The present work was submitted to the Institute for Data Science in Mechanical Engineering and is a joint collaboration between the Institute for Data Science in Mechanical Engineering and the Institute for Dynamic Systems and Control.
Diese Arbeit wurde vorgelegt am Institut für Data Science im Maschinenbau und ist eine Zusammenarbeit zwischen dem Institut für Data Science im Maschinenbau und dem Institute for Dynamic Systems and Control.

Master Thesis

Masterarbeit

Approximate Model Predictive Control with Kernel-based Methods

Approximative modellprädiktive Regelung mit kernbasierten Methoden

Presented by / Vorgelegt von

Abdullah Tokmak [1,2]

Matr.Nr.: 368331

Supervised by / Betreut von	Dr. Johannes Köhler[1]
	Christian Fiedler M.Sc.[2]
	Univ.-Prof. Dr. Sebastian Trimpe[2]
	Prof. Dr. Melanie N. Zeilinger[1]
1st Examiner / 1. Prüfer	Univ.-Prof. Dr. Sebastian Trimpe[2]
2nd Examiner / 2. Prüfer	Dr. Friedrich Solowjow[2]

[1] Institute for Dynamic Systems and Control, ETH Zurich

[2] Institute for Data Science in Mechanical Engineering, RWTH Aachen

Aachen, November 24, 2022

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne die Benutzung anderer als der angegebenen Hilfsmittel selbstständig verfasst habe; die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe des Literaturzitats gekennzeichnet.

Aachen, den November 24, 2022

Contents

Abstract

In this work, we develop an algorithm to automatically determine an explicit function that approximates a robust nonlinear model predictive controller (MPC). In contrast to the online optimization needed for MPC, the approximating function is cheap to evaluate, also on relatively inexpensive hardware. We approximate an MPC that is robust w.r.t. input disturbances. By combining a uniform bound on the approximation error with the robust MPC design, we can preserve all control theoretic guarantees induced by the MPC for the resulting approximating function.

We use kernel interpolation to learn the approximating function by sampling the MPC offline. We employ a local approximation setting, resulting in guaranteed bounds on the approximation error and a fast-to-evaluate approximating function. Moreover, we ensure scalability to millions of samples by employing an adaptive sub-domain partitioning that leads to well-conditioned covariance matrices. Furthermore, we use a novel heuristic to extrapolate an upper bound on the reproducing kernel Hilbert space (RKHS) norm of the ground truth from samples. In combination, this yields ALKIA-X, our adaptive and localized kernel interpolation algorithm with extrapolated RKHS norm. We provide a worst-case sample complexity bound for ALKIA-X and ensure that the desired bound on the approximation error holds under suitable conditions.

In a numerical experiment, ALKIA-X successfully approximates a nonlinear MPC benchmark with the desired accuracy. Compared to a current state-of-the-art nonlinear MPC approximation method, ALKIA-X has a significantly reduced offline training time, results in a notably faster online evaluation, and works automatically at the push of a button without requiring any human interaction. Furthermore, we provide an implementation in `Python` that is entirely parallelized. Although we developed ALKIA-X for MPC schemes, we expect that the algorithm is equally capable of automatically approximating a wide range of black-box functions with guaranteed error bounds.

Kurzzusammenfassung

In dieser Arbeit entwickeln wir einen Algorithmus zur automatischen Bestimmung einer expliziten Funktion, die einen robusten nichtlinearen modellprädiktiven Regler (engl. model predictive controller, MPC) approximiert. Im Gegensatz zur Lösung des Optimierungsproblems des MPC, ist die approximierende Funktion sogar auf relativ günstiger Hardware schnell auswertbar. Wir approximieren einen MPC, welcher robust gegenüber Eingangsstörungen ist. Daher ist die Gewährleistung eines Approximationsfehlers, welcher kleiner als die zulässige Eingangsstörung des robusten MPC ist, ausreichend, um die regelungstechnischen Garantien des MPC auf die resultierende Approximationsfunktion zu übertragen.

Wir verwenden kernbasierte Interpolation, um die Approximationsfunktion zu lernen. Wir entwickeln lokalisierte Approximationsfunktionen, die es ermöglichen, garantierte Grenzen für den Approximationsfehler zu bestimmen und zu einer schnell auswertbaren Approximationsfunktion führen. Außerdem gewährleisten wir Skalierbarkeit durch eine adaptive Aufteilung des globalen Gebiets in Teilgebiete. Darüber hinaus verwenden wir eine neue Heuristik zur Extrapolation einer oberen Schranke der Norm bzgl. des Hilbertraums mit reproduzierendem Kern (engl. reproducing kernel Hilbert space, RKHS) der wahren Funktion. Wir präsentieren ALKIA-X, unseren Algorithmus für lokalisierte und adaptive kernbasierte Interpolation mit extrapolierter RKHS-Norm.

In einem numerischen Experiment approximiert ALKIA-X ein nichtlineares MPC Beispiel erfolgreich mit der gewünschten Genauigkeit. Im Vergleich zu einem existierenden nichtlinearen MPC Approximationsverfahren hat ALKIA-X eine deutlich reduzierte Trainingszeit und führt zu einer Approximationsfunktion, die online schneller auswertbar ist. Außerdem läuft ALKIA-X automatisch auf Knopfdruck, ohne jegliche menschliche Interaktion. Wir stellen eine parallelisierte Implementierung von ALKIA-X in `Python` zur Verfügung, die zur Approximation von vielen Black-Box Funktionen genutzt werden kann.

Notations

Next, we present abbreviations and mathematical symbols used throughout the thesis. Further notations will be introduced in the respective sections.

Abbreviations

s.t.	such that
w.l.o.g.	without loss of generality
w.r.t.	with respect to
a.o.	among others
e.g.	for example
i.e.	that is
approx.	approximately, approximation, approximate, or approximating

Mathematical Symbols

$x \in \mathbb{R}^n$	input
$\mathcal{X} \subseteq \mathbb{R}^n$	domain
$w : \mathcal{X} \to \mathbb{R}$	width of domain $\mathcal{X}$, i.e., $w(\mathcal{X}) = \max_{x_i, x_j \in \mathcal{X}} \|x_i - x_j\|_\infty$
$X \subseteq \mathcal{X}$	set of samples $\{x_1, \dots, x_N\}$
$f : \mathcal{X} \to \mathbb{R}$	ground truth
f_X	f evaluated at X
$h_X : \mathcal{X} \to \mathbb{R}$	given samples X, resulting approximating function
$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$	kernel function as a function of two inputs x and x'
$\widetilde{k} : \mathcal{X} \to \mathbb{R}$	kernel function as a function of the eucledian distance between two inputs $\|x - x'\|_2$
ℓ	length scale
$\widetilde{k}_\ell$	scaled kernel s.t. $\widetilde{k}_\ell(\|x - x'\|_2) = \widetilde{k}\left(\frac{\|x - x'\|_2}{\sqrt{2}\ell}\right)$
$k_X : \mathcal{X} \to \mathbb{R}$	given samples X, resulting covariance vector
K_X	given samples X, resulting covariance matrix
$P_X : \mathcal{X} \to \mathbb{R}$	given samples X, resulting power function
$\overline{\epsilon}$	maximum approximation error
Δx_c	edge length of a local cube
L	Lipschitz constant
N	number of total samples
M	number of samples per axis
$\|\cdot\|_k$	RKHS norm w.r.t. the RKHS of kernel k
$\overline{\Gamma}$	upper bound on the RKHS norm of the ground truth
$\hat{\Gamma}_X$	overestimation of the RKHS norm of the approximating function given samples X
$\widetilde{\Gamma} : \mathbb{N}_+ \to \mathbb{R}$	extrapolating function of the RKHS norm of the approximating function
μ_X	mean value of function evaluations f_X
$\mathrm{diag}(\cdot)$	diagonal matrix
$\mathbb{P}$	probability
$\mathbb{R}$	set of all real numbers
$\mathbb{N}$	set of all natural numbers
$\mathrm{conv}(\cdot)$	convex hull
$\mathbb{I}_N$	identity matrix $\in \mathbb{R}^{N \times N}$
$\mathbf{1}_N$	$[1, \dots, 1]^\top \in \mathbb{R}^N$
$\mathbf{0}_N$	$[0, \dots, 0]^\top \in \mathbb{R}^N$
$\mathrm{cond}(A)$	condition number of a matrix, $\mathrm{cond}(A) = \|A^{-1}\|_\infty \|A\|_\infty$
$\overline{\kappa}$	upper bound on the condition number of a matrix
$\mathrm{card}(\cdot)$	cardinality
t	time
$\mathcal{O}(\cdot)$	order of
$\frac{\partial}{\partial x} g(x)\|_{x'}$	derivative of a function $g(x)$ w.r.t. x, evaluated at x'

1 Introduction

In this chapter, we present the motivation for the thesis, the related work, and our contributions.

1.1 Motivation

Model predictive control (MPC) [1] is a modern control method based on repeatedly solving a multivariate and constrained optimization problem online to determine the optimal control input for every step. There exist robust MPC approaches that ensure robustness with, e.g., constraint tightening [2] and thus guarantee constraint satisfaction, stability, and recursive feasibility, also in presence of uncertainty or disturbance [3]. For nonlinear systems, the application of MPC requires solving a nonlinear program (NLP) online at each time step [4]. Therefore, the MPC law, which returns the optimal input given a state, is implicitly defined by the solution of an NLP. For this reason, MPC is computationally expensive and thus challenging to evaluate under real-time requirements. Therefore, we aim to determine an explicit function that approximates the MPC law. Particularly, the approximating function shall be cheap to evaluate, also on relatively inexpensive hardware. To nevertheless ensure safety and robustness, we require guarantees on the approximation error.

1.2 Related Work

Approximate MPC

In this thesis, we aim to automatically approximate a nonlinear MPC law with guarantees on the approximation error. For linear systems, approaches exist to obtain explicit control laws via multi-parametric approaches [5], [6]. However, these methods have scalability limitations. References [7], [8], [9], and [10] use neural networks

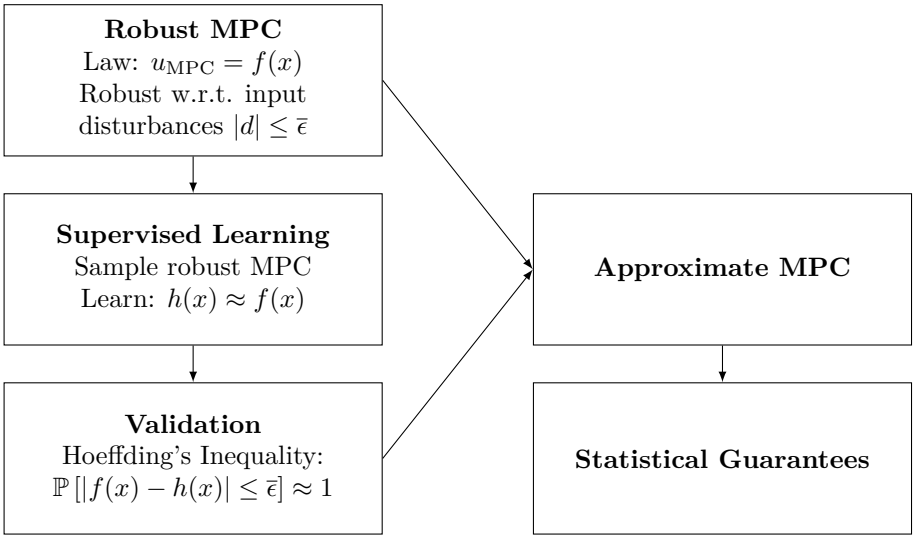

Figure 1.1: MPC approximation setting presented in [12] and [13]. The figure is taken from [12].

(NNs) [11] to approximate linear MPC laws and guarantee stability, constraint satisfaction, and recursive feasibility by adding a suitable projection or using primal-dual methods. Nevertheless, extensions to nonlinear systems are not straightforward. One approach to approximate MPC laws for nonlinear systems with closed-loop guarantees is presented in [12] and [13]. In particular, these references employ NNs to learn an explicit function that approximates a nonlinear MPC offline and use Hoeffding's inequality ([14], [15]) to give statistical guarantees on the approximation error.

Figure 1.1 shows the high-level idea presented in [12] and [13]. The MPC law given by $f : \mathcal{X} \to \mathbb{R}$ is implicitly defined by the underlying NLP and returns the optimal input for a given state. The approximating function $h : \mathcal{X} \to \mathbb{R}$ is learned from samples of the MPC law f. The main idea is to approximate an MPC law that can handle imperfect input. Therefore, [12] and [13] approximate a *robust* MPC, which is robust w.r.t. input disturbances up to a magnitude of $\bar{\epsilon}$. Therefore, if

$$|f(x) - h(x)| \overset{!}{\leq} \bar{\epsilon} \quad \forall x \in \mathcal{X}, \tag{1.1}$$

i.e., the uniform approximation error is smaller or equal to the allowed input error

given by the robust MPC scheme, then the approximating function preserves all control theoretic guarantees induced by the robust MPC. During the offline training, the robust MPC acts as a black-box and returns noise-free evaluations of the optimal input defined by the solution of the NLP. After approximating the MPC law and validating the approximation with Hoeffding's inequality, the explicitly defined approximate MPC has statistical guarantees on the approximation error. However, the separate learning and validation present in [12] and [13] lead to an iterative and non-trivial offline design that may limit automation. Thus, we use non-parametric regression approaches that inherently supply error bounds to automatically determine an explicit function that approximates a nonlinear robust MPC with guarantees on the approximation error.

Non-parametric Regression

For applications in control, non-parametric regression is primarily divided into Lipschitz- [16] and kernel-based [17] methods. Lipschitz-based approaches include Kinky Inference and nonlinear set membership concepts [18], [19]. References [20] and [21] use Lipschitz-based approximation to approximate an MPC. However, we choose to focus on kernel-based methods, which include support vector regression [22], [23], Gaussian processes (GPs) [24], kernel ridge regression [25], and kernel interpolation [26], [27], [28]. Reference [29] uses GPs to approximate an MPC, albeit only a linear MPC is considered.

Furthermore, the scalability of non-parametric approaches is crucial to retain prediction quality with increasing sample size. Reference [30] presents, a.o., a local approximation concept, in which the global domain is divided into sub-domains to ensure scalability. Our work uses kernel interpolation to approximate a nonlinear MPC and addresses scalability by partitioning the global domain into sub-domains.

Lazy Regression

Since we use the kernel-based approximating function to replace the online evaluation of the MPC law, we need to enable evaluation under real-time requirements. References [31] and [32] present suitable approaches and follow the idea of lazy regression, i.e., only using a localized batch of data for the online computation. We also follow an approach that only uses localized data for the online evaluation.

Error Bounds

We need to attach guarantees on the approximation error (cf. (1.1)) to ensure a safe and robust operation. Therefore, we require upper bounding it. For kernel interpolation, obtaining error bounds is accompanied by upper bounding the power function. This is translatable to upper bounding the variance for GP-based methods like, e.g., Bayesian optimization (BO) [33]. In BO, the problem of finding the maximum variance is usually addressed by gridding the domain to obtain an approximation of the maximum or by directly considering discrete domains [34], [35], [36]. However, neither approach fits our problem. Since we require a guaranteed upper bound on the power function to develop robust error bounds, we cannot settle for approximate upper bounds obtained by gridding. Moreover, we aim to approximate a continuous domain. Therefore, we propose a different approach to derive a *guaranteed* upper bound on the power function.

To develop error bounds, we additionally require an upper bound on the reproducing kernel Hilbert space (RKHS) norm of the ground truth. Comparatively, GP-based BO methods, which work in a Bayesian setup but require frequentist bounds [37], also need an upper bound on the RKHS norm. However, the RKHS norm is a fundamental characteristic of the unknown ground truth, and upper bounding it is still an open research question [37]. In BO, an upper bound on the RKHS norm of the unknown ground truth is often assumed to be known [34], [35], [38]. However, for Lipschitz-based non-parametric regression, heuristic approaches exist to obtain an approximation of the Lipschitz-constant of the unknown ground truth by acquiring samples and determining an overestimation [39], [40]. We follow a similar heuristic to overestimate the RKHS norm from samples and use an extrapolation, which leads to an implementable algorithm that does not require an upper bound on the RKHS norm of the ground truth a priori.

1.3 Contributions

In this thesis, we present an algorithm to automatically determine an explicit function that approximates a robust nonlinear MPC with guarantees on the approximation error. We use the same setting as [12] and [13] (cf. Figure 1.1), i.e., we approximate a nonlinear MPC that is robust w.r.t. input disturbances. Therefore,

it suffices to ensure that the error bound (1.1) holds to preserve all control theoretic guarantees induced by the robust MPC for the resulting approximating function. In contrast to [12] and [13], we use kernel interpolation to enable automated offline training by combining learning and validation.

Our first contribution is introducing local cubes and defining individual approximating functions on each local cube (cf. Chapter 3). This leads to a piecewise-defined approximating function and follows the idea of lazy regression. As a result, we derive guaranteed bounds on the approximation error. Additionally, the localized approximating functions are fast-to-evaluate.

Moreover, we enable scalability to millions of samples by adaptively partitioning the domain into sub-domains (cf. Chapter 4). Hence, we upper bound the maximum number of samples per sub-domain. As a result, we ensure well-conditioned covariance matrices, which leads to numerically reliable computations without requiring any regularization.

Additionally, we do not depend on an oracle that returns an upper bound on the RKHS norm of the ground truth. Specifically, we present a novel exponential extrapolation formula to obtain an upper bound on the RKHS norm of the ground truth from samples (cf. Chapter 5). In combination, this yields ALKIA-X, our adaptive and localized kernel interpolation algorithm with extrapolated RKHS norm. We provide a worst-case sample complexity bound and ensure that the desired bound on the approximation error holds under suitable conditions (cf. Theorem 4).

We provide an implementation of ALKIA-X in `Python` that is entirely parallelized and runs on a Linux server (cf. Chapter 6). ALKIA-X successfully approximates a two-dimensional nonlinear robust MPC benchmark with the desired accuracy in a numerical experiment. Compared to [12], ALKIA-X requires significantly less time for the offline learning and results in a notably faster online evaluation. Furthermore, in contrast to [12], ALKIA-X runs automatically at the push of a button without requiring any iterative human interaction (cf. Chapter 7). Moreover, we expect that ALKIA-X is capable of automatically approximating a wide range of black-box functions with guaranteed error bounds.

1.4 Outline

Next, we describe the structure of this thesis. In Chapter 2, we provide a concise background on kernel interpolation. Moreover, in Chapter 3, we present our localized kernel interpolation framework and derive guaranteed error bounds. In Chapter 4, we extend the framework by introducing an adaptive sub-domain partitioning. In Chapter 5, we present ALKIA-X, which yields from combining the proposed framework with a heuristic to extrapolate an upper bound on the RKHS norm of the ground truth. In Chapter 6, we describe the implementation of ALKIA-X in `Python`. Chapter 7 contains numerical experiments conducted with ALKIA-X to approximate an MPC benchmark. The thesis ends with a conclusion and an outlook presented in Chapter 8.

2 Background on Kernel Interpolation

In this chapter, we present a brief introduction to kernel interpolation based on [26]. For more details, we refer to [27] and [28].

We aim to approximate the ground truth $f : \mathcal{X} \to \mathbb{R}$ on the domain $\mathcal{X} \subseteq \mathbb{R}^n$. We query it and receive noise-free function evaluations

$$f_x = f(x) \quad x \in \mathcal{X}. \tag{2.1}$$

We can always consider scalar function evaluations f_x. For vector-valued functions, we choose to approximate each dimension individually, which results in multiple scalar approximation problems. We denote by $X = \{x_1, \dots, x_N\} \subseteq \mathcal{X}$ a set of samples. In this thesis, we consider the domain $\mathcal{X}$ to be a compact cube with a non-empty interior. Since we can arbitrarily scale the domain and solve an equivalent approximation problem, w.l.o.g, we set $\mathcal{X} = [0, 1]^n$. First, we introduce the kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, which is the central object in kernel interpolation.

Assumption 1.

1. *k is continuous*
2. *k is (strictly) positive definite, i.e., $\sum_{i=i}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j k(x_i, x_j) > 0 \; \forall \alpha_1, \dots, \alpha_N \in \mathbb{R} \setminus \{0\} \; \forall (x_i, x_j) \in X \times X$.*
3. *k is only a function of the euclidean distance, i.e., $\exists \widetilde{k} : \mathbb{R} \to \mathbb{R} : k(x, x') = \widetilde{k}(\|x - x'\|_2) \; \forall x, x' \in \mathcal{X}$*
4. *$\widetilde{k}$ is monotonically decreasing*
5. $\widetilde{k}(0) = 1$

The first two statements in Assumption 1 are standard assumptions for kernels. Statement three implies shift-invariance (cf. [41]) and a uniform length scale in all

dimensions. The fourth condition states that the value of the kernel function for two samples decreases as their euclidean distance increases. We can assume the final condition w.l.o.g., as it only influences the prefactor of the kernel.

Given samples X, the covariance matrix $K_X \in \mathbb{R}^{N\times N}$ is defined as

$$K_X = \begin{pmatrix} k(x_1,x_1) & k(x_1,x_2) & \cdots & k(x_1,x_N) \\ k(x_2,x_1) & k(x_2,x_2) & \cdots & k(x_2,x_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_N,x_1) & k(x_N,x_2) & \cdots & k(x_N,x_N) \end{pmatrix}. \tag{2.2}$$

In the following, we assume that the elements of the sample set $X \subseteq \mathcal{X}$ are pairwise distinct. Thus, K_X is positive definite. Given samples X, the covariance vector $k_X : \mathcal{X} \to \mathbb{R}^N$ is given by

$$k_X(x) = [k(x,x_1), k(x,x_2), \cdots, k(x,x_N)]^\top. \tag{2.3}$$

We denote by

$$P_X(x) := \sqrt{1 - k_X(x)^\top K_X^{-1} k_X(x)} \tag{2.4}$$

the power function $P_X : \mathcal{X} \to [0,1]$ given samples X, which is equal to the posterior standard deviation formula for GPs [24]. It holds that $P_X(x) = 0$ for all $x \in X$ [42]. Given samples X, the approximating function $h_X : \mathcal{X} \to \mathbb{R}$ is a weighted sum of kernels $k(x,x_i),\ x_i \in X$ and interpolates the given function samples, i.e., $h(x_i) = f_{x_i}\ \forall x_i \in X$. The resulting approximating function is the unique solution of a variational problem [26, Section 3.1] and is given by

$$h_X(x) = f_X^\top K_X^{-1} k_X(x). \tag{2.5}$$

Remark 1. *The squared-exponential (SE)-kernel [43] is defined as*

$$k(x,x') = \exp\left(-\|x-x'\|_2^2\right) \tag{2.6}$$

and the Matérn kernel [24] is given by

$$k(x,x') = \frac{2^{1-\nu}}{\gamma(\nu)} \left(\sqrt{2\nu}\|x-x'\|_2\right)^\nu K_\nu \left(\sqrt{2\nu}\|x-x'\|_2\right). \tag{2.7}$$

Assumption 1 holds for both kernels. We denote by $K_\nu(\cdot)$ a modified Bessel function, by $\gamma(\cdot)$ the gamma-function, and by ν the smoothness parameter. Moreover, we abbreviate the Matérn kernel with, e.g., $\nu = \frac{3}{2}$, as Ma32.

The kernel k is a *reproducing* kernel of the reproducing kernel Hilbert space (RKHS) H_k, if H_k is a Hilbert space of functions on $\mathcal{X}$ equipped with the inner-product $\langle \cdot, \cdot \rangle_k$ and it holds that

$$\begin{aligned} k(\cdot, x') &\in H_k \quad \forall x' \in \mathcal{X} \\ f(x') &= \langle f, k(\cdot, x') \rangle_k \quad \forall x' \in \mathcal{X},\ \forall f \in H_k \end{aligned} \tag{2.8}$$

[41], [44]. In fact, kernels and RKHSs have one-to-one correspondence, i.e., for every positive (semi) definite kernel k, there exists an RKHS for which k is the unique reproducing kernel [45].

Assumption 2. *The ground truth f is a member of the RKHS of the chosen kernel k, i.e., $\|f\|_k = \sqrt{\langle f, f \rangle_k} < \infty$.*

The RKHS norm of the approximating function w.r.t. the RKHS of kernel k is given by

$$\|h_X\|_k = \sqrt{f_X^\top K_X^{-1} f_X} \tag{2.9}$$

and it holds that $\|f\|_k \geq \|h_X\|_k$ for all $X \subseteq \mathcal{X}$ [26, Remark 2].

