelities.

Keyword Arguments:

- `param_scale` (default = 1e-4): *float*. A value for scaling the initial guess for the kernel parameters. Setting this value too high will make the kernel-matrix singular to working precision.
- `lr` (default = `1e-6`): The learning rate for the optimization algorithm.
- `max_iter` (default = `500`): The maximum number of iterations for the optimization algorithm.

Example Usage:

```
model.fit(param_scale = 1e-6, lr= 1e-3, max_iter = 2500)
```

4.3.5 `AR1.predict(), NARGP.predict(), PFSGP.predict()`

For making online predictions using a trained multi-fidelity model.

Arguments:

- `Xtest`: *array-like*. The input data for which to make predictions.

Returns:

- `Yhat`: *array-like*. The predicted output data.

Example Usage:

```
Yhat = model.predict(Xtest)
```

Chapter 5

Preliminary Results

In this chapter, we provide an illustrative example to show the implementation of Hyperkriging on an analytical test problem, then compare the performance of Hyperkriging to single-fidelity Kriging, AR1, and NARGP methods.

5.1 Step-by-Step Numerical Example

To outline the use-case and clearly illustrate the performance of Hyperkriging, we will use a synthetic example. Consider the following scenario in which we have a scalar input, **x**:

$$f_{\text{hi}}(\mathbf{x}) = \sin(2\pi\mathbf{x}) \cdot \exp(-\mathbf{x}) \tag{5.1}$$

$$f_2(\mathbf{x}) = \sin(2\pi\mathbf{x}) \tag{5.2}$$

$$f_1(\mathbf{x}) = \exp(-\mathbf{x}) \tag{5.3}$$

Note that the high-fidelity function cannot be reproduced by any linear combination of f_1 and f_2; we must encode the nonlinear relationship between fidelities to form an accurate approximation $f_{\text{hi}}(\mathbf{x})$. Further, we only have $N_{\text{hi}} = 7$ high-fidelity training data points. We will generate observations of $\mathbf{x} \in \mathbb{R}$ via a uniform distribution on the interval [0,5] to produce training observations shown in Figure 5.1:

