TECHNISCHE UNIVERSITÄT WIEN

AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

# Online function approximation with streaming data for electric drives

## DIPLOMA THESIS

Conducted in partial fulfillment of the requirements for the degree of a

Diplom-Ingenieur (Dipl.-Ing.)

supervised by

Univ.-Prof. Dr. techn. T. Pock
Univ.-Prof. Dr. Ing. W. Kemmetmüller

submitted at the

TU Wien
Faculty of Electrical Engineering and Information Technology
Automation and Control Institute

by
Aaron Raffeiner, BSc
Matriculation number 01227323

Vienna, April 2023

**Complex Dynamical Systems Group**
A-1040 Wien, Gußhausstr. 27–29, Internet: https://www.acin.tuwien.ac.at

# Preamble

The present diploma thesis with the title *Online function approximation with streaming data for electric drives* emerged from a research project at the Complex Dynamic Systems competence unit of the Austrian Institute of Technology (AIT). At this point, I would like to thank Dr. T. Glück for giving me the opportunity to work on this interesting thesis.

Next, I would like to sincerely thank my supervisors Prof. T. Pock and Prof. W. Kemmetmüller for their effort, support and constructive feedback over the whole course of the thesis.

Moreover, I would like to thank Dipl. Ing. M. Gurtner for his effort and many interesting discussions.

Vienna, April 2023

# Abstract

The accurate control of inverter-fed Permanent-Magnet Synchronous-Motors is essential for many industrial applications such as robotics and process automation. In order to improve to control performance, an accurate model of the system nonlinearities is required.

This thesis investigates methods for online static function approximation from streaming data and evaluates them based on typical application scenarios form the electric drive domain. Kernel methods and Bayesian regression methods are investigated in detail since these models can be trained efficiently using convex optimization and allow to incorporate prior knowledge. Furthermore, the Bayesian framework provides a prediction uncertainty and systematic way for model selection. Simulation experiments are performed to evaluate the performance of the online approximators with artificially generated and measurement data. The robustness of the approximators against constant and heteroscedastic noise is tested. Additionally, the performance of the approximators is evaluated with a slowly time varying function.

It is found that the Bayesian-Kernel-Recursive-Least-Squares (B-KRLS) is best suited for the considered scenarios since it obtained the most accurate approximations. Furthermore, the computational complexity of the B-KRLS can be defined a priori and it is able to approximate slowly time varying functions. Additionally, it is robust against heteroscedastic noise and it provides a predictive variance which can be used to evaluate the reliability of the prediction.

# Kurzzusammenfassung

Die genaue Regelung von umrichtergespeisten Permanentmagnet-Synchronmotoren ist für viele industrielle Anwendungen wie Robotik und Prozessautomatisierung unerlässlich. Um die Regelgüte zu verbessern, ist ein genaues Modell der Nichtlinearitäten des Systems erforderlich.

In dieser Arbeit werden Methoden zur Online-Approximation statischer Funktionen auf Basis von Streaming-Daten untersucht und anhand typischer Anwendungsszenarien aus dem Bereich der elektrischen Antriebe evaluiert. Kernel-Methoden und Bayes'sche Regressionsmethoden werden im Detail untersucht, da diese Modelle mit Hilfe von konvexer Optimierung effizient trainiert werden können und es erlauben, Vorwissen miteinzubeziehen. Darüber hinaus bietet der Bayes'sche Ansatz eine Prädiktive-Varianz und einen systematischen Weg zur Modellauswahl. Es werden Simulationsexperimente durchgeführt, um die Performance der Online-Approximatoren mit künstlich erzeugten Daten und Messdaten zu bewerten. Die Robustheit der Approximatoren gegenüber konstantem und heteroskedastischem Rauschen wird getestet. Zusätzlich wird die Performance der Approximatoren mit einer langsam zeitlich variierenden Funktion bewertet.

Es zeigt sich, dass der Bayesian-Kernel-Recursive-Least-Squares (B-KRLS) am besten geeignet ist, da er die genauesten Approximationen für die betrachteten Szenarien liefert. Darüber hinaus kann die Rechenkomplexität des B-KRLS a priori definiert werden und er ist in der Lage, langsam zeitlich variierende Funktionen zu approximieren. Außerdem ist er robust gegenüber heteroskedastischem Rauschen und liefert eine Prädiktive-Variannz, die zur Bewertung der Zuverlässigkeit der Vorhersage verwendet werden kann.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

Inverter-fed electric drives are the standard drive-solution in many industrial applications, such as robotics or process automation. In order to control the drive system in an optimal manner, an accurate mathematical model of the motor and the inverter is needed. However, an accurate description of all relevant physical quantities is time demanding and often even infeasible due to complex motor geometries, parameter variations, unavoidable uncertainties, etc.. Furthermore, for real-time applications and high dynamic control performance, a computational efficient model is required.

The industrial standard to control AC-drives is based on a fundamental wave or $dq$-model [1]. The model assumes a symmetrically constructed motor with an ideal sinusoidal flux distribution in the air gap. Nonlinear effects such as saturation, hysteresis, asymmetries, etc., are neglected. The model captures the general behavior in the magnetic linear operating range with acceptable accuracy. Since the fundamental wave model is linear, well established linear control methods [2], e. g., proportional integral (PI) control, are applicable. For many real use cases, the assumption of an unsaturated, magnetic linear motor is not justified. Thus, the nonlinear relation between the flux-linkage $\boldsymbol{\Psi}_{dq}$ and the currents $\mathbf{i}_{dq}$ in the the $dq$-frame has to be modelled, see, e. g., [3–6]. Assuming that the nonlinear function is known a priori or approximated online during operation, nonlinear control methods, e. g., differential flatness based two-degrees-of-freedom control [7, 8] allow to compensate the flux-linkage nonlinearity. Thus, the accurate approximation of static nonlinearities is crucial for high performance control applications.

When concerning not only the motor, but the whole drive system, the assumption of an ideal inverter is not valid anymore. Nonlinear effects such as dead times, switching losses, parasitic effects, etc., produce errors between the desired and the actual output-voltages and in turn degrade control performance [9]. Thus, in order to improve the control performance of the whole drive system, the non-ideal behavior of the inverter has to be modelled as well.

In the last decades the usage of machine-learning methods for function approximation has gained popularity in the engineering and control domain because accurate approximations of complex nonlinear functions can be obtained from data only. A recent survey of machine-learning methods in the field of electric drives is given in [10–12]. In the electric drive domain, prior knowledge about the function to be modeled is typically available. The incorporation of such knowledge into the modelling process increases the interpretability and reliability of the data-driven model and is therefore considered as particular useful for control applications. However, the quality of the data-based model significantly depends on the quality of the data. Thus, the generation of data is considered one of the most

crucial steps for data-based function approximation [13].

In order to increase the flexibility and adaptability of the model, online function approximation during motor-operation is required. In an online setting, the function is approximated based on a stream of data. This is challenging because the model processes each sample only once and hence it has to extract the contained information as efficient as possible without significantly affecting the real-time performance of the control system. Furthermore, the online approach allows to adapt to changing operation conditions, such as parameter or temperature drift.

## 1.1 Goal of the Thesis

In this thesis, data-driven methods for online nonlinear function approximation shall be investigated and evaluated with a focus on control applications for electric drives. A literature review of state-of-the-art online function approximators serves as the baseline. Thereby, approximators which allow to incorporate prior domain knowledge into the approximation shall be highlighted. To be suitable in a real-time control system, the online approximator must have a bounded computational complexity per time step. The selected approximators shall be compared with respect to their computational complexity and achievable prediction accuracy. Furthermore, methods for model selection, i. e., selecting the parameters of the model, are mandatory. To cope with parameter changes, online approximators that allow to model slowly time varying function shall be investigated as special case.

After the theoretical discussion of suitable online function approximators, they shall be evaluated with streaming data from two typical application scenarios in the electric-drive domain. The first scenario deals with the approximation of the magnetic model of a Permanent-Magnet Synchronous-Motor (PMSM). In order to obtain realistic data, a typical control strategy of PMSMs shall be discussed. Furthermore, the robustness of the approximators against measurement noise shall be investigated and the effect of a changing operating condition shall be analyzed. The second scenario considers the approximation of an average model of a Voltage Source Inverter (VSI). Here, the dataset is obtained from online measurements at certain motor operating points. The selection of an appropriate model which properly interpolates the measurement data shall be discussed and the performance of the online approximators with streaming data shall be investigated.

## 1.2 Outline

The structure of the thesis and the main contents of each chapter are as follows. Chapter 2 provides a brief overview of function approximation algorithms and starts by defining the offline function approximation problem from data. Kernel methods and Gaussian processes are introduced for non-parametric function approximation. Furthermore, the online function approximation setting is defined and a literature overview of kernel methods and Gaussian processes with reduced computational complexity is given. Chapter 3 is

dedicated to frequentist function approximation. First, linear basis function models are reviewed. Moreover, commonly used kernel functions are summarized and hyperparameter-tuning with cross-validation is introduced. Finally, two online kernel methods from the kernel adaptive filter framework are discussed. Bayesian regression methods are the key topic in Chapter 4. First, the linear basis function model is reviewed from a Bayesian perspective. Furthermore, Bayesian hyperparameter-tuning is discussed and two Bayesian online approximators are discussed. The selected online function approximators are evaluated in Chapter 5 based on data from the electric drive domain. In Section 5.1, the approximation of the flux-linkage, an important physical quantity of a PMSM, is investigated. This includes the motivation of an typical application scenario, the derivation of a measurement noise model and the discussion on changing operating conditions. Offline model selection is discussed. The online approximators are then evaluated based on random sampled data and artificially generated operating data. In Section 5.2 an average model of an inverter is approximated based on measurement data. A typical application scenario is motivated and offline model selection is discussed. The section closes by evaluating the online approximators based on sequential measurement data.

# 2 State of the Art Function Approximation

This chapter gives an overview of the field of data-based function approximation. To begin with, the offline function approximation problem is stated with the theory of statistical learning in Section 2.1. Section 2.2 motivates kernel methods for non-parametric regression from the viewpoint of regularization. In Section 2.3, kernel methods are derived from a Bayesian regularization perspective, leading to the Gaussian process framework. The online function approximation setting is defined in Section 2.4. Finally, Section 2.5 gives an overview of scalable kernel methods and Gaussian processes with reduced computational complexity.

## 2.1 Statistical Learning Theory

In this thesis, the approximation of a static multiple-input single-output (MISO) function

$$f(\mathbf{x}) : \mathbb{X} \to \mathbb{Y} \ , \tag{2.1}$$

with an $p$-dimensional input-space $\mathbb{X} \subseteq \mathbb{R}^p$ and an one-dimensional output-space $\mathbb{Y} \subseteq \mathbb{R}$, from data is considered. The assumption of a MISO function is not a strong restriction, since a function with multi-dimensional output $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^q$ can be constructed by combining multiple MISO functions, according to

$$\mathbf{f}^{\mathrm{T}}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) & \cdots & f_q(\mathbf{x}) \end{bmatrix} . \tag{2.2}$$

It is assumed that only noisy observations $y$ of the function $f(\mathbf{x})$ are available. The relationship between the observations $y$ and the inputs $\mathbf{x}$ is described by the statistical model

$$y = f(\mathbf{x}) + \varepsilon \ , \tag{2.3}$$

where $\varepsilon$ is an additive random error [14, Section 2.6]. In this context, $f(\mathbf{x})$ is referred to as *latent* function. The random error $\varepsilon$ serves as a model for measurement noise and potentially unknown inputs which also contribute to $y$.

The function approximation[1] problem from data is stated very generally within the theory of statistical learning, introduced by Vapnik [16]. The learning framework assumes that the *inputs* $\mathbf{x}$ and the *observations* $y$ are related by a probabilistic relationship. This relationship is modelled with a joint probability distribution $p(\mathbf{x}, y)$. The learning problem

[1] The function approximation setting is referred to as regression in the statistics [14] and as supervised learning in machine learning [15]. Note the difference between regression and classification.

is defined as minimization[2] of the expected risk functional $R(f)$

$$f^* = \arg\min_{f \in \mathbb{F}^*} R(f) = \int \int L(y, f(\mathbf{x}))\, p(\mathbf{x}, y)\, \mathrm{d}\mathbf{x}\, \mathrm{d}y \ , \tag{2.4}$$

with respect to functions $f$ of an arbitrary large function space $\mathbb{F}^*$ [16]. The minimizer $f^*$ is further referred to as *target* function and $L(y, f(\mathbf{x}))$ is a loss function which assesses the quality of approximation. For the special case of the squared-error loss function $L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$, the minimizer of (2.4) is given by the *regression function* [17, Section 3.9], according to

$$f^* = \int y\, p(y \mid \mathbf{x})\, \mathrm{d}y \ . \tag{2.5}$$

In most practical cases the joint probability $p(\mathbf{x}, y)$ is unknown and therefore (2.4) cannot be solved exactly. However, the joint probability can be approximated with independent and identical distributed samples $\{(\mathbf{x}_i, y_i) \sim p(\mathbf{x}, y) \mid i = 1, \ldots, N\}$ from $p(\mathbf{x}, y)$. This corresponds to a *frequentist* view of probability, since it interprets probability as the relative frequency of events [18, Chapter 2.1]. In an *offline* or batch setting, all the data is available at once. Thus, (2.4) can be solved approximately by minimizing the average loss, given by the empirical risk functional $R_e(f)$ according to

$$\hat{f} = \arg\min_{f \in \mathbb{F}^*} R_e(f) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i)) \ . \tag{2.6}$$

Statistical learning theory studies under which conditions the *estimate* $\hat{f}$ converges to the target function $f^*$. One of the key statements of the theory is that in order to obtain an estimate $\hat{f}$ that performs well on unseen data, i. e., has small *generalization error*, it is necessary to properly constrain the function space in which (2.6) is minimized [16]. This can be expressed as the minimization of the constrained empirical risk functional, according to

$$\hat{f}_{\mathbb{F}} = \arg\min_{f \in \mathbb{F}} R_e(f) \ , \tag{2.7}$$

where $\mathbb{F} \subset \mathbb{F}^*$ is the constraint function space. However, the restriction to $\mathbb{F}$ imposes a fundamental trade-off between the generalization error and complexity[3] of the approximator. The trade-off can be formalized by expanding the expected generalization error [17, Section 3.14], according to

$$\mathbb{E}\Big[R(\hat{f}_{\mathbb{F}}) - R(f^*)\Big] = \underbrace{\mathbb{E}[R(f^*_{\mathbb{F}}) - R(f^*)]}_{\text{approximation error}} + \underbrace{\mathbb{E}\Big[R(\hat{f}_{\mathbb{F}}) - R(f^*_{\mathbb{F}})\Big]}_{\text{estimation error}} \ . \tag{2.8}$$

Here, $\mathbb{E}[\cdot]$ is the expectation w.r.t. $p(\mathbf{x}, y)$ and $f^*_{\mathbb{F}} \in \mathbb{F}$ is the minimizer of the expected risk functional (2.4) in the space $\mathbb{F}$ [17, Section 3.14]. The *approximation error* is a measure of the deviation between the target function $f^*$ and its approximation $f^*_{\mathbb{F}}$ in a constrained

[2] In this thesis, the notation $\mathbf{x}^* = \arg\min_{\mathbf{x}} J(\mathbf{x}) = f(\mathbf{x})$ is an abbreviation of $\mathbf{x}^* = \arg\min_{\mathbf{x}} f(\mathbf{x})$, where $\mathbf{x}^*$ is the minimizer and $J(\mathbf{x})$ is a loss function which is defined by the function $f(\mathbf{x})$.