Lemma 1. *[26, Proposition 1] Let Assumption 1 and 2 hold. Further assume that the elements of the sample set $X \subseteq \mathcal{X}$ are pairwise distinct. Then, given X, we have that*

$$|f(x) - h_X(x)| \leq P_X(x) \sqrt{\|f\|_k^2 - \|h_X\|_k^2} \tag{2.10}$$

for all $x \in \mathcal{X}$.

3 Localized Kernel Interpolation

In this chapter, we develop an approximating function using kernel interpolation that ensures a uniform approximation error smaller or equal to the predefined maximum error $\bar{\epsilon}$ (cf. (1.1)). In Section 3.1, we discuss different sampling schemes and their challenges before presenting a suitable sampling approach and the resulting approximation framework. Furthermore, we develop guaranteed error bounds in Section 3.2 and describe the resulting pseudo algorithm in Section 3.3. In Section 3.4, we conclude the chapter by providing a proof on convergence after a finite number of samples (cf. Theorem 2) for the presented pseudo algorithm. Additionally, we prove that the resulting approximating function guarantees the desired accuracy.

3.1 Sampling Scheme and Approximation Framework

The ground truth f, which we aim to approximate, acts as a black-box function. Therefore, to obtain information to develop an approximating function, we have to sample f. Correspondingly, the black-box setting is also present in many BO approaches (cf. [46], [47]), where popular sampling schemes classify into two general classes: greedy sampling and uniform sampling. Following a greedy sampling scheme would require determining the maximum of the power function (cf. (2.4)).

Figure 3.1 illustrates the power function of the Ma32-kernel given 16 random samples. The resulting power function is nonconcave. In a greedy sampling setting, we would have to solve a challenging optimization problem to find the maximum of the nonconcave power function to acquire the next sample. Conversely, a uniform sampling scheme does not require the solution of an optimization problem to determine the next samples, as the sample acquisition is defined a priori.

However, we nevertheless need the maximum of the power function to upper bound the approximation error (cf. (2.10)). Approaches mentioned in Section 1.2 that use gridding to determine an approximate upper bound on the power function or that

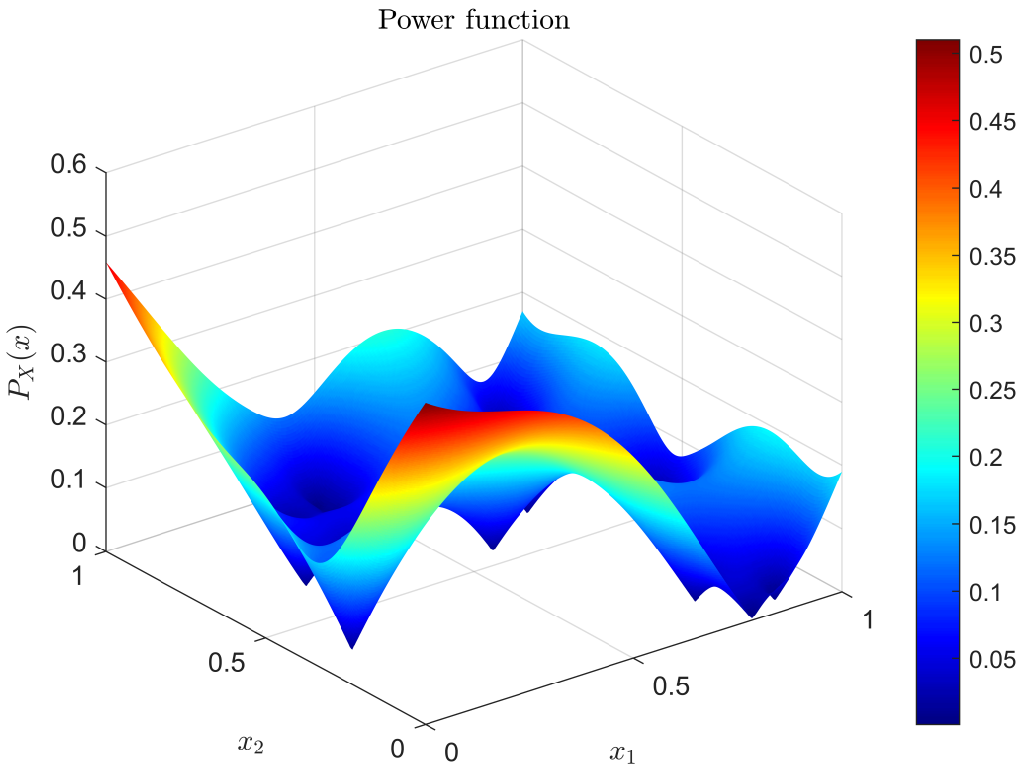

Figure 3.1: Power function (cf. (2.4)) of the Ma32-kernel given 16 random samples.

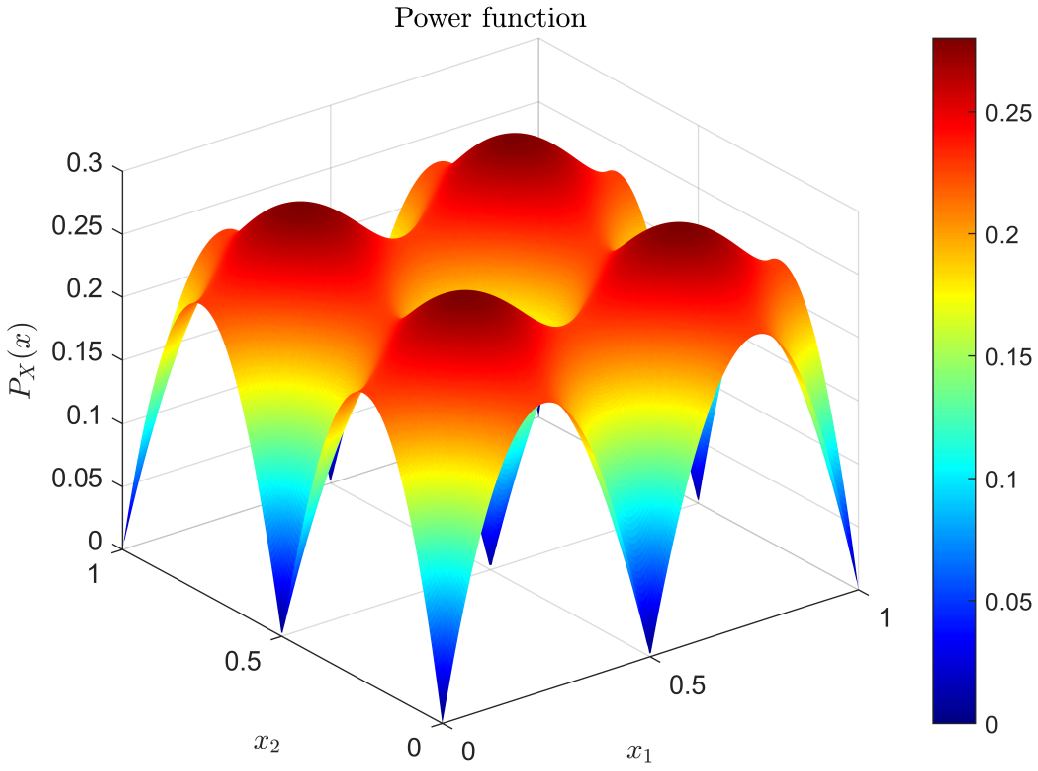

Figure 3.2: Power function (cf. (2.4)) of the Ma32-kernel given nine uniform samples.

assume a discrete domain are not applicable to our problem setting. Therefore, we have to find a guaranteed upper bound on the power function on a continuous domain.

Figure 3.2 shows the power function of the Ma32-kernel given nine uniform samples. Since the power function is nonconcave, finding its maximum to develop guaranteed bounds on the approximation error (cf. (2.10)) still requires the solution of a demanding optimization problem.

Furthermore, another important challenge is to ensure that the resulting approximating function is cheap to evaluate. Evaluating the approximating function (2.5) using all N samples requires an N-dimensional vector-matrix-vector multiplication. Since matrix multiplications scale with a complexity of approximately $\mathcal{O}\left(N^3\right)$ [24],

we choose to only use a subset of the available samples when constructing the approximating function. We presented the idea in Section 1.2 under the term lazy regression, where only a localized batch of data is used for the online prediction.

We now present the solution to the challenges of having to solve a difficult optimization problem to greedily acquire samples or, independent from the sampling scheme to upper bound the approximation error, and the necessity of ensuring a fast-to-evaluate approximating function. We sample *uniformly* to acquire samples without requiring the solution of a nontrivial optimization problem. Furthermore, we divide the domain $\mathcal{X}$ into *uniform local cubes* $\mathcal{X}_c$. Each local cube $\mathcal{X}_c$ is of the same size and has an edge length of Δx_c. Moreover, each local cube has 2^n samples $X_c \subseteq X$. The samples X_c are located at the vertices of $\mathcal{X}_c$. It holds that

$$\begin{aligned} \cup_{c\in\mathcal{C}} \mathcal{X}_c &= \mathcal{X} \\ \cup_{c\in\mathcal{C}} X_c &= X \\ \operatorname{card}(X_c) &= 2^n \\ \operatorname{conv}(X_c) &= \mathcal{X}_c, \end{aligned} \tag{3.1}$$

with $\mathcal{C} \subseteq \mathbb{N}$. For each local cube $\mathcal{X}_c$, we define an individual local approximating function that only uses the samples X_c. For the online prediction of an input x', only 2^n samples X_c of the local cube $\mathcal{X}_c \subseteq \mathcal{X}$ containing x' are used. As a result, evaluating the approximating function (2.5) boils down to evaluating a vector-matrix-vector product with 2^n samples, which scales significantly better than using all N samples. We denote by $h_{X_c}(x)$ the approximating function that is defined by the samples X_c and is therefore only valid on the local cube $\mathcal{X}_c$. This approach leads to a piecewise-defined approximating function. If we pick any $x' \in \mathcal{X}$, there will always be a local cube $\mathcal{X}_c \subseteq \mathcal{X}$ s.t. $x \in \mathcal{X}_c$. Therefore, there always exists a local approximating function $h_{X_c}(x)$ that is employed to approximate x'. Our original goal was to ensure that the uniform approximation error between the ground truth f and the approximating function is smaller or equal to $\bar{\epsilon}$ (cf. (1.1)). However, we now employ a piecewise-defined approximation framework with local approximating functions $h_{X_c}(x)$. Therefore, we have to ensure that the uniform error between the local approximating functions and the ground truth f is smaller or equal to $\bar{\epsilon}$, i.e.,

$$|f(x) - h_{X_c}(x)| \overset{!}{\leq} \bar{\epsilon} \quad \forall c \in \mathcal{C},\, \forall x \in \mathcal{X}_c. \tag{3.2}$$

The presented approximation framework enables developing a guaranteed bound on the approximation error (3.2), which we will present in Section 3.2.

In Chapter 4, we will introduce an adaptive sampling scheme. However, locally we will always sample uniformly and we will preserve defining individual approximating functions for each local cube.

3.2 Guaranteed Error Bounds

In this section, we develop guaranteed bounds on the approximation error. Since we use the introduced piecewise-defined approximation framework on uniform local cubes (cf. Section 3.1), the approximation error (cf. Lemma 1) is now upper bounded by

$$|f(x) - h_{X_c}(x)| \leq P_{X_c}(x)\|f\|_k \quad \forall c \in \mathcal{C},\ \forall x \in \mathcal{X}_c. \tag{3.3}$$

We first devote ourselves to finding the maximum of the power function $P_{X_c}(x)$ on $\mathcal{X}_c$. This enables upper bounding the approximation error (3.3) to ensure that (3.2) holds, for which we provide a proof in Theorem 1.

Assumption 3. *For any sample set X_c according to* (3.1)*, it holds that $P_{X_c}(x) \leq P_{X_c}(x_{cm})$ for all $x \in \mathcal{X}_c$, where x_{cm} is the center of X_c, i.e., $x_{cm} = \frac{1}{2^n}\sum_{i=1}^{2^n} x_i$.*

Figure 3.3 shows the power function of the Ma32-kernel with four samples X_c on the vertices of a local cube $\mathcal{X}_c$. Moreover, Figure 3.4 illustrates the sublevel sets of Figure 3.3. As can be seen, Assumption 3 holds for the presented case. The maximum of the power function is located at the center of the domain and is highlighted by the magenta point in Figure 3.4.

After investigating the maximum of the power function for the Ma32-kernel in Figures 3.3 and 3.4 and visually seeing that Assumption 3 holds there, we now look at it more formally. For this, we look into the resulting relationship between the covariance matrix given samples X_c and the covariance vector given samples X_c evaluated at x_{cm}.

Proposition 1. *Let Assumption 1 hold. Let X_c be a sample set according to* (3.1)*. Then,*

$$K_{X_c}^{-1} k_{X_c}(x_{cm}) = \frac{1}{\beta} k_{X_c}(x_{cm}), \tag{3.4}$$

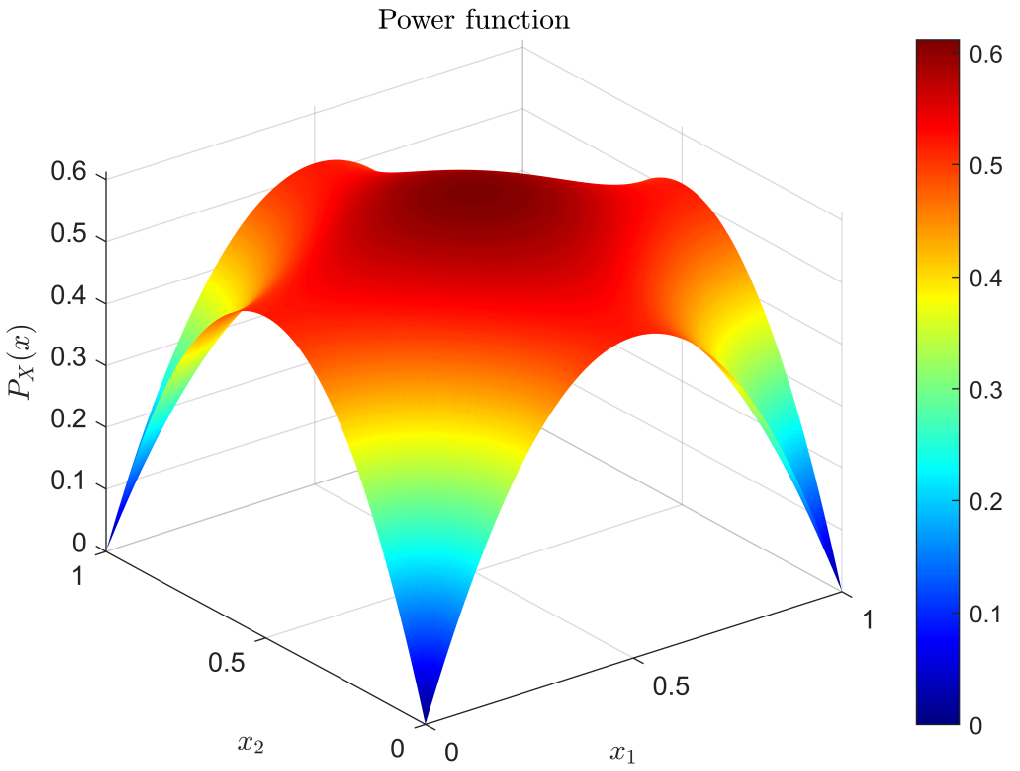

Figure 3.3: Power function of the Ma32-kernel given samples X_c that are vertices of a local cube $\mathcal{X}_c$. The maximum of the power function is at the center of the domain.

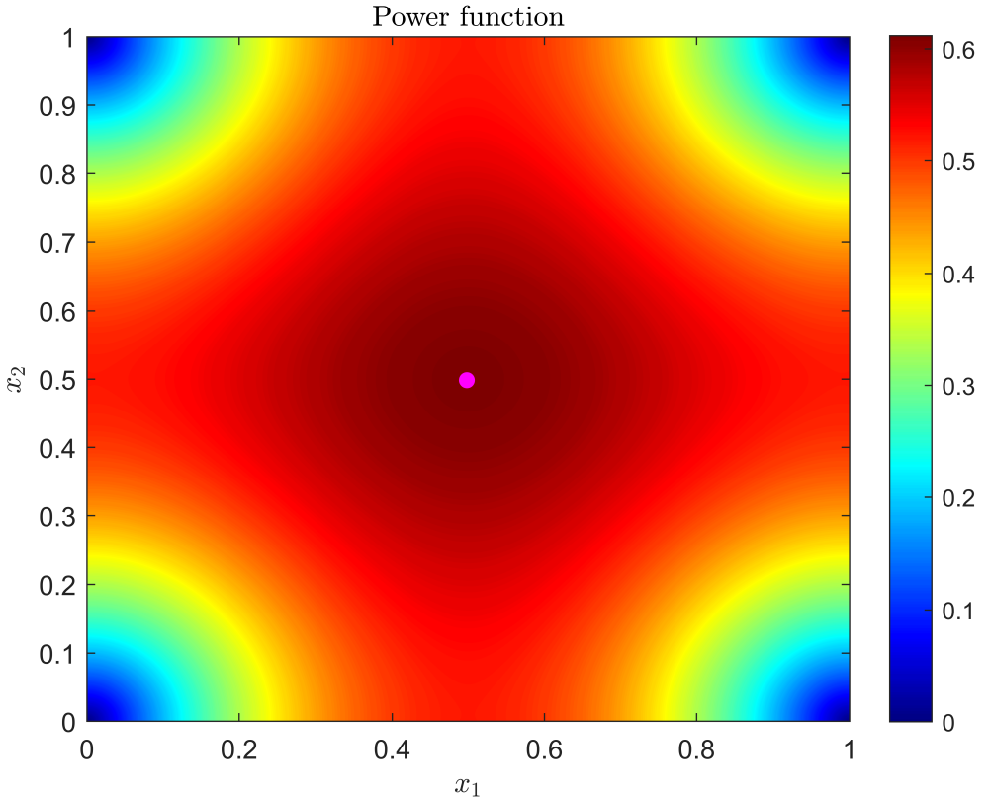

Figure 3.4: Power function of the Ma32-kernel given samples X_c that are vertices of a local cube $\mathcal{X}_c$. This plot illustrates the sublevel sets of Figure 3.3. The maximum of the power function is at the center of the domain and is portrayed by the magenta point.

with

$$\beta = \sum_{x_i \in X_c} k(x, x_i) \leq 2^n \quad x \in X_c, \tag{3.5}$$

where x_{cm} *is the center of* X_c*, i.e.,* $x_{cm} = \frac{1}{2^n} \sum_{x_i \in X_c} x_i$*.*

Proof. Consider an n-dimensional cube $\mathcal{X}_c$ with vertices $X_c = \{x_1, \ldots, x_{2^n}\}$. W.l.o.g., suppose that the center of the cube x_{cm} is at the origin, i.e., $x_{cm} = 0$. Thus, it holds that $x_i = \frac{\Delta x_c}{2}[\pm 1, \ldots, \pm 1]^\top$ for all $x_i \in X_c$, with Δx_c being the length of the cube's edges. Furthermore, $\|x_{cm} - x_i\|_2 = \|x_{cm} - x_j\|_2 = \frac{\sqrt{n}\Delta x_c}{2}$ for all $i, j \in \{1, \ldots, 2^n\}$. Since $k(x_1, x_2) = k(x_3, x_4)$ for all $x_1, \ldots, x_4$ s.t. $\|x_1 - x_2\|_2 = \|x_3 - x_4\|_2$ (cf. Assumption 1), it follows that

$$k(x_{cm}, x_i) = \widetilde{k}\left(\frac{\sqrt{n}\Delta x_c}{2}\right) \tag{3.6}$$

for all $x_i \in X_c$. Thus, given X_c, the covariance vector evaluated at x_{cm} (cf. (2.3)) is given by

$$k_{X_C}(x_{cm}) = \widetilde{k}\left(\frac{\sqrt{n}\Delta x_c}{2}\right) \cdot \mathbf{1}_{2^n}. \tag{3.7}$$

Furthermore, since $x_i = \frac{\Delta x_c}{2}[\pm 1, \ldots, \pm 1]^\top$ for all $x_i \in X_c$, it holds that $\sum_{j=1}^{2^n} k(x_i, x_j) = \sum_{j=1}^{2^n} k(x_{i+1}, x_j)$ for all $i \in \{1, \ldots, 2^n - 1\}$. Therefore, given samples X_c, the covariance matrix K_{X_c} (cf. (2.2)) has constant row sum. Thus, the covariance vector $k_{X_c}(x_{cm})$ is an eigenvector of the covariance matrix K_{X_c} and

$$K_{X_c} k_{X_c}(x_{cm}) = \beta k_{X_c}(x_{cm}) \tag{3.8}$$

holds with

$$\beta = \sum_{i=1}^{2^n} k(x, x_i) \leq 2^n \quad x \in X_c.$$

The inequality arises directly from $k \leq \widetilde{k}(0) = 1$ (cf. Assumption 1). Therefore, $k_{X_c}(x_{cm})$ is also an eigenvector of $K_{X_c}^{-1}$ with

$$K_{X_c}^{-1} k_{X_c}(x_{cm}) = \frac{1}{\beta} k_{X_c}(x_{cm}),$$

with $\beta > 0$, since k is (strictly) positive definite (cf. Assumption 1). ■

Assumption 4. *For any sample set X_c according to* (3.1)*, the power function $P_{X_c}(x)$ (cf.* (2.4)*) is quasi-concave on $\mathcal{X}_c$.*

We can see that Assumption 4 holds for the case presented in Figures 3.3 and 3.4. We use this assumption to now formally show for the special case of k being the SE-kernel that the maximum of the power function $P_{X_c}(x)$ is at the center of the local cube.

Proposition 2. *Let Assumption 4 hold. Let k be the SE-kernel (cf.* (2.6)*). Then,*

$$P_{X_c}(x) \leq P_{X_c}(x_{cm}) \; \forall x \in \mathcal{X}_c, \tag{3.9}$$

where x_{cm} is the center of X_c.

Proof. It holds that

$$\begin{aligned}
&\frac{\partial}{\partial x}\left(k_{X_c}(x)^\top K_{X_c}^{-1} k_{X_c}(x)\right)\Big|_{x_{cm}} = 2k_{X_c}(x_{cm})^\top K_{X_c}^{-1}\frac{\partial}{\partial x}k_{X_c}(x)\Big|_{x_{cm}} \\
&\overset{(2.6)}{=} -4k_{X_c}(x_{cm})^\top K_{X_c}^{-1}\operatorname{diag}\left(K_{X_c}(x_{cm})\right)\begin{pmatrix}(x_{cm}-x_1)^\top\\ \vdots \\ (x_{cm}-x_{2^n})^\top\end{pmatrix} \\
&\overset{(3.4)}{=} \frac{-4}{\beta}k_{X_c}(x_{cm})^\top\operatorname{diag}\left(K_{X_c}(x_{cm})\right)\begin{pmatrix}(x_{cm}-x_1)^\top\\ \vdots \\ (x_{cm}-x_{2^n})^\top\end{pmatrix} \\
&\overset{(2.3)}{=} \frac{-4}{\beta}\begin{pmatrix}k(x_{cm},x_1)^2\\ \vdots \\ k(x_{cm},x_{2^n})^2\end{pmatrix}^\top\begin{pmatrix}(x_{cm}-x_1)^\top\\ \vdots \\ (x_{cm}-x_{2^n})^\top\end{pmatrix} \\
&\overset{(3.7)}{=} \frac{-4\widetilde{k}\left(\frac{\sqrt{n}\Delta x_c}{2}\right)^2\Delta x_c}{\sqrt{n}\beta}\mathbf{1}_{2^n}^\top\begin{pmatrix}1 & \cdots & 1\\ 1 & \cdots & -1\\ \vdots & \ddots & \vdots\\ -1 & \cdots & -1\end{pmatrix} \\
&= 0.
\end{aligned} \tag{3.10}$$

The final equality arises because the sum of row sums of the matrix is equal to zero. That is since $x_i,\ i \in \{1,\dots,2^n\}$ is on average x_{cm}, wherefore the sum of the mean

deviations equals zero. With

$$\frac{\partial}{\partial x} P_{X_c}(x)|_{x_{cm}} \overset{(2.4)}{=} \frac{\partial}{\partial x} \sqrt{1 - k_{X_c}(x)^\top K_{X_c}^{-1} k_{X_c}(x)}\Big|_{x_{cm}} \overset{(3.10)}{=} 0,$$

it follows that x_{cm} is a stationary point of $P_{X_c}(x)$. Since the power function $P_{X_c}(x)$ is quasi-concave on $\mathcal{X}_c$ (cf. Assumption 4), x_{cm} is the only stationary point and the stationarity is a necessary and sufficient condition of x_{cm} being the global maximizer of $P_{X_c}(x)$ on $\mathcal{X}_c$. Thus, we showed that $P_{X_c}(x_{cm}) \geq P_{X_c}(x)\ \forall x \in \mathcal{X}_c$. ■

Since we divide the domain into *uniform* local cubes, all local cubes $\mathcal{X}_c \subseteq \mathcal{X}$ are of the same size (cf. (3.1)). Thus, $P_{X_c}(x_{cm})$ has the same value for any $c \in \mathcal{C}$. Therefore, we define $\overline{P}_X \coloneqq P_{X_c}(x_{cm})$.