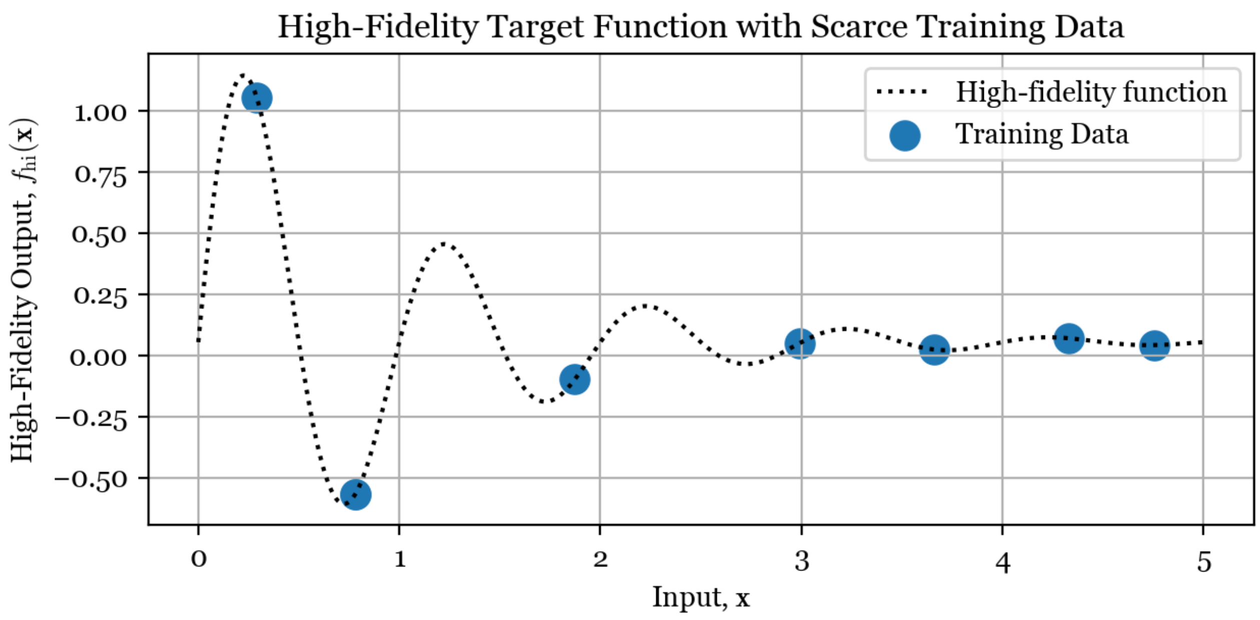

FIGURE 5.1: The target high-fidelity function with seven training observations.

Then, we randomly generate $N_2 = 25$ and $N_1 = 50$ training datapoints for f_2 and f_1, respectively. Suppose our desired test points, $\mathbf{X}_*$, are 1000 linearly spaced points along the interval [0,5]. This will give us a clear picture of the final approximation of f_{hi}. The first step is approximating the low-fidelity functions at these test points using the method described in Section 3.1. In this example, we will use Gaussian Processes to form the regressions for each fidelity level. Figure 5.2 shows the GPR approximation of $f_1 = \exp(-\mathbf{x})$.

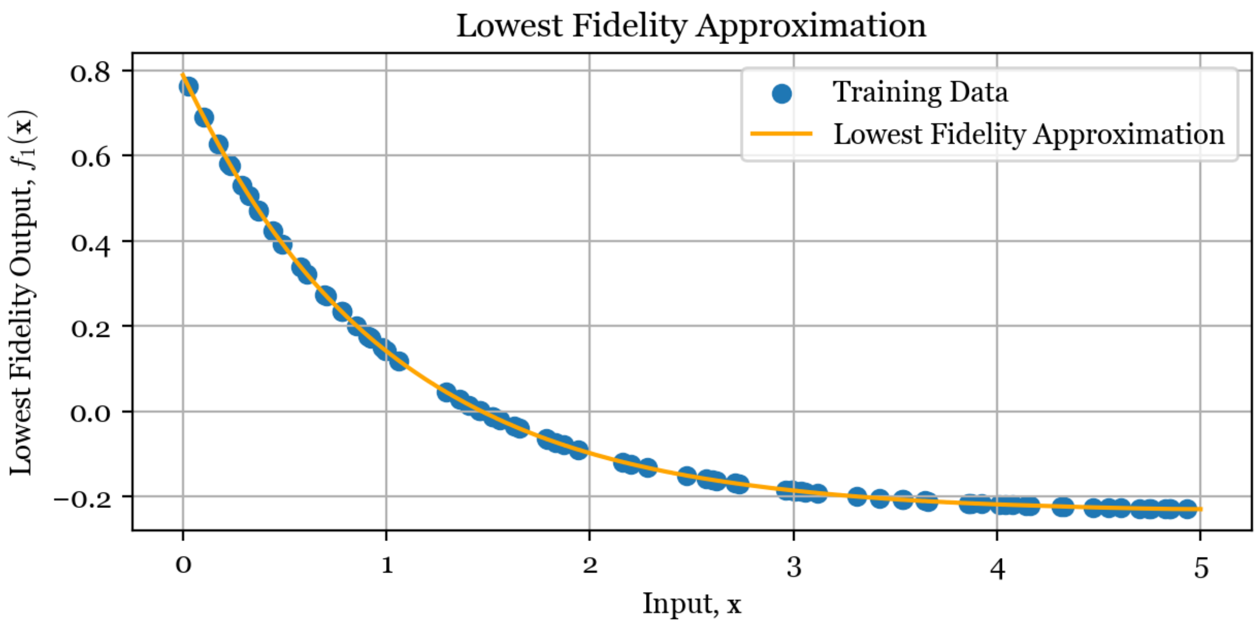

FIGURE 5.2: The lowest-fidelity function ($f_1 = \exp(-\mathbf{x})$) approximated with GPR.