[3] The complexity or capacity of a function is defined within the theory of statistical learning, see [16].

function space $\mathbb{F}$. Thus, it quantifies the inability of the function space $\mathbb{F}$ to perfectly represent $f^*$. The *estimation error* is introduced since the empirical risk (2.6) is minimized instead of the expected risk (2.4). A geometric interpretation of the generalization error is depicted in Figure 2.1. The approximation error is independent of the dataset size $N$ and decreases as the complexity of the function space $\mathbb{F}$ increases. However, for a fixed $N$, increasing $\mathbb{F}$ also increases the estimation error [17, Section 3.14]. Thus, for fixed $N$, it is not possible to minimize both error terms simultaneously. The only way to minimize both terms is to increase the dataset size $N$ and the function space $\mathbb{F}$ [17, Section 3.9]. For the special case of the squared-error loss function and linear models, the generalization-complexity trade-off is also know as bias-variance trade-off [17, Section 3.9].

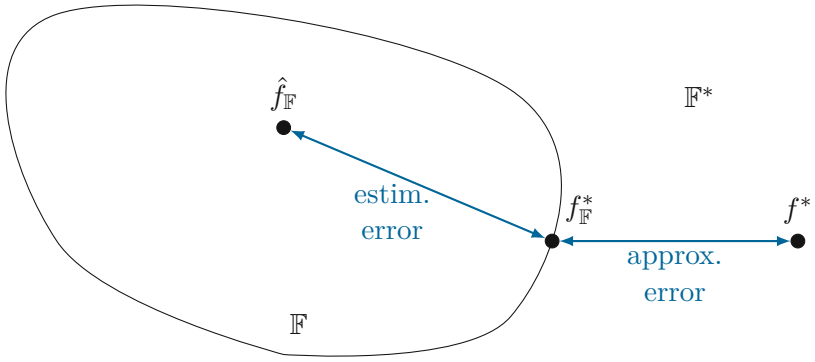



Figure 2.1: Geometric visualization of the generalization error.

A possible approach to constrain the function space is to restrict the function to a parametric model $f(\mathbf{x}) = f(\mathbf{x}; \mathbf{w})$ with parameters $\mathbf{w} \in \mathbb{R}^M$. In this case, the variational problem (2.10) is transformed to a $M$-dimensional static optimization problem for the parameters $\mathbf{w}$, where the optimal parameter vector $\mathbf{w}^*$ is given as

$$\mathbf{w}^* = \arg\min_{\mathbf{w} \in \mathbb{R}^M} R_e(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i; \mathbf{w})) \ . \tag{2.9}$$

In case of a parametric linear basis function model $f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})$ and the squared-error loss function, the global minimizer $\mathbf{w}^*$ can be efficiently calculated with least-squares (for more details see Section 3.1). In the parametric nonlinear case (e. g., Deep Neural Network [19]), the nonlinear optimization problem (2.9) has to be solved with numerical methods, such as stochastic gradient descent [19, Section 5.9]. However, the approximation error, according to (2.8), is large if the target function $f^*$ is not a member of the function space of the parametric model. Another approach to constrain the function space is based on regularization theory [20]. The regularization approach will be discussed in detail in the next section.

## 2.2 Regularization Theory

Regularization is commonly used to constrain the function space in (2.7) and thus to improve the generalization performance of the approximator. Henceforth, the special class of regularizers, which constrain the norm of the function $f$ in the *reproducing kernel Hilbert space* (RKHS) $\mathbb{H}$, are considered. Applying this regularizer to (2.6) leads to the constrained empirical risk functional

$$\hat{f}_{\mathbb{H}} = \arg\min_{f} = \frac{1}{N}\sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i)) \tag{2.10a}$$

$$\text{subject to} \quad \|f\|_{\mathbb{H}}^2 \leq C \ , \tag{2.10b}$$

where $\|f\|_{\mathbb{H}}$ is the norm of the function $f$ in the RKHS and the parameter $C > 0$ controls the complexity of the function space. The RKHS is a Hilbert space of functions which is uniquely defined by a kernel function $k(\mathbf{x}, \mathbf{x}')$. Informally, the kernel function may be interpreted as similarity measure between two inputs $\mathbf{x}, \mathbf{x}' \in \mathbb{X}$ and thus allows to incorporate prior knowledge into the approximation [21, Section 2]. Commonly used kernel function and their properties will be discussed later in Section 3.2. The kernel $k(\mathbf{x}, \mathbf{x}')$ is a symmetric and positive semi-definite function which has the *reproducing property*[4] [22]

$$f(\mathbf{x}) = \langle f \, , \, k(\mathbf{x}, \cdot)\rangle_{\mathbb{H}} \ , \quad \forall f \in \mathbb{H} \ . \tag{2.11}$$

Here, $\langle \cdot, \cdot \rangle_{\mathbb{H}}$ denotes the inner product in $\mathbb{H}$. A kernel function is positive semi-definite if and only if

$$\sum_{i=1}^{N}\sum_{j=1}^{N} c_i c_j k(\mathbf{x}_i, \mathbf{x}'_j) = \mathbf{c}^{\mathrm{T}}\mathbf{K}(\mathcal{X}, \mathcal{X}')\mathbf{c} \geq 0 \ , \tag{2.12}$$

holds for any vector $\mathbf{c} \in \mathbb{R}^N$ and for arbitrary input sets $\mathcal{X} = \{\mathbf{x}_i \mid i = 1, \ldots, a\}$ and $\mathcal{X}' = \{\mathbf{x}'_j \mid j = 1, \ldots, b\}$. The $a \times b$ dimensional matrix $\mathbf{K}(\mathcal{X}, \mathcal{X}')$, with entires

$$\left[\mathbf{K}(\mathcal{X}, \mathcal{X}')\right]_{ij} = k(\mathbf{x}_i, \mathbf{x}'_j) \ , \qquad \text{with} \qquad \mathbf{x}_i \in \mathcal{X} \ , \quad \mathbf{x}'_j \in \mathcal{X}' \ , \tag{2.13}$$

is further referred to as *kernel* or *Gram* matrix. Since the RKHS is a Hilbert space with an inner product, one can compute projections and derive optimization efficient algorithms [23, Chapter 5]. The constraint optimization problem (2.10) can be reformulated in terms of a Lagrange function [24] which leads to the regularized empirical risk functional

$$\hat{f}_{\mathbb{H}} = \arg\min_{f \in \mathbb{H}} \frac{1}{N}\sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i)) + \lambda \|f\|_{\mathbb{H}}^2 \ , \tag{2.14}$$

where $\lambda \geq 0$ is the regularization parameter. The parameter $\lambda$ trades generalization performance against the complexity of the function space $\mathbb{H}$. Note that $\lambda$ should not be optimized using the training data[5], since this leads to poor generalization performance

[4] With the reproducing property the kernel function behaves in $\mathbb{H}$ similar to the delta function $\delta(x)$ in $L_2$ spaces, i. e., $\int f(x)\delta(x-a)\,\mathrm{d}x = f(a)$. The kernel is the representer of evaluation. [20, Section 3].

[5] The parameter $\lambda$ can be estimated for example with cross-validation, which will be discussed in Section 3.2

[18, Section 6.5]. Methods which solve the regularized empirical risk functional (2.14) in the RKHS are hereafter referred to as *kernel methods*.

Kimeldorf and Wahba [22] proved that for functions $f \in \mathbb{H}$ from the RKHS, the minimizer of the variational problem (2.14) lies in a finite-dimensional subspace which is spanned by kernel functions $k(\mathbf{x}, \mathbf{x}'_i)$ evaluated at all training inputs $\mathbf{x}'_i \in \mathcal{X}$. This remarkable result is known as the *Representer Theorem*. It allows to express the minimizer of (2.14) as linear combination of kernel functions

$$\hat{f}_{\mathbb{H}}(\mathbf{x}) = \sum_{i=1}^{N} v_i k(\mathbf{x}, \mathbf{x}'_i) \ , \tag{2.15}$$

with data dependent coefficients $v_i$. For special case of the squared-error loss function, the optimization problem (2.14) can be reformulated using the Representer Theorem (2.15), according to

$$\mathbf{v}^* = \arg\min_{\mathbf{v}} J(\mathbf{v}) = \|\mathbf{y} - \mathbf{K}(\mathcal{X}, \mathcal{X})\mathbf{v}\|_2^2 + \lambda \mathbf{v}^{\mathrm{T}} \mathbf{K}(\mathcal{X}, \mathcal{X})\mathbf{v} \tag{2.16}$$

where $\mathcal{X}$ is a set of training inputs, $\mathbf{y} \in \mathbb{R}^N$ are the corresponding observations and $\mathbf{v} \in \mathbb{R}^N$ are the coefficients. The optimization problem (2.16) is further referred to as *kernel ridge regression*. Since the problem is convex and quadratic in $\mathbf{v}$, the global minimizer $\mathbf{v}^*$ can be obtained by solving the linear equation system[6]

$$(\mathbf{K}(\mathcal{X}, \mathcal{X}) + \lambda \mathbf{E})\mathbf{v} = \mathbf{y} \ , \tag{2.17}$$

where $\mathbf{E}$ is the identity matrix. Thus, computing the minimizer scales with a computational complexity of $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ space [25, Section 2.2]. Note that the regularization parameter $\lambda > 0$ improves the numerical stability when the kernel matrix $\mathbf{K}(\mathcal{X}, \mathcal{X})$ is ill-conditioned.

By virtue of the reproducing property (2.11) it can be shown that any positive semi-definite kernel function defines an inner product in the RKHS [21, Chapter 2]

$$k(\mathbf{x}, \mathbf{x}') = \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}') \rangle_{\mathbb{H}} \ . \tag{2.18}$$

This in known as the *kernel trick*. In this context the function $\boldsymbol{\phi}(\mathbf{x}) = k(\cdot, \mathbf{x}) \in \mathbb{H}$ is referred to as *feature map* and the RKHS $\mathbb{H}$ as *feature space*. The kernel trick implies that every algorithm that is solely based on inner products can be extended to its non-linear version in the feature space by replacing the inner product with a kernel function. For certain kernels, the dimension of the feature space can be high- or even infinite-dimensional. In this cases, the kernel function (2.18) enables to efficiently compute inner products in high-dimensional $\mathbb{H}$ via kernel evaluations in the low dimensional input space $\mathbb{X}$.

[6] Solving a linear equation system is numerically more stable compared to matrix inversion, see [25, Section 2.2].

An introduction to the RKHS and regularization theory is given in [20, 22], whereas a comprehensive introduction to offline kernel methods can be found in [21]. The later derives kernel methods within a functional analytic framework. A mathematically more accessible derivation of kernel methods, based on parametric linear basis functions models, can be found in [15]. A comprehensive introduction how to construct a kernel is given in [23]. Here, also elementary algorithms for computing distances, projections and Cholesky factorization in the RKHS are derived.

## 2.3 Bayesian Regularization

In this section, the function approximation problem is reformulated within a *Bayesian* framework [17, 18]. Although the Bayesian approach is based on a different interpretation of probability [18, Section 12.2], the Bayesian learning problem can also be cast as the minimization of the regularized risk functional (2.14), see [20]. Thus, Bayesian learning is closely related the regularization. However, in contrast to the frequentist approach from Section 2.2, Bayesian learning aims to *infer* the probability density of the latent function $f$ from data [17, Section 3.11]. This is beneficial since from the inferred distribution, a predictive variance, which allows to quantify the uncertainty of the prediction, can be computed. Furthermore, the Bayesian framework provides a systematic way to estimate the hyperparameters, e. g., kernel parameters, from data.

Bayesian inference over functions is performed by introducing a *prior* distribution $p(f)$ over the latent function $f$. The prior reflects the prior knowledge about the function. Thus, the prior can be interpreted as a regularizer which constrains the function space. A popular choice is to use a *Gaussian Process* (GP) prior [25, 26], according to

$$p(f) = \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \ . \tag{2.19}$$

A GP is completely defined by an arbitrary mean[7] function $m(\mathbf{x})$ and a positive semi-definite covariance or kernel function $k(\mathbf{x}, \mathbf{x}')$. A Gaussian process defines a probability distribution over the function $f(\mathbf{x})$, such that any finite set of latent function values $\mathbf{f}^{\mathrm{T}} = \begin{bmatrix} f(\mathbf{x}_1) \dots f(\mathbf{x}_N) \end{bmatrix}$ with $\mathbf{x}_i \in \mathcal{X}$ is a jointly Gaussian distribution [25, Section 2.2].

In order to incorporate the information that a set of inputs $\mathcal{X}$ and corresponding observations $\mathbf{y}$ provides, *Bayes Theorem* is applied to compute the *posterior* distribution $p(\mathbf{f} \mid \mathcal{X}, \mathbf{y})$ [26], according to

$$p(\mathbf{f} \mid \mathbf{y}, \mathcal{X}) = \frac{p(\mathbf{y} \mid \mathbf{f}, \mathcal{X})\, p(\mathbf{f}, \mathcal{X})}{p(\mathbf{y}, \mathcal{X})} = \frac{p(\mathbf{y} \mid \mathbf{f}, \mathcal{X})\, p(\mathbf{f} \mid \mathcal{X})}{p(\mathbf{y} \mid \mathcal{X})} \cdot \frac{p(\mathcal{X})}{p(\mathcal{X})} \ . \tag{2.20}$$

The identity $p(\mathbf{y} \mid \mathbf{f}, \mathcal{X})$ in (2.20) is referred to as *likelihood*. The likelihood expresses the probability of observations $\mathbf{y}$, given the latent function values $\mathbf{f}$ at training inputs $\mathcal{X}$. Thus, the likelihood is model of the random error $\varepsilon$ in (2.3). Furthermore, $p(\mathbf{y} \mid \mathcal{X})$ is the

[7] The mean function $m(\mathbf{x})$ is usually set to zero, since a GP is flexible enough to approximate the mean arbitrarily well, see [18, Section 15.2]

*marginal-likelihood* which can be obtained through marginalization of the latent function values from the numerator in (2.20) according to

$$p(\mathbf{y} \mid \mathcal{X}) = \int p(\mathbf{y} \mid \mathbf{f}, \mathcal{X})\, p(\mathbf{f} \mid \mathcal{X})\, \mathrm{d}\mathbf{f} \ . \tag{2.21}$$

The marginal-likelihood is a constant which ensures that the posterior $p(\mathbf{f} \mid \mathbf{y}, \mathcal{X})$ is a valid probability density and integrates to one. The integral in (2.21) has an analytic solution only in special cases, e. g., Gaussian prior and Gaussian likelihood. Furthermore, the marginal-likelihood can be used for hyperparameter tuning (for more details see Section 4.2).