Now that we dealt with guaranteeing an upper bound on the power function and provided a proof for the special case of k being the SE-kernel, we turn to finding guaranteed bounds on the approximation error (3.3) to make sure that (3.2) holds. In addition to the upper bound on the power function, (3.3) requires an upper bound on the RKHS norm of the ground truth f on the domain $\mathcal{X}$.

Assumption 5. *We have access to an oracle, which returns $\overline{\Gamma} > 0$, an upper bound on the RKHS norm of the ground truth f on the domain $\mathcal{X}$.*

Theorem 1. *Let Assumptions 1, 2, 3, and 5 hold and suppose that $\overline{P}_X \overline{\Gamma} \leq \bar{\epsilon}$. Then $|f(x) - h_{X_c}(x)| \leq \bar{\epsilon}\ \forall c \in \mathcal{C},\ \forall x \in \mathcal{X}_c$, i.e., (3.2) holds.*

Proof. For any $c \in \mathcal{C}$ and for any $x \in \mathcal{X}_c$ it holds that

$$|f(x) - h_{X_c}(x)| \overset{(3.3)}{\leq} P_{X_c}(x)\overline{\Gamma} \overset{\text{Asm. 3}}{\leq} \overline{P}_X \overline{\Gamma} \leq \bar{\epsilon},$$

i.e., (3.2) holds. ■

3.3 Pseudo Algorithm

In this section, we present the pseudo algorithm of the developed localized kernel interpolation with known RKHS norm (cf. Assumption 5) and guaranteed bounds on the approximation error (cf. Theorem 1).

Algorithm 1 Upper bound on power function

Require: k, X_M

1: $X_{M,c} \leftarrow$ vertices of a cube in X_M (cf. (3.1)) with 2^n samples
2: $K^{-1}_{X_{M,c}} \leftarrow k, X_{M,c}$ in (2.2) ▷ Inverse covariance matrix
3: $k_{X_{M,c}}(x) \leftarrow k, X_{M,c}$ in (2.3) ▷ Covariance vector
4: $x_{cm} \leftarrow$ center of $X_{M,c}$
5: $\overline{P}_{X_M} \leftarrow K^{-1}_{X_{M,c}}, k_{X_{M,c}}(x_{cm})$ in (2.4) ▷ Upper bound on the power function
6: **return** $\overline{P}_{X_M}$, $K^{-1}_{X_{M,c}}$

In Algorithm 1, we determine an upper on the power function (cf. Assumption 3). For this, we require the chosen kernel k and the set of samples on the domain $\mathcal{X}$ with M samples per axis, which we denote by X_M. To determine the upper bound on the power function $\overline{P}_{X_M}$, we first define the set of 2^n samples $X_{M,c} \subseteq X_M$ on a local cube $\mathcal{X}_c$. Since the local cubes are of the same size (cf. (3.1)), it suffices to pick an *arbitrary* local cube $\mathcal{X}_c \subseteq \mathcal{X}$ with corresponding samples $X_{M,c}$. Furthermore, we calculate the inverse covariance matrix $K^{-1}_{X_{M,c}}$ and the covariance vector $k_{X_{M,c}}(x)$. After defining the center x_{cm} of sample set $X_{M,c}$, we evaluate the covariance vector at x_{cm} and determine the upper bound on the power function $\overline{P}_{X_M}$.

In Algorithm 2, we increase the number of samples to achieve the desired accuracy, i.e., to guarantee a uniform approximation error smaller or equal to $\bar{\epsilon}$ (cf. (3.2), (3.3)). First, we initialize the current number of samples per axis M to the minimum number of samples per axis $\underline{M} > 1$. The minimum number of samples per axis $\underline{M}$ sets the lower bound on how many uniform samples we use in total to define the approximating functions to $\underline{M}^n$. Algorithm 2 calls Algorithm 1 to receive the maximum of the power function $\overline{P}_{X_M}$ given the current number of samples per axis M. If the desired accuracy is achieved, i.e., $\overline{P}_{X_M}\overline{\Gamma} \leq \bar{\epsilon}$ (cf. Theorem 1), we break out of the while-True loop and do not have to increase the number of samples. If, however, the desired accuracy is not yet achieved, we increase the number of samples and define the sample set for the next iteration. Let us denote the current iteration

Algorithm 2 Sufficient samples to achieve desired accuracy

Require: $\bar{\epsilon}$, k, $\mathcal{X}$, $\underline{M}$, $\overline{\Gamma}$

Initialize: $M \leftarrow \underline{M}$, $X_M \leftarrow$ set of samples on $\mathcal{X}$ with M samples per axis
while True **do**
  $\overline{P}_{X_M}, K^{-1}_{X_{M,c}} \leftarrow$ Algorithm 1(k, X_M) ▷ Upper bound on power function
  if $\overline{P}_{X_M}\overline{\Gamma} \leq \bar{\epsilon}$ **then** ▷ Desired accuracy achieved
    break
  else ▷ Desired accuracy not yet achieved
    $M \leftarrow 2M - 1$ ▷ Increase number of samples
    $X_M \leftarrow$ set of samples on $\mathcal{X}$ with M samples per axis
  end if
end while
return X_M, M, $K^{-1}_{X_{M,c}}$

by index i and the next iteration by $i+1$. The update rule for the number of samples per axis is

$$M_{i+1} = 2M_i - 1. \tag{3.11}$$

Since we sample uniformly, update rule (3.11) ensures that $X_{M_i} \subseteq X_{M_{i+1}}$, i.e., no samples are discarded. The described procedure is repeated until the sufficient number of samples is achieved to guarantee the desired accuracy.

Algorithm 3 Construct local approximating functions

Require: f, X_M, $K^{-1}_{X_{M,c}}$, k

Initialize: dictionary $\leftarrow$ empty dictionary
$\mathcal{C}_M \leftarrow$ set of positive integers s.t. $\cup_{c' \in \mathcal{C}_M} X_{M,c'} = X_M$
$f_{X_M} \leftarrow f(X_M)$ ▷ Function evaluation
for $c' \in \mathcal{C}_M$ **do** ▷ Split samples into cubes
  $k_{X_{M,c'}}(x) \leftarrow k, X_{M,c'}$ in (2.3) ▷ Covariance vector
  $h_{X_{M,c'}}(x) \leftarrow f_{X_{M,c'}}, K^{-1}_{X_{M,c}}, k_{X_{M,c'}}(x)$ in (2.5) ▷ Approximating function
  dictionary$(X_{M,c'}) \leftarrow h_{X_{M,c'}}(x)$ ▷ Store approximating function
end for
return dictionary

In Algorithm 3 we construct the local approximating functions that satisfy (3.2). For this, we require the ground truth f, the chosen kernel k, and the set of samples X_M that is sufficient to guarantee the desired accuracy (cf. Algorithm 2). Moreover,

we have $K_{X_{M,c}}^{-1}$ (cf. Algorithm 1), the inverse covariance matrix given the samples of an arbitrary local cube. We initialize Algorithm 3 by defining an empty dictionary. A dictionary for our algorithms is defined akin to a `Python`-dictionary, i.e., for storing data values for a given key. After initializing the dictionary, we determine the set of integers for the local cubes $\mathcal{C}_M$. Furthermore, we evaluate the ground truth f at X_M and receive f_{X_M}. Then, we iterate over each local cube and define the respective covariance vector. Since all local cubes are of the same size, their inverse covariance matrices are identical. Therefore, we can use $K_{X_{M,c}}^{-1}$, the inverse covariance matrix on an arbitrary local cube to define the approximating function for all local cubes. After determining the approximating functions, we store them in the dictionary. The key of each approximating function is the set of vertices $X_{M,c}$, which are used to generate the approximating functions. After all local cubes have been processed, we return the dictionary that contains the piecewise-defined localized approximating functions that in union cover the whole domain $\mathcal{X}$.

Algorithm 4 Localized Kernel Interpolation: offline learning

Require: f, $\mathcal{X}$, k, $\underline{M}$, $\overline{\epsilon}$

1: $\overline{\Gamma} \leftarrow$ Upper bound on RKHS norm of f on $\mathcal{X}$ (cf. Assumption 5)
2: $K_{X_c}^{-1}, X \leftarrow$ Algorithm 2$(\overline{\epsilon}, \overline{\Gamma}, k, \mathcal{X}, \underline{M})$
▷ Compute sufficient samples to achieve desired accuracy
3: dictionary $\leftarrow$ Algorithm 3$(f, X, K_{X_c}^{-1}, k)$
▷ Evaluate f, generate approximating functions, and save in a dictionary
4: **return** dictionary, X

Algorithm 4 summarizes the offline learning procedure to generate sufficiently accurate piecewise-defined approximating functions for the domain $\mathcal{X}$. First, we query the oracle (cf. Assumption 5) to receive an upper bound on the RKHS norm of the ground truth f on the domain $\mathcal{X}$. We receive the set of sufficient samples X to ensure the desired accuracy from Algorithm 2 and the dictionary containing the resulting piecewise-defined approximating function from Algorithm 3. Algorithm 4 is a one-shot algorithm, as we can a priori, i.e., before evaluating the ground truth f, determine which samples we need to evaluate to achieve the desired accuracy.

Algorithm 5 describes the online evaluation procedure. We require the set of all samples X, the dictionary containing the local approximating functions, and an input $x' \in \mathcal{X}$. To make an online prediction for an input x', we first determine the

Algorithm 5 Localized Kernel Interpolation: online evaluation

Require: X, x', dictionary

1: $X_c \leftarrow$ relevant cube of X that contains x'
2: $h_{X_c}(x) \leftarrow$ dictionary(X_c) ▷ Get local approximating function
3: **return** $h_{X_c}(x')$

relevant samples X_c in whose convex hull x' is contained. Since we are in a uniform sampling setting, determining X_c is a trivial and cheap computation. Furthermore, the local approximating function $h_{X_c}(x)$ is obtained from the dictionary. Finally, we evaluate the approximating function and return $h_{X_c}(x')$.

3.4 Theoretical Analysis

We now prove that Algorithm 4 terminates after a finite number of samples and the resulting piecewise-defined approximating function (cf. Algorithm 5) has a uniform approximation error smaller or equal to $\bar{\epsilon}$.

Theorem 2. *Let Assumptions 1,2, 3, and 5 hold. Then, Algorithm 4 terminates after a finite number of samples. Additionally, if k is the SE-kernel, then Algorithm 4 terminates after* $N = \mathcal{O}\left(\left(\frac{-n}{\ln\sqrt{1-\left(\frac{\bar{\epsilon}}{\Gamma}\right)^2}}\right)^{\frac{n}{2}}\right)$. *Furthermore, the resulting approximating functions (cf. Algorithm 5) have a uniform approximation error smaller or equal to $\bar{\epsilon}$, i.e.,* (3.2) *holds.*

Proof. **Part 1:** First, we find an upper bound for $k_{X_c}(x_{cm})^\top K_{X_c}^{-1} k_{X_c}(x_{cm})$. It holds that

$$\begin{aligned} k_{X_c}(x_{cm})^\top K_{X_c}^{-1} k_{X_c}(x_{cm}) &\overset{(3.4),(3.7)}{=} \frac{\widetilde{k}\left(\frac{\sqrt{n}\Delta x_c}{2}\right)^2}{\beta} \cdot \mathbf{1}_{2^n}^\top \mathbf{1}_{2^n} \\ &\overset{(3.5)}{\geq} \frac{\widetilde{k}\left(\frac{\sqrt{n}\Delta x_c}{2}\right)^2}{2^n} \cdot 2^n \\ &= \widetilde{k}\left(\frac{\sqrt{n}\Delta x_c}{2}\right)^2. \end{aligned} \tag{3.12}$$

Given continuity of $\widetilde{k}$ and $\widetilde{k}(0) = 1$, there exists a sufficiently small constant $\Delta x_c > 0$,

s.t.

$$\widetilde{k}\left(\frac{\sqrt{n}\Delta x_c}{2}\right) \geq \sqrt{1-\left(\frac{\overline{\epsilon}}{\overline{\overline{\Gamma}}}\right)^2}. \tag{3.13}$$

If (3.13) holds, then for any $c \in \mathcal{C}$ it holds that

$$\begin{aligned}\overline{P}_X\overline{\Gamma} &\overset{(2.4),\text{Asm. } 3}{\leq} \sqrt{1-k_{X_c}(x_{cm})^\top K_{X_c}^{-1}k_{X_c}(x_{cm})}\overline{\Gamma} \\ &\overset{(3.12)}{\leq} \sqrt{1-\widetilde{k}\left(\frac{\sqrt{n}\Delta x_c}{2}\right)^2}\,\overline{\Gamma} \\ &\overset{(3.13)}{\leq} \overline{\epsilon}.\end{aligned} \tag{3.14}$$

Therefore, Algorithm 4 terminates after a finite number of steps, as $\Delta x_c > 0$ implies $N < \infty$.

Part 2: Now, suppose that $\widetilde{k}$ is the SE-kernel. Then,

$$\widetilde{k}\left(\frac{\sqrt{n}\Delta x_c}{2}\right) \overset{(2.6)}{=} \exp\left(\frac{-\Delta x_c^2 n}{4}\right) \overset{!}{\geq} \sqrt{1-\left(\frac{\overline{\epsilon}}{\overline{\overline{\Gamma}}}\right)^2}. \tag{3.15}$$

Solving (3.15) for Δx_c yields

$$\Delta x_c^2 \leq \frac{-4}{n}\ln\sqrt{1-\left(\frac{\overline{\epsilon}}{\overline{\overline{\Gamma}}}\right)^2}, \tag{3.16}$$

wherefore

$$\Delta x_c = \mathcal{O}\left(\left(\frac{-\ln\sqrt{1-\left(\frac{\overline{\epsilon}}{\overline{\overline{\Gamma}}}\right)^2}}{n}\right)^{\frac{1}{2}}\right). \tag{3.17}$$

Achieving Δx_c requires $N = \mathcal{O}\left(\left(\frac{1}{\Delta x_c}\right)^n\right)$. Thus, for the SE-kernel, Algorithm 4 terminates after $N = \mathcal{O}\left(\left(\frac{-n}{\ln\sqrt{1-\left(\frac{\overline{\epsilon}}{\overline{\overline{\Gamma}}}\right)^2}}\right)^{\frac{n}{2}}\right)$.

Part 3: Now, we omit the restriction that $\widetilde{k}$ is the SE-kernel and again look at

all kernels that satisfy Assumption 1. It holds for any $c \in \mathcal{C}$ and for any $x \in \mathcal{X}_c$ that

$$\begin{aligned} |f(x) - h_{X_c}(x)| &\overset{(3.3)}{\leq} P_{X_c}(x)\overline{\Gamma} \\ &\overset{\text{Asm. } 3}{\leq} \overline{P}_X\overline{\Gamma} \\ &\overset{(3.14)}{\leq} \overline{\epsilon}, \end{aligned} \tag{3.18}$$

i.e., (3.2) holds.

■

4 Adaptive and Localized Kernel Interpolation

Although we presented an approximation framework with guaranteed bounds on the uniform approximation error and a proof for convergence after a finite number of samples in Chapter 3 (cf. Theorem 2), employing Algorithm 4 leads to numerically unreliable calculations due to ill-conditioned covariance matrices. In this chapter, we introduce a scaled kernel and an adaptive partitioning into sub-domains to ensure well-conditioned covariance matrices in Section 4.1. Moreover, we present the resulting pseudo algorithm in Section 4.2. Finally, we prove convergence after a finite number of samples for the presented pseudo algorithm and prove that the resulting approximation framework with well-conditioned covariance matrices guarantees the desired approximation accuracy in Section 4.3 (cf. Theorem 3).

4.1 Ensuring well-conditioned Covariance Matrices

While executing Algorithm 4, the number of sufficient samples per axis M to guarantee the desired approximation accuracy is determined by Algorithm 2. The higher the number of samples per axis M, the denser we sample. As a result, the eucledian distance between the vertices of a local cube decreases. Thus, for increasing M, this results in $\widetilde{k}(\|x_i - x_j\|_2) \to 1$ for all $x_i, x_j \in X_c$ (cf. Assumption 1). Therefore, if M approaches infinity, the covariance matrix given samples on a local cube approaches (cf. (2.2))

$$K_{X_c} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix}, \tag{4.1}$$

i.e., a rank-one matrix that is endowed with a condition number of infinity. Therefore, employing Algorithm 4 as presented in Chapter 3 results in numerical complications. This is a well-known problem in kernel interpolation (cf. [27], [28]). To ensure numerically reliable calculations, we require that the condition number of the covariance matrix K_{X_c} is upper bounded by a constant $\overline{\kappa}_c$, i.e.,

$$\operatorname{cond}(K_{X_c}) \overset{!}{\leq} \overline{\kappa}_c \quad \forall c \in \mathcal{C}, \tag{4.2}$$

with $\operatorname{cond}(K_{X_c}) = \|K_{X_c}\|_\infty \|K_{X_c}^{-1}\|_\infty$.

To establish a setting that enables that (4.2) holds, we introduce *scaled* kernels. Given a kernel k, we defined a scaled version k_ℓ s.t.

$$\widetilde{k}_\ell(\|x - x'\|_2) = \widetilde{k}\left(\frac{\|x - x'\|_2}{\sqrt{2}\ell}\right), \tag{4.3}$$

where we denote by ℓ the length scale. Analogously, we define scaled covariance matrices and scaled covariance vectors, which we denote by $K_{\ell,X}$ and $k_{\ell,X}(x)$, respectively. One possible way of ensuring well-conditioned covariance matrices K_{ℓ,X_c} is to let the length scale ℓ be arbitrarily close to zero. Then

$$\lim_{\ell \to 0} \widetilde{k}_\ell(\|x_i - x_j\|_2) = \begin{cases} 1 & \text{if } x_i = x_j, \\ 0 & \text{otherwise,} \end{cases} \tag{4.4}$$

for all $x_i, x_j \in X_c$. However (cf. Assumption 1, (2.4)),

$$\begin{aligned} \lim_{\ell \to 0} P_{X_c}(x) &= \sqrt{1 - \mathbf{0}_{2^n}^\top \mathbb{I}_{2^n} \mathbf{0}_{2^n}} \\ &= 1 \end{aligned} \tag{4.5}$$

for all $x \in \mathcal{X}_c \setminus X_c$, which impedes ensuring the desired uniform approximation error (cf. (3.2), (3.3)).

Consequently, we develop a different approach that leads to well-conditioned covariance matrices (cf. (4.2)) while still achieving an arbitrarily small approximation error (cf. (3.3)). We choose to partition the domain $\mathcal{X}$ into sub-domains $\mathcal{X}_a \subseteq \mathcal{X}$. The piecewise-defined approximation framework with local cubes introduced in Chapter 3 is present in every sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$. Notably, for each

sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$, the task of developing approximating functions that guarantee a uniform approximation error smaller or equal to $\bar{\epsilon}$ (cf. (3.2)) is equivalent to the setting described in Chapter 3. Correspondingly to (3.1), it now holds that

$$\begin{aligned} \cup_{c\in\mathcal{C}_a}\mathcal{X}_{a,c} &= \mathcal{X}_a \\ \cup_{c\in\mathcal{C}_a}X_{a,c} &= X_a \\ \operatorname{conv}(X_a) &= \mathcal{X}_a \\ \operatorname{card}(X_{a,c}) &= 2^n \\ \operatorname{conv}(X_{a,c}) &= \mathcal{X}_{a,c} \\ \cup_{a\in\mathcal{A}}\mathcal{X}_a &= \mathcal{X} \\ \cup_{a\in\mathcal{A}}X_a &= X, \end{aligned} \tag{4.6}$$

with $\mathcal{A} \subseteq \mathbb{N}$ and $\mathcal{C}_a \subseteq \mathbb{N}$ for all $a \in \mathcal{A}$.

We execute the sub-domain partitioning as follows. If we decide to partition the domain $\mathcal{X}$, which is a cube, it results in 2^n sub-domains $\mathcal{X}_a$ that are cubes of the same size. We can choose to partition any sub-domain $\mathcal{X}_a$ further into 2^n more sub-domains and continue with that procedure as often as desired. Therefore, the resulting sub-domain partitioning does not necessarily cause that all sub-domains are of the same size, since sub-domains that are split more frequently are smaller than sub-domains that are split less frequently. Since a sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$ is by definition a cube, its size is fully specified by its width

$$\begin{aligned} \operatorname{width}(\mathcal{X}_a) &= \max_{x_i,x_j\in\mathcal{X}_a} \|x_i - x_j\|_\infty \\ w_a &\coloneqq \operatorname{width}(\mathcal{X}_a). \end{aligned} \tag{4.7}$$

We choose to set length scale ℓ_a proportional to the width of the sub-domain $\mathcal{X}_a$, i.e.,

$$\ell_a = C_\ell \cdot w_a, \tag{4.8}$$

with C_ℓ being the length scale hyperparameter. Therefore, smaller sub-domains have smaller length scales. Now, every sub-domain $\mathcal{X}_a$ is endowed with an individual scaled kernel $\widetilde{k}_{\ell_a}$ (cf. (4.3)).

For scaled kernels $\widetilde{k}_{\ell_a}$ (cf. (4.3)), the ratio $\frac{\|x_i - x_j\|_2}{\ell_a}$, $x_i, x_j \in X_{a,c}$ fully defines the value of the covariance matrix $K_{\ell_a, X_{a,c}}$ and its condition number. Smaller sub-

domains with smaller length scales thus allow for a denser sampling, i.e., smaller 2-norm distances between the samples for a given ratio $\frac{\|x_i - x_j\|_2}{\ell_a}$. Furthermore, a lower bound on $\frac{\|x_i - x_j\|_2}{\ell_a}$ can be defined to upper bound the condition number of $K_{\ell_a, X_{a,c}}$ (cf. (4.2)). Since $\|x_i - x_j\|_2$ is proportional to the edge length of a local cube $\Delta x_{a,c}$, a corresponding lower bound for $\frac{\Delta x_{a,c}}{\ell_a}$ can be obtained to upper bound the condition number. As the number of uniform samples per axis M_a on sub-domain $\mathcal{X}_a$ determines $\Delta x_{a,c}$, we introduce the maximum number of samples per axis $\overline{M}$ to lower bound $\frac{\Delta x_{a,c}}{\ell_a}$ and therefore upper bound the condition number of $K_{\ell_a, X_{a,c}}$. Due to the scaling introduced by the length scale that is proportional to the width of the sub-domain $\mathcal{X}_a$, the maximum number of samples per axis $\overline{M}$ is a constant for every sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$ and we fix $\overline{M}$ s.t. (4.2) holds for every $K_{\ell_a, X_{a,c}}$. The upper bounding of the condition number ensures well-conditioned covariance matrices without requiring any regularization. In Chapter 6, we will present quantitative results on the condition number, on the choice of the maximum number of samples per sub-domain $\overline{M}$, and the length scale parameter C_ℓ.

If Algorithm 4 requires more than $\overline{M}$ samples per axis to ensure the desired approximation accuracy on the domain $\mathcal{X}$, we split $\mathcal{X}$ into 2^n sub-domains. These sub-domains have a smaller width and thus a smaller length scale. For this reason, they allow for a denser sampling while still ensuring well-conditioned covariance matrices. Since the procedure is repeated until the desired approximation accuracy is achieved, the domain $\mathcal{X}$ is *adaptively* partitioned further whenever the resulting approximation with $\overline{M}$ uniform samples per axis is not sufficiently accurate. The adaptive sub-domain partitioning leads to bigger sub-domains for areas of the domain that are easy to approximate and to smaller sub-domains for harder-to-approximate areas to allow for a denser sampling.

Extending the proposed approximation framework of Chapter 3 with the adaptive sub-domain partitioning results in a sampling scheme that is not completely uniform anymore. However, in each sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$, we still sample uniformly and we preserve defining individual approximating functions for each local cube $\mathcal{X}_{a,c} \subseteq \mathcal{X}_a$. In addition to the advantages of choosing a uniform sampling scheme with local cubes, which we presented in Section 3.1, we now additionally assess how challenging it is to approximate a specific area and locally sample denser whenever necessary. Furthermore, the extension of adaptively partitioned sub-domains guarantees well-

conditioned covariance matrices and therefore a numerically reliable computation. Hence, we enable scalability w.r.t the total number of samples without requiring any regularization.