Once we have an approximation of f_1, we use the training outputs for f_1 as features for a regression of $f_2 = \sin(2\pi\mathbf{x})$. The approximation of f_2 is shown in 5.3:

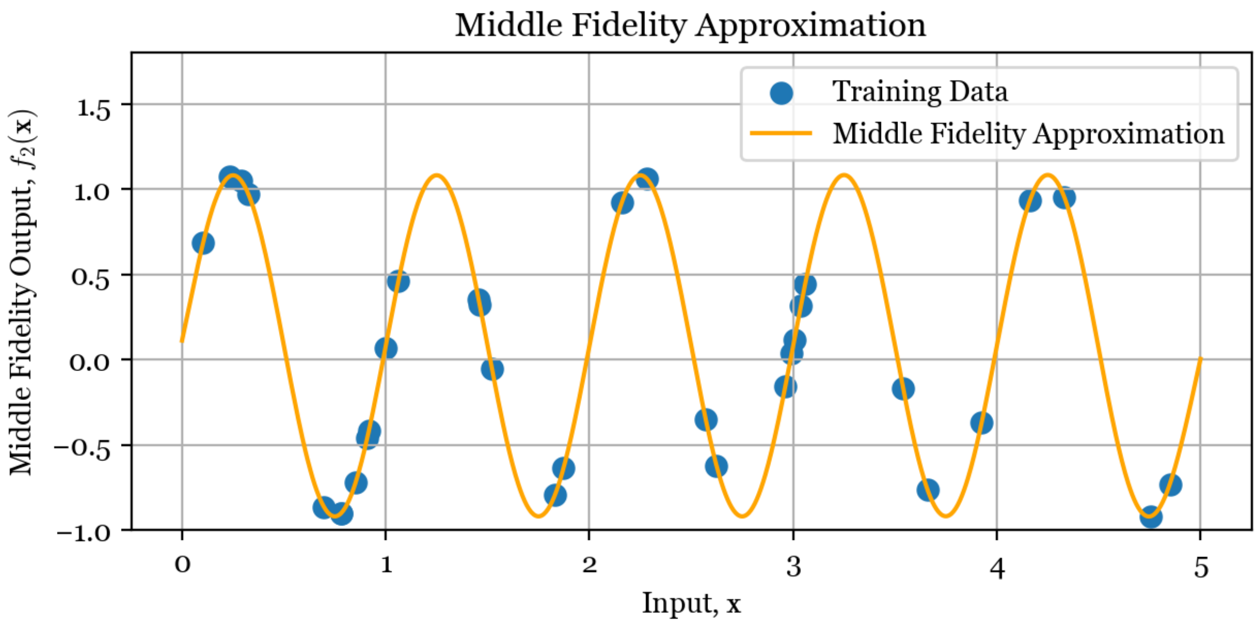

FIGURE 5.3: The highest low-fidelity function ($f_2 = \sin(2\pi\mathbf{x})$) approximated with GPR.

We observe accurate approximations of f_1 and f_2 in Figures 5.3 and 5.2. This is due to the larger number of training datapoints available at the lower fidelity levels. These approximations form additional features used to approximate f_{hi}. The final high-fidelity approximation using the Hyperkriging procedure is shown in Figure 5.4. A 95% Confidence Interval for the value of f_{hi} was constructed using the analytical estimator variance and is shown in Figure 5.4 by the orange shaded region.