A prediction[8] $f_* = \hat{f}(\mathbf{x}_*)$ at a test input $\mathbf{x}_*$ is made by evaluating the *predictive distribution* $p(f_* \mid \mathbf{y})$. The distribution is obtained through marginalization of the training latent function values $\mathbf{f}$ from the conditioned joint probability $p(\mathbf{f}, f_* \mid \mathbf{y})$, according to

$$p(f_* \mid \mathbf{y}) = \int p(\mathbf{f}, f_* \mid \mathbf{y})\, \mathrm{d}\mathbf{f} \ . \tag{2.22}$$

The conditioned joint probability can be further decomposed using Bayes Theorem, which yields

$$p(\mathbf{f}, f_* \mid \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{f})}{p(\mathbf{y})}\, p(\mathbf{f}, f_*) = \frac{p(\mathbf{y} \mid \mathbf{f})}{p(\mathbf{y})}\, p(f_* \mid \mathbf{f})\, p(\mathbf{f}) \ . \tag{2.23}$$

The second equality in (2.23) follows by applying the chain rule to the joint probability. From the definition of a GP follows, that the joint probability $p(\mathbf{f}, f_*)$ between the training $\mathbf{f}$ and test $f_*$ latent function values is a *Gaussian distribution*, given by

$$\begin{bmatrix} \mathbf{f} \\ f_* \end{bmatrix} \sim p(\mathbf{f}, f_*) = \mathcal{N}\left(\mathbf{0},\, \begin{bmatrix} \mathbf{K}(\mathcal{X}, \mathcal{X}) & \mathbf{K}(\mathcal{X}, \mathbf{x}_*) \\ \mathbf{K}(\mathbf{x}_*, \mathcal{X}) & k(\mathbf{x}_*, \mathbf{x}_*) \end{bmatrix}\right) \ , \tag{2.24}$$

where $\mathbf{K}(\mathcal{X}, \mathcal{X})$ is defined in (2.13) and $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ refers to a Gaussian with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. In order to obtain a predictive distribution in closed form, the likelihood $p(\mathbf{y} \mid \mathbf{f})$ is assumed to be Gaussian[9], according to

$$p(\mathbf{y} \mid \mathbf{f}) = \mathcal{N}\left(\mathbf{f},\, \sigma_n^2 \mathbf{E}\right) = \frac{1}{\sqrt{2\pi\sigma_n^2}} \exp\left(-\frac{1}{2\sigma_n^2} \|\mathbf{y} - \mathbf{f}\|_2^2\right) \ . \tag{2.25}$$

Finally, a Gaussian process prior and a Gaussian likelihood lead to a predictive distribution $p(f_* \mid \mathbf{y})$, according to (2.22), which is also Gaussian given as

$$p(f_* \mid \mathbf{y}) = \mathcal{N}\left(\hat{f}(\mathbf{x}_*),\, \hat{\sigma}^2(\mathbf{x}_*)\right) \ , \tag{2.26a}$$

$$\hat{f}(\mathbf{x}_*) = \mathbf{K}(\mathbf{x}_*, \mathcal{X})\left(\mathbf{K}(\mathcal{X}, \mathcal{X}) + \sigma_n^2 \mathbf{E}\right)^{-1} \mathbf{y} \ , \tag{2.26b}$$

$$\hat{\sigma}^2(\mathbf{x}_*) = k(\mathbf{x}_*, \mathbf{x}_*) + \sigma_n^2 - \mathbf{K}(\mathbf{x}_*, \mathcal{X})\left(\mathbf{K}(\mathcal{X}, \mathcal{X}) + \sigma_n^2 \mathbf{E}\right)^{-1} \mathbf{K}(\mathcal{X}, \mathbf{x}_*) \ . \tag{2.26c}$$

[8] The notation $\mathbf{f}$ and $f_*$ follows from [25]. The vector $\mathbf{f}$ corresponds to the latent function values at the training inputs $\mathcal{X}$, whereas $f_*$ is the latent function value at the test input $\mathbf{x}_*$. Therefore, the conditioning on the inputs is omitted to lighten the notation.

[9] This corresponds to a zero mean Gaussian noise $p(\varepsilon) = \mathcal{N}\left(0,\, \sigma_n^2\right)$ in (2.3).

Details on frequently applied manipulations for Gaussian distributions are given in the Appendix A.2.3. The prediction of the mean $\hat{f}(\mathbf{x}_*)$ is identical to the solution of the kernel ridge regression (2.17) with $\lambda = \sigma_n^2$. However, the Bayesian derivation additionally provides a predictive variance $\hat{\sigma}^2(\mathbf{x}_*)$. The predictive variance is composed of two terms: $k(\mathbf{x}_*, \mathbf{x}_*) + \sigma_n^2$ is the prior covariance and the positive definite term $\mathbf{K}(\mathbf{x}_*, \mathcal{X})(\mathbf{K}(\mathcal{X}, \mathcal{X}) + \sigma_n^2 \mathbf{E})^{-1}\mathbf{K}(\mathcal{X}, \mathbf{x}_*)$ represents the uncertainty reduction due to observations. Since the matrix $\mathbf{K}(\mathcal{X}, \mathcal{X}) + \lambda \mathbf{E}$ is positive definite, the numerically stable Cholesky decomposition can be used to solve the linear system with a computational complexity of $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ space [25, Section 2.2].

A comprehensive introduction to Gaussian process can be found in [25]. Here, a discussion of commonly used kernel functions and their properties is given and model selection for Gaussian processes is discussed. The connection of GP and regularization theory is explained in detail in [20]. A derivation of Gaussian process from linear basis function models can be found in [15]. Furthermore, the connection between Gaussian Processes and Neural Networks is discussed in [26].

## 2.4 Online Function Approximation

In an online function approximation setting, the latent function is approximated from a stream of potentially infinite input-observation pairs $\{(\mathbf{x}_t, y_t) \mid t = 1, \ldots, N\}$ with $N \to \infty$. With each acquired pair, the estimate $\hat{f}(\mathbf{x})$ has to be updated without the need to store the whole training dataset in memory [17, Section 5.1]. Thus, it is required that an online algorithm has a bounded time and space complexity per update. Motivated by the famous Recursive-Least-Squares (RLS) [17, Section 6.6] algorithm, an online algorithm should have a maximal complexity of $\mathcal{O}(m^2)$ time and space per update, where $m \ll N$ is a constant which is independent of the dataset size $N$. Furthermore, it is assumed that each input-observation pair is only presented once to the algorithm. Thus, algorithms which require multiple iterations of the dataset are not considered as online in this thesis.

In contrast to the offline setting, described in Section 2.1, the online setting allows that the joint distribution from which the data is drawn, may change over time, i. e., $(\mathbf{x}_t, y_t) \sim p_t(\mathbf{x}, y)$ [21, Section 10.6.3]. In general, a time varying distribution $p_t(\mathbf{x}, y)$ can be modeled with a probabilistic state-space model [27, Chapter 3]

$$\mathbf{s}_{t+1} = \mathbf{g}(\mathbf{s}_t) + \boldsymbol{\delta}_t \ , \tag{2.27a}$$

$$y_t = f(\mathbf{x}_t, \mathbf{s}_t) + \varepsilon_t \ , \tag{2.27b}$$

where the function $\mathbf{g}$ defines the dynamics of the states $\mathbf{s}_t$, $f$ is the observation model, $\varepsilon_t$ is the measurement noise and $\boldsymbol{\delta}_t$ the process noise. The goal is to obtain an estimate of the state[10] $\mathbf{s}_t$, given a stream of inputs $\mathbf{x}_t$ and the observations $y_t$. This setting is known in the control theory community as *state estimation* [13, 28]. Assuming the state-space

[10] The states $\mathbf{s}_t$ may represent the time varying coefficients of a linear basis function model $f(\mathbf{x}; \mathbf{s}_t) = \mathbf{s}_t^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})$.

model is known and linear, i. e., $f$ and $\mathbf{g}$ are linear functions of $\mathbf{s}$, and the noise is Gaussian, then the optimal state estimator is the Kalman filter [17, Section 4.10]. For the nonlinear case, only approximate solutions are available [27].

However, the approximation of arbitrary time varying function is out of scope of this thesis. Henceforth, the approximation of static functions, characterized by $\mathbf{s}_t = \mathbf{s}_{t-1}$, is considered. As a special case, drifting functions, i. e., time varying functions with slow dynamics, will be discussed. In order to approximate a static function with an online kernel method, the regularized empirical risk functional (2.14) has to be minimized from a stream of data.

## 2.5 Scalable Kernel and Bayesian Regression

The aim of this section is to give an literature overview of scalable kernel regression and Gaussian processes regression, with a computational complexity smaller than $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ space, where $N$ is the size of the training dataset.

### Kernel Matrix Approximation

As discussed in Section 2.2, training the kernel ridge regression model (2.17) involves the inversion of $N \times N$ matrix. An early attempt to reduce its the computational complexity is based on the following approximation of the kernel matrix

$$\mathbf{K}(\mathcal{X}, \mathcal{X}) \approx \mathbf{K}_{Nm}\mathbf{K}_{mm}\mathbf{K}_{mN} \ , \tag{2.28}$$

where $\mathbf{K}_{Nm} = \mathbf{K}_{mN}^{\mathrm{T}} \in \mathbb{R}^{N\times m}$, $\mathbf{K}_{mm} \in \mathbb{R}^{m\times m}$ and $m \ll N$. With this approximation, the inversion of a $N \times N$ matrix is transformed to the inversion of a $m \times m$ matrix by using the matrix inversion lemma (A.1)

$$(\mathbf{K}_{Nm}\mathbf{K}_{mm}\mathbf{K}_{mN} + \mathbf{A})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{K}_{Nm}\Big(\mathbf{K}_{mm}^{-1} + \mathbf{K}_{mN}\mathbf{A}^{-1}\mathbf{K}_{Nm}\Big)^{-1}\mathbf{K}_{mN}\mathbf{A}^{-1} \ , \tag{2.29}$$

where $\mathbf{A} = \lambda\mathbf{E}$. Thus, the complexity is reduced to $\mathcal{O}(Nm^2)$ time and $\mathcal{O}(Nm)$ space. Williams and Seeger [29] proposed to use the Nyström approximation to compute (2.28). The computation of the Nyström approximation additionally scales with $\mathcal{O}(Nm^2)$ time.

### Sparse Kernel Matrix

In [30] kernel functions with local support are utilized. These kernels lead to a sparse kernel matrix. An algorithm is proposed which solves the Cholesky factorization of the kernel matrix iteratively using Givens rotations. The algorithm scales with $\mathcal{O}(N)$ time and $\mathcal{O}(N^2)$ space complexity.

### Kernel Function Approximation

Another approach to reduce the complexity is to approximate the kernel function by a finite dimensional expansion

$$k(\mathbf{x}, \mathbf{x}') \approx \mathbf{z}^{\mathrm{T}}(\mathbf{x})\mathbf{z}(\mathbf{x}') \ , \tag{2.30}$$

where $\mathbf{z}(\mathbf{x}) \in \mathbb{R}^m$. For translation invariant kernels, i. e., $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{r})$ with $\mathbf{r} = \mathbf{x} - \mathbf{x}'$, the approximation can be obtained with Random Fourier Features [17, Section 11.12]

$$\mathbf{z}^{\mathrm{T}}(\mathbf{x}) = \sqrt{\frac{2}{m}} \left[ \cos\left(\boldsymbol{\omega}_1^{\mathrm{T}} \mathbf{x} + b_1\right) \ \dots \ \cos\left(\boldsymbol{\omega}_m^{\mathrm{T}} \mathbf{x} + b_m\right) \right] , \tag{2.31}$$

where $b_i$ are uniform distributed random variables in the range $[0, 2\pi]$ and $\boldsymbol{\omega}_i$ is sampled from $p(\boldsymbol{\omega}) = \mathbb{F}[k(\mathbf{r})]$, the Fourier transform of a translation invariant kernel. The $m$ coefficients $\mathbf{w}$ of the resulting parametric model

$$\hat{f}(\mathbf{x}) \approx \sum_{i=1}^{N} v_i \mathbf{z}^{\mathrm{T}}(\mathbf{x}_i)\mathbf{z}(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{z}(\mathbf{x}) \tag{2.32}$$

can be calculated with parametric online methods e. g., Recursive Least Squares [17] with $\mathcal{O}(m^2)$ time and $\mathcal{O}(m^2)$ space complexity. This approach fits in the defined online setting if the kernel can be approximated well with $m \ll N$. However, the approach is limited to translation invariant kernels.

### Stochastic Approximation in Function Space

A simple and computationally cheap online kernel method is given in [31]. The algorithm performs stochastic gradient descent in the reproducing kernel Hilbert space (RKHS) space

$$f_{t+1}(\mathbf{x}) = (1 - \eta\lambda) f_t(\mathbf{x}) - \eta \frac{\partial L(f_t, y_t)}{\partial f_t} k(\mathbf{x}_t, \mathbf{x}) \ . \tag{2.33}$$

A forgetting mechanism is implemented by setting the learning rate $\eta < \frac{1}{\lambda}$. This ensure that $(1 - \eta\lambda) < 1$ holds and consequently past samples are given less importance. The algorithm has a complexity of $\mathcal{O}(m)$ time and space. However, the achievable accuracy is poor because the information of past samples is completely forgotten.

### Support Vector Machine

The Support Vector Machine (SVM) [32] obtains a sparse representation by minimizing (2.14) using the $\epsilon$-insensitivity loss function

$$L(y_i, f(\mathbf{x}_i)) = \max(0, |y_i - f(\mathbf{x}_i)| - \varepsilon) \ . \tag{2.34}$$

The loss function ignores all errors which are smaller than $\epsilon > 0$. The resulting optimization problem is usually solved in the dual form by introducing slack variables and Lagrange multipliers [23, Section 7.3]. This results in constraint quadratic optimization problem which scales with $\mathcal{O}(N^3)$ time and $\mathcal{O}(N^2)$ space. The problem may be solved more efficiently with $\mathcal{O}(N^2)$ time and $\mathcal{O}(N)$ space complexity with the Sequential-Minimal-Optimization algorithm, introduced by Platt [33]. As discussed in [34], the problem can also be solved in the primal formulation. Since the $\epsilon$-insensitivity loss function is not continuously differentiable, a stochastic sub-gradient algorithm is proposed in [35] to solve the optimization problem.