4.2 Pseudo Algorithm

In this section, we present the pseudo algorithm for our adaptive and localized kernel interpolation, which extends the pseudo algorithm presented in Section 3.3 by incorporating an adaptive partitioning into sub-domains to enforce well-conditioned covariance matrices. Working with the introduced adaptive sub-domain partitioning leads to an approximation problem that is equivalent to the problem described in Chapter 3 for every sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$. Therefore, we now aim to ensure that (3.2) holds for every sub-domain $\mathcal{X}_a$. Thus, we require an upper bound on the RKHS norm of the ground f on every sub-domain $\mathcal{X}_a$ to upper bound (3.3).

Assumption 6. *We have access to an oracle, which returns $\overline{\Gamma}_a > 0$, an upper bound on the RKHS norm of the ground truth f on sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$.* [1]

Algorithm 6 summarizes the offline learning procedure to obtain approximating functions that guarantee a uniform approximation error smaller or equal to $\bar{\epsilon}$ while still ensuring a well-conditioned computation. We require the ground truth f, the domain $\mathcal{X}$, the chosen kernel k, the length scale parameter C_ℓ, the maximum allowed uniform approximation error $\bar{\epsilon}$, the minimum number of samples per axis $\underline{M}$, and the maximum number of samples per axis $\overline{M}$. The maximum number of samples per axis $\overline{M}$ sets the upper bound on how many uniform samples we use per sub-domain to $\overline{M}^n$. We begin by initializing the set of tasks $\mathcal{X}_{\text{tasks}}$. The set of tasks $\mathcal{X}_{\text{tasks}}$ is a dynamic set of sub-domains on which we still have to define the piecewise-defined approximating functions. We initialize $\mathcal{X}_{\text{tasks}} = \{\mathcal{X}\}$, i.e., we first investigate the whole input domain $\mathcal{X}$. Moreover, we initialize an empty set X_{total}, which will contain all samples obtained during the offline learning. To conclude, we initialize $\text{dictionary}_{\text{total}}$, an empty dictionary defined akin to the dictionary in Algorithm 4.

To define approximating functions on all adaptively created sub-domains $\mathcal{X}_a$, we iterate with a dynamic for-loop over every $\mathcal{X}_a$ in $\mathcal{X}_{\text{tasks}}$. First, we define the width w_a of the current sub-domain $\mathcal{X}_a$ and the corresponding length scale ℓ_a. Furthermore,

[1] Note that the ground truth f is only defined on the respective sub-domain $\mathcal{X}_a$.

Algorithm 6 Adaptive and Localized Kernel Interpolation: offline learning

Require: f, $\mathcal{X}$, k, C_ℓ, $\underline{M}$, $\overline{M}$, $\overline{\epsilon}$

Initialize: $\mathcal{X}_{\text{tasks}} \leftarrow \{\mathcal{X}\}$, $X_{\text{total}} \leftarrow \{\}$, $\text{dictionary}_{\text{total}} \leftarrow$ empty dictionary
for $\mathcal{X}_a \in \mathcal{X}_{\text{tasks}}$ **do** ▷ Dynamic for loop
    $w_a \leftarrow \text{width}(\mathcal{X}_a)$ (cf. (4.7))
    $\ell_a \leftarrow C_\ell \cdot w_a$
    Define scaled kernel k_{ℓ_a} given k and ℓ_a (cf. (4.3))
    $\overline{\Gamma}_a \leftarrow$ Upper bound on RKHS norm of f on $\mathcal{X}_a$ (cf. Assumption 6)
    $M_a, X_{a,M_a}, K^{-1}_{\ell_a, X_{a,M_a,c}} \leftarrow$ Algorithm 2$(\overline{\epsilon}, \overline{\Gamma}_a, k_{\ell_a}, \mathcal{X}_a, \underline{M})$
        ▷ Determine sufficient number of samples to get the desired accuracy
    if $M_a > \overline{M}$ **then** ▷ Maximum number of samples per domain exceeded
        $\mathcal{X}_{a,i} \leftarrow$ Create 2^n uniform subdomains of $\mathcal{X}_a$
        $\mathcal{X}_{\text{tasks}} \leftarrow \mathcal{X}_{\text{tasks}} \cup_{i \in \{1,\dots,2^n\}} \{\mathcal{X}_{a,i}\}$ ▷ Add subdomains to set of tasks
    else
        $\text{dictionary}_a \leftarrow$ Algorithm 3$(f, X_{a,M_a}, K^{-1}_{\ell_a, X_{a,M_a,c}}, k_{\ell_a})$
            ▷ Function evaluation, generate and save the approximating functions
        $\text{dictionary}_{\text{total}}(X_{a,M_a}) \leftarrow \text{dictionary}_a$
        $X_{\text{total}} \leftarrow X_{\text{total}} \cup X_{a,M_a}$
    end if
    $\mathcal{X}_{\text{tasks}} \leftarrow \mathcal{X}_{\text{tasks}} \setminus \{\mathcal{X}_a\}$ ▷ Remove $\mathcal{X}_a$ from the set of tasks
end for
return $\text{dictionary}_{\text{total}}$, X_{total}

we define the scaled kernel k_{ℓ_a} (cf. (4.3)). Moreover, we query the oracle (cf. Assumption 6) to receive an upper bound on the RKHS norm of the ground truth f on sub-domain $\mathcal{X}_a$. Next, Algorithm 2 computes the number of uniform samples per axis M_a that is sufficient to achieve the desired accuracy on the current sub-domain $\mathcal{X}_a$. We additionally receive the set of samples with M_a uniform samples per axis on $\mathcal{X}_a$, which we denote by X_{a,M_a}, and the scaled inverse covariance matrix $K^{-1}_{\ell_a,X_{a,M_a,c}}$ on an arbitrary local cube $\mathcal{X}_{a,c} \subseteq \mathcal{X}_a$. If the number of required samples per axis M_a is greater than the maximum number of samples per axis $\overline{M}$, the covariance matrix $K^{-1}_{\ell_a,X_{a,M_a,c}}$ would have a condition number greater than $\overline{\kappa}_c$, i.e., (4.2) would not hold. To mitigate ill-conditioned covariance matrices, we partition the current domain $\mathcal{X}_a$ into 2^n sub-domains that are of the same size, add them to $\mathcal{X}_{\text{tasks}}$ and repeat the described procedure on the resulting smaller sub-domains, which allow for a denser sampling.

Once the required number of samples M_a is smaller or equal to $\overline{M}$, Algorithm 3 evaluates the ground truth and computes the dictionary that contains the localized approximating functions for the current sub-domain $\mathcal{X}_a$. The dictionary for $\mathcal{X}_a$ is stored in $\text{dictionary}_{\text{total}}$. The key for obtaining dictionary_a from $\text{dictionary}_{\text{total}}$ is the sample set X_{a,M_a} with M_a uniform samples per axis on the current sub-domain $\mathcal{X}_a$. Moreover, X_{a,M_a} is added to the set of total samples X_{total}.

In essence, either we already approximated the current sub-domain $\mathcal{X}_a$ successfully or we split $\mathcal{X}_a$ into 2^n new sub-domains and added them to $\mathcal{X}_{\text{tasks}}$. Therefore, in both cases our approximation procedure on the current sub-domain $\mathcal{X}_a$ is concluded. Thus, we remove $\mathcal{X}_a$ from the set of tasks $\mathcal{X}_{\text{tasks}}$. The algorithm terminates once $\mathcal{X}_{\text{tasks}}$ is empty, i.e., once the domain $\mathcal{X}$ has been successfully partitioned into sub-domains with approximating functions defined on local cubes that ensure the desired approximation accuracy (cf. (3.2)). Then, $\text{dictionary}_{\text{total}}$ contains the local dictionaries with piecewise-defined local approximating functions for all sub-domains. Moreover, the samples that were necessary to generate all approximating functions are contained in X_{total}.

The online evaluation of the pseudo algorithm for our adaptive and localized kernel interpolation is based on Algorithm 7. We require the set of total samples X_{total}, $\text{dictionary}_{\text{total}}$, and an input $x' \in \mathcal{X}$. In contrast to the online evaluation in Chapter 3 (cf. Algorithm 5), Algorithm 7 first determines the relevant sample set $X_a \subseteq X_{\text{total}}$ on whose convex hull x' is contained. Moreover, we determine the

Algorithm 7 Adaptive and Localized Kernel Interpolation: online evaluation

Require: $\text{dictionary}_{\text{total}}$, X_{total}, x'

1: $X_a \leftarrow$ relevant samples of X_{total} that contains x'
2: $X_{a,c} \leftarrow$ relevant cube of X_a that contains x'
3: $\text{dictionary}_a \leftarrow \text{dictionary}_{\text{total}}(X_a)$
4: $h_{X_{a,c}}(x) \leftarrow \text{dictionary}_a(X_{a,c})$
5: **return** $h_{X_{a,c}}(x')$

vertices $X_{a,c}$ of the relevant local cube $\mathcal{X}_{a,c} \subseteq \mathcal{X}_a$ that contains x'. Determining the corresponding set of samples $X_{a,c}$ or X_a is a trivial and cheap computation for the considered uniform sampling scheme. Furthermore, we follow the same procedure as in Algorithm 5 to obtain the value of the approximating function for input x'. Algorithm 7 is still a one-shot algorithm, as it is possible to a priori, i.e., before evaluating the ground truth, determine how many function evaluations are needed to achieve the desired accuracy.

4.3 Theoretical Analysis

We now prove that Algorithm 6 terminates after a finite number of samples. Moreover, we provide a sample complexity. Furthermore, we prove that the piecewise-defined approximating function (cf. Algorithm 7) is sufficiently accurate (cf. (3.2)). To achieve these properties, we introduce further assumptions on the ground truth f.

Assumption 7. *The ground truth is Lipschitz-continuous, i.e., there exists a constant $L > 0$, s.t. $|f(x) - f(x')| \leq L \cdot \|x - x'\|_\infty \; \forall x, x' \in \mathcal{X}$.*

Assumption 8. *For any $a \in \mathcal{A}$, there exists a fixed constant $C^* \in (0, \infty)$, s.t. the upper bound on the RKHS norm of the ground truth f on sub-domain $\mathcal{X}_a$ is bounded by*

$$\overline{\Gamma}_a \leq C^* \cdot L \cdot w_a. \tag{4.9}$$

Theorem 3. *Let Assumptions 1, 2, 3, 6, 7, and 8 hold. Then, Algorithm 6 terminates after $N = \mathcal{O}\left(\left(\frac{L}{\bar{\epsilon}}\right)^n\right)$. Furthermore, the resulting approximating functions (cf. Algorithm 7) have a uniform approximation error smaller or equal to $\bar{\epsilon}$, i.e., (3.2) holds.*

Proof. **Part 1:** Consider any $a \in \mathcal{A}$. Suppose that

$$w_a \leq \frac{\overline{\epsilon}}{C^* \cdot L}. \tag{4.10}$$

If (4.10) holds, then it holds that

$$\begin{aligned} \overline{P}_{X_a}\overline{\Gamma}_a &\overset{(2.4)}{\leq} \overline{\Gamma}_a \\ &\overset{\text{Asm. } 8}{\leq} C^* \cdot L \cdot w_a \\ &\overset{(4.10)}{\leq} \overline{\epsilon}. \end{aligned} \tag{4.11}$$

Therefore, Algorithm 6 terminates.

Part 2: Achieving w_a requires $N = \mathcal{O}\left(\left(\frac{1}{w_a}\right)^n\right)$. Thus, Algorithm 6 terminates after (cf. (4.10)) $N = \mathcal{O}\left(\left(\frac{L}{\overline{\epsilon}}\right)^n\right)$.

Part 3: It holds for any $c \in \mathcal{C}_a$ and for any $x \in \mathcal{X}_{a,c}$ that

$$\begin{aligned} |f(x) - h_{X_{a,c}}(x)| &\overset{(3.3)}{\leq} P_{X_{a,c}}(x)\overline{\Gamma}_a \\ &\overset{\text{Asm. } 3}{\leq} \overline{P}_{X_a}\overline{\Gamma}_a \\ &\overset{(4.11)}{\leq} \overline{\epsilon}, \end{aligned} \tag{4.12}$$

i.e., (3.2) holds. ∎

5 ALKIA-X: Adaptive and Localized Kernel Interpolation Algorithm with extrapolated RKHS Norm

In Chapter 4, we presented a one-shot algorithm that learns sufficiently accurate localized approximating functions after a finite number of samples. Furthermore, we proposed an approach to avoid ill-conditioned covariance matrices by adaptively partitioning the domain $\mathcal{X}$ into sub-domains $\mathcal{X}_a$ and using scaled kernels (cf. (4.3)). However, we still assumed access to an oracle that returns an upper bound on the RKHS norm of the unknown ground truth f on each sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$ (cf. Assumption 6, 8). In this chapter, we extend the ideas presented in Chapter 4 by dropping the assumption of having access to an oracle by incorporating a heuristic to obtain an upper bound on the unknown RKHS norm of the ground truth. In particular, we employ an exponential extrapolation from samples of the RKHS norm of the approximating function, which we describe in Section 5.1. This extension leads to an implementable algorithm. In Section 5.2, we present the resulting pseudo algorithm. Finally, in Section 5.3, we conclude the chapter by proving that the presented pseudo algorithm terminates after a finite number of samples and that (3.2) holds for the resulting piecewise-defined approximating function (cf. Theorem 4).

5.1 RKHS Norm Extrapolation

In this section, we replace the oracle that returns an upper bound on the RKHS norm of the ground truth f for each sub-domain $\mathcal{X}_a$ (cf. Assumption 6). In particular, we extrapolate $\overline{\Gamma}_a$ from samples of the RKHS norm of the approximating function $\|h_{X_a}\|_{k_{\ell_a}}$ for all sub-domains $\mathcal{X}_a \subseteq \mathcal{X}$. As introduced in Section 4.1, we continue working with scaled kernels k_{ℓ_a} (cf. (4.3)). To define $\|h_{X_a}\|_{k_{\ell_a}}$ (cf. (2.9)), we require

function evaluations f_{X_a} and the inverse covariance matrix given samples X_a on sub-domain $\mathcal{X}_a$, i.e., $K^{-1}_{\ell_a,X_a}$. In contrast, in Chapters 3 and 4 we only used covariance matrices given samples on a local cube $\mathcal{X}_{a,c} \subseteq \mathcal{X}_a$ to build the local approximating functions. We still continue using covariance matrices defined on local cubes $\mathcal{X}_{a,c} \subseteq \mathcal{X}_a$ to build the local approximating functions. However, we now additionally require the covariance matrix given all samples on sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$ to extrapolate $\overline{\Gamma}_a$ from $\|h_{X_a}\|_{k_{\ell_a}}$.

As we already introduced in Section 4.1, we aim for numerically reliable calculations by ensuring well-conditioned covariance matrices, for which we introduced $\overline{M}$. Now, the maximum number of samples per axis $\overline{M}$ on a sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$ does not only have to ensure well-conditioned covariance matrices on a local cube $\mathcal{X}_{a,c} \subseteq \mathcal{X}_a$, but also well-conditioned covariance matrices given all samples on a sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$, which we need calculate $\|h_{X_a}\|_{k_{\ell_a}}$ in order to extrapolate $\overline{\Gamma}_a$. Therefore, in addition to (4.2), $\overline{M}$ further ensures that

$$\operatorname{cond}\left(K_{\ell_a,X_{a,\overline{M}}}\right) \leq \overline{\kappa} \quad \forall a \in \mathcal{A} \tag{5.1}$$

holds, i.e., that the condition number of the covariance matrix given $\overline{M}$ samples per axis on a sub-domain $\mathcal{X}_a$ is smaller or equal to $\overline{\kappa}$ for all $a \in \mathcal{A}$. We provide additional information on the choice of $\overline{\kappa}$ and $\overline{M}$ in Chapter 6.

In Chapter 4, we did not only assume access to an oracle that returns an upper bound on the RKHS norm of the ground truth f on sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$ (cf. Assumption 6) but also required a specific behavior of $\overline{\Gamma}_a$ (cf. Assumption 8). Assumption 8 states that $\overline{\Gamma}_a$ decreases as the width w_a of sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$ decreases. Furthermore, we used this property in Theorem 3 to prove convergence after a finite number of samples and to prove that the resulting piecewise-defined approximating function is sufficiently accurate. Therefore, we aim to resemble this behavior for the extrapolated $\overline{\Gamma}_a$. In the following, we work with *shifted* function values, which ensure that $\|h_{X_a}\|_{k_{\ell_a}} \to 0$ as $w_a \to 0$, which will induce that $\overline{\Gamma}_a \to 0$ as $w_a \to 0$, equivalent to Assumption 8.

We denote values that have been shifted w.r.t. function evaluations on X_a by the subscript s_{X_a}. We define shifted function evaluations as

$$f_{X_a,s_{X_a}} = f_{X_a} - \mu_{X_a}, \tag{5.2}$$

with

$$\begin{aligned} \operatorname{mean}(f_{X_a}) &= \frac{\sum_{x_i \in X_a} f_{x_i}}{\operatorname{card}(X_a)} \\ \mu_{X_a} &\coloneqq \operatorname{mean}(f_{X_a}). \end{aligned} \tag{5.3}$$

Next, we first provide some intuition on how the RKHS norm of the shifted approximating function on a sub-domain approaches zero as the width of the sub-domain approaches zero. If $w_a \to 0$, it holds that $\|x_i - x_j\|_2 \to 0$ for all $x_i, x_j \in X_a$. Therefore, as we assume Lipschitz-continuity (cf. Assumption 7), it follows that

$$\lim_{w_a \to 0} |f(x_i) - f(x_j)| = 0 \quad \forall x_i, x_j \in X_a. \tag{5.4}$$

Thus (cf. (5.2)),

$$\lim_{w_a \to 0} |f_{x, s_{X_a}}| = 0 \quad \forall x \in X_a \tag{5.5}$$

and

$$\begin{aligned} \lim_{w_a \to 0} \|h_{X_a, s_{X_a}}\|_{k_{\ell_a}} &= \sqrt{\mathbf{0}^\top_{\operatorname{card}(X_a)} K^{-1}_{\ell_a, X_a} \mathbf{0}_{\operatorname{card}(X_a)}} \\ &= 0. \end{aligned} \tag{5.6}$$

For this reason, when working with shifted function evaluations and a shifted $\|h_{X_a, s_{X_a}}\|_{k_{\ell_a}}$ to approximate an upper bound on the RKHS norm of the shifted ground truth on sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$, we can resemble the behavior stated in Assumption 8. Therefore, working with shifted function evaluations enables achieving arbitrarily small error bounds (cf. (3.2)), which we will present in Section 5.3. Since we now work with shifted function evaluations, we require $\overline{\Gamma}_{a,s}$, an upper bound on the *shifted* ground truth f on sub-domain $\mathcal{X}_a$.

Before devoting ourselves to formulating the extrapolating function to obtain $\overline{\Gamma}_{a,s}$, we introduce an overestimation of the RKHS norm of the shifted approximating function defined on sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$ to introduce conservatism to the upcoming extrapolation. To clarify how many samples per axis are used to define the objects of interest, we again introduce the explicit dependence on M. The overestimation of $\|h_{X_{a,M}, s_{X_{a,M}}}\|_{k_{\ell_a}}$ is given by

$$\hat{\Gamma}_{\ell_a, X_{a,M}, s_{X_{a,M}}} = \|h_{X_{a,M}, s_{X_{a,M}}}\|_{k_{\ell_a}} + C_\Gamma, \tag{5.7}$$

with

$$C_\Gamma = \frac{\overline{\epsilon}}{2^{\overline{\lambda}+1}} \tag{5.8}$$

and

$$\begin{aligned} \lambda &:= \frac{1}{1-\frac{M_i}{M_{i+1}}} \\ &\overset{(3.11)}{=} 2 + \frac{1}{M_i - 1} \\ &\leq 2 + \frac{1}{\underline{M} - 1} \\ &=: \overline{\lambda}. \end{aligned} \tag{5.9}$$

We denote by C_Γ the overestimation constant. Moreover, we define M_i and M_{i+1} as two consecutive numbers of uniform samples per axis (cf. (3.11)). To extrapolate the overestimation of the RKHS norm of the shifted approximating function on sub-domain $\mathcal{X}_a$, we use the exponential function

$$\widetilde{\Gamma}_{\ell_a, X_{a,M_{i,i+1}}, s_{X_{a,M_{i+1}}}}(M) = \overline{\Gamma}_{a,s} \cdot \exp\left(\frac{-\tau_{a,s}}{M}\right). \tag{5.10}$$

The goal of (5.10) is to predict the trend of the RKHS norm of the shifted approximating function given M uniform samples on a sub-domain $\mathcal{X}_a$. The extrapolation (5.10) is defined by the parameters $\overline{\Gamma}_{a,s}$ and $\tau_{a,s}$, which are determined based on the two samples

$$\begin{aligned} \widetilde{\Gamma}_{\ell_a, X_{a,M_{i,i+1}}, s_{X_{a,M_{i+1}}}}(M_i) &= \hat{\Gamma}_{\ell_a, X_{a,M_i}, s_{X_{a,M_i}}} \\ \widetilde{\Gamma}_{\ell_a, X_{a,M_{i,i+1}}, s_{X_{a,M_{i+1}}}}(M_{i+1}) &= \hat{\Gamma}_{\ell_a, X_{a,M_{i+1}}, s_{X_{a,M_{i+1}}}}. \end{aligned} \tag{5.11}$$

Note that in (5.11), we use one sample that is shifted w.r.t. X_{a,M_i} and another sample that is shifted w.r.t. $X_{a,M_{i+1}}$ to construct the extrapolating function (5.10). However, the extrapolation formula is shifted w.r.t. $X_{a,M_{i+1}}$ samples. [1] We can state explicit expressions for the parameters $\tau_{a,t}$ and $\overline{\Gamma}_{a,t}$, which yield

$$\tau_{a,s} = \frac{\ln \frac{\hat{\Gamma}_{\ell_a, X_{a,M_{i+1}}, s_{X_{a,M_{i+1}}}}}{\hat{\Gamma}_{\ell_a, X_{a,M_i}, s_{X_{a,M_i}}}}}{\frac{1}{M_i} - \frac{1}{M_{i+1}}}, \tag{5.12}$$

[1] We recommend shifting both samples and the RKHS norm extrapolation (cf. Assumption 9, (5.10)) by the same mean $\mu_{X_{a,M_i}}$. This would slightly change the pseudo algorithm in Section 5.2 and the proof in Section 5.3.

and

$$\begin{aligned}\overline{\Gamma}_{a,s} &= \hat{\Gamma}_{\ell_a, X_{a,M_i}, s_{X_{a,M_i}}} \cdot \exp\left(\frac{\tau_{a,s}}{M_i}\right) \\ &\overset{(5.12)}{=} \hat{\Gamma}_{\ell_a, X_{a,M_i}, s_{X_{a,M_i}}} \cdot \exp\left(\frac{\ln \frac{\hat{\Gamma}_{\ell_a, X_{a,M_{i+1}}, s_{X_{a,M_{i+1}}}}}{\hat{\Gamma}_{\ell_a, X_{a,M_i}, s_{X_{a,M_i}}}}}{1 - \frac{M_i}{M_{i+1}}}\right) \\ &\overset{(5.9)}{=} \hat{\Gamma}_{\ell_a, X_{a,M_i}, s_{X_{a,M_i}}} \left(\frac{\hat{\Gamma}_{\ell_a, X_{a,M_{i+1}}, s_{X_{a,M_{i+1}}}}}{\hat{\Gamma}_{\ell_a, X_{a,M_i}, s_{X_{a,M_i}}}}\right)^{\lambda}.\end{aligned} \tag{5.13}$$

Figure 5.1 illustrates the applied RKHS norm extrapolation. The starred data points show the sampled overestimations of the RKHS norm of the shifted approximating function with $M \in \{5, 9, 17, 33, 65\}$ samples per axis (cf. (5.7)). The solid line is the exponential extrapolation formula (cf. (5.10)) with $M_i = 5$ and $M_{i+1} = 9$. The experiment has been conducted on a toy example and illustrates that the trend of the RKHS norm of the shifted approximating function is captured reasonably well by the extrapolation. Although the extrapolating function is only defined by the samples of $M_i = 5$ and $M_{i+1} = 9$, the resulting extrapolation fits almost perfectly through the shifted data points obtained by $M \in \{17, 33, 65\}$ as well. The dashed line in Figure 5.1 shows $\overline{\Gamma}_{a,s}$ (cf. (5.13)), the resulting limit value of the exponential function (5.10).