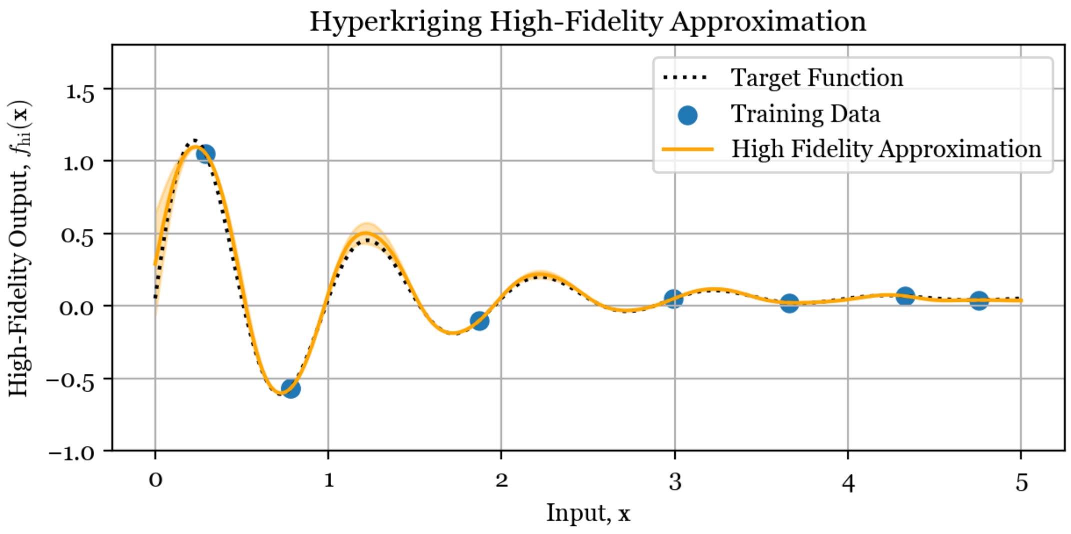

FIGURE 5.4: The multi-fidelity approximation of the high-fidelity function achieved using both low-fidelity approximations as features to aid the regression. The shaded region represents a 95% confidence interval for the model's predictions.

Figure 5.4 shows that in the presence of seven high-fidelity training data points, the Hyperkriging model can generate a narrow confidence interval that contains the true f_{hi} over almost the entire domain of $\mathbf{x}$.

5.2 Comparing Performance Across Methods

We will compare our Hyperkriging approach to single-fidelity Gaussian Process Regression (Kriging), 1st-order Autoregressive Cokriging (AR1) described [11], and Nonlinear Autoregressive Gaussian Processes (NARGP) described in [16]. We consider the last method to be state of the art in this problem. We will consider four multi-fidelity examples drawn from [18], with a third level of fidelity added to illustrate the advantage of Hyperkriging. We consider the functions described in table 5.1, the left column as the highest fidelity and the rightmost column as the lowest fidelity and being evaluated on the interval $x \in [0, 1]$:

Name	High-Fidelity	Middle-Fidelity	Low-Fidelity
Linear-A	$(6x-2)^2\sin(12x-4)$	$\frac{1}{2}y_h(x)+10(x-\frac{1}{2})+5$	$y_h(x)-12x^2$
Linear-B	$5x^2\sin(12x)$	$2y_h+(x^3-\frac{1}{2}\sin(3x-\frac{1}{2}))+4\cos(2x)$	$y_h+12\tanh(x-\frac{1}{2})$
Nonlinear-A	$(x-\sqrt{2})\,(y_2(x))^2$	$\sin(8\pi x)$	$e^{-\pi x}\cdot y_2(x)$
Nonlinear-B	$x\cdot e^{y_2(2x-0.2)}-1$	$\cos(15x)$	$\tanh(y_2(x)^2)$

TABLE 5.1: The four test problems used to evaluate the multi-fidelity models ranging with high-fidelity functions on the left and lowest-fidelity functions on the right.

Note that Linear-A and Linear-B are instances where the levels of fidelities are linearly related to the high-fidelity function, making them more conducive to the model form of AR1. This is not the case with Nonlinear-A and Nonlinear-B. All Gaussian Processes used in the model evaluations had their parameters optimized until convergence, to simulate "optimal" parameters across all models. Convergence was defined as 250 parameter updates without an improvement in the loss function. We plot the means and 95% Confidence intervals (shaded regions) produced by each method across the test problems in Figure 5.5.