### Kernel Adaptive Filters

In the kernel adaptive filters framework [36], popular linear filters, such as the Recursive Least Squares [17, Chapter 5], are extended to their kernel-based variants using the kernel trick (2.18). In order to limit the number of kernel functions in (2.15), sparsification methods are proposed. Sparsification reduces the number of kernel functions in (2.15) by actively selecting an informative subset or *dictionary* of $m \ll N$ inputs $\mathcal{D} = \{\tilde{\mathbf{x}}_i \mid 1, \dots m\}$, according to

$$\hat{f}(\mathbf{x}) \approx \sum_{i=1}^{m} \tilde{v}_i k(\mathbf{x}, \tilde{\mathbf{x}}_i) \quad \text{with} \quad \tilde{\mathbf{x}}_i \in \mathcal{D} \; . \tag{2.35}$$

The dictionary $\mathcal{D}$ is chosen in a way, that the solution of the optimization problem (2.16) can be approximated with arbitrary accuracy. Training and prediction based on this dictionary scales with $\mathcal{O}(m^2)$ and $\mathcal{O}(m)$ complexity, respectively. The dictionary $\mathcal{D}$ can be selected online in a greedy manner, based on a heuristic sparsification criterion [37]. It was argued in [38], that sparsity is a desirable property of an learning algorithm because it improves the generalization ability of the model. An overview of different sparsification criteria together with a theoretical analysis is given in [37]. Many algorithms from the kernel adaptive filter family are implemented in MATLAB in the Kernel-Adaptive-Filter-Toolbox [39].

The Kernel Recursive Least Squares (KRLS), introduced by Engel et al. [38], solves the linear equation system (2.17) with $\lambda = 0$ recursively using the matrix inversion lemma (A.1). The Approximate Linear Dependency (ALD) sparsification criterion was proposed to limit the number of kernel functions. A similar criterion based on Gaussian processes was also derived in [40]. Engel et al. proved that the ALD sparsification criterion performs an computational efficient online approximation of Principal Component Analysis (PCA) in kernel feature space [41]. PCA is a statistical method that can identify a low-dimensional approximation of the data by computing a set of orthogonal directions of highest variance [17, Section 19.3]. The KRLS is derived in a Bayesian framework in [42]. The Bayesian-KRLS is able to approximate drifting functions.

The Kernel Affine Projection algorithm (KAP) algorithm [43] provides a unified view to many kernel adaptive filter algorithms, including the kernel least-mean-squares (KLMS), the sliding-window KRLS and many others, see [44]. The KAP uses the coherence sparsification criterion [43] to limit the number of kernel functions. The algorithm updates its coefficients by a stochastic approximation based on the recent $q$ data points. The algorithm scales with $\mathcal{O}(mq^2)$ time and $\mathcal{O}(m + q)$ space. The KLMS algorithm is a special case of the KAP where only the most recent sample ($q = 1$) is used. The KAP is capable of approximating drifting functions.

### Sparse Gaussian Processes

A framework that unifies many sparse approximations of Gaussian processes was proposed by Quinonero-Candela and Rasmussen [45]. A set of $m \ll N$ inducing variables $\mathbf{f}_u$ is introduced, based on which the full GP is approximated. Thus, the complexity is reduced

to $\mathcal{O}(Nm^2)$ time and $\mathcal{O}(Nm)$ space. The inducing variables correspond to latent function values $f(\mathbf{x})$ evaluated at a set of inducing-inputs $\mathcal{X}_u$. After observing a set of inputs $\mathcal{X}$ and observation $\mathbf{y}$, a prediction $f_*$ at a test input $\mathbf{x}_*$ is made through marginalization of the inducing variables from the joint probability

$$p(f_*, \mathbf{f}) = \int p(f_*, \mathbf{f}, \mathbf{f}_u)\, \mathrm{d}\mathbf{f}_u = \int p(f_*, \mathbf{f} \mid \mathbf{f}_u)\, p(\mathbf{f}_u)\, \mathrm{d}\mathbf{f}_u \ , \tag{2.36}$$

where $p(\mathbf{f}_u) = \mathcal{N}(\mathbf{0}, \mathbf{K}(\mathcal{X}_u, \mathcal{X}_u))$ is the prior of the inducing variables. Sofar, this corresponds to an full GP. A sparse approximation is obtained, by assuming that the joint prior $p(f_*, \mathbf{f} \mid \mathbf{f}_u)$ is conditionally independent given $\mathbf{f}_u$

$$p(f_*, \mathbf{f}) \approx \tilde{p}(f_*, \mathbf{f}) = \int p(f_* \mid \mathbf{f}_u)\, p(\mathbf{f} \mid \mathbf{f}_u)\, p(\mathbf{f}_u)\, \mathrm{d}\mathbf{f}_u \ , \tag{2.37}$$

The various sparse GP approximations differ in how they approximate the conditional distributions $p(f_* \mid \mathbf{f}_u)$ and $p(\mathbf{f} \mid \mathbf{f}_u)$. Examples of spares approximations are the Subset-of-Regressor (SoR), the Deterministic-Training-Conditional (DTC) and Fully-Independent-Training-Conditional (FITC) approximation. Details about these approximations can be found in [45]. Furthermore, the sparse approximations reduce the computational complexity for hyperparameter tuning from $\mathcal{O}(N^3)$ time to $\mathcal{O}(Nm^2)$, see Section 4.2. Many sparse GP approximations are implemented in MATLAB in the GPML-toolbox [46].

As shown in [45], the Nyström approximation (2.28) is not based on a proper probabilistic model and thus may lead to a non-positive predictive variance in the sparse Gaussian process view.

An early approach to obtain an online GPR was proposed by Csato and Opper [40]. The algorithm is closely related to the KRLS from the Kernel Adaptive Filter framework [38]. However, Csato and Opper derived the algorithm within a Bayesian framework by recursive application of Bayes rule. They also suggest to use an variational approach to handle non-Gaussian likelihoods. Additionally, a greedy sparsification criterion to include and delete points from the dictionary is proposes. Thus, the algorithm allows to approximate drifting functions.

An online implementation of the Fully-Independent-Training-Conditional (FITC) sparse GP approximation is derived in [47]. The FITC uses a set of $m$ inducing inputs $\mathcal{X}_u$ and approximates the conditionals in (2.37), according to

$$p(\mathbf{f} \mid \mathbf{f}_u) = \mathcal{N}\Big(\mathbf{K}(\mathcal{X}, \mathcal{X}_u)\mathbf{K}^{-1}(\mathcal{X}_u, \mathcal{X}_u)\mathbf{f}_u,\, \mathrm{diag}(\mathbf{K}(\mathcal{X}, \mathcal{X}) - \mathbf{Q})\Big) \tag{2.38a}$$

$$p(\mathbf{f}_* \mid \mathbf{f}_u) = p(\mathbf{f}_* \mid \mathbf{f}_u) \ , \tag{2.38b}$$

with $\mathbf{Q} = \mathbf{K}(\mathcal{X}, \mathcal{X}_u)\mathbf{K}^{-1}(\mathcal{X}_u, \mathcal{X}_u)\mathbf{K}(\mathcal{X}_u, \mathcal{X})$. An online update rule based on the matrix inversion lemma is derived which allows to update to model with $\mathcal{O}(m^2)$ time and $\mathcal{O}(m^2)$ space complexity.

### Variational Gaussian Processes

A sparse approximation based on a variational formulation was proposed by Titsias [48]. Within the variational framework, the kernel parameters as well as the $m$ inducing inputs $\mathcal{X}_u$ are treated as variational parameters which are obtained by minimizing the Kullback-Leibler divergence (see Section A.2.4) between the sparse approximation and the exact posterior distribution over the latent function values [48]. Furthermore, it is shown that the solution of the variational formulation converges to the full GP as the number of inducing point $m$ increases. The variational problem can also be formulated to handle non-Gaussian likelihoods. Similar to sparse Gaussian processes, the complexity is reduced to $\mathcal{O}(Nm^2)$ time and $\mathcal{O}(Nm)$ space. A framework that unifies many sparse inducing input approximations was proposed by Bui et al. [49].

An algorithm based on stochastic variational inference is derived in [50]. The algorithm scales with $\mathcal{O}(m^3)$ time and $\mathcal{O}(m^2)$ space and enables to tune the hyperparameters and inducing inputs with each new incoming sample. However, for a reasonable approximation, many iterations of the dataset may be required. Thus, this algorithm is not suited for the online setting.

### Kalman Filter

In [51], the latent function is inferred based on an a priori defined set of inputs $\mathcal{D} = \{\mathbf{x}_i \mid i = 1, \ldots, m\}$. The location of the inputs is fixed over time, which allows to derive a Kalman filter [27] update for the predictive mean and variance. The algorithm scales with $\mathcal{O}(m^2)$ time and $\mathcal{O}(m^2)$ space complexity. Additionally, a strategy for online hyperparameter-tuning, based on the unscented Kalman filter [27], is proposed.

# 3 Frequentist Regression

This chapter summarizes the theoretical background for nonlinear function approximation with kernel methods from a frequentist point of view. Kernel-based algorithms are powerful non-parametric modelling tools. These algorithms map the input data to a feature space where the nonlinear function approximation problem can be solved efficiently with linear algorithms and convex optimization techniques. The kernel function defines the inner product in the feature space and thus also provides the tool to perform computations in that space. Moreover, the kernel function allows to incorporate prior knowledge into the approximation and thus leads to interpretable modelling results.

Section 3.1 gives an introduction to linear basis function models and states the function approximation problem. The problem can be solved offline with complexity $\mathcal{O}(M^3)$ where $M$ is the number of basis functions. A discussion about commonly used kernel functions, how to incorporate prior knowledge and hyperparameter tuning is given in Section 3.2. Finally, online kernel algorithms with a bounded computational complexity, independent of the number of training data points $N$, are introduced in Section 3.3.

## 3.1 Linear Basis Function Models

A linear basis function model [15, Chapter 3] is defined as linear combination of $M$ basis functions $\phi_i(\mathbf{x})$, in the form

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{M} w_i \phi_i(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}) \ , \tag{3.1}$$

with coefficients $\mathbf{w}^{\mathrm{T}} = \begin{bmatrix} w_1 \dots w_M \end{bmatrix}$ and basis functions $\boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}} = \begin{bmatrix} \phi_1(\mathbf{x}) \dots \phi_M(\mathbf{x}) \end{bmatrix}$. The function $\boldsymbol{\phi} : \mathbb{X} \to \mathbb{H}$ is a map from the input-space $\mathbb{X}$ to the feature-space $\mathbb{H}$. If the basis functions $\boldsymbol{\phi}(\mathbf{x})$ are nonlinear functions, then the resulting model $\hat{f}(\mathbf{x})$ is a nonlinear function of the input vector $\mathbf{x}$. Commonly used basis functions are e. g., polynomials $\phi_i(x) = x^n$ or the Gaussian basis function $\phi_i(\mathbf{x}) = \exp\left(-\frac{1}{2l^2}\|\mathbf{x} - \boldsymbol{\mu}_i\|^2\right)$ with center $\boldsymbol{\mu}_i$ and lengthscale $l$.

To obtain the coefficients $\mathbf{w}$, the model (3.1) is trained by solving the optimization problem (2.9). The most basic offline optimization formulation finds an optimal $\mathbf{w}^*$ by minimizing the squared-error between the observations $y_i$ and prediction $\hat{f}(\mathbf{x}_i)$ over $N$ training points according to

$$\min_{\mathbf{w}} \frac{1}{2} \sum_{i=1}^{N} \left( y_i - \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_i) \right)^2 \ . \tag{3.2}$$

However, minimizing (3.2) is prune to overfitting the training data. In order to improve the robustness against overfitting, the $L_2$-regularized squared-error loss function $J_\lambda(\mathbf{w})$ (or ridge regression) is minimized according to

$$\min_{\mathbf{w}} J_\lambda(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\Big(y_i - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_i)\Big)^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} \ . \tag{3.3}$$

The regularization term $\frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$ with the regularization parameter $\lambda > 0$ penalizes components of $\mathbf{w}$. Consequently, the coefficients $w_i$ of the basis functions $\phi_i(\mathbf{x})$ which are not supported by data are penalized, thus reducing overfitting and improving generalization performance. Introducing the *design matrix* $\boldsymbol{\Phi} \in \mathbb{R}^{M\times N}$

$$\boldsymbol{\Phi} = \begin{bmatrix}\boldsymbol{\phi}(\mathbf{x}_1) & \dots & \boldsymbol{\phi}(\mathbf{x}_N)\end{bmatrix} \ , \tag{3.4}$$

a set of training inputs $\mathcal{X} = \{\mathbf{x}_i \mid i = 1, \dots, N\}$ with corresponding observations $\mathbf{y}^{\mathrm{T}} = \begin{bmatrix}y_1 \dots y_N\end{bmatrix}$ leads to the compact matrix form

$$\min_{\mathbf{w}} \frac{1}{2}\Big\|\mathbf{y} - \boldsymbol{\Phi}^{\mathrm{T}}\mathbf{w}\Big\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 \ . \tag{3.5}$$

Since the loss function in (3.5) is convex, the necessary first-order optimality condition

$$(\nabla J_\lambda)(\mathbf{w}^*) = \mathbf{0} \tag{3.6}$$

for a local minima is also sufficient for a global minimum [24, Section 4.1]. Hence, taking the gradient of (3.5) with respect to $\mathbf{w}$

$$(\nabla J_\lambda)(\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} - \mathbf{y}^{\mathrm{T}}\boldsymbol{\Phi}^{\mathrm{T}} + \lambda\mathbf{w}^{\mathrm{T}} \ , \tag{3.7}$$

and setting it equal to zero leads to the minimizer

$$\mathbf{w}^* = \Big(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} + \lambda\mathbf{E}\Big)^{-1}\boldsymbol{\Phi}\mathbf{y} \ , \tag{3.8}$$

where $\mathbf{E}$ is a $M \times M$ the identity matrix. From a computational point of view, model training involves the inversion of an $M \times M$ matrix which scales with $\mathcal{O}(M^3)$ time and due to $\boldsymbol{\Phi}$ with $\mathcal{O}(MN)$ space. Predicting with the model (3.1) scales with $\mathcal{O}(M)$ time. For large $N$, maintaining $\boldsymbol{\Phi}$ in memory is often not feasible. In this case, online algorithms with constant computational complexity, independent of the dataset size $N$, can be used.