Assumption 9. *For any $M_i, M_{i+1} \in \{\underline{M}, \ldots, \overline{M}\}$ according to* (3.11) *and for any $a \in \mathcal{A}$, $\overline{\Gamma}_{a,s}$ according to* (5.13) *is an upper bound on the RKHS norm of the shifted ground truth $f_{s_{X_{a,M_{i+1}}}}$ on sub-domain $\mathcal{X}_a$, i.e., $\overline{\Gamma}_{a,s} \geq \|f_{s_{X_{a,M_{i+1}}}}\|_{k_{\ell_a}}$.* [2]

Our heuristic (cf. Assumption 9) claims that using the exponential extrapolating function (5.10) returns an upper bound on the RKHS norm of the shifted ground truth $f_{s_{X_{a,M_{i+1}}}}$ that is defined on sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$. We use Assumption 9 to extend the framework presented in Chapter 4 by dropping the dependence on an oracle that returns an upper bound on the RKHS norm of the unknown ground truth f on each sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$. Next, we present the resulting pseudo algorithm in Section 5.2.

[2] Note that the ground truth f is only defined on the respective sub-domain $\mathcal{X}_a$ and is shifted by $\mu_{X_{a,M_{i+1}}}$.

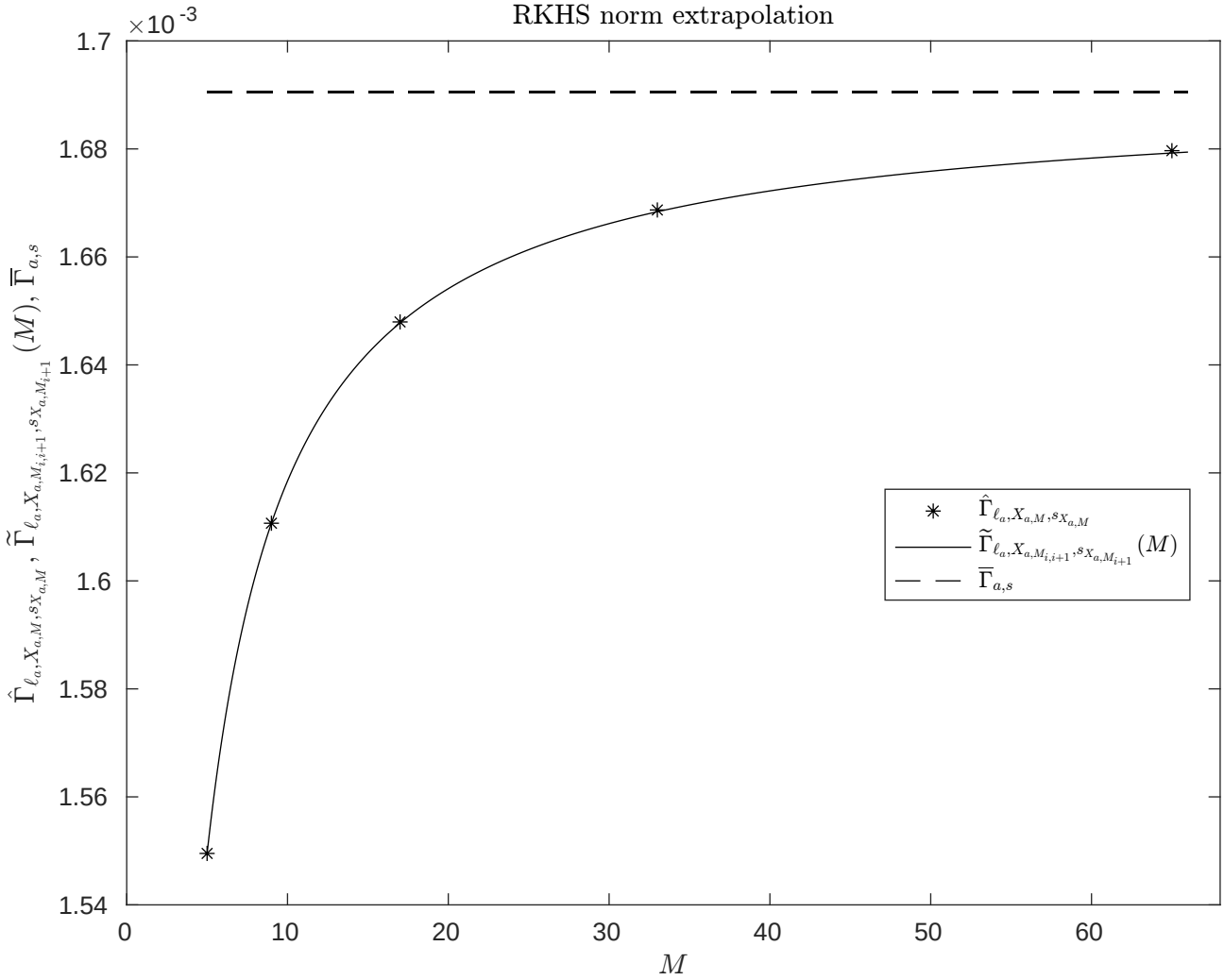

Figure 5.1: The starred data points show evaluated values of (5.13). The solid line shows the extrapolating function (5.10) constructed based on the first two data points, i.e., $M_i = 5$ and $M_{i+1} = 9$. The dashed line illustrates the limit value of the extrapolating function (5.13).

5.2 Pseudo Algorithm

We now present the pseudo algorithm of ALKIA-X, our adaptive and localized kernel interpolation algorithm with extrapolated RKHS norm. ALKIA-X employs the local approximation approach presented in Chapter 3 to have a guaranteed bound on the uniform approximation error and to ensure a fast-to-evaluate approximating function. Moreover, ALKIA-X enables scalability through well-conditioned covariance matrices by employing an adaptive sub-domain partitioning (cf. Chapter 4). Furthermore, ALKIA-X applies the extension presented in Section 5.1 and does not rely on a given RKHS norm upper bound of the unknown ground truth. Thus, ALKIA-X is an implementable algorithm. ALKIA-X works with shifted function values to ensure that $\overline{\Gamma}_{a,s}$ decreases as the width of the sub-domain $\mathcal{X}_a$ decreases (cf. Section 5.1). Therefore, ALKIA-X creates shifted localized approximating functions for each sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$. The resulting shifted approximation is shifted back in the online evaluation (cf. Algorithm 11) to return the absolute value of the approximation.

Algorithm 8 ALKIA-X: Evaluate function and extrapolate RKHS norm

Require: f, $\mathcal{X}_a$, C_Γ, M_i, M_{i+1}, k_{ℓ_a}

1: **for** $M' \in \{M_i, M_{i+1}\}$ **do**
2: $X_{a,M'} \leftarrow$ set of uniform samples on $\mathcal{X}_a$ with M' samples per axis
3: $f_{X_{a,M'}} \leftarrow f(X_{a,M'})$ ▷ Function evaluation for M'
4: $\mu_{X_{a,M'}} \leftarrow \text{mean}(f_{X_{a,M'}})$
5: $f_{X_{a,M'},s_{X_{a,M'}}} \leftarrow f_{X_{a,M'}} - \mu_{X_{a,M'}}$ ▷ Shift function evaluations around mean
6: $K^{-1}_{\ell_a,X_{a,M'}} \leftarrow k_{\ell_a}, X_{a,M'}$ in (2.2) ▷ Inverse covariance matrix
7: $k_{\ell_a,X_{a,M'}}(x) \leftarrow k_{\ell_a}, X_{a,M'}$ in (2.3) ▷ Covariance vector
8: $\|h_{X_{a,M'},s_{X_{a,M'}}}\|_{k_{\ell_a}} \leftarrow k_{\ell_a,X_{a,M'}}(x), K^{-1}_{\ell_a,X_{a,M'}}, f_{X_{a,M'},s_{X_{a,M'}}}$ in (2.9)
▷ RKHS norm of the shifted approximating function
9: $\hat{\Gamma}_{\ell_a,X_{a,M'},s_{X_{a,M'}}} \leftarrow \|h_{X_{a,M'},s_{X_{a,M'}}}\|_{k_{\ell_a}}, C_\Gamma$ in (5.13)
▷ Overestimation of the RKHS norm of the shifted approximating function
10: **end for**
11: $\overline{\Gamma}_{a,s} \leftarrow \hat{\Gamma}_{\ell_a,X_{a,M_i},s_{X_{a,M_i}}}, \hat{\Gamma}_{\ell_a,X_{a,M_{i+1}},s_{X_{a,M_{i+1}}}}, M_i, M_{i+1}$ in (5.13)
▷ Extrapolated RKHS norm upper bound (cf. Assumption 9)
12: **return** $\overline{\Gamma}_{a,s}$, $\mu_{X_{a,M_{i+1}}}$, $f_{X_{a,M_{i+1}},s_{X_{a,M_{i+1}}}}$, $X_{a,M_{i+1}}$, M_{i+1}, $K^{-1}_{\ell_a,X_{a,M_{i+1}}}$

Algorithm 8 summarizes the function evaluation and the calculation of an upper

bound on the RKHS norm of the shifted ground truth (cf. (5.13), Assumption 9). First, we loop over M_i and M_{i+1} and determine, for the respective number of samples per axis on the current sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$, the sample set, the function evaluations, the mean values, the shifted function evaluations, the covariance vector, and the inverse covariance matrix. Then, Algorithm 8 calculates the RKHS norm of the shifted approximating function and its overestimation. Moreover, we obtain the extrapolated upper bound on the RKHS norm of the shifted ground truth $\overline{\Gamma}_{a,s}$ for the current sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$ (cf. Assumption 9). Since we use update rule (3.11), we do not discard any samples. Furthermore, we continue the approximation procedure with quantities obtained by M_{i+1} samples per axis. Thus, returning the values given by M_{i+1} uniform samples per axis is sufficient.

Algorithm 9 ALKIA-X: Local approximating functions

Require: $f_{X_{a,M_{i+1}},s_{X_{a,M_{i+1}}}}$, $X_{a,M_{i+1}}$, $K^{-1}_{\ell_a,X_{a,M_{i+1},c}}$, k_{ℓ_a}

1: Initialize: dictionary$_a \leftarrow$ empty dictionary
2: $\mathcal{C}_{a,M_{i+1}} \leftarrow$ set of positive integers s.t. $\cup_{c' \in \mathcal{C}_{a,M_{i+1}}} X_{a,M_{i+1},c'} = X_{a,M_{i+1}}$
3: **for** $c' \in \mathcal{C}_{a,M_{i+1}}$ **do** ▷ Split samples into cubes
4: $\quad f_{X_{a,M_{i+1},c},s_{X_{a,M_{i+1}}}} \leftarrow$ Extract function evaluations for $X_{a,M_{i+1},c'}$
5: $\quad k_{\ell_a,X_{a,M_{i+1},c'}}(x) \leftarrow k_{\ell_a}, X_{a,M_{i+1},c'}$ in (2.3) ▷ Covariance vector
6: $\quad h_{X_{a,M_{i+1},c'},s_{X_{a,M_{i+1}}}}(x)$
$\quad\quad \leftarrow f_{X_{a,M_{i+1},c},s_{X_{a,M_{i+1}}}}, K^{-1}_{\ell_a,X_{a,M_{i+1},c}}, k_{\ell_a,X_{a,M_{i+1},c'}}(x)$ in (2.5)
7: $\quad$ dictionary$_a(X_{a,M_{i+1},c'}) \leftarrow h_{X_{a,M_{i+1},c'},s_{X_{a,M_{i+1}}}}(x)$ ▷ Store approximating functions
8: **end for**
9: **return** dictionary$_a$

Algorithm 9 shows how the shifted local approximating functions for a sub-domain $\mathcal{X}_a$ are created. The procedure is almost identical to Algorithm 3. However, Algorithm 9 works with shifted function evaluations to create a shifted piecewise-defined approximating function. After defining the shifted local approximating functions for sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$, they are stored in dictionary$_a$.

Algorithm 10 shows the complete offline learning procedure of ALKIA-X. We start by initializing $\mathcal{X}_{\text{tasks}}$, X_{total}, and dictionary$_{\text{total}}$. Moreover, we execute the dynamic for-loop over $\mathcal{X}_{\text{tasks}}$ to define approximating functions for all sub-domains $\mathcal{X}_a \subseteq \mathcal{X}$. First, we define the width, the corresponding length scale, and the scaled

Algorithm 10 ALKIA-X: offline learning

Require: $\underline{M}, \overline{M}, \mathcal{X}, k, \bar{\epsilon}, C_\ell, C_\Gamma$

Initialize: $\mathcal{X}_{\text{tasks}} \leftarrow \{\mathcal{X}\}$, $X_{\text{total}} \leftarrow \{\}$, dictionary_total $\leftarrow$ empty dictionary
for $\mathcal{X}_a \in \mathcal{X}_{\text{tasks}}$ **do** ▷ Dynamic for loop
  $w_a \leftarrow \text{width}(\mathcal{X}_a)$
  $\ell_a \leftarrow C_\ell \cdot w_a$
  Define scaled kernel k_{ℓ_a} given k and ℓ_a (cf. (4.3))
  $M_i \leftarrow \underline{M}$, $M_{i+1} \leftarrow 2\underline{M} - 1$
  while True **do**
    $\overline{\Gamma}_{a,s}$, $\mu_{X_{a,M_{i+1}}}$, $f_{X_{a,M_{i+1}},s_{X_{a,M_{i+1}}}}$, $X_{a,M_{i+1}}$,$K^{-1}_{\ell_a,X_{a,M_{i+1}}} \leftarrow$ Algorithm 8(f, $\mathcal{X}_a$, C_Γ, M_i, M_{i+1}, k_{ℓ_a})
      ▷ Function evaluation and RKHS norm extrapolation
    $\overline{P}_{X_{a,M_{i+1}}}, K^{-1}_{\ell_a,X_{a,M_{i+1},c}} \leftarrow$Algorithm 1($k_{\ell_a}, X_{a,M_{i+1}}$)
      ▷ Upper bound on power function
    if $\overline{P}_{X_{a,M_{i+1}}} \overline{\Gamma}_{a,s} \leq \bar{\epsilon}$ **then** ▷ Desired accuracy achieved
      dictionary$_a \leftarrow$ Algorithm 9($f_{X_{a,M_{i+1}},s_{X_{a,M_{i+1}}}}, X_{a,M_{i+1}}, K^{-1}_{\ell_a,X_{a,M_{i+1},c}}, k_{\ell_a}$)
      ▷ Construct shifted local approximating functions of sub-domain $\mathcal{X}_a$
      dictionary$_{\text{total}}(X_{a,M_{i+1}}) \leftarrow$ dictionary$_a, \mu_{X_{a,M_{i+1}}}$
      $X_{\text{total}} \leftarrow X_{\text{total}} \cup X_{a,M_{i+1}}$
      $\mathcal{X}_{\text{tasks}} \leftarrow \mathcal{X}_{\text{tasks}} \setminus \{\mathcal{X}_a\}$ ▷ Remove $\mathcal{X}_a$ from the set of tasks
      break ▷ Approximated $\mathcal{X}_a$ successfully
    else ▷ Desired accuracy not yet achieved
      $M_i \leftarrow M_{i+1}$, $M_{i+1} \leftarrow 2M_{i+1} - 1$ ▷ Increase the number of samples
    end if
    if $M_{i+1} > \overline{M}$ **then** ▷ Maximum number of samples per domain exceeded
      $\mathcal{X}_{a,i} \leftarrow$ Create 2^n uniform sub-domains of $\mathcal{X}_a$
      $\mathcal{X}_{\text{tasks}} \leftarrow \mathcal{X}_{\text{tasks}} \cup_{i\in\{1,\dots,2^n\}} \{\mathcal{X}_{a,i}\}$
      $\mathcal{X}_{\text{tasks}} \leftarrow \mathcal{X}_{\text{tasks}} \setminus \{\mathcal{X}_a\}$ ▷ Remove $\mathcal{X}_a$ from the set of tasks
      break ▷ Approximate separately on sub-domains $X_{a,i}$
    end if
  end while
end for
return dictionary$_{\text{total}}$, X_{total}

kernel of the current sub-domain $\mathcal{X}_a$. Then, we initialize two consecutive numbers of samples per axis M_i and M_{i+1}. In contrast to the algorithms presented in Chapters 3 and 4, we now have to define the number of samples per axis in the main offline-learning algorithm, as not only the power function but also the upper bound on the RKHS norm of the ground truth depends on M_i and M_{i+1} (cf. (5.13), Assumption 9). Hence, the while-True loop is now also embedded in the main offline-learning algorithm.

We receive the function evaluations and the extrapolated upper bound on the RKHS norm of the shifted ground truth $\overline{\Gamma}_{a,s}$ on the current sub-domain $\mathcal{X}_a$ from Algorithm 8. Furthermore, Algorithm 1 determines the maximum of the power function for a local cube given the current set of samples $X_{a,M_{i+1}}$. If the desired approximation accuracy is achieved for the current sub-domain, the shifted piecewise-defined approximating function is created by Algorithm 9. Moreover, we save the local dictionary containing the local approximating functions and the mean function value in $\text{dictionary}_{\text{total}}$. Furthermore, we add $X_{a,M_{i+1}}$ to X_{total} and remove $\mathcal{X}_a$ from $\mathcal{X}_{\text{tasks}}$. As we created sufficiently accurate approximating functions for all local cubes of the current sub-domain $\mathcal{X}_a \subseteq \mathcal{X}$, we break out of the while-True loop and move to the next sub-domain.

In case the current number of samples per axis does not yield a uniform approximation error smaller or equal to $\overline{\epsilon}$, we increase the number of samples per axis and go through the described procedure again. Note that we do not want to have more than $\overline{M}$ samples per axis to avoid ill-conditioned convariance matrices (cf. (4.2), (5.1)). Hence, we split the sub-domain $\mathcal{X}_a$ into 2^n further sub-domains whenever more than $\overline{M}$ samples are required to achieve the desired accuracy. Then, the described procedure is repeated. The algorithm terminates after $\mathcal{X}_{\text{tasks}}$ is empty, i.e., when we created sufficiently accurate localized approximating functions for all sub-domains $\mathcal{X}_a \subseteq \mathcal{X}$.

Algorithm 10 is *not* a one-shot algorithm anymore, as we have to evaluate the ground truth before computing the error bound, which leads to the described iterative procedure. Therefore, it is not possible to a priori define the samples needed for the offline learning. However, it is still completely automated and in Theorem 4 we will show that Algorithm 10 terminates after a finite number of samples and results in a piecewise-defined approximating function that guarantees sufficient accuracy.

For the online evaluation in Algorithm 11, we require $\text{dictionary}_{\text{total}}$, X_{total}, and an

Algorithm 11 ALKIA-X: online evaluation

Require: $\text{dictionary}_{\text{total}}$, X_{total}, x'

1: $X_a \leftarrow$ relevant samples of X_{total} that contain x'
2: $X_{a,c} \leftarrow$ relevant cube of X_a that contains x'
3: $\text{dictionary}_a, \mu_{X_a} \leftarrow \text{dictionary}_{\text{total}}(X_a)$
4: $h_{X_{a,c},s_{X_a}}(x) \leftarrow \text{dictionary}_a(X_{a,c})$ ▷ Shifted approximating function
5: $h_{X_{a,c}}(x') \leftarrow h_{X_{a,c},s_{X_a}}(x') + \mu_{X_a}$ ▷ Shift back the prediction
6: **return** $h_{X_{a,c}}(x')$

input $x' \in \mathcal{X}$. The procedure is very similar to Algorithm 7. However, $\text{dictionary}_{\text{total}}$ now also returns the mean value in addition to dictionary_a. After obtaining the relevant shifted approximating function, we evaluate it and shift the evaluation back by adding the mean value. Finally, we return the absolute approximation value.

5.3 Theoretical Analysis

After seeing pseudo algorithms of the offline approximation and the online evaluation of ALKIA-X in Section 5.2, we now prove that Algorithm 10 terminates after a finite number of samples. Moreover, we provide a worst-case sample complexity and show that the resulting piecewise-defined approximating function (cf. Algorithm 11) is sufficiently accurate, i.e., (3.2) holds.

Theorem 4. *Let Assumptions 1, 2, 3, 7, and 9 hold. Then, Algorithm 10 terminates after* $N = \mathcal{O}\left(\left(\frac{L}{\bar{\epsilon}}\right)^n\right)$. *Furthermore, the resulting piecewise-defined approximating function (cf. Algorithm 11) has a uniform approximation error that is smaller or equal to* $\bar{\epsilon}$, *i.e.,* (3.2) *holds.*

Proof. **Part 1:** Consider any $a \in \mathcal{A}$. First, we prove that $|f_{x,s_{X_a}}| \leq L \cdot w_a$ for all $x \in \mathcal{X}_a$. Consider an arbitrary $x \in \mathcal{X}_a$, where, w.l.o.g., $f_{x,s_{X_a}} \geq 0$. Pick any other $x' \in \mathcal{X}_a$, where $f_{x',s_{X_a}} \leq 0$, which exists since $\text{mean}(f_{X_a,s_{X_a}}) = 0$. It holds that

$$|f_{x,s_{X_a}}| \leq |f_{x,s_{X_a}} - f_{x',s_{X_a}}| \overset{\text{Asm. 7}}{\leq} L \cdot \|x - x'\|_\infty \leq L \cdot w_a. \tag{5.14}$$

Furthermore, for all $M \in \{\underline{M}, \dots, \overline{M}\}$, it holds that

$$\begin{aligned}
\|h_{X_{a,M},s_{X_{a,M}}}\|_{k_{\ell_a}} &\overset{(2.9)}{=} \sqrt{f^\top_{X_{a,M},s_{X_{a,M}}} K^{-1}_{\ell_a,X_{a,M}} f_{X_{a,M},s_{X_{a,M}}}} \\
&\leq \sqrt{\|f_{X_{a,M},s_{X_{a,M}}}\|_1 \|f_{X_{a,M},s_{X_{a,M}}}\|_\infty \|K^{-1}_{\ell_a,X_{a,M}}\|_\infty} \\
&\overset{(5.14)}{\leq} L \cdot w_a \sqrt{\overline{M}\|K^{-1}_{\ell_a,X_{a,M}}\|_\infty} \\
&\overset{(5.1)}{\leq} L \cdot w_a \sqrt{\overline{M}\frac{\overline{\kappa}}{\|K_{\ell_a,X_{a,M}}\|_\infty}} \\
&\leq L \cdot w_a \sqrt{\overline{M}\overline{\kappa}}.
\end{aligned} \tag{5.15}$$

The last inequality used the fact that $\|K_{\ell_a,X_{a,M}}\|_\infty \geq \max_{i,j} K_{X_{a,M}}(i,j) = 1$ (cf. Assumption 1).

Suppose

$$w_a \leq \frac{\overline{\epsilon}}{L \cdot \sqrt{\overline{M}\overline{\kappa}} \cdot 2^{\overline{\lambda}+1}} =: \overline{w}. \tag{5.16}$$

Then, for all $M_i, M_{i+1} \in \{\underline{M}, \dots, \overline{M}\}$ according to (3.11), it holds that

$$\begin{aligned}
\overline{P}_{X_{a,M_{i+1}}} \overline{\Gamma}_{a,s} &\leq \overline{\Gamma}_{a,s} \\
&\overset{(5.13),(5.9)}{\leq} \hat{\Gamma}_{X_{a,M_i},s_{X_{a,M_i}}} \left(\frac{\hat{\Gamma}_{X_{a,M_{i+1}},s_{X_{a,M_{i+1}}}}}{\hat{\Gamma}_{X_{a,M_i},s_{X_{a,M_i}}}}\right)^{\overline{\lambda}} \\
&\overset{(5.7)}{\leq} \left(\|h_{X_{a,M_i},s_{X_{a,M_i}}}\|_{k_{\ell_a}} + C_\Gamma\right)\left(1 + \frac{\|h_{X_{a,M_{i+1}},s_{X_{a,M_{i+1}}}}\|_{k_{\ell_a}}}{C_\Gamma}\right)^{\overline{\lambda}} \\
&\overset{(5.15)}{\leq} \left(L \cdot w_a \cdot \sqrt{\overline{M}\overline{\kappa}} + C_\Gamma\right)\left(1 + \frac{L \cdot w_a \cdot \sqrt{\overline{M}\overline{\kappa}}}{C_\Gamma}\right)^{\overline{\lambda}} \\
&\overset{(5.16)}{\leq} \left(L \cdot \overline{w} \cdot \sqrt{\overline{M}\overline{\kappa}} + C_\Gamma\right)\left(1 + \frac{L \cdot \overline{w} \cdot \sqrt{\overline{M}\overline{\kappa}}}{C_\Gamma}\right)^{\overline{\lambda}} \\
&\overset{(5.8),(5.16)}{=} \left(\frac{2\overline{\epsilon}}{2^{\overline{\lambda}+1}}\right) 2^{\overline{\lambda}} \\
&= \overline{\epsilon},
\end{aligned} \tag{5.17}$$

i.e., Algorithm 10 terminates.