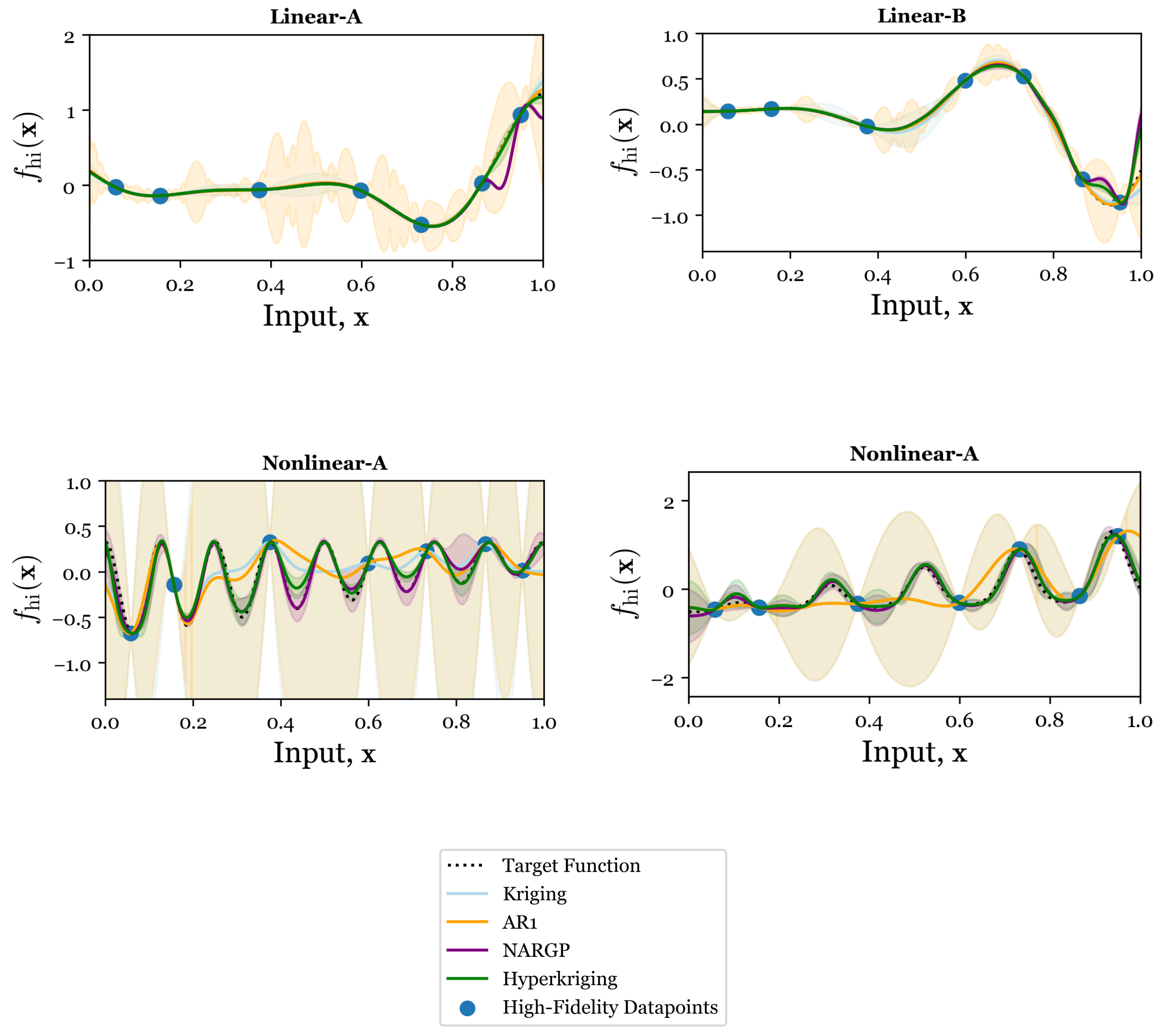

FIGURE 5.5: Kriging [19], AR1 [11], [20], NARGP [16], and Hyperkriging evaluated on the four test problems.

The Root Mean Squared Errors (RMSE) for each method evaluated on each of the test problems is illustrated in Table 5.2. As shown in Figure 5.5, when the levels of fidelity are linearly related, the Kriging and AR1 methods are comparably accurate with the NARGP and Hyperkriging methods. However, when there is a nonlinear relationship between fidelities, the accuracies of Kriging and AR1 decrease by approximately an order of magnitude. Not only this, but the confidence intervals associated with Kriging and AR1 predictions become about an order of magnitude wider (i.e. less confident). The Hyperkriging method performs consitently across all test problems, having the most accurate test RMSE on the Linear-A and Nonlinear-B problems. The NARGP method appears to fall into a local minimum during its parameter optimization in the Linear-A problem, evidenced by its deviation from the target function. The jaggedness of the confidence intervals produced by the AR1 method comes from the linear propagation of uncertainty up multiple levels of fidelity.