### Online Learning

A simple online learning algorithm for the model (3.1) is derived by solving the optimization problem iteratively with stochastic-gradient-descent (SGD) [15, Section 3.1.3]. If the loss function is defined as a sum over data points $J_{se}(\mathbf{w}) = \sum_t J_t(\mathbf{w})$, see (3.2), then the SGD algorithm updates the coefficients according to

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta\nabla J_{t-1}(\mathbf{w}_{t-1}) \ , \tag{3.9}$$

where $t$ denotes the time index and $\eta$ is the learning rate. The method is also known as *least-mean-squares* (LMS) and $J_t(\mathbf{w}_{t-1})$ is referred to as instantaneous loss. The algorithm scales with $\mathcal{O}(M)$ time and $\mathcal{O}(M)$ space per time index which is much more efficient compared to the offline solution. The optimization problem (3.2) can also be solved with the *recursive-least-squares* (RLS) algorithm [17]. The RLS solves the matrix inversion in (3.8) iteratively using the *matrix inversion lemma*, see (A.1). This leads to the recursion

$$\mathbf{k}_t = \frac{\mathbf{P}_{t-1}\boldsymbol{\phi}(\mathbf{x}_t)}{1 + \boldsymbol{\phi}(\mathbf{x}_t)^{\mathrm{T}}\mathbf{P}_{t-1}\boldsymbol{\phi}(\mathbf{x}_t)} \tag{3.10a}$$

$$\mathbf{P}_t = \mathbf{P}_{t-1} - \mathbf{k}_t\boldsymbol{\phi}(\mathbf{x}_t)^{\mathrm{T}}\mathbf{P}_{t-1} \tag{3.10b}$$

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \mathbf{k}_t\Big(y_t - \boldsymbol{\phi}(\mathbf{x}_t)^{\mathrm{T}}\mathbf{w}_{t-1}\Big)\ . \tag{3.10c}$$

The algorithm is usually initialized with $\mathbf{w}_0^* = \mathbf{0}$ and $\mathbf{P}_0 = \alpha\mathbf{E}$, with a sufficiently large $\alpha > 0$. The RLS algorithm scales with $\mathcal{O}(M^2)$ time and space per update. In general, the RLS converges much faster than the LMS algorithm [17, Section 6.6]. The RLS is closely related to Newton's method, see [17, Section 6.7].

## 3.2 Model Selection

The kernel function is the essential part in the theory of kernel methods. Since kernel functions define a similarity measure between data points, they allow to encode prior knowledge. A well chosen kernel improves generalization performance and interpretability of the approximation. To begin with, some commonly used kernels for regression and their properties are discussed. This is followed by the construction of more advanced kernels from base kernels. Finally, kernel hyperparameter tuning with cross-validation is described in the end of this section.

### Kernel Functions and Kernel Construction

A kernel functions requires two properties in order to be useful for practical applications [23, Chapter 9]:

- The kernel evaluation should require less computation than the basis function approach from Section 3.1.
- The kernel should be chosen to capture an appropriate measure of similarity for the specific application. If prior knowledge is available, then the kernel should be chosen accordingly.

In the following, commonly used kernels are discussed.

#### Gaussian kernel

The Gaussian kernel

$$k(\mathbf{x}, \mathbf{x}') = \sigma_v^2 \exp\left(-\frac{1}{2l^2}\|\mathbf{x} - \mathbf{x}'\|_2^2\right) \tag{3.11}$$

is one of the most popular kernels. The kernel is shown in Figure 3.1 for different lengthscales $l$. For a too small $l$, all the data looks distinct, i. e., $k(\mathbf{x}, \mathbf{x}') \approx 0$ and for a too large $l$, all the data looks similar, i. e., $k(\mathbf{x}, \mathbf{x}') \approx \sigma_v^2$. Thus, for small $l$ the kernel is more likely to overfit the data, while for large $l$ the data is smoothed. The kernel has universal approximation capability, i. e., any function can be approximated with arbitrarily accuracy given enough data [52]. A Taylor expansion of the Gaussian kernel shows that it is a polynomial kernel of infinite degree [23, Section 3.4.1]

$$\exp(x) = \sum_{n=0}^{\infty} \frac{1}{n!} x^n \ . \tag{3.12}$$

Thus, the kernel has an infinite dimensional feature space, i. e., $\dim \mathbb{H} = \infty$. Furthermore, the kernel is infinitely differentiable, thus resulting in an infinitely differentiable approximation [25, Section 4.2.1]. As discussed in [21, Section 4.2.2], the kernel acts as a regularizer which penalizes derivatives of all orders depending on the lengthscale $l$, and thus enforces a smoothness of the approximation. The parameter $\sigma_v$ scales the magnitude of the kernel. The magnitude becomes important when multiple kernels are combined and when predictive variances in the Bayesian framework are considered.

Instead of the Euclidean norm, any weighted norm with a symmetric and positive-definite weighting matrix $\mathbf{\Sigma}$ can be used

$$k(\mathbf{x}, \mathbf{x}') = \sigma_v^2 \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^{\mathrm{T}} \mathbf{\Sigma} (\mathbf{x} - \mathbf{x}')\right) \ . \tag{3.13}$$

### Rational Quadratic Kernel

The rational quadratic (RQ) kernel, defined as

$$k(\mathbf{x}, \mathbf{x}') = \sigma_v^2 \left(1 + \frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{2\alpha l^2}\right)^{-\alpha} \tag{3.14}$$

is equivalent to summing infinite Gaussian kernels (3.11) with different lengthscales $l_i$ [25, Section 4.2]. Like the Gaussian kernel, the RQ kernel the has universal approximation capability [52]. For $\alpha \to \infty$ the kernel is even equivalent to the Gaussian kernel (3.11). Again, the parameter $l$ defines the lengthscale of the kernel. Figure 3.1 shows the kernel for different values $\alpha$ and $l = 1$.

### Mátern Kernels

Mátern kernels are more complex kernels constructed by using the Gamma function $\Gamma(x)$ and the modified Bessel function [25, Section 4.2]. The kernel is a function of the distance $d = \|\mathbf{x} - \mathbf{x}'\|_2$ and it is equivalent to a product of exponential and polynomial functions, according to

$$k(d)_{\nu = p + \frac{1}{2}} = \sigma_v^2 \exp\left(-\frac{\sqrt{2\nu} d}{l}\right) \frac{\Gamma(p+1)}{\Gamma(2p+1)} \sum_{i=0}^{p} \frac{(p+1)!}{i!(p-1)!} \left(\frac{\sqrt{8\nu} d}{l}\right)^{p-i} \ , \tag{3.15}$$

in the special case with $\nu = p + \frac{1}{2}$. The corresponding function is $p$-times differentiable [25, Section 4.2]. The kernel is depicted in Figure 3.1 for different lengthscale $l$.

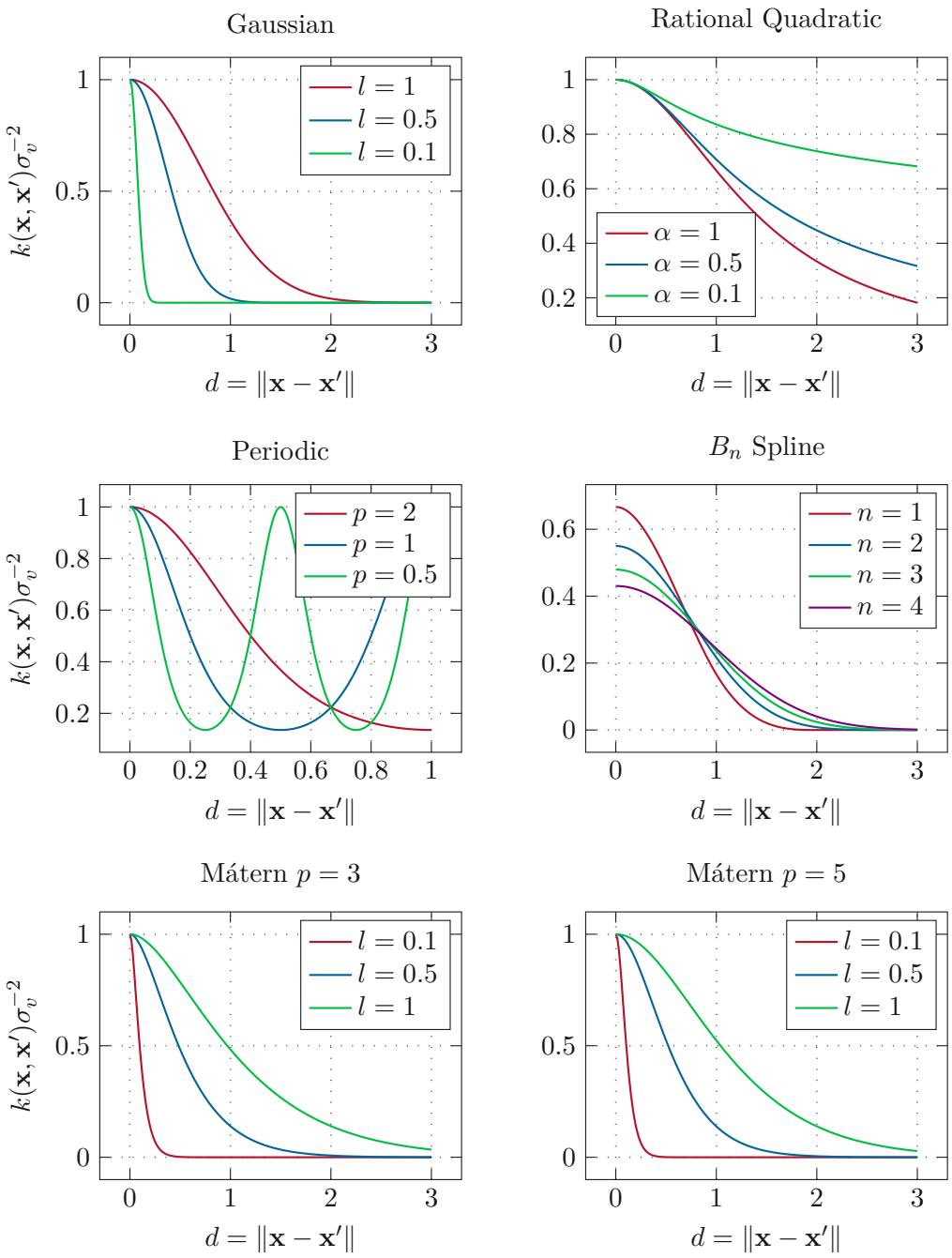



Figure 3.1: Comparison of different kernel functions.

### Piecewise Polynomial Kernels with Compact Support

Piecewise polynomial kernels with compact support (PPKCS) [25, Section 4.2] is a positive definite kernel for inputs $\mathbf{x} \in \mathbb{R}^D$. A PPKCS of order $q = 0$ and $q = 1$ is defined as

$$k(\mathbf{x}, \mathbf{x}')_{D,0} = \sigma_v^2 (1 - d)_+^j \tag{3.16a}$$

$$k(\mathbf{x}, \mathbf{x}')_{D,1} = \sigma_v^2 (1 - d)_+^{j+1} ((j + 1)d + 1) \tag{3.16b}$$

where $d = \|\mathbf{x} - \mathbf{x}'\|_2$ is the distance, $(x)_+ = \max(x, 0)$ and $j = \lfloor \frac{D}{2} \rfloor + q + 1$. The square brackets $\lfloor x \rfloor$ refer to the floor function. Compact support means that the kernel becomes zero if the distance $d$ exceeds a certain threshold, i.e, $d > 1$. The kernel corresponding process is $q$ times mean-square differentiable.

### Periodic Kernels

One possible form of prior knowledge is periodicity. Therefore [26] derives a translation invariant kernel which enforces periodic functions. The kernel is defined as

$$k(\mathbf{x}, \mathbf{x}') = \sigma_v^2 \exp\left( -\frac{2}{l^2} \sin^2\left( \frac{\pi}{p} \|\mathbf{x} - \mathbf{x}'\|_2^2 \right) \right) . \tag{3.17}$$

where $p$ defines the periodicity of the function and $l$ its lengthscale. The influence of the parameter $p$ is depicted in Figure 3.1 for a constant $l = 1$.

### B-Spline Kernels

A one-dimensional $B_n$-spline kernel [32] can be constructed from a $B_n$-spline of order $n$

$$B_n(x) = \sum_{r=0}^{n+1} \frac{(-1)^r}{n!} \binom{n+1}{r} \left( x + \frac{n+1}{2} - r \right)_+^n \tag{3.18}$$

with $(x)_+ = \max(x, 0)$ by solving the integral equation

$$k(x, x') = \sigma_v^2 \int_{-\infty}^{\infty} B_n(x - \zeta) B_n(x' - \zeta) \, \mathrm{d}\zeta = \sigma_v^2 B_{2n+1}(x - x') . \tag{3.19}$$

The kernel is equivalent to a $B$-spline of order $2n + 1$ evaluated for $x - x'$. Again, the $B_n$-spline kernel has compact support. The kernel is shown in Figure 3.1 for different orders $n$.

### Spline Kernels

A one-dimensional spline Kernel [32] of order $d$ with an infinite number of knots is obtained by

$$k(x, x') = \sum_{r=0}^{d} x^r (x')^r + \int_0^{\min(x,x')} (x - t)_+^d (x' - t)_+^d \, \mathrm{d}t . \tag{3.20}$$

The kernel has $\dim \mathbb{H} = \infty$. In the linear case with $d = 1$ the kernel is given by

$$k(x, x') = 1 + xx' + xx' \min(x, x') - \frac{1}{2}(x + x') \min(x, x')^2 + \frac{1}{3} \min(x, x')^3 . \tag{3.21}$$

### Polynomial kernel

A polynomial kernel [21, Section 2.3] of degree $d$ is defined as

$$k(\mathbf{x}, \mathbf{x}') = \sigma_v^2 \Big(\mathbf{x}^{\mathrm{T}} \mathbf{x}' + c\Big)^d \,, \tag{3.22}$$

where $c \geq 0$ and $d \in \mathbb{N}$. With an $p$ dimensional input $\mathbf{x}$, the kernel operates in a feature space with dimension $\dim \mathbb{H} = \binom{d+p-1}{d}$ [21, Section 2.1]. For $d = 5$ and $p = \dim \mathbf{x} = 2$ follows $\dim \mathbb{H} = 15$. Thus, for low dimensional inputs $p$ its computational more efficient to use the linear basis function approach, see Section 3.1, rather than the dual representation.

The computation of the kernel gets more efficient with increasing input dimension $p$. Polynomial kernels are mostly used for image processing since the dimension of the feature space of a $p = 16 \times 16$ pixel image and $d = 5$ is $\dim \mathbb{H} \approx 10^{10}$.

### Constructing Kernels

More advanced kernels can be constructed by combining multiple bases kernels [23, Section 3.4]. If $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$ are two valid kernels, than the following operations lead to a valid kernel $k(\mathbf{x}, \mathbf{x}')$:

$$k(\mathbf{x}, \mathbf{x}') = c\, k_1(\mathbf{x}, \mathbf{x}') \,, \tag{3.23a}$$
$$k(\mathbf{x}, \mathbf{x}') = c_1 k_1(\mathbf{x}, \mathbf{x}') + c_2 k_2(\mathbf{x}, \mathbf{x}') \,, \tag{3.23b}$$
$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')\, k_2(\mathbf{x}, \mathbf{x}') \,, \tag{3.23c}$$
$$k(\mathbf{x}, \mathbf{x}') = g(\mathbf{x})\, g(\mathbf{x}') \,, \tag{3.23d}$$
$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{h}(\mathbf{x}), \mathbf{h}(\mathbf{x}')) \,, \tag{3.23e}$$

where $c, c_1, c_2 \in \mathbb{R}^+$ and $g(\cdot)$, $\mathbf{h}(\cdot)$ are arbitrary real valued functions. An introduction how to combine kernels to model complex functions can be found in [53].