Part 2: Achieving w_a requires $N = \mathcal{O}\left(\left(\frac{1}{w_a}\right)^n\right)$. Thus, Algorithm 10 terminates

after (cf. (5.16)) $N = \mathcal{O}\left(\left(\frac{L}{\overline{\epsilon}}\right)^n\right)$.

Part 3: For all $M_{i+1} \in \{2\underline{M} - 1, \ldots, \overline{M}\}$ according to (3.11), for all $c \in \mathcal{C}_a$, and for all $x \in \mathcal{X}_{a,c}$, it holds that

$$\begin{aligned} |f(x) - h_{X_{a,M_{i+1},c}}(x)| &= |f_{s_{X_{a,M_{i+1}}}}(x) - h_{X_{a,M_{i+1},c},s_{X_{a,M_{i+1}}}}(x)| \\ &\overset{(3.3)}{\leq} P_{X_{a,M_{i+1}}}(x)\|f_{s_{X_{a,M_{i+1}}}}\|_{k\ell_a} \\ &\overset{\text{Asm. } 3}{\leq} \overline{P}_{X_{a,M_{i+1}}}\|f_{s_{X_{a,M_{i+1}}}}\|_{k\ell_a} \\ &\overset{\text{Asm. } 9}{\leq} \overline{P}_{X_{a,M_{i+1}}}\overline{\Gamma}_{a,s} \\ &\overset{(5.17)}{\leq} \overline{\epsilon}, \end{aligned} \tag{5.18}$$

i.e., (3.2) holds.

■

6 Implementation

In this chapter, we provide a high-level description of parts of the implementation of ALKIA-X. In Section 6.1, we present the chosen hyperparameters to ensure well-conditioned covariance matrices (cf. Chapters 4 and 5). Moreover, in Section 6.2, we present the implementation of ALKIA-X in `Python`. In particular, we describe how the adaptive sub-domain partitioning (cf. Chapter 4) is translated into a dynamic tree implementation. Furthermore, we present methods to make ALKIA-X more efficient. In Section 6.3, we demonstrate the parallelization of the offline training. In Section 6.4, we provide information on the implementation of the MPC, which we will use in Chapter 7 to perform numerical experiments.

6.1 Choice of Parameters to ensure well-conditioned Covariance Matrices

In Chapters 4 and 5, we enabled numerically reliable computations and scalability by ensuring well-conditioned covariance matrices without requiring any regularization. To achieve this, we introduced scaled kernels (cf. (4.3)) and an adaptive sub-domain partitioning, where each sub-domain contains at most $\overline{M}$ samples. First, choices on the scaled kernel k_ℓ and the corresponding length scale parameter C_ℓ (cf. (4.8)) are presented.

Figure 6.1 illustrates the condition number of the covariance matrix of the SE-kernel with M uniform samples on the domain $\mathcal{X} = [0, 1]^2$. The investigation was conducted with a length scale parameter of $C_\ell = 1$. Since the domain $\mathcal{X} = [0, 1]^2$ has a width of $w = 1$ (cf. (4.7)), the length scale results in $\ell = 1$ (cf. (4.8)). As can be seen, the condition number of the covariance matrix grows exponentially and approaches a value of 10^{12} with only $M = 5$ samples per axis. In other words, if we employ more than 25 samples per sub-domain to create approximating functions for the presented two-dimensional example, the covariance matrices would be severely

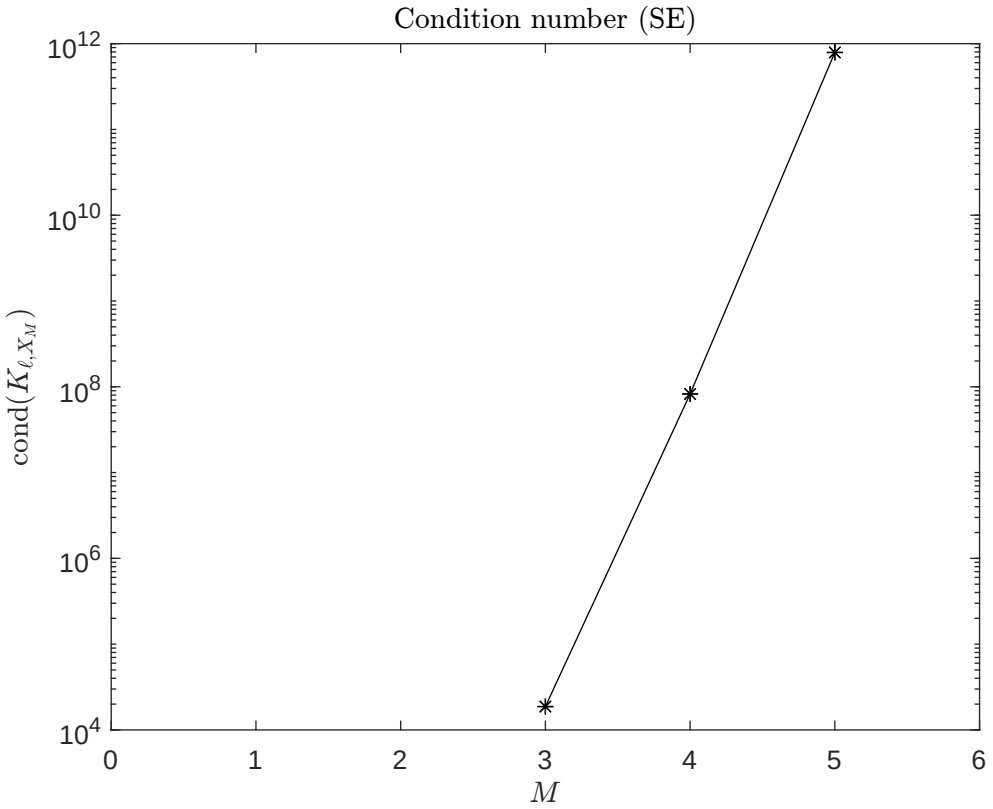

Figure 6.1: Condition number of the covariance matrix of the SE-kernel for different values of M uniform samples on $\mathcal{X} = [0,1]^2$. We chose $C_\ell = 1$, i.e., $\ell = 1$.

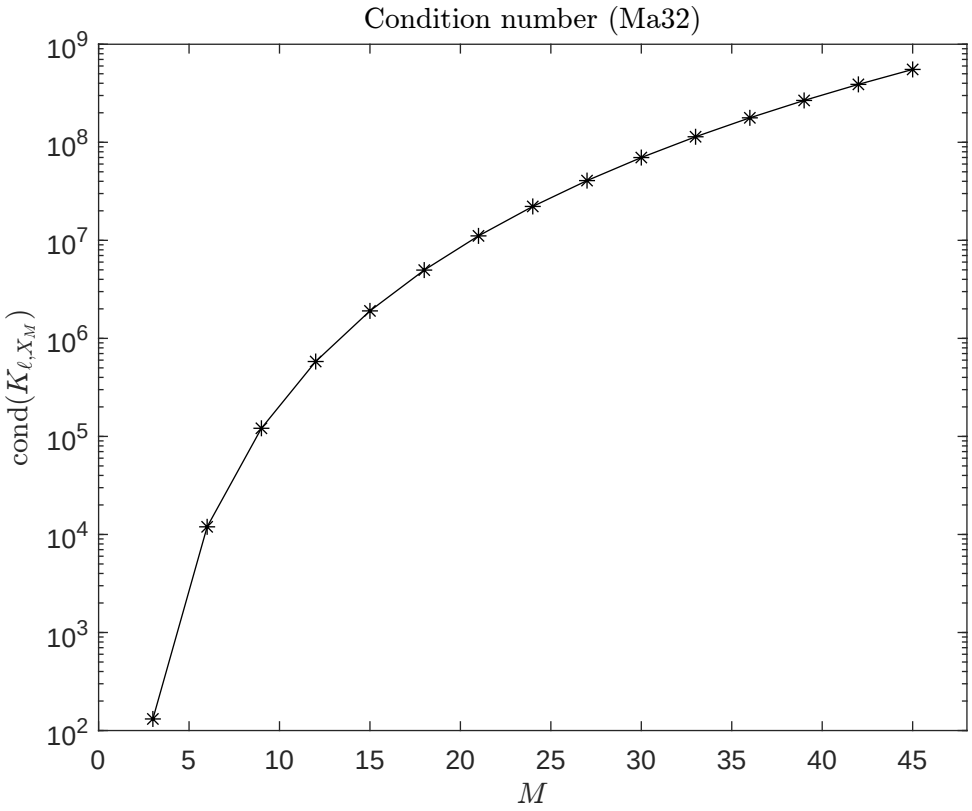

Figure 6.2: Condition number of the covariance matrix of the Ma32-kernel for different values of M uniform samples on $\mathcal{X} = [0,1]^2$. We use $C_\ell = 1$, i.e., $\ell = 1$.

ill-conditioned. Hence, the resulting kernel interpolation computations would become unreliable.

In Figure 6.2, we present the same investigations for the Ma32-kernel. The condition number of the covariance matrix of the Ma32-kernel grows significantly slower compared to the SE-kernel. For instance, $M = 30$ samples per axis still result in a condition number of less than 10^8. Because of the more favorable conditioning properties when compared to the SE-kernel, we choose k to be the Ma32 kernel for the remainder of this work. Nevertheless, there exist numerous other kernels (cf. e.g., [43]) that may also be suitable. However, we did not investigate the kernel choice any further.

Figure 6.3 shows the Ma32-kernel with different length scale parameters C_ℓ. Since

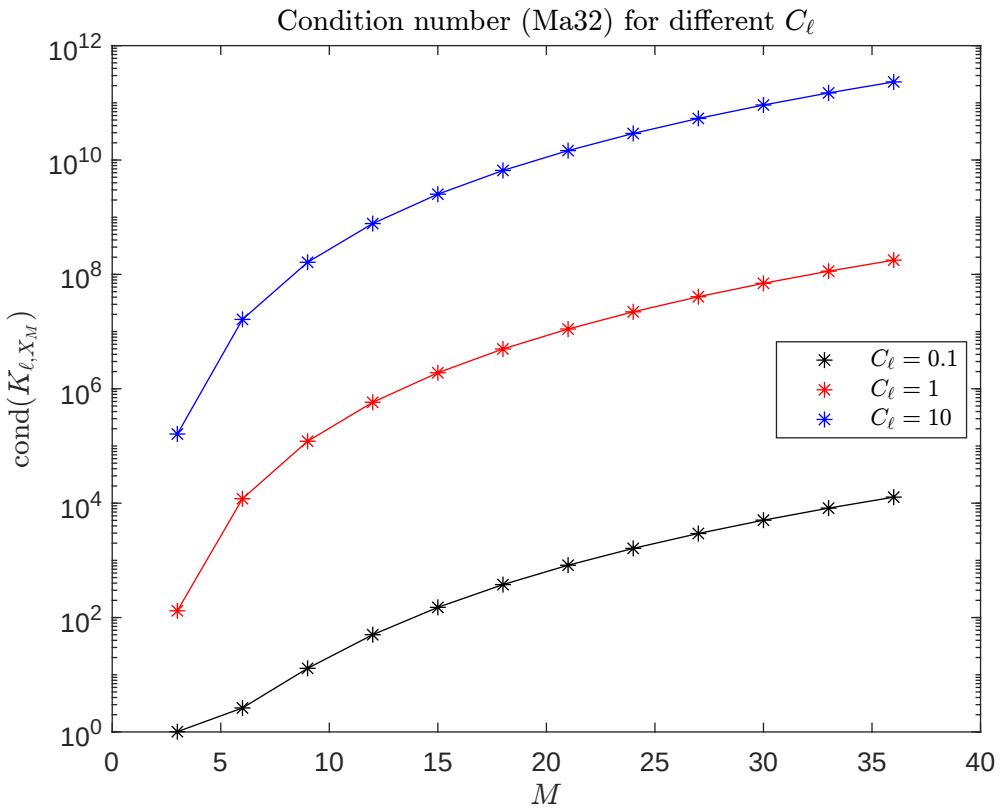

Figure 6.3: Condition number of covariance matrices given M uniform samples per axis on $\mathcal{X} = [0, 1]^2$. We used the Ma32-kernel and three different length scale parameters C_ℓ.

we still conduct the experiments on the domain $\mathcal{X} = [0, 1]^2$ with a width of $w_a = 1$, the length scale ℓ is equal to the length scale parameter C_ℓ (cf. (4.8)). Figure 6.3 indicates that a larger length scale parameter results in a larger condition number. In Section 4.1, we demonstrated that the condition number of a covariance matrix for a scaled kernel k_ℓ (cf. (4.3)) decreases as ℓ decreases. However, we also saw that if ℓ approaches zero, using the resulting covariance matrices impedes creating approximating functions that have an arbitrarily small error bound (cf. (4.5)). Therefore, we conclude that it is sensible to choose C_ℓ between 0.1 and 1, as this allows for having more than 30 samples per axis while still ensuring well-conditioned covariance matrices with a condition number of approximately 10^8. Henceforth, we choose $C_\ell = 0.8$ and conduct the upcoming experiments with this parameter.

Furthermore, we investigate the condition number for a covariance matrix that is defined on a local cube and for a covariance matrix that uses all samples on a sub-domain. Covariance matrices for local cubes are used in Chapters 3, 4, and 5 to construct the local approximating functions. Moreover, we use the covariance matrix given all samples on a sub-domain in Chapter 5 to determine an upper bound on the RKHS norm of the ground truth from samples. We choose to set the maximum number of samples per axis to $\overline{M} = 33$ to simultaneously allow for a rather dense sampling and well-conditioned covariance matrices (cf. Figure 6.3). Thus, every sub-domain has at most $\overline{M}^n$ samples. Although every local cube always contains 2^n samples, the maximum number of samples per axis $\overline{M}$ determines the sample density. Choosing $\overline{M} = 33$ results in an upper bound on the condition number of a covariance matrix on a local cube of $\overline{\kappa}_c = 4.961 \cdot 10^4$ (cf. (4.2)). Correspondingly, the condition number for covariance matrices that use all $\overline{M}^n$ samples on a sub-domain results in $\overline{\kappa} = 5.258 \cdot 10^7$ (cf. (5.1)). Note that we only investigated the choice of $\overline{M}$ for $n = 2$. For higher dimensional applications, the resulting condition numbers change, and thus is it recommended to investigate the resulting values of $\overline{\kappa}$ and $\overline{\kappa}_c$. To conclude, the largest condition number that we use when employing ALKIA-X with the presented choice of hyperparameters on a two-dimensional experiment is smaller than 10^8, which results in numerically reliable computations. Therefore, we can use millions of samples without requiring any regularization.

In addition to specifying $\overline{M}$, we also have to choose $\underline{M}$. The specification of $\underline{M}$ does not influence the maximum condition number. We choose to set $\underline{M} = 5$, which results in at least 9^n samples (cf. (3.11)) used per sub-domain to create the

approximating function (cf. Algorithm 10).

6.2 Implementation in `Python`

In this section, we present the implementation of ALKIA-X in `Python`. First, we show how the adaptive sub-domain partitioning introduced in Chapter 4 is translated to a dynamic tree implementation. Furthermore, we briefly present additional information on how to make ALKIA-X more efficient.

In Chapters 4 and 5, we defined the maximum number of samples per axis for a sub-domain to ensure well-conditioning. The resulting pseudo algorithms (cf. Algorithms 6 and 10) employ an adaptive sub-domain partitioning and split a domain into 2^n further sub-domains whenever $\overline{M}$ samples per axis do not yield a sufficiently accurate approximation. To implement the adaptive sub-domain partitioning, we employ a dynamic tree implementation in `Python` using [48].

In our implementation, every sub-domain corresponds to a node. If the number of maximum samples per axis $\overline{M}$ is exceeded for a sub-domain, we branch the corresponding node and create 2^n children nodes. If $\overline{M}$ or fewer samples per axis yield sufficiently accurate approximating functions, we prune the corresponding node. As a result, the leaf nodes correspond to the final sub-domain partitioning and contain the local approximating functions that approximate the whole domain with the desired accuracy.

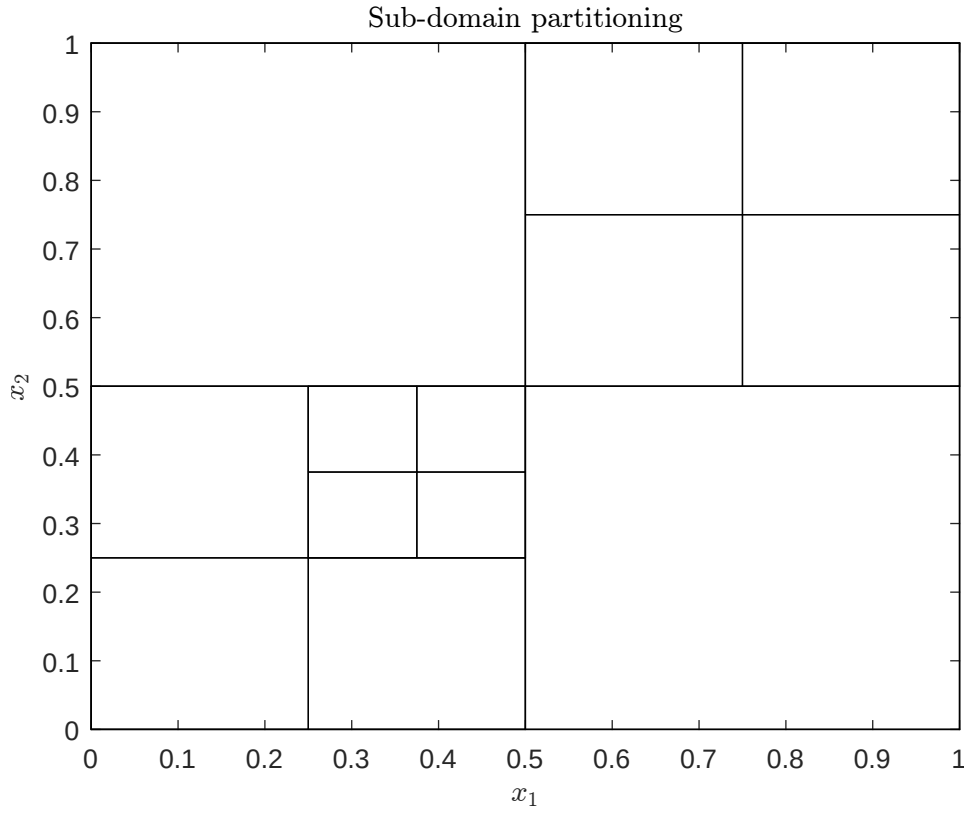

Figure 6.4: Example of a resulting sub-domain partitioning with ALKIA-X.

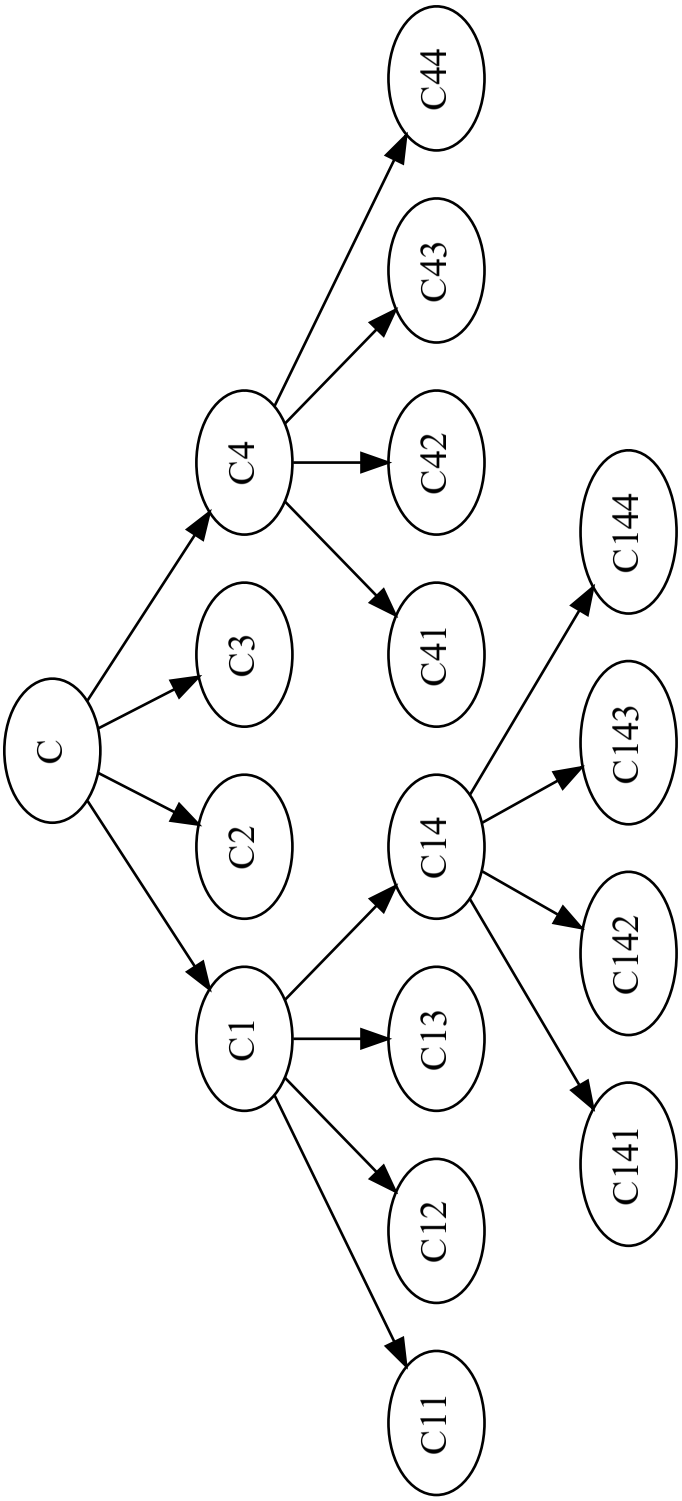

Figure 6.5: Resulting tree structure of the sub-domain partitioning in Figure 6.4.

The tree implementation is illustrated in Figures 6.4 and 6.5, which show the sub-domain partitioning and the resulting tree structure for a two-dimensional example. The root node C (cf. Figure 6.5) corresponds to the whole input domain $\mathcal{X} = [0,1]^2$ (cf. Figure 6.4). Creating the four children nodes C1-C4 leads to the partitioning of $\mathcal{X} = [0,1]^2$ into four uniform squares in Figure 6.4. Nodes C1 and C4 are partitioned further, which leads to four more uniform squares, respectively. The adaptive sub-domain partitioning is concluded with the partitioning of C14 into C141-C144. Every node branching adds depth to the resulting tree, wherefore the tree in Figure 6.5 has a depth of three.

The piecewise-defined approximating function that is used for the online evaluation (cf. Algorithm 11) is located at the leaf nodes of the n-ary tree. Therefore, Algorithm 11 executes an n-ary tree search through the depth levels of the tree to find the relevant leaf node that corresponds to the sub-domain for the input that we aim to evaluate.

To make the offline learning of ALKIA-X more efficient, we store the function evaluations obtained by a node in a `Python`-dictionary. This dictionary is handed over to the children nodes to use previously obtained function evaluations.

The covariance matrix, the covariance vector, the power function, and the resulting approximating function (cf. Chapter 2) are calculated using the GP-library of [49]. Determining the approximating function (cf. (2.5)) requires inverting the covariance matrix. However, [49] employs the Cholesky decomposition and solves a linear system between the function evaluations f_X and the covariance matrix K_X to omit inverting the matrix directly. The solution of this linear system is a vector that is equal to $f_X^\top K_X^{-1}$. For the online evaluation of an input x', we first have to create the covariance vector evaluated at x', i.e., $k_X(x')$. Furthermore, we need to multiply the offline determined vector $f_X^\top K_X^{-1}$ with the covariance vector $k_X(x')$. Therefore, the complete online evaluation boils down to finding the relevant leaf node, constructing the covariance vector, and calculating a 2^n-dimensional vector-vector multiplication.

6.3 Parallelization

In this section, we present the parallelization of ALKIA-X. We use the `Python` package [50] to implement the parallelization. In particular, we parallelize the exe-

cution of the dynamic for-loop in Algorithm 10 over the dynamic set of tasks $\mathcal{X}_{\text{tasks}}$. The set of tasks $\mathcal{X}_{\text{tasks}}$ contains the adaptively created sub-domains for which we still have to define the piecewise-defined approximating function.