Method	**Linear-A**	**Linear-B**	**Nonlinear-A**	**Nonlinear-B**
Kriging	0.02001	0.03176	0.24515	0.31912
AR1	0.01126	<u>0.00628</u>	0.23825	0.31926
NARGP	0.10625	0.08283	<u>0.07224</u>	0.08846
Hyperkriging	<u>0.00828</u>	0.06205	0.08161	<u>0.07362</u>

TABLE 5.2: RMSE on Testing Data. Lowest errors are underlined for each test problem.

Chapter 6

Conclusions & Discussion

6.1 Summary of Problem

In this report, we discuss a multi-fidelity machine learning problem, in which we seek to accurately approximate a high-fidelity function. This is accomplished by combining data collected from the high-fidelity function and K low-fidelity functions. We assume we have a small number of training datapoints for the high-fidelity function due to its experimental infeasibility or computational cost. We also assume that the low-fidelity functions are arranged in order of increasing availability of training data such that $f_1(\cdot)$ has the most training data and $f_K(\cdot)$ has the least.

Three existing methods are discussed: the first-order autoregressive method utilized by Kennedy & O'Hagan [11], Nonlinear Auto Regressive Gaussian Processes [16], and Multi-Fidelity Deep Gaussian Processes [18]. These methods can provide accurate high-fidelity approximations from multi-fidelity data, but are limited by their construction of the Markovian property. By assuming an accuracy hierarchy, these methods imply that the fidelity-level immediately below a given level contains a sufficient amount of information to make accurate predictions. This is limiting because inaccurate low-fidelity functions can still produce measurable improvements in estimator accuracy.

6.2 Summary of Contributions

Our main contribution is the method we call Hyperkriging which takes advantage of a specific sampling strategy (the "nested" sampling style). This sampling style allows us to form a progressively larger set of features formed from low-fidelity function evaluations. The nested training data is necessary to introduce zero error in the offline model training. Some uncertainty is naturally introduced into the inputs during online prediction because we approximate the low-fidelity functions with surrogates instead of using their exact outputs. This progressive feature space is what provides a richer set of features used to approximate the high-fidelity function (conceptually illustrated in Figure 3.1). We give a detailed analysis of the computational cost of this method, whose offline cost is cubic in training data like the other existing methods and whose online cost is linear in its training data. Many methods exist to reduce the computational cost of kernel methods, some of which which are mentioned in the introduction.

Additionally, we present a computational artifact, which includes a Gaussian Process Regression package and a Multi-Fidelity Toolbox package, both implemented in Python. The GPR package is written using the high-performance array-based computing language, `jax`, and supports custom kernels, offline training, online prediction, and parameter optimization using the ADAM optimization algorithm. The Multi-Fidelity Toolbox package implements single-fidelity Kriging, AR1, NARGP, and Hyperkriging algorithms, making use of the GPR package.

6.3 Summary of Empirical Results

We compare the performances of the aforementioned methods on four test examples in Figure 5.5. Single-fidelity Kriging and AR1 methods performed more accurately on the Linear-A and Linear-B functions in which the high-fidelity function had a linear relationship with the low-fidelity observations. The AR1 model tended to have much higher model uncertainty than the other methods, largely because of how the uncertainty linearly propagates up from the lowest-fidelity function to the high-fidelity function in a way that is amplified as the number of levels of fidelity increases. We observe that the NARGP estimator strays from the target function in the Linear-A problem.

On the Nonlinear-A and Nonlinear-B problems, Kriging and AR1 produced less accurate point estimates than NARGP and Hyperkriging due to the nonlinear relationship between fidelity levels. The NARGP and Hyperkriging methods produced an order of magnitude more accurate approximations on the nonlinear test problems, with an order of magnitude narrower confidence intervals.