The addition of two kernels $k_1(\mathbf{x}, \mathbf{x}') = \langle \boldsymbol{\phi}_1(\mathbf{x}) \,,\, \boldsymbol{\phi}_1(\mathbf{x}') \rangle$ and $k_2(\mathbf{x}, \mathbf{x}') = \langle \boldsymbol{\phi}_2(\mathbf{x}) \,,\, \boldsymbol{\phi}_2(\mathbf{x}') \rangle$ corresponds to a concatenation of the individual feature vectors according to $\boldsymbol{\phi}^{\mathrm{T}}(\mathbf{x}) = \begin{bmatrix} \boldsymbol{\phi}_1(\mathbf{x}) & \boldsymbol{\phi}_2(\mathbf{x}) \end{bmatrix}$ [23, Section 3.4].

The multiplication of the kernels corresponds to a product of all pairs of features the base kernels according to $\boldsymbol{\phi}(\mathbf{x})_{ij} = \boldsymbol{\phi}_{1,i}(\mathbf{x})\, \boldsymbol{\phi}_{2,j}(\mathbf{x})$ for all $i = 1, \ldots, M_1$ and $j = 1, \ldots, M_2$ where $M_1$ and $M_2$ is the dimension of the corresponding feature spaces. For more details see [23, Section 3.4]. A multi dimensional kernel with e. g., $\mathbf{x}^{\mathrm{T}} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}$ and $(\mathbf{x}')^{\mathrm{T}} = \begin{bmatrix} x_1' & x_2' \end{bmatrix}$ can be constructed from one dimensional kernels according to

$$k(\mathbf{x}, \mathbf{x}') = k(x_1, x_1') k(x_2, x_2') \,. \tag{3.24}$$

### Hyperparameter Tuning

All kernel functions from Section 3.2 have one or multiple tuning parameters, further denoted as $\boldsymbol{\theta}$, which determine the shape and approximation behavior of the kernel. Choosing these hyperparameters is not trivial. In the kernel method framework, the kernel

parameter are usually tuned with $K$-fold cross-validation (CV), which is covered in detail by [36, Chapter 2.3] or [54]. In general, CV is a frequentist approach to estimate the prediction error of a static model, where the root-mean-square (RMS) error

$$\mathrm{RMS}(\mathbf{e})^2 = \frac{1}{n}\sum_{i=1}^{n} e_i^2 \quad \text{with} \quad \mathbf{e} \in \mathbb{R}^n \tag{3.25}$$

is commonly used as performance metric. $K$-fold CV randomly divides the dataset of $n$ inputs $\mathcal{X} = \{\mathbf{x}_i \mid i = 1, \dots, n\}$ with corresponding observations $\mathbf{y}$ into $K$ equal parts of size $n_k$. The model is trained with combined $K-1$ parts, denoted as $\mathcal{X}_{-k}$ and $\mathbf{y}_{-k}$. Then the prediction RMS error based on the remaining part $\mathcal{X}_k$ and $\mathbf{y}_k$ is calculated. The procedure is repeated $K$-times and the RMS prediction errors are averaged, according to

$$\mathrm{CV}(\hat{f}_{\boldsymbol{\theta}}, \mathcal{X}, \mathbf{y}) = \sum_{k=1}^{K} \frac{n_k}{n} \mathrm{RMS}\Big(\mathbf{y}_k - \hat{\mathbf{f}}_{\boldsymbol{\theta}}(\mathcal{X}_k, \mathcal{X}_{-k}, \mathbf{y}_{-k})\Big) \ . \tag{3.26}$$

Henceforth, the notation $\hat{\mathbf{f}}_{\boldsymbol{\theta}}(\mathcal{X}_*, \mathcal{X}, \mathbf{y})$ is used to refer to a model with hyperparameters $\boldsymbol{\theta}$ which was trained with the sets $\mathcal{X}$ and $\mathbf{y}$ and is then evaluated at a set of test inputs $\mathcal{X}_*$, according to

$$\hat{\mathbf{f}}_{\boldsymbol{\theta}}(\mathcal{X}_*, \mathcal{X}, \mathbf{y}) = \Big[\hat{f}(\mathbf{x}_i)\Big] \quad \text{with} \quad \mathbf{x}_i \in \mathcal{X}_* \ , \tag{3.27}$$

where the prediction $\hat{f}(\mathbf{x})$ is defined as in (2.15). A grid-search over the hyperparameters is usually performed to select those hyperparameters, which correspond to the smallest CV score (3.26). When many parameters have to be tuned, the computational complexity of CV with grid-search increases exponentially and thus often hinders the practical usage[1]. In Section 4.2 an alternative Bayesian approach for hyperparameter-tuning will be discussed.

## 3.3 Online Kernel Regression

In this section, two online algorithms from the kernel adaptive filter framework [36] are introduced. In order to prevent the increasing of the number of kernel functions in (2.15), two sparsification methods are given in Section 3.3.1. An online kernel method is then derived by combining a sparsification method with a method to solve the optimization problem (2.16) online. Motivated by the preliminary literature, see Chapter 2, the kernel recursive least squares algorithm is derived in Section 3.3.2 and the kernel affine projection algorithm in Section 3.3.3.

### 3.3.1 Sparsification Methods

Sparsification reduces the number of kernel functions in (2.15) by actively selecting an informative subset $\mathcal{D}_t = \{\tilde{\mathbf{x}}_i \mid i = 1, \dots, m_t\}$, where $m_t = |\mathcal{D}_t|$ is the cardinality of the set. At time step $t$, a training sample $\mathbf{x}_t$ is added to the dictionary $\mathcal{D}_t = \{\mathcal{D}_{t-1} \cup \mathbf{x}_t\}$ if

[1] Suppose $p$ hyperparameters should be tuned using CV where each hyperparameter is discretized into $d$ values. Therefore one has to compute the prediction error with CV $d^p$-times.

and only if a given sparsification criterion is fulfilled. Two common sparsification criteria, namely the Approximate Linear Dependency Criterion and the Coherence Criterion, are discussed next.

#### Approximate Linear Dependency Criterion

The approximate linear dependency (ALD) criterion [38] computes the minimum distance $\delta_t$ between the feature $\boldsymbol{\phi}(\mathbf{x}_t) = k(\cdot, \mathbf{x}_t) \in \mathbb{H}$ and the linear span of the present dictionary, according to

$$\delta_t = \min_{\mathbf{a}} \left\| \sum_{i=1}^{m_{t-1}} a_i \boldsymbol{\phi}(\tilde{\mathbf{x}}_i) - \boldsymbol{\phi}(\mathbf{x}_t) \right\|_2^2 \leq \nu \ . \tag{3.28}$$

The parameter $\nu > 0$ controls the level of sparsity of the solution. If $\delta_t > \nu$ holds, then $\boldsymbol{\phi}(\mathbf{x}_t)$ is not well approximated by the current dictionary and $\mathbf{x}_t$ is added to it. The minimizer $\mathbf{a}_t^*$ of (3.28) is obtained by exploiting the first order optimality condition $(\nabla \delta_t)(\mathbf{a}_t^*) = 0$ and applying the kernel trick (2.18) which leads to

$$\mathbf{a}_t^* = \mathbf{K}^{-1}(\mathcal{D}_{t-1}, \mathcal{D}_{t-1}) \mathbf{K}(\mathcal{D}_{t-1}, \mathbf{x}_t) \ . \tag{3.29}$$

Substituting (3.29) into (3.28) yields to the ALD criterion in terms of kernel functions

$$\delta_t = k(\mathbf{x}_t, \mathbf{x}_t) - \mathbf{K}^{\mathrm{T}}(\mathcal{D}_{t-1}, \mathbf{x}_t) \mathbf{K}^{-1}(\mathcal{D}_{t-1}, \mathcal{D}_{t-1}) \mathbf{K}(\mathcal{D}_{t-1}, \mathbf{x}_t) \leq \nu \tag{3.30}$$

The ALD criterion only adds an input to the dictionary if the model is uncertain about its prediction from the viewpoint of Gaussian process regression [36, Section 4.7]. If the matrix inversion in (3.29) is performed iteratively using the block matrix inversion identity (A.2), the ALD criterion can be computed with $\mathcal{O}(m_t^2)$ complexity.

#### Coherence Criterion

Richard et al. [43] introduced the coherence criterion

$$\mu = \max_{\tilde{\mathbf{x}} \in \mathcal{D}_t} |k(\mathbf{x}_t, \tilde{\mathbf{x}})| \leq \mu_0 \ . \tag{3.31}$$

The criterion calculates the largest cross-correlation in the dictionary. Thus, it is equal to zero for every orthonormal basis. A new element $\mathbf{x}_t$ is added to the dictionary if $\mu \leq \mu_0$ holds. It was shown in [43] that the coherence criterion is a computationally efficient approximation of the ALD criterion and scales with $\mathcal{O}(m_t)$ complexity instead of $\mathcal{O}(m_t^2)$.

### 3.3.2 Kernel Recursive Least Squares

The kernel recursive least squares (KRLS) algorithm, introduced by Engel et al. [38], is the kernel-based variant of the recursive least squares algorithm [17]. The algorithm uses the ALD criterion for sparsification and iteratively solves the optimization problem in the dual representation

$$\mathbf{v}_t^* = \arg\min_{\mathbf{v}_t} \frac{1}{2} \left\| \mathbf{y}_t - \boldsymbol{\Phi}_t^{\mathrm{T}} \boldsymbol{\Phi}_t \mathbf{v}_t \right\|_2^2 \ , \tag{3.32}$$

where $t$ indicates the actual time index and

$$\mathbf{y}_t^{\mathrm{T}} = \begin{bmatrix} y_1 \dots y_t \end{bmatrix} , \quad \mathbf{v}_t^{\mathrm{T}} = \begin{bmatrix} v_1 \dots v_t \end{bmatrix} , \quad \boldsymbol{\Phi}_t = \begin{bmatrix} \boldsymbol{\phi}(\mathbf{x}_1) \dots \boldsymbol{\phi}(\mathbf{x}_t) \end{bmatrix} . \tag{3.33}$$

Note that the dimension of the involved vectors and matrices increases with each time index $t$. If the ALD sparsification method from Section 3.3.1 is used, then the full design matrix $\boldsymbol{\Phi}_t$ can be expressed as the direct sum of the subspace spanned by the dictionary vectors $\tilde{\boldsymbol{\Phi}}_t \mathbf{A}_t^{\mathrm{T}}$, and its orthogonal complement $\boldsymbol{\Phi}_R$, according to

$$\boldsymbol{\Phi}_t = \tilde{\boldsymbol{\Phi}}_t \mathbf{A}_t^{\mathrm{T}} + \boldsymbol{\Phi}_R , \tag{3.34}$$

where $\tilde{\boldsymbol{\Phi}}_t = \begin{bmatrix} \boldsymbol{\phi}_t(\tilde{\mathbf{x}}_1) \dots \boldsymbol{\phi}_t(\tilde{\mathbf{x}}_{m_t}) \end{bmatrix}$ is the design matrix of all elements in the dictionary $\tilde{\mathbf{x}} \in \mathcal{D}_t$. The matrix

$$\mathbf{A}_t^{\mathrm{T}} = \begin{bmatrix} \mathbf{a}_1^*, \dots, \mathbf{a}_t^* \end{bmatrix} \in \mathbb{R}^{m_t \times t} , \tag{3.35}$$

stores the optimal expansion coefficients, defined by the ALD criterion (3.29). The resulting kernel matrix is given by

$$\mathbf{K}(\mathcal{X}_t, \mathcal{X}_t) = \boldsymbol{\Phi}_t^{\mathrm{T}} \boldsymbol{\Phi}_t = \mathbf{A}_t \mathbf{K}(\mathcal{D}_t, \mathcal{D}_t) \mathbf{A}_t^{\mathrm{T}} + \boldsymbol{\Phi}_R^{\mathrm{T}} \boldsymbol{\Phi}_R , \tag{3.36}$$

where the cross terms vanish due to orthogonality. By choosing $\nu$ in the ALD criterion (3.28) sufficiently small, the kernel matrix is approximated well by

$$\mathbf{K}(\mathcal{X}_t, \mathcal{X}_t) \approx \mathbf{A}_t \mathbf{K}(\mathcal{D}_t, \mathcal{D}_t) \mathbf{A}_t^{\mathrm{T}} . \tag{3.37}$$

Substituting the kernel approximation (3.37) into the optimization problem (3.32) leads to

$$\tilde{\mathbf{v}}_t^* = \arg\min_{\tilde{\mathbf{v}}_t} \frac{1}{2} \| \mathbf{y}_t - \mathbf{A}_t \mathbf{K}(\mathcal{D}_t, \mathcal{D}_t) \tilde{\mathbf{v}}_t \|_2^2 , \tag{3.38}$$

where $\tilde{\mathbf{v}}_t = \mathbf{A}_t^{\mathrm{T}} \mathbf{v}_t$. The minimizer $\tilde{\mathbf{v}}_t^*$ is obtained by least-squares

$$\tilde{\mathbf{v}}_t^* = \mathbf{K}^{-1}(\mathcal{D}_t, \mathcal{D}_t) \left( \mathbf{A}_t^{\mathrm{T}} \mathbf{A}_t \right)^{-1} \mathbf{A}_t^{\mathrm{T}} \mathbf{y}_t . \tag{3.39}$$

A prediction with the model is made according to

$$\hat{f}_t(\mathbf{x}_t) = \mathbf{K}^{\mathrm{T}}(\mathcal{D}_{t-1}, \mathbf{x}_t) \tilde{\mathbf{v}}_{t-1}^* \tag{3.40}$$

At each time step $t$, the algorithm faces either one of the following two cases:

1. The ALD criterion is fulfilled, i. e., $\delta_t \leq \nu$.
   In this case $\mathbf{A}_t$ changes but $\mathcal{D}_t = \mathcal{D}_{t-1}$ and $\mathbf{K}(\mathcal{D}_t, \mathcal{D}_t) = \mathbf{K}(\mathcal{D}_{t-1}, \mathcal{D}_{t-1})$ remain unchanged. The new expansion coefficients are appended according to $\mathbf{A}_t^{\mathrm{T}} = \begin{bmatrix} \mathbf{A}_{t-1}^{\mathrm{T}}, \mathbf{a}_t^* \end{bmatrix}$ where $\mathbf{a}_t^*$ is defined by the ALD criterion (3.29). The unbounded growth of $\mathbf{A}_t$ is overcome by defining

$$\mathbf{P}_t = \left( \mathbf{A}_t^{\mathrm{T}} \mathbf{A}_t \right)^{-1} = \left( \mathbf{A}_{t-1}^{\mathrm{T}} \mathbf{A}_{t-1} + \mathbf{a}_t^* (\mathbf{a}_t^*)^{\mathrm{T}} \right)^{-1} \tag{3.41}$$

and applying the *matrix inversion lemma*, see (A.1). Note that the matrix $\mathbf{P}_t$ has dimension $m_t \times m_t$. This leads to the recursive formula

$$\mathbf{P}_t = \mathbf{P}_{t-1} - \frac{\mathbf{P}_{t-1}\mathbf{a}_t^*(\mathbf{a}_t^*)^{\mathrm{T}}\mathbf{P}_{t-1}}{1 + (\mathbf{a}_t^*)^{\mathrm{T}}\mathbf{P}_{t-1}\mathbf{a}_t^*} \,. \tag{3.42}$$

Substituting (3.42) into (3.39) yields the update rule for the coefficients

$$\tilde{\mathbf{v}}_t^* = \tilde{\mathbf{v}}_{t-1}^* + \mathbf{K}^{-1}(\mathcal{D}_t, \mathcal{D}_t)\mathbf{q}_t\Big(y_t - \hat{f}_t(\mathbf{x}_t)\Big) \,, \tag{3.43}$$

where the definition

$$\mathbf{q}_t = \frac{\mathbf{P}_{t-1}\mathbf{a}_t^*}{1 + (\mathbf{a}_t^*)^{\mathrm{T}}\mathbf{P}_{t-1}\mathbf{a}_t^*} \,, \tag{3.44}$$

is introduced.