Next, we explain the parallel execution of the dynamic for-loop in Algorithm 10 for the example illustrated by Figures 6.4 and 6.5. The set of tasks $\mathcal{X}_{\text{tasks}}$ initially only contains the whole domain $\mathcal{X} = [0, 1]^2$. The domain $\mathcal{X}$ corresponds to the root node C in Figure 6.5. As $\overline{M}$ samples per axis were not enough to create sufficiently accurate approximating functions that cover the whole domain, we split the domain into four sub-domains of the same size. These sub-domains correspond to the nodes C1-C4, which are children nodes of the root node C. The sub-domains are added to $\mathcal{X}_{\text{tasks}}$, while the domain of C is removed. Next, ALKIA-X *parallelly* executes the approximation tasks on C1, C2, C3, and C4. Further, C1-C4 are removed from $\mathcal{X}_{\text{tasks}}$, while C11-C14 and C41-C44 are added and parallelly approximated. Finally, the last parallelly executed task tackles the approximation with $\mathcal{X}_{\text{tasks}}$ containing C141-C144. In conclusion, ALKIA-X parallelly executes the nodes contained in $\mathcal{X}_{\text{tasks}}$ on every depth-level of the tree (cf. Figure 6.5).

Our parallel implementation runs fully on a Linux server. Furthermore, we can freely choose how many cores are used for the parallel execution. For our experiments in Chapter 7, we used a 32GB Linux server and used 16 cores for the parallel execution. A heuristic that we employed for our experiments (cf. Chapter 7) was to a priori split the domain corresponding to node C into the 16 sub-domains, such that the resulting tree has 16 nodes in its first depth-level. This enables assigning a sub-domain to every core at the beginning of the approximation procedure, which enhances the parallelization.

6.4 Application to MPC

In this section, we present the application of ALKIA-X to approximate an MPC. Moreover, in Chapter 7, we apply ALKIA-X to approximate a two-dimensional nonlinear robust MPC benchmark. The MPC is implemented in `Matlab` and formulated using CasADi [51]. The underlying NLP of the MPC is solved using IPOPT [52]. To receive function evaluations of the MPC during the offline learning of ALKIA-X, we use the `Matlab`-engine interface.

ALKIA-X requires the domain $\mathcal{X}$ to be a cube (cf. Chapter 2). Therefore, we

need solutions of the underlying NLP of the MPC on the whole cube $\mathcal{X}$. However, evaluations of the MPC are only defined on the feasible set $\mathcal{X}_{N_f} \subseteq \mathcal{X}$. To also define function evaluations of the MPC for infeasible states on $\mathcal{X}$, we incorporate soft constraints and slack variables into the problem formulation of the MPC. The slack variables have a large penalty in the objective function of the NLP. Then, under suitable conditions, this exactly returns the solution of the original MPC for any feasible state $x \in \mathcal{X}_{N_f}$ (cf. [53]), while also providing a return for infeasible states. Moreover, for simplicity's sake, we assume that the solver always returns a unique minimizer. In practice, however, also local minima can occur.

Although we receive function evaluations for the whole domain $\mathcal{X}$ during the offline learning, we do not require approximating the MPC on infeasible parts of the domain. ALKIA-X deals with infeasible states and resulting infeasible areas of the domain during the offline training. When receiving function evaluations of the MPC, ALKIA-X further receives the information on whether the state is feasible or whether a solution with a slack variable greater than zero is returned. If a sub-domain only contains infeasible states, ALKIA-X defines that sub-domain as infeasible. Moreover, we prune the node corresponding to that sub-domain and thus do not continue approximating it. Therefore, ALKIA-X differentiates between feasible and infeasible solutions of the MPC and determines its own feasible domain $\mathcal{X}_{N_h}$ for which it returns an approximation value with the desired accuracy (cf. Chapter 7). The feasible domain of the approximating function should contain the feasible domain of the ground truth to ensure that the approximating function is well-defined for all feasible states $x \in \mathcal{X}_{N_f}$.

7 Numerical Experiment

In this chapter, we investigate numerical experiments of approximating a nonlinear robust MPC with ALKIA-X. In Section 7.1, we first look into the ground truth MPC before investigating the resulting approximation. Furthermore, we investigate different values for $\bar{\epsilon}$ to draw conclusions on the scaling with changing accuracy. In Section 7.2, we compare our results with the results of a current state-of-the-art nonlinear MPC approximation presented in [12] and discuss both approaches.

All experiments were performed on a Linux server with 32GB RAM and 16 cores. The offline learning and the online evaluation with ALKIA-X are implemented in `Python` (cf. Section 6.2). We use the `Matlab`-engine interface in `Python` to receive function evaluations of the MPC during the offline learning (cf. Section 6.4). We performed all numerical experiments with the Ma32-kernel a length scale parameter of $C_\ell = 0.8$. Furthermore, we set $\underline{M} = 5$ and $\overline{M} = 33$ (cf. Section 6.1).

The evaluations in this chapter were obtained by using $9 \cdot 10^4$ uniform samples on the domain $\mathcal{X} = [-0.2, 0.2]^2$. We define these samples as the evaluation set and denote it by X_e.

7.1 Results

Before investigating the resulting approximating function, we first present the ground truth MPC. The two-dimensional MPC benchmark was also used for the numerical experiment in [12]. We use this benchmark to conduct all numerical experiments presented in this chapter.

Figure 7.1 shows the ground truth MPC $f(x)$ over the domain $\mathcal{X} = [-0.2, 0.2]^2$, which we aim to approximate. The domain $\mathcal{X} = [-0.2, 0.2]^2$ also contains infeasible states whose function values were obtained by the soft-constrained MPC formulation with slack variables (cf. Section 6.4). In Figure 7.2, the dashed line corresponds to the domain $\mathcal{X}$. The feasible domain $\mathcal{X}_{N_f} \subseteq \mathcal{X}$ is illustrated by the gray-filled area in

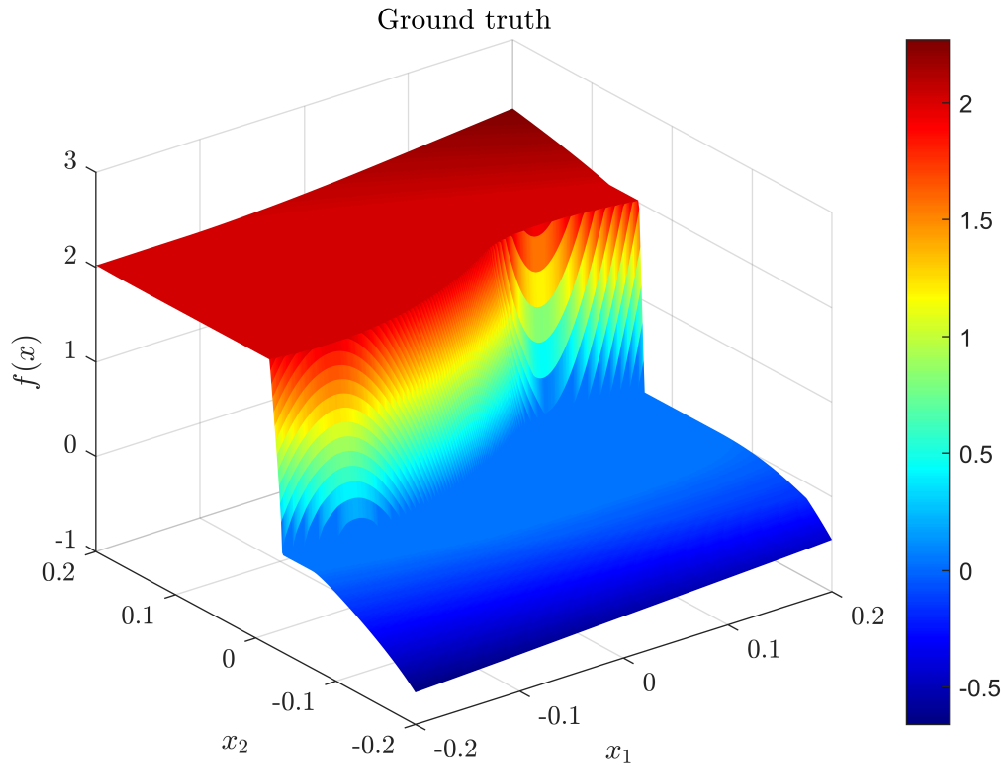

Figure 7.1: Ground truth MPC $f(x)$ over $\mathcal{X} = [-0.2, 0.2]^2$ with soft constraints (cf. Section 6.4).

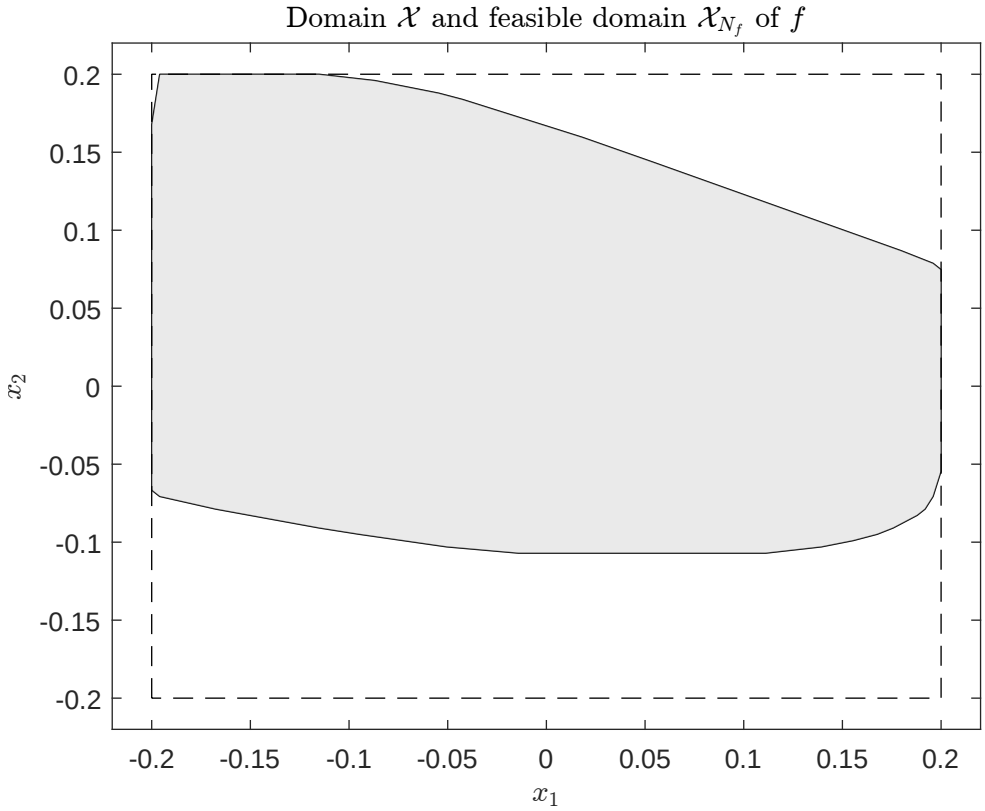

Figure 7.2: Feasible domain $\mathcal{X}_{N_f}$ (gray-filled area) of ground truth MPC $f(x)$ and input domain $\mathcal{X} = [-0.2, 0.2]^2$ (dashed lines).

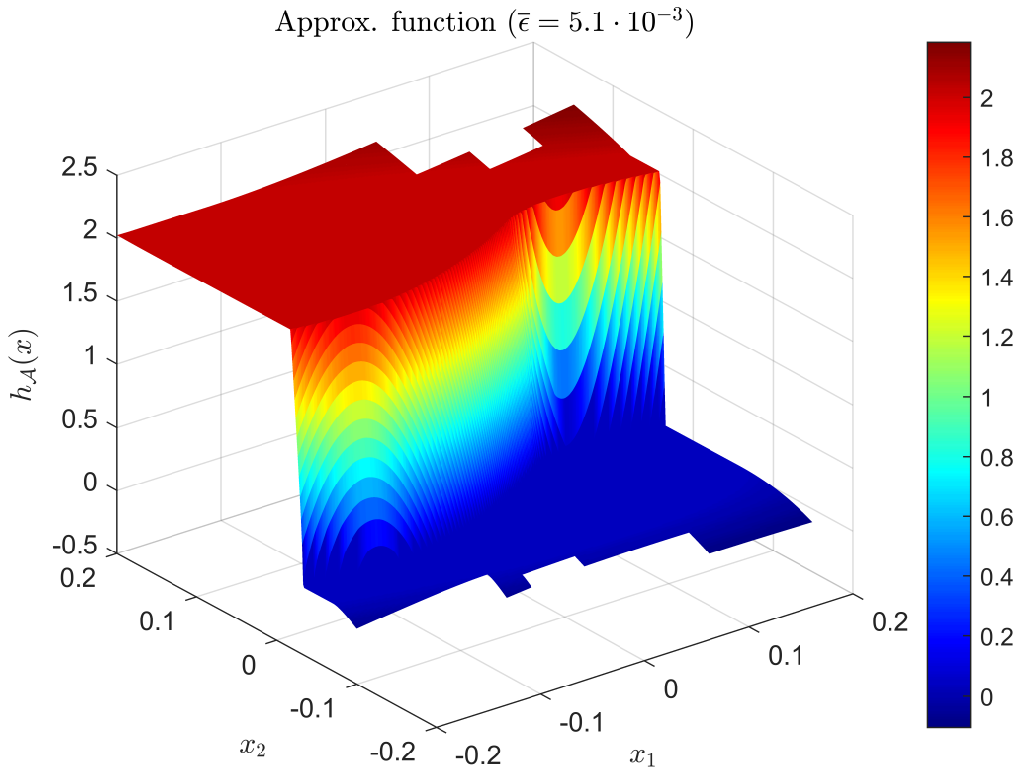

Figure 7.3: Resulting approximation function of ALKIA-X with a maximum uniform approximation error of $\bar{\epsilon} = 5.1 \cdot 10^{-3}$.

Figure 7.2 and is not known a priori. Nevertheless, we deal with infeasible states and the resulting feasible domain of the approximating function $\mathcal{X}_{N_h}$ during the offline learning (cf. Section 6.4).

In the following, we denote the resulting piecewise-defined approximating function by $h_{\mathcal{A}}(x)$. Figure 7.3 depicts the approximating function $h_{\mathcal{A}}(x)$ that results after the offline learning of ALKIA-X with a maximum uniform approximation error of $\bar{\epsilon} = 5.1 \cdot 10^{-3}$. During the offline learning, ALKIA-X adaptively partitions the whole input domain into sub-domains, which are all cubes. Sub-domains that only contain infeasible states are pruned during the offline learning (cf. Section 6.4). In Figure 7.3, the approximating function is only plotted on the feasible domain of the approximating function $\mathcal{X}_{N_h}$, where the approximation error of $\bar{\epsilon}$ is enforced. Since

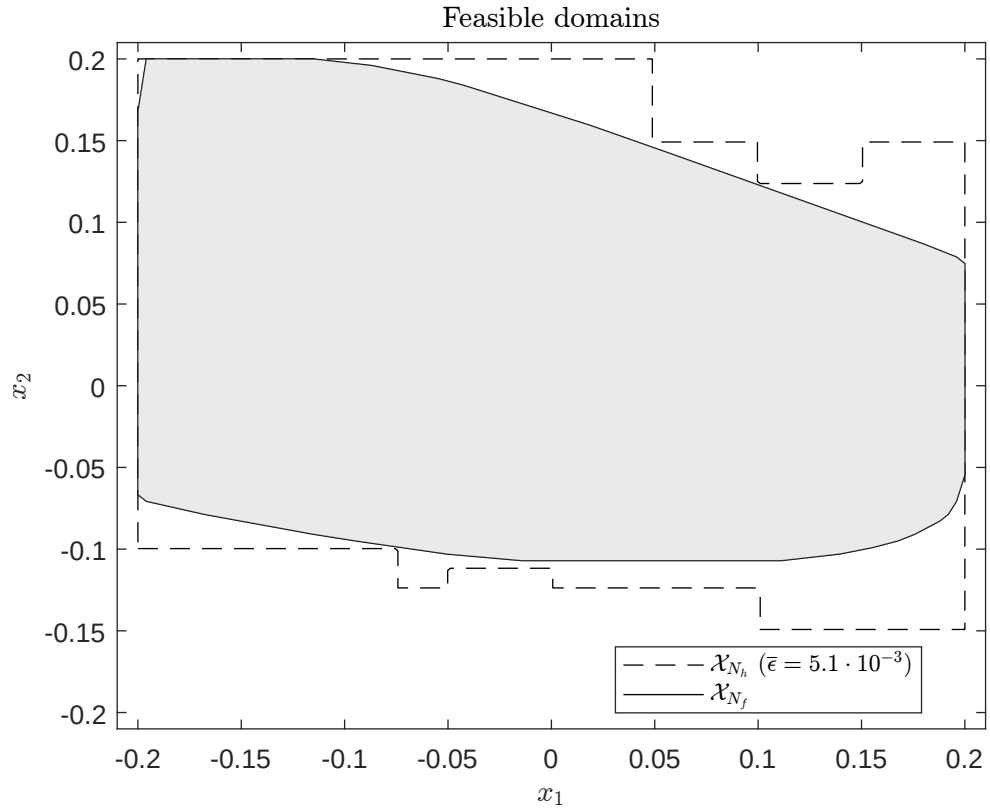

Figure 7.4: Feasible domains of the ground truth $\mathcal{X}_{N_f}$ (gray-filled area) and the resulting feasible domain of the approximating function $\mathcal{X}_{N_h}$ (dashed lines) with $\bar{\epsilon} = 5.1 \cdot 10^{-3}$.

ALKIA-X only declares entire sub-domains as infeasible, the feasible and infeasible domains in Figure 7.3 are also cubes.

The shape of the feasible domain of the resulting approximation is further clarified in Figure 7.4, where the gray-filled area represents the feasible domain of the ground truth $\mathcal{X}_{N_f}$ (cf. Figure 7.2) and the angular-shaped dashed lines correspond to the feasible domain of the approximating function $\mathcal{X}_{N_h}$. The feasible domain of the approximating function contains the feasible domain of the ground truth, which ensures that the approximating function $h_{\mathcal{A}}(x)$ is well-defined for all feasible states $x \in \mathcal{X}_{N_f}$.

Figure 7.5 portrays the approximation error $|f(x) - h_{\mathcal{A}}(x)|$ evaluated at $x \in X_e$. The maximum of the discretely evaluated approximation error is $2 \cdot 10^{-3}$, which is

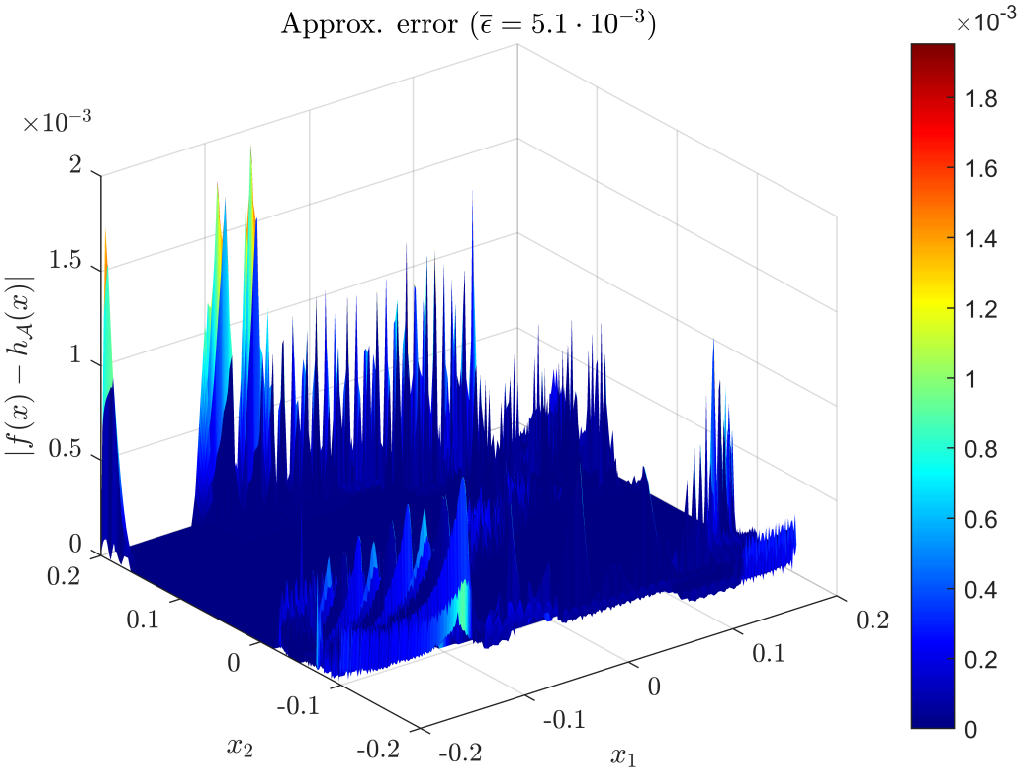

Figure 7.5: Resulting approximation error of the approximating function with $\overline{\epsilon} = 5.1 \cdot 10^{-3}$.

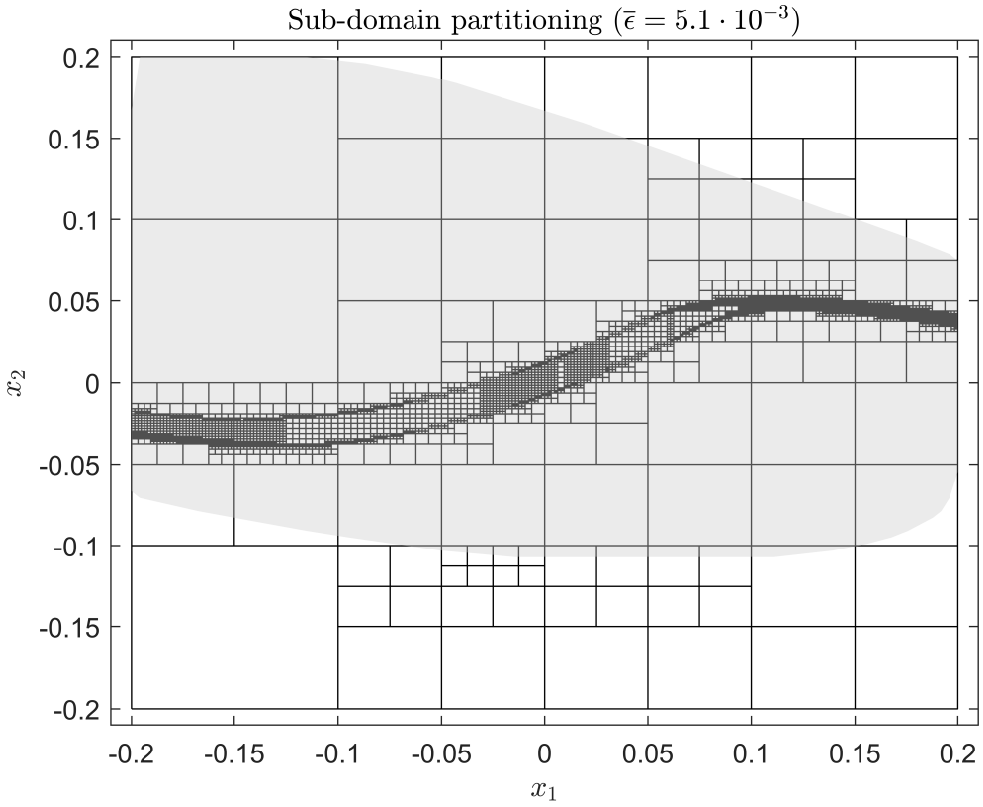

Figure 7.6: Sub-domain partitioning of the approximating function with $\bar{\epsilon} = 5.1 \cdot 10^{-3}$. The boundaries of the sub-domains are illustrated by the black squares. The feasible domain of the ground truth is represented by the gray-filled area.

smaller than the allowed approximation error $\bar{\epsilon} = 5.1 \cdot 10^{-3}$. Furthermore, we define the conservatism factor as $\bar{\epsilon}$ divided by the maximum approximation error in X_e. Thus, a conservatism factor of smaller than one would mean that we did not achieve the desired approximation accuracy. In contrast, a conservatism factor of greater than one indicates that the used bounds are rather conservative. The approximation with $\bar{\epsilon} = 5.1 \cdot 10^{-3}$ is endowed with a conservatism factor of 2.61.