6.4 Discussion and Future Work

The results show Hyperkriging as a promising alternative to multi-fidelity regression that can produce measurable accuracy improvements over state of the art. Further, we generalize Hyperkriging by removing the computational burden of Gaussian Process Regression when the amount of low-fidelity training data is large. Using methods like linear regression, K-Nearest Neighbors, or neural networks can reduce both offline training and online prediction costs for small (if any) sacrifices in accuracy. Exploring optimal models to accomplish this is an area of ongoing investigation.

As discussed throughout the paper, the optimization of the kernel hyperparameters is a highly nontrivial task which can improve in the generality of the high-fidelity surrogate model. Currently, we explore gradient-based optimization, but work to accelerate this process using heuristic algoritms or approximation using geometric/statistical interpretations of the kernels could speed up the optimization methods discussed in this paper. Further, only three existing methods were implemented in this investigation, and comparing the GPR-based methods to other regression frameworks like neural-networks is necessary to fully comment on the true state of the art in multi-fidelity surrogate modeling.

Bibliography

[1] L. Tang, F. Liu, A. Wu, *et al.*, "A combined modeling method for complex multi-fidelity data fusion," en, *Machine Learning: Science and Technology*, vol. 5, no. 3, p. 035 071, Sep. 2024, Publisher: IOP Publishing, ISSN: 2632-2153. DOI: 10.1088/2632-2153/ad718f. [Online]. Available: https://dx.doi.org/10.1088/2632-2153/ad718f (visited on 11/18/2024).

[2] M. Raissi and G. Karniadakis, *Deep Multi-fidelity Gaussian Processes*, arXiv:1604.07484 [cs], Apr. 2016. DOI: 10.48550/arXiv.1604.07484. [Online]. Available: http://arxiv.org/abs/1604.07484 (visited on 12/31/2024).

[3] J. Kou and W. Zhang, "Multi-fidelity modeling framework for nonlinear unsteady aerodynamics of airfoils," *Applied Mathematical Modelling*, vol. 76, pp. 832–855, Dec. 2019, ISSN: 0307-904X. DOI: 10.1016/j.apm.2019.06.034. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0307904X19303968 (visited on 12/31/2024).

[4] S. Z. Ashtiani, M. Sarabian, K. Laksari, and H. Babaee, "Reconstructing blood flow in data-poor regimes: A vasculature network kernel for Gaussian process regression," *Journal of The Royal Society Interface*, vol. 21, no. 217, p. 20 240 194, Aug. 2024, Publisher: Royal Society. DOI: 10.1098/rsif.2024.0194. [Online]. Available: https://royalsocietypublishing.org/doi/full/10.1098/rsif.2024.0194 (visited on 11/18/2024).

[5] A. J. Smola and B. Schökopf, "Sparse Greedy Matrix Approximation for Machine Learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, ser. ICML '00, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Jun. 2000, pp. 911–918, ISBN: 978-1-55860-707-1. (visited on 03/13/2025).

[6] C. Williams and M. Seeger, "Using the Nyström Method to Speed Up Kernel Machines," in *Advances in Neural Information Processing Systems*, vol. 13, MIT Press, 2000. [Online]. Available: https://papers.nips.cc/paper_files/paper/2000/hash/19de10adbaa1b2ee13f77f679fa1483a-Abstract.html (visited on 03/13/2025).

[7] S. Fine, "Efficient SVM Training Using Low-Rank Kernel Representations," en,

[8] B. Dai, B. Xie, N. He, *et al.*, "Scalable Kernel Methods via Doubly Stochastic Gradients," in *Advances in Neural Information Processing Systems*, vol. 27, Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/hash/95d309f0b035d97f69902e7972c2b2e6-Abstract.html (visited on 12/20/2024).

[9] D. J. Toal, N. W. Bressloff, A. J. Keane, and C. M. Holden, "The development of a hybridized particle swarm for kriging hyperparameter tuning," *Engineering Optimization*, vol. 43, no. 6, pp. 675–699, Jun. 2011, Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/0305215X.2010.508524, ISSN: 0305-215X. DOI: 10.1080/0305215X.2010.508524. [Online]. Available: https://doi.org/10.1080/0305215X.2010.508524 (visited on 03/10/2025).