2. The ALD criterion is not fulfilled, i. e., $\delta_t > \nu$.
Thus $\mathbf{x}_t$ is added to the dictionary $\mathcal{D}_t = \{\mathcal{D}_{t-1} \cup \mathbf{x}_t\}$ and the kernel matrix grows according to

$$\mathbf{K}(\mathcal{D}_t, \mathcal{D}_t) = \begin{bmatrix} \mathbf{K}(\mathcal{D}_{t-1}, \mathcal{D}_{t-1}) & \mathbf{K}(\mathcal{D}_{t-1}, \mathbf{x}_t) \\ \mathbf{K}^{\mathrm{T}}(\mathcal{D}_{t-1}, \mathbf{x}_t) & k(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix} \,. \tag{3.45}$$

Applying the *block matrix inversion identity*, see (A.2), leads to the recursive formula

$$\mathbf{K}^{-1}(\mathcal{D}_t, \mathcal{D}_t) = \begin{bmatrix} \mathbf{K}^{-1}(\mathcal{D}_{t-1}, \mathcal{D}_{t-1}) & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & 0 \end{bmatrix} + \frac{1}{\delta_t}\begin{bmatrix} \mathbf{a}_t^* \\ -1 \end{bmatrix}\begin{bmatrix} \mathbf{a}_t^* \\ -1 \end{bmatrix}^{\mathrm{T}} \,. \tag{3.46}$$

Since a new element is added to the dictionary, the matrix

$$\mathbf{A}_t = \begin{bmatrix} \mathbf{A}_{t-1} & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} \tag{3.47}$$

has to be augmented as well. The minimizer (3.39) is then given as

$$\tilde{\mathbf{v}}_t^* = \delta_t^{-1}\begin{bmatrix} \delta_t\tilde{\mathbf{v}}_{t-1}^* - \mathbf{a}_t^*\Big(y_t - \hat{f}_t(\mathbf{x}_t)\Big) \\ y_t - \hat{f}_t(\mathbf{x}_t) \end{bmatrix} \,. \tag{3.48}$$

Engel et al. [38] proved that with the ALD criterion, the maximum number of kernel functions is bounded even for $t \to \infty$. The algorithm is numerically stable because the ALD criterion $\delta_t > \nu$ prevents the inverse kernel matrix (3.46) from being ill-conditioned [36, Appendix B]. A MATLAB implementation of the algorithm can be found in the Kernel Adaptive Filter Toolbox [39].

### 3.3.3 Kernel Affine Projection

The kernel affine projection (KAP) algorithm, introduced by Richard et al. [43], uses the coherence criterion for sparsification and solves the least squares problem using a stochastic gradient approach. Instead of solving the optimization problem for all previous $t$ data points, only the recent $q$ points are considered. Thus, it is assumed that the recent $q$ points provide a reasonable stochastic approximation of the training data. The resulting kernel matrix, evaluated at all $m_t$ elements of the dictionary $\mathcal{D}_t$ and the last $q$ points $\mathcal{X}_q = \{\mathbf{x}_i \mid i = t-q+1, \dots, t\}$, is given by

$$\mathbf{K}_{q,t} = \mathbf{K}(\mathcal{X}_q, \mathcal{D}_t) = \begin{bmatrix} k(\mathbf{x}_t, \tilde{\mathbf{x}}_1) & \cdots & k(\mathbf{x}_t, \tilde{\mathbf{x}}_{m_t}) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_{t-q+1}, \tilde{\mathbf{x}}_1) & \cdots & k(\mathbf{x}_{t-q+1}, \tilde{\mathbf{x}}_{m_t}) \end{bmatrix}, \tag{3.49}$$

where $\mathbf{K}_{q,t}$ is a $q \times m_t$ matrix. A prediction with the model is made, according to

$$\hat{f}_t(\mathbf{x}_t) = \mathbf{K}(\mathcal{X}_q, \mathcal{D}_{t-1})\tilde{\mathbf{v}}_{t-1} \ . \tag{3.50}$$

The KAP algorithm updates the coefficients $\tilde{\mathbf{v}}_t$ according to the optimization problem

$$\tilde{\mathbf{v}}_t^* = \arg\min_{\tilde{\mathbf{v}}_t} \quad \frac{1}{2}\|\tilde{\mathbf{v}}_t - \tilde{\mathbf{v}}_{t-1}^*\|_2^2 \tag{3.51a}$$

$$\text{subject to} \quad \mathbf{y}_q - \mathbf{K}_{q,t}\tilde{\mathbf{v}}_t = \mathbf{0} \tag{3.51b}$$

where $\mathbf{y}_q^{\mathrm{T}} = \begin{bmatrix} y_t \dots y_{t-q+1} \end{bmatrix}$. Each of the $q$ constraints in (3.51) defines a hyperplane in a $m_t$-dimensional space [17, Section 5.6]. The coefficients $\tilde{\mathbf{v}}_t^*$ are obtained by projecting $\tilde{\mathbf{v}}_{t-1}^*$ on the $q$ hyperplanes. Thus, $\tilde{\mathbf{v}}_t^*$ lies on the intersections of these planes. However, noise affects the position of the hyperplanes and thus, forcing $\tilde{\mathbf{v}}_t^*$ to lie exactly on those planes is not necessarily good. Therefore a learning rate $\eta$ will be introduced later on to improve the robustness against noise.

At each time index $t$ the algorithm faces either one of the following two cases:

1. Sparsification criterion fulfilled, i.e $\mu > \mu_0$.
   In this case $\mathbf{x}_t$ is not added to the dictionary. The solution of the constrained optimization problem (3.51) is obtained by minimizing the Lagrange function [24, Section 5.1]
   $$L(\tilde{\mathbf{v}}_t, \boldsymbol{\lambda}) = \frac{1}{2}\|\tilde{\mathbf{v}}_t - \tilde{\mathbf{v}}_{t-1}\|_2^2 + \boldsymbol{\lambda}^{\mathrm{T}}(\mathbf{y}_q - \mathbf{K}_{q,t}\tilde{\mathbf{v}}_t) \ , \tag{3.52}$$
   where $\boldsymbol{\lambda}$ is the Lagrange multiplier. The necessary first-order conditions for a local minima are obtained by setting the derivatives of (3.52) with respect to $\tilde{\mathbf{v}}$ and $\boldsymbol{\lambda}$ to zero
   $$\frac{\mathrm{d}L}{\mathrm{d}\tilde{\mathbf{v}}_t}(\tilde{\mathbf{v}}_t, \boldsymbol{\lambda}) = \tilde{\mathbf{v}}_t - \tilde{\mathbf{v}}_{t-1} - \mathbf{K}_{q,t}^{\mathrm{T}}\boldsymbol{\lambda} = \mathbf{0} \tag{3.53a}$$
   $$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\lambda}}(\tilde{\mathbf{v}}_t, \boldsymbol{\lambda}) = \mathbf{y}_q - \mathbf{K}_{q,t}\tilde{\mathbf{v}}_t = \mathbf{0} \ . \tag{3.53b}$$

Solving (3.53a) and (3.53b) for $\tilde{\mathbf{v}}_t$ leads to the update rule for the coefficients

$$\tilde{\mathbf{v}}_t^* = \tilde{\mathbf{v}}_{t-1} + \mathbf{K}_{q,t}^{\mathrm{T}}\left(\mathbf{K}_{q,t}\mathbf{K}_{q,t}^{\mathrm{T}}\right)^{-1}(\mathbf{y}_q - \mathbf{K}_{q,t}\tilde{\mathbf{v}}_{t-1}) \ . \tag{3.54}$$

The computational complexity of the KAP is $\mathcal{O}(m_t q^2)$ time and $\mathcal{O}(m_t q)$ space. To improve the numerical stability of the algorithm a regularization parameter $\epsilon \geq 0$ is introduced

$$\tilde{\mathbf{v}}_t^* = \tilde{\mathbf{v}}_{t-1} + \eta\mathbf{K}_{q,t}^{\mathrm{T}}\left(\mathbf{K}_{q,t}\mathbf{K}_{q,t}^{\mathrm{T}} + \epsilon\mathbf{E}\right)^{-1}(\mathbf{y}_q - \mathbf{K}_{q,t}\tilde{\mathbf{v}}_{t-1}) \ . \tag{3.55}$$

Furthermore, a learning rate $0 \leq \eta$ is introduced to cope with noise.

2. Sparsification criterion not fulfilled, i.e $\mu \leq \mu_0$.
In this case $\mathbf{x}_t$ is added to the dictionary, thus $m_t = m_{t-1} + 1$ and the matrix $\mathbf{K}_{q,t}$ is updated according to (3.49). To account for the new element in the coefficients, the optimization problem (3.51) is restated to

$$\min_{\tilde{\mathbf{v}}_t} \quad \|\tilde{\mathbf{v}}_{1:m_{t-1}} - \tilde{\mathbf{v}}_{t-1}\|_2^2 + v_{m_t}^2 \tag{3.56a}$$

$$\text{subject to} \quad \mathbf{y}_q - \mathbf{K}_{q,t}\tilde{\mathbf{v}}_t = \mathbf{0} \tag{3.56b}$$

where the notation $\tilde{\mathbf{v}}_{1:m_{t-1}}$ indicates the first $m_{t-1}$ entries and $v_{m_t}$ is the last entry of the vector $\tilde{\mathbf{v}}_t$. Thus, the minimizer is

$$\tilde{\mathbf{v}}_t^* = \begin{bmatrix}\tilde{\mathbf{v}}_{t-1} \\ 0\end{bmatrix} + \eta\mathbf{K}_{q,t}^{\mathrm{T}}\left(\mathbf{K}_{q,t}\mathbf{K}_{q,t}^{\mathrm{T}} + \epsilon\mathbf{E}\right)^{-1}\left(\mathbf{y}_q - \mathbf{K}_{q,t}\begin{bmatrix}\tilde{\mathbf{v}}_{t-1} \\ 0\end{bmatrix}\right) . \tag{3.57}$$

A MATLAB implementation of the algorithm can be found in the Kernel Adaptive Filter Toolbox [39].

# 4 Bayesian Regression

In this chapter, the function approximation problem is reformulated within a Bayesian framework. In the Bayesian interpretation, a probability distribution reflects the uncertainty of a stochastic quantity. In the context of function approximation, the stochastic quantities are the parameters of a parametric model or functions. The uncertainty about these quantities can be reduced by observing data. Bayes Theorem serves as the mathematical tool to incorporate the information that the data provides. Online algorithms can be derived by recursive application of Bayes Theorem. Compared to the frequentist approach of Chapter 3, the Bayesian framework allows to quantify the uncertainty of a prediction and provides a systematic framework for model selection.

This chapter is structured as follows: The linear basis function model is reviewed in Section 4.1 from a Bayesian perspective. In order to obtain a closed form solution, a Gaussian prior and Gaussian likelihood is assumed throughout this chapter. The offline solution of the Bayesian linear regression problem can be calculated with $\mathcal{O}(M^3)$ complexity, where $M$ is the number of basis functions. Bayesian model selection is discussed in Section 4.2. Finally, Section 4.3 closes with Gaussian process based online algorithms, with a computational complexity independent of number of training data points $N$.

## 4.1 Bayesian Linear Regression

The linear basis function model (3.1) is now reviewed from a Bayesian perspective [25, Section 2.1]. In the Bayesian framework, the model coefficients $\mathbf{w}$ are not deterministic any more. Rather, they are described through a probability distribution $p(\mathbf{w})$. The distribution $p(\mathbf{w})$ is known as the prior distribution because it reflects the prior beliefs about the model coefficients. After receiving a set inputs $\mathcal{X} = \{\mathbf{x}_i \mid i = 1, \dots, N\}$ with corresponding observations $\mathbf{y} = \{y_i \mid i = 1, \dots, N\}$, the information that the data provides, is incorporated by updating the prior $p(\mathbf{w})$ according to Bayes Theorem

$$p(\mathbf{w} \mid \mathcal{X}, \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{w}, \mathcal{X}) p(\mathbf{w})}{p(\mathbf{y} \mid \mathcal{X})} \,. \tag{4.1}$$

Here, $p(\mathbf{w} \mid \mathcal{X}, \mathbf{y})$ is the posterior distribution of the model coefficients $\mathbf{w}$ conditioned on the inputs $\mathcal{X}$ and the observations $\mathbf{y}$. The likelihood $p(\mathbf{y} \mid \mathbf{w}, \mathcal{X})$ expresses the probability of observations given the coefficients $\mathbf{w}$ and inputs $\mathcal{X}$. The marginal-likelihood $p(\mathbf{y} \mid \mathcal{X})$ is obtained through marginalization of the numerator in (4.1) leading to

$$p(\mathbf{y} \mid \mathcal{X}) = \int p(\mathbf{y} \mid \mathbf{w}, \mathcal{X}) p(\mathbf{w}) \, \mathrm{d}\mathbf{w} \,. \tag{4.2}$$

The integral (4.2) has an analytic solution only in special cases, e. g., Gaussian prior and Gaussian likelihood. In order to make a prediction at a test point $\mathbf{x}_*$, the coefficients are marginalized out according to

$$p\Big(\hat{f} \mid \mathbf{x}_*, \mathcal{X}, \mathbf{y}\Big) = \int p\Big(\hat{f} \mid \mathbf{x}_*, \mathbf{w}\Big) p(\mathbf{w} \mid \mathcal{X}, \mathbf{y}) \, \mathrm{d}\mathbf{w} \; . \tag{4.3}$$

The marginalization operation averages the output of all possible models, weighted by their posterior (4.1) [25, Section 2.1]. The distribution $p\Big(\hat{f} \mid \mathbf{x}_*, \mathbf{w}\Big)$ is the likelihood of $\hat{f}$ conditioned on the model with coefficients $\mathbf{w}$ at a test point $\mathbf{x}_*$.

### Gaussian Prior and Gaussian Likelihood

For the special case of a Gaussian prior and a Gaussian likelihood the integral (4.3) can be solved analytically. A common assumption for regression is an additive Gaussian noise model according to

$$y = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}) + \epsilon \quad \text{with} \quad p(\epsilon) = \mathcal{N}\Big(0, \, \sigma_n^2\Big) \; , \tag{4.4}$$

where $\mathcal{N}\big(0, \, \sigma_n^2\big)$ is the Gaussian or Normal distribution with zero mean and constant[1] variance $\sigma_n^2$. For additive Gaussian noise, the likelihood of a single observation $y_i$, given the input $\mathbf{x}_i$ and the coefficients $\mathbf{w}$, is Gaussian and can be described as

$$p(y_i \mid \mathbf{w}, \mathbf{x}_i) = \mathcal{N}\Big(\mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}), \, \sigma_n^2\Big) \; . \tag{4.5}$$

For independent Gaussian noise the likelihood factorizes according to

$$p(\mathbf{y} \mid \mathbf{w}, \mathcal{X}) = \prod_{i=1}^{N} p\Big(y_i \mid \mathbf{w}, \mathbf{x}_i, \sigma_n^2\Big) = \mathcal{N}\Big(\boldsymbol{\Phi}^{\mathrm{T}} \mathbf{w}, \, \sigma_n^2 \mathbf{E}\Big) \; . \tag{4.6}$$

A Gaussian likelihood together with a Gaussian prior $p(\mathbf{w}) = \mathcal{N}(\boldsymbol{\mu}_0, \, \boldsymbol{\Sigma}_0)$ with mean $\boldsymbol{\mu}_0$ and covariance $\boldsymbol{\Sigma}_0$ leads to a posterior distribution (4.1) over the coefficients, according to

$$p(\mathbf{w} \mid \mathcal{X}, \mathbf{y}) = \mathcal{N}(\mathbf{w}^*, \, \boldsymbol{\Sigma}_w) \; , \tag{4.7a}$$

$$\boldsymbol{\Sigma}_w^{-1} = \sigma_n^{-2} \boldsymbol{\Phi} \boldsymbol{\Phi}^{\mathrm{T}} + \boldsymbol{\Sigma}_0^{-1} \; , \tag{4.7b}$$

$$\mathbf{w}^* = \boldsymbol{\Sigma}_w \Big( \boldsymbol{\Sigma}_0^{-1} \boldsymbol{\mu}_0 + \sigma_n^{-2} \boldsymbol{\Phi} \mathbf{y} \Big) \; , \tag{4.7c}$$

which is Gaussian [15, Section 3.3]. For the special case $\boldsymbol{\Sigma}_0 = \sigma_0^2 \mathbf{E}$ and $\boldsymbol{\mu}_0 = \mathbf{0}$ the mean $\mathbf{w}^*$ is identical to (3.8) with $\lambda = \left(\frac{\sigma_n}{\sigma_0}\right)^2$. The log of the marginal likelihood (4.2), which is often used for hyperparameter tuning (see [15, Section 3.5] and Section 4.2), is given as

$$\begin{aligned} \ln p(\mathbf{y} \mid \mathcal{X}) = &- \frac{1}{2\sigma_n^2} \Big\| \mathbf{y} - \boldsymbol{\Phi}^{\mathrm{T}} \mathbf{w}^* \Big\|_2^2 - \frac{1}{2\sigma_0^2} \| \mathbf{w}^* \|_2^2 \\ &- \frac{1}{2} \ln \det \Big( \boldsymbol{\Sigma}_w^{-1} \Big) - \frac{M}{2} \ln \sigma_0^2 - \frac{N}{2} \ln \Big( 2\pi \sigma_n^2 \Big) \; , \end{aligned} \tag{4.8}$$

[1] It is also possible to consider heteroscedastic noise, i. e., noise which depends on the inputs $\sigma_n^2 = \sigma_n^2(\mathbf{x})$.

with $\mathbf{w}^*$ and $\mathbf{\Sigma}_w$ according to (4.7). Finally, the predictive equation (4.3) is also Gaussian

$$p\Big(\hat{f} \mid \mathbf{x}_*, \mathcal{X}, \mathbf{y}\Big) = \mathcal{N}\Big(\hat{f}(\mathbf{x}_*),\, \hat{\sigma}^2(\mathbf{x}_*)\Big) \;, \tag{4.9a}$$

$$\hat{f}(\mathbf{x}_*) = \boldsymbol{\phi}(\mathbf{x}_*)^{\mathrm{T}}\mathbf{w}^* \;, \tag{4.9b}$$

$$\hat{\sigma}^2(\mathbf{x}_*) = \sigma_n^2 + \boldsymbol{\phi}(\mathbf{x}_*)^{\mathrm{T}}\mathbf{\Sigma}_w\boldsymbol{\phi}(\mathbf{x}_*) \;. \tag{4.9c}$$

For basis functions which converge to zero outside the main influence region, e. g., the Gaussian basis function, the contribution from $\boldsymbol{\phi}(\mathbf{x}_*)^{\mathrm{T}}\mathbf{\Sigma}_w\boldsymbol{\phi}(\mathbf{x}_*)$ in (4.9) will also converge to zero outside the main influence region of the basis function. Hence, the model becomes very confident outside the effective region of the basis function [15, Section 3.3.2]. This quite unintuitive understanding of confidence is avoided in the framework auf Gaussian processes, introduced in Section 2.3.

## 4.2 Model Selection

The Bayesian approach provides a systematic framework to infer the hyperparameter of the model from data [25, Chapter 5]. Hyperparameters, further denoted as $\boldsymbol{\theta}$, are free parameters like the noise variance $\sigma_n^2$ and kernel parameters, e. g., the length scale of the Gaussian kernel (3.11). In a fully Bayesian treatment [25, Chapter 5.2], a prior distribution $p(\boldsymbol{\theta})$ over the hyperparameter is introduced and than marginalized out according to

$$p_{\boldsymbol{\theta}}(\mathbf{y}) = \int p(\mathbf{y} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta} \;, \tag{4.10}$$

where $\boldsymbol{\theta}$ in $p(\mathbf{y} \mid \boldsymbol{\theta})$ indicates the dependency of the marginal-likelihood (2.21) on the hyperparameters $\boldsymbol{\theta}$. The integral is analytically intractable. Thus, it is usually assumed that the prior $p(\boldsymbol{\theta})$ is peaked[2] around the value $\hat{\boldsymbol{\theta}}$. In this case, the integral (4.10) can be well approximated by the marginal-likelihood

$$p_{\boldsymbol{\theta}}(\mathbf{y}) \approx p\Big(\mathbf{y} \mid \hat{\boldsymbol{\theta}}\Big) \;. \tag{4.11}$$

Thus, the optimal hyperparameters $\boldsymbol{\theta}^*$ of the Gaussian process are obtained by maximizing the log-marginal-likelihood (2.21) according to

$$\max_{\boldsymbol{\theta}} -\frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{K}_{\boldsymbol{\theta}}^{-1}\mathbf{y} - \frac{1}{2}\ln\det(\mathbf{K}_{\boldsymbol{\theta}}) - \frac{N}{2}\ln 2\pi \;. \tag{4.12}$$

The subscript in $\mathbf{K}_{\boldsymbol{\theta}} = \mathbf{K}(\mathcal{X}, \mathcal{X}) + \sigma_n^2\mathbf{E}$ is used to emphasize the dependency on the hyperparameters. The first term in (4.12) measures the data-fit, while the second is a complexity penalty term which reduces overfitting. The nonlinear optimization problem (4.12) can be efficiently solved with gradient-based solvers [24]. This is possible, since the partial derivatives of the log-marginal-likelihood w.r.t. the hyperparameters can be obtained in closed form via

$$\frac{\partial}{\partial\theta_i}\ln p(\mathbf{y} \mid \mathcal{X}, \boldsymbol{\theta}) = \frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{K}_{\boldsymbol{\theta}}^{-1}\frac{\partial\mathbf{K}_{\boldsymbol{\theta}}}{\partial\theta_i}\mathbf{K}_{\boldsymbol{\theta}}^{-1}\mathbf{y} - \frac{1}{2}\operatorname{Tr}\Big(\mathbf{K}_{\boldsymbol{\theta}}^{-1}\frac{\partial\mathbf{K}_{\boldsymbol{\theta}}}{\partial\theta_i}\Big) \;. \tag{4.13}$$

[2] For example $p(\boldsymbol{\theta})$ is a Gaussian with small variance.

The computational overhead of computing the gradients is small because the most expensive operation, the inversion of $\mathbf{K}_{\boldsymbol{\theta}}$, has to be computed anyway in (4.12) with $\mathcal{O}(N^3)$ complexity. Once the inverse is computed, the computation of the derivative requires $\mathcal{O}(N^2)$ time per hyperparameter. The computational complexity could also be reduced if a sparse GP, introduced in Section 2.5 is used. From an optimization point of view, the nonlinear optimization problem (4.12) may have multiple local maxima, each corresponding to a particular interpretation of the data. In [25, Section 5.4] it is empirically observed that the log-marginal-likelihood gets more peaked with more data, leading to more robust hyperparameter estimates.

## 4.3 Online Gaussian Processes

Based on the preliminary introduction to Gaussian processes, this section summarizes two possible implementations of online Gaussian processes. First, the Kernel Recursive Least Squares (KRLS) algorithm is derived from a Bayesian perspective. In order to limit the number of kernel functions in (2.15), a sparsification criterion is introduced. Second, the recursive GP (rec-GP) algorithm is introduced, which uses an a priori defined dictionary of fixed size and therefore does not need a sparsification criterion.

### 4.3.1 Bayesian Kernel Recursive Least Squares

The KRLS algorithm is now derived from a Bayesian perspective [40, 42]. In contrast to Section 3.3.2, the Bayesian interpretation allows to systematically account for noise and additionally provides an estimate for the prediction uncertainty. Therefore, a recursive Bayesian update rule is derived and a sparsification method to limit the number of kernel functions is introduced.

Suppose that at time step $t$, the algorithm has processed the inputs $\mathcal{X}_t = \{\mathbf{x}_i \mid i = 1, \dots, t\}$ with corresponding observations $\mathbf{y}_t$. The information that a new sample $(\mathbf{x}_{t+1}, y_{t+1})$ provides is incorporated by recursive application of Bayes rule

$$p(\mathbf{f}_{t+1} \mid \mathbf{y}_{t+1}) = p(\mathbf{f}_t,\, f_{t+1} \mid \mathbf{y}_t, y_{t+1}) = \frac{p(\mathbf{y}_t, y_{t+1} \mid \mathbf{f}_t, f_{t+1})\, p(\mathbf{f}_t, f_{t+1})}{p(\mathbf{y}_t, y_{t+1})} \,. \tag{4.14}$$

The first term in the numerator can be further simplified due to the assumed conditional independency between the observations and latent function values at time $t$ and $t+1$, according to

$$p(\mathbf{y}_t, y_{t+1} \mid \mathbf{f}_t, f_{t+1}) = p(\mathbf{y}_t \mid \mathbf{f}_t)\, p(y_{t+1}, \mid f_{t+1}) \,. \tag{4.15}$$

After expanding the joint probabilities in (4.14) with the chain rule of probability and rearranging terms, the recursive update equation is given by

$$p(\mathbf{f}_{t+1} \mid \mathbf{y}_{t+1}) = \frac{p(y_{t+1} \mid f_{t+1})\, p(f_{t+1} \mid \mathbf{f}_t)}{p(y_{t+1} \mid \mathbf{y}_t)}\, p(\mathbf{f}_t \mid \mathbf{y}_t) \,, \tag{4.16}$$

where $p(\mathbf{f}_t \mid \mathbf{y}_t) = \mathcal{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ is the Gaussian posterior of the previous time step $t$. Specifically, the posterior of the previous estimate can be taken directly as the prior for the

current estimate. It should be noted that the recursive update rule (4.16) shows resemblance to the conditioned joint probability (2.20) of the offline GP.

In general, the recursive update rule (4.16) is valid for arbitrary likelihoods $p(y_{t+1} \mid f_{t+1})$. However, an analytical solution is only obtained for Gaussian likelihoods like

$$p(y_{t+1} \mid f_{t+1}) = \mathcal{N}\left(f_{t+1},\, \sigma_n^2\right) . \tag{4.17}$$

The conditional distribution $p(f_{t+1} \mid \mathbf{f}_t)$ is obtain by conditioning of the joint probability (2.24) using (A.5), according to

$$p(f_{t+1} \mid \mathbf{f}_t) = \mathcal{N}\left(\hat{f}_{t+1},\, \gamma_{t+1}^2\right) , \tag{4.18a}$$

$$\hat{f}_{t+1} = \mathbf{K}^{\mathrm{T}}(\mathcal{X}_t, \mathbf{x}_{t+1})\mathbf{Q}_t\boldsymbol{\mu}_t \tag{4.18b}$$

$$\gamma_{t+1}^2 = k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) - \mathbf{K}^{\mathrm{T}}(\mathcal{X}_t, \mathbf{x}_{t+1})\mathbf{Q}_t\mathbf{K}(\mathcal{X}_t, \mathbf{x}_{t+1}) , \tag{4.18c}$$

where $\mathbf{Q}_t = \mathbf{K}^{-1}(\mathcal{X}_t, \mathcal{X}_t)$ is the inverse kernel matrix. Furthermore, $\gamma_{t+1}^2$ is uncertainty of the projection of $f_{t+1}$ onto the previous latent function values $\mathbf{f}_t$. The marginal-likelihood $p(y_{t+1} \mid \mathbf{y}_t)$ is obtained by marginalization of the numerator in (4.16), according to

$$p(y_{t+1} \mid \mathbf{y}_t) = \int\int p(y_{t+1} \mid f_{t+1})\, p(f_{t+1} \mid \mathbf{f}_t)\, p(\mathbf{f}_t \mid \mathbf{y}_t)\, \mathrm{d}\mathbf{f}_t\, \mathrm{d}f_{t+1} . \tag{4.19}$$

The marginal-likelihood is again Gaussian, given by

$$p(y_{t+1} \mid \mathbf{y}_t) = \mathcal{N}\left(\hat{f}_{t+1},\, \hat{\sigma}_{t+1}^2\right) , \tag{4.20a}$$

$$\hat{f}_{t+1} = \mathbf{K}^{\mathrm{T}}(\mathcal{X}_t, \mathbf{x}_{t+1})\mathbf{Q}_t\boldsymbol