Moreover, Figure 7.6 shows the sub-domain partitioning of the approximating function and the feasible domain of the ground truth $\mathcal{X}_{N_f}$, which is colored in gray. For this experiment, we a priori divide the input domain $\mathcal{X}$ into 16 sub-domains

to simplify parallelization (cf. Section 6.3). After that, ALKIA-X adaptively continues with the sub-domain partitioning during the offline training. For instance, the lower left sub-domain $\mathcal{X}_a = [-0.2, -0.1]^2$ only contains infeasible states. Hence, it is an infeasible sub-domain. Thus, ALKIA-X prunes the corresponding node, does not partition the sub-domain any further, and stops the offline learning on that sub-domain. Another example of a sub-domain that is not partitioned is $\mathcal{X}_a = [-0.2, -0.1] \times [0.1, 0.2]$ on the upper left corner of Figure 7.6. Although this sub-domain almost exclusively contains feasible states, the ground truth on that sub-domain approximately resembles a constant function (cf. Figure 7.1). For this reason, this sub-domain is trivial to approximate. Therefore, no partitioning is necessary to guarantee the desired approximation error on that sub-domain. In contrast, sub-domains in the neighborhood of $x = (0.12, 0.04)$ are quite small, i.e., they have been partitioned multiple times. The area around these sub-domains corresponds to a rapidly changing value of the ground truth (cf. Figure 7.1), which is harder to approximate than regions with a constant function value. This property is also reflected by Theorem 4. Since regions with non-constant function values are endowed with a bigger local Lipschitz-constant, the sample complexity $N = \mathcal{O}\left(\left(\frac{L}{\bar{\epsilon}}\right)^n\right)$ is also higher compared to regions with relatively constant function values.

ALKIA-X samples the MPC during the offline learning. To obtain the piecewise-defined approximating function that ensures a uniform approximation error smaller or equal to $\bar{\epsilon} = 5.1 \cdot 10^{-3}$, the MPC was evaluated at $N = 3.15 \cdot 10^6$ samples. The offline learning procedure took 8.46 hours with the implemented parallelization on 16 cores. For the online evaluation, we only perform an n-ary tree search and evaluate the already created approximating function. Since we sample uniformly, the tree search is trivial. For a two-dimensional benchmark, evaluating the approximating function results in constructing the four-dimensional covariance vector and evaluating a four-dimensional vector-vector multiplication (cf. Section 6.2). Thus, the localized approximation approach leads to a fast online evaluation time of about $3 \cdot 10^{-5} s$ per evaluation. All corresponding values for the numerical experiment with $\bar{\epsilon} = 5.1 \cdot 10^{-3}$ are summarized in Table 7.3.

To investigate how the approximation scales, we conducted experiments with a maximum error of $\bar{\epsilon}_1 = 5.1 \cdot 10^{-1}$, $\bar{\epsilon}_2 = 10^{-1}$, $\bar{\epsilon}_3 = 5.1 \cdot 10^{-2}$, $\bar{\epsilon}_4 = 10^{-2}$, and the already discussed case of $\bar{\epsilon}_5 = 5.1 \cdot 10^{-3}$.

Figure 7.7 shows the number of samples N needed until Algorithm 10 terminates

Quantity	Value
$\bar{\epsilon}$	$5.1 \cdot 10^{-3}$
$\mathcal{X}$	$[-0.2, 0.2]^2$
N	$3.15 \cdot 10^6$
t_{offline}	$8.46h$
t_{online}	$3 \cdot 10^{-5}s$
Conservatism	2.61

Table 7.1: Summarized values for the approximation with $\bar{\epsilon} = 5.1 \cdot 10^{-3}$.

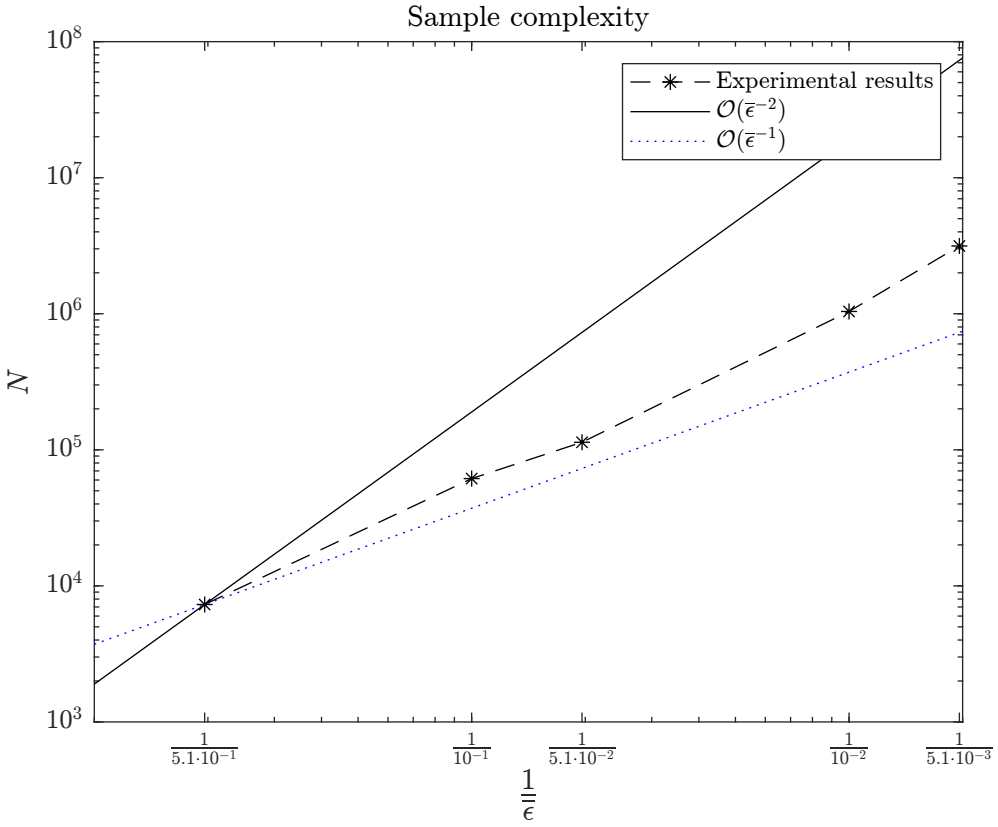

Figure 7.7: Sample complexity in a log-log plot. The worst-case sample complexity (cf. Theorem 4) is shown by the black solid line. The blue dotted line illustrates the sample complexity corresponding to $\mathcal{O}\left(\epsilon^{-1}\right)$. The required samples N for the experiments with the respective maximum approximation error are illustrated by the star points, which are connected through the black dashed line.

$\overline{\epsilon}$	Conservatism	N
$5.1 \cdot 10^{-1}$	1.65	$7.30 \cdot 10^{3}$
10^{-1}	2.16	$6.15 \cdot 10^{4}$
$5.1 \cdot 10^{-2}$	3.14	$1.14 \cdot 10^{5}$
10^{-2}	2.11	$1.04 \cdot 10^{6}$
$5.1 \cdot 10^{-3}$	2.61	$3.15 \cdot 10^{6}$

Table 7.2: Conservatism and total samples N different values of $\overline{\epsilon}$.

given the respective maximum error. In Theorem 4, we developed the worst-case sample complexity, which resulted in $N = \mathcal{O}\left(\left(\frac{L}{\overline{\epsilon}}\right)^{n}\right)$. Since we investigate the same ground truth, L is identical for every experiment and can thus be ignored when plotting the derived sample complexity to investigate the number of samples for changing values of $\overline{\epsilon}$. Furthermore, $n = 2$, which yields $N = \mathcal{O}\left(\overline{\epsilon}^{-2}\right)$, represented by the solid black line in Figure 7.7. Moreover, the sample complexity $\mathcal{O}\left(\overline{\epsilon}^{-1}\right)$ is shown by the dotted blue line. The number of samples needed for the experiments is illustrated by the star points, which are connected by the dashed black line. It can be seen that the underlying slope of the actual sample complexity is smaller than the derived slope of the worst-case sample complexity in Theorem 4. Since the actual sample complexity is between the black solid line and the blue dotted line, we can conclude that the number of samples scales with $N = \mathcal{O}(\overline{\epsilon}^{-r}),\ r \in (1, 2)$.

The online evaluation time was approximately constant for all experiments, i.e., $3 \cdot 10^{-5} s$ per evaluation. The depth of the tree and therefore the complexity of the n-ary tree search depends on the number of samples N. However, evaluating the local approximating functions always results in evaluating a four-by-four vector-vector multiplication. This shows that the bottleneck of the online evaluation is not the n-ary tree search, but the actual evaluation of the approximating function, which is still considerably fast (cf. Section 7.2) because of the localized approximation setting.

Figure 7.8 shows the conservatism factor for the five experiments. All five experiments have a conservatism factor between 1.5 and 3.5. Furthermore, there is no indication of an increase in conservatism with decreasing maximum approximation error $\overline{\epsilon}$. The corresponding values of the conservatism factor and the required number of samples for different values of $\overline{\epsilon}$ are summarized in Table 7.2.

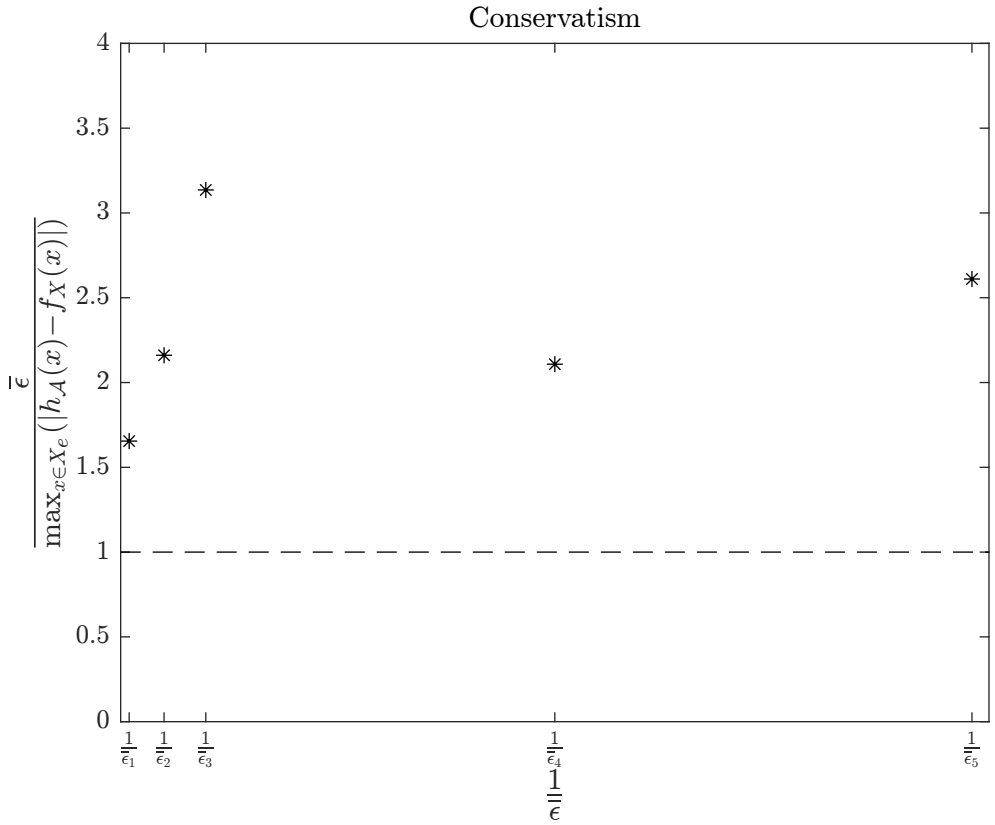

Figure 7.8: Conservatism factor of ALKIA-X for $\bar{\epsilon}_1 = 5.1 \cdot 10^{-1}$, $\bar{\epsilon}_2 = 10^{-1}$, $\bar{\epsilon}_3 = 5.1 \cdot 10^{-2}$, $\bar{\epsilon}_4 = 10^{-2}$, and $\bar{\epsilon}_5 = 5.1 \cdot 10^{-3}$.

7.2 Discussion

After investigating the resulting approximation of ALKIA-X with a maximum approximation error of $\bar{\epsilon} = 5.1 \cdot 10^{-3}$ and examining the development for different maximum errors in Section 7.1, we now compare our results with the results obtained by the NN approximation framework presented in [12]. Reference [12] also tackles approximating the same ground truth (cf. Figure 7.1) on $\mathcal{X} = [-0.2, 0.2]^2$ with a maximum approximation error of $\bar{\epsilon} = 5.1 \cdot 10^{-3}$. The values comparing the NN approach in [12] and ALKIA-X are shown in Table 7.3.

We needed $N = 3.15 \cdot 10^6$ samples to construct the approximating function with the desired accuracy. Since ALKIA-X employs a kernel-based method, we do not require an additional validation step and complete the offline procedure after $N = 3.15 \cdot 10^6$ samples. In contrast, [12] only required $N = 1.6 \cdot 10^6$ samples for the offline learning. However, learning and validation are separated when approximating the ground truth with NNs. Thus, [12] required additional samples to execute the validation step. Therefore, [12] sampled the ground truth in total significantly more than $N = 1.6 \cdot 10^6$ times.

Additionally, ALKIA-X is completely parallelizable (cf. Section 6.3). Our localized approach with adaptive sub-domains not only enables sampling the ground truth in parallel but learning the approximating functions independently. Although sampling the ground truth could have also been parallelized in [12], building the approximating function with the proposed NN is inherently a global task. The advantages of the combined learning and validation and the complete parallelization of ALKIA-X is underlined by the total offline learning time t_{offline} of under nine hours, as opposed to $t_{\text{offline}} = 500h$ in [12]. Moreover, in contrast to ALKIA-X, NNs require nontrivial hyperparameter optimizations by hand regarding their architecture.

The localized approach not only benefits the offline learning time but also the online evaluation time. The online evaluation with ALKIA-X is 100 times faster than evaluating the NN of [12]. Although the online evaluation in [12] could be improved and accelerated up to less than 10^{-3} seconds (cf. [13]), the speed-up when using ALKIA-X is still significant.

Quantity	ALKIA-X	NN [12]
$\overline{\epsilon}$	$5.1 \cdot 10^{-3}$	$5.1 \cdot 10^{-3}$
$\mathcal{X}$	$[-0.2, 0.2]^2$	$[-0.2, 0.2]^2$
N	$3.15 \cdot 10^6$	$1.6 \cdot 10^6$
t_{offline}	$8.46h$	$500h$
t_{online}	$3 \cdot 10^{-5}s$	$3 \cdot 10^{-3}s$

Table 7.3: Comparing the approximation results of ALKIA-X to the NN approach in [12].

8 Summary and Outlook

The goal of this thesis was to automatically determine an explicit function that approximates a nonlinear robust MPC with guarantees. Since the MPC is robust w.r.t input disturbances, it sufficed to guarantee bounds on the uniform approximation error to preserve the control theoretic properties induced by the robust MPC for the resulting approximating function (cf. Section 1.2). The robust MPC acts as a black-box function and we used kernel interpolation (cf. Chapter 2) to approximate it. We developed approximating functions on local cubes, which led to a piecewise-defined approximation setting (cf. Chapter 3). Furthermore, the restriction on local cubes allowed deriving guaranteed bounds on the approximation error. Moreover, the localized approach resulted in a fast-to-evaluate approximating function.

Nevertheless, this localized kernel interpolation approach had two deficiencies: ill-conditioned covariance matrices and the reliance on an oracle that returns an upper bound on the RKHS norm of the ground truth. To ensure well-conditioned covariance matrices, we introduced an adaptive sub-domain partitioning (cf. Chapter 4). Every sub-domain has a bounded number of samples, leading to a uniform bound on the condition number of the covariance matrix. The incorporation of adaptively partitioned sub-domains therefore enables scalability while ensuring numerically reliable computations. To no longer depend on a given upper bound on the RKHS norm of the ground truth, we extrapolated it by using the RKHS norm of the approximating function. In particular, we introduced a novel exponential extrapolation formula to obtain an upper bound on the RKHS norm of the ground truth (cf. Chapter 5). With these extensions, we developed ALKIA-X, our adaptive and localized kernel interpolation algorithm with extrapolated RKHS norm. We provided a worst-case sample complexity bound and ensured that the desired bound on the approximation error holds under suitable conditions.

We implemented ALKIA-X as an n-ary tree in `Python` (cf. Chapter 6). Whenever a sub-domain is partitioned, children nodes are created. Hence, the resulting

localized approximating functions are located at the leaf nodes. We provided an implemented framework of ALKIA-X, which is completely parallelized. In a numerical experiment (cf. Chapter 7), ALKIA-X successfully approximated a two dimensional nonlinear robust MPC benchmark with the desired accuracy. Compared to a current state-of-the-art framework of approximating a nonlinear robust MPC with guarantees (cf. [12]), ALKIA-X needed significantly less time for the offline learning and provided a notably faster online evaluation. Furthermore, in contrast to [12], ALKIA-X runs automatically at the push of a button without requiring any iterative human interaction. Although using ALKIA-X to approximate a robust nonlinear MPC is intriguing, ALKIA-X is a flexible and automated algorithm, capable of approximating a wide range of black-box functions with guarantees.

Future work should consider scalability w.r.t. higher dimensional problems, e.g., by applying ALKIA-X to approximate the robust MPC controller of a real-world robotic system (cf. e.g., [13]). Furthermore, we suggest carefully investigating the suggested RKHS norm extrapolation to examine whether the introduced heuristic (cf. Assumption 9) is a suitable assumption on the ground truth and whether it can be applied to a wide range of RKHS functions.

Bibliography

[1] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017, third printing.

[2] J. Köhler, M. A. Müller, and F. Allgöwer, "A novel constraint tightening approach for nonlinear robust model predictive control," *Annual American Control Conference (ACC)*, pp. 728–734, 2018.

[3] J. Köhler, R. Soloperto, M. A. Müller, and F. Allgöwer, "A computationally efficient robust model predictive control framework for uncertain nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 66, no. 2, pp. 794–801, 2021.

[4] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control: Theory and Algorithms*. Springer, 2017, second edition.

[5] D. Munoz de la Pena, A. Bemporadb, and C. Filippi, "Robust explicit mpc based on approximate multi-parametric convex programming," *43rd IEEE Conference on Decision and Control*, pp. 2491–2496, 2004.

[6] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, pp. 3–20, 2002.

[7] S. Chen, K. Saulnier, N. Atansov, *et al.*, "Approximating explicit model predictive control using constrained neural networks," *Annual American Conference*, 2018.

[8] X. Zhang, M. Bujarbaruah, and F. Borrelli, "Near-optimal rapid mpc using neural networks: A primal-dual policy learning framework," *IEEE Transactions on Control Systems Technology*, no. 5, pp. 2102–2114, 2021.

[9] B. Karg and S. Lucia, "Efficient representation and approximation of model predictive control laws via deep learning," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3866–3878, 2020.

[10] S. W. Chen, T. Wang, N. Atanasov, V. Kumar, and M. Morari, "Large scale model predictive control with neural networks and primal active sets," *Automatica*, vol. 135, 2019.

[11] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[12] M. Hertneck, J. Köhler, S. Trimpe, and F. Allgöwer, "Learning an approximate model predictive controller with guarantees," *IEEE Control Systems Letters*, vol. 2, no. 3, pp. 543–548, 2018.

[13] J. Nubert, J. Köhler, V. Berenz, F. Allgöwer, and S. Trimpe, "Safe and fast tracking on a robot manipulator: Robust MPC and neural network control," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3050–3057, 2020.

[14] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, no. 58, pp. 13–30,

[15] B. Karg, T. Alamo, and S. Lucia, "Probabilistic performance validation of deep learning-based robust nmpc controllers," *International journal of robust and nonlinear control*, vol. 31, no. 18, pp. 8855–8876, 2021.

[16] J.-P. Calliess, "Conservative decision-making and inference in uncertain dynamical systems," Ph.D. dissertation, 2014.

[17] T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel methods in machine learning," *The Annals of Statistics*, vol. 36, no. 3, pp. 1171–1220, 2008.

[18] J.-P. Calliess, S. Roberts, C. Rasmussen, and J. Maciejowski, "Nonlinear set membership regression with adaptive hyper-parameter estimation for online learning and control," *2018 European Control Conference (ECC)*, pp. 1–6, 2018.

[19] E. T. Maddalena and C. Jones, "Learning non-parametric models with guarantees: A smooth lipschitz interpolation approach," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 965–970, 2020, ISSN: 2405-8963.

[20] M. Canale, L. Fagiano, and M. Milanese, "Fast nonlinear model predictive control via set membership approximation: An overview," in *Nonlinear Model Predictive Control: Towards New Challenging Applications*. Springer Berlin Heidelberg, 2009, vol. 12, pp. 461–470.

[21] ——, "Efficient model predictive control for nonlinear systems via function approximation techniques," *IEEE Transactions on Automatic Control*, vol. 55, no. 8, pp. 1911–1916,

[22] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector MIT Press Institute of Technology*. 2002.

[23] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, no. 14, pp. 199–222, 2004.

[24] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*. MIT Press, 2006.

[25] V. Vovk, "Kernel ridge regression," in *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*. Springer Berlin Heidelberg, 2013, pp. 105–116.

[26] E. T. Maddalena, P. Scharnhorst, and C. N. Jones, "Deterministic error bounds for kernel-based learning techniques under bounded noise," 2021, arXiv:2008.04005v3.

[27] G. M. Fasshauer, *Meshfree approximation methods with MATLAB*. World Scientific, 2007, vol. 6.

[28] G. M. Fasshauer and M. J. McCourt, *Kernel-based approximation methods using MATLAB*. World Scientific, 2015, vol. 19.

[29] M. Binder, G. Darivianakis, A. Eichler, and J. Lygeros, "Approximate explicit model predictive controller using gaussian processes," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 841–846.

[30] H. Liu, Y.-S. Ong, X. Shen, and J. Cai, "When gaussian process meets big data: A review of scalable gps," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 11, 2020.

[31] A. Lederer, K. Maier, J. Umlauft, A. J. O. C. Conejo, X. Wenxin, and S. Hirche, "Real-time regression with dividing local gaussian processes," arXiv:2006.09446, 2021.

[32] A. Blaas, J. Manzano, D. Limon, and J.-P. Calliess, "Localized kinky inference," *18th European Control Conference (ECC)*, 2019.

[33] P. I. Frazier, "A tutorial on bayesian optimization," 2018.

[34] N. Srinivas, A. Krause, S. M. Kakde, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," *International Conference on Machine Learning*, no. 27, 2010.

[35] Y. Sui, A. Gotovos, J. Burdick, and A. Krause, "Safe exploration for optimization with gaussian processes," *Proceedings of the 32nd International Conference on Machine Learning*, vol. 37, pp. 997–1005, 2015.

[36] F. Berkenkamp, A. P. Schoellig, and A. Krause, "Safe controller optimization for quadrotors with gaussian processes," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 491–496, 2016.

[37] C. Fiedler, C. W. Scherer, and S. Trimpe, "Practical and rigorous uncertainty bounds for gaussian process regression," *Proc. AAAI Conference on Artificial Intelligence*, vol. 35, no. 8, pp. 7439–7447, 2021.

[38] T. Koller, F. Berkenkamp, M. Turchetta, and A. Krause, "Learning-based model predictive control for safe exploration," *IEEE 57th Conference on Decision and Control (CDC)*, 2018.

[39] J.-P. Calliess, S. Roberts, C. Rasmussen, and J. Maciejowski, "Lazily adapted constant kinky inference for nonparametric regression and model-reference adaptive control," *Automatica*, 2020.

[40] Z. Jin, M. Khajenejad, and S. Z. Yong, "Data-driven model invalidation for unknown lipschitz continuous systems via abstraction," *2020 American Control Conference (ACC)*, pp. 2975–2980, 2020.

[41] M. Kanagawa, P. Henning, D. Sejdinovic, and B. K. Sriperumbudur, "Gaussian processes and kernel methods: A review on connections and equivalences," arXiv:1807.02582, 2018.

[42] H. Wendland, *Scattered Data Approximation*, ser. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2004.

[43] D. K. Duvenaud, "Automatic model construction with gaussian processes," Ph.D. dissertation, University of Cambridge, 2014.

[44] A. Gretton, *Introduction to rkhs, and some simple kernel algorithms*, 2016.

[45] N. Aronszajn, "Theory of reproducing kernels," *Transactions of the American Mathematical Society*, no. 3, pp. 337–404, 1950.

[46] R. Turner, D. Eriksson, M. McCourt, *et al.*, "Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020," in *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, vol. 133, 2021, pp. 3–26.

[47] S. N. Shukla, A. K. Sahu, D. Willmott, and J. Z. Kolter, "Black-box adversarial attacks with bayesian optimization," *CoRR*, 2019, arXiv:1909.13857.

[48] *Any python tree data*, `https://anytree.readthedocs.io/en/latest/`, Accessed: 2022-11-20.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[50] *Concurrent.futures — launching parallel tasks*, `https://docs.python.org/3/library/concurrent.futures.html`, Accessed: 2022-11-20.

[51] J. Andersson, "A general-purpose software framework for dynamic optimization," Ph.D. dissertation, Arenberg Doctoral School, KU Leuven, 2013.

[52] A. Wächter and L. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Math. Program.*, A, pp. 25–57, 2006.

[53] E. Rosenberg, "Exact penalty functions and stability in locally lipschitz programming," *Journal of Optimization Theory and Applications*, vol. 30, pp. 340–356, 1984.