[10] J. H. S. de Baar, R. P. Dwight, and H. Bijl, "Speeding up Kriging through fast estimation of the hyperparameters in the frequency-domain," *Computers & Geosciences*, vol. 54, pp. 99–106, Apr. 2013, ISSN: 0098-3004. DOI: 10.1016/j.cageo.2013.01.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0098300413000307 (visited on 03/25/2025).

[11] M. C. Kennedy and A. O'Hagan, "Predicting the Output from a Complex Computer Code When Fast Approximations Are Available," *Biometrika*, vol. 87, no. 1, pp. 1–13, 2000, Publisher: [Oxford University Press, Biometrika Trust], ISSN: 0006-3444. [Online]. Available: https://www.jstor.org/stable/2673557 (visited on 11/23/2024).

[12] Q. Zhou, Y. Wu, Z. Guo, J. Hu, and P. Jin, "A generalized hierarchical co-Kriging model for multi-fidelity data fusion," en, *Structural and Multidisciplinary Optimization*, vol. 62, no. 4, pp. 1885–1904, Oct. 2020, ISSN: 1615-1488. DOI: 10.1007/s00158-020-02583-7. [Online]. Available: https://doi.org/10.1007/s00158-020-02583-7 (visited on 11/18/2024).

[13] Q. Zhou, Y. Wang, S.-K. Choi, P. Jiang, X. Shao, and J. Hu, "A sequential multi-fidelity metamodeling approach for data regression," *Knowledge-Based Systems*, vol. 134, pp. 199–212, Oct. 2017, ISSN: 0950-7051. DOI: 10.1016/j.knosys.2017.07.033. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705117303556 (visited on 11/18/2024).

[14] M. Xiao, G. Zhang, P. Breitkopf, P. Villon, and W. Zhang, "Extended Co-Kriging interpolation method based on multi-fidelity data," *Applied Mathematics and Computation*, vol. 323, pp. 120–131, Apr. 2018, ISSN: 0096-3003. DOI: 10.1016/j.amc.2017.10.055. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0096300317307646 (visited on 10/28/2024).

[15] B. Yang, B. Chen, Y. Liu, and J. Chen, "Gaussian process fusion method for multi-fidelity data with heterogeneity distribution in aerospace vehicle flight dynamics," *Engineering Applications of Artificial Intelligence*, vol. 138, p. 109 228, Dec. 2024, ISSN: 0952-1976. DOI: 10.1016/j.engappai.2024.109228. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197624013861 (visited on 11/18/2024).

[16] P. Perdikaris, M. Raissi, A. Damianou, N. D. Lawrence, and G. E. Karniadakis, "Nonlinear information fusion algorithms for data-efficient multi-fidelity modelling," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 473, no. 2198, p. 20 160 751, Feb. 2017, Publisher: Royal Society. DOI: 10.1098/rspa.2016.0751. [Online]. Available: https://royalsocietypublishing.org/doi/full/10.1098/rspa.2016.0751 (visited on 11/18/2024).

[17] L. Brevault, M. Balesdent, and A. Hebbal, *Overview of Gaussian process based multi-fidelity techniques with variable relationship between fidelities*, arXiv:2006.16728 [cs], Jun. 2020. DOI: 10.48550/arXiv.2006.16728. [Online]. Available: http://arxiv.org/abs/2006.16728 (visited on 01/30/2025).

[18] K. Cutajar, M. Pullin, A. C. Damianou, N. D. Lawrence, and J. I. González, "Deep Gaussian Processes for Multi-fidelity Modeling," *ArXiv*, Mar. 2019. [Online]. Available: https://www.semanticscholar.org/paper/Deep-Gaussian-Processes-for-Multi-fidelity-Modeling-Cutajar-Pullin/484ddd91f273edf4201e3d001d0b29a52fa27eac (visited on 12/08/2024).

[19] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning* (Adaptive computation and machine learning), en, 3. print. Cambridge, Mass.: MIT Press, 2008, ISBN: 978-0-262-18253-9.

[20] L. Le Gratiet, "Recursive co-kriging model for Design of Computer experiments with multiple levels of fidelity with an application to hydrodynamic," Sep. 2012. [Online]. Available: https://hal.science/hal-00737332 (visited on 10/30/2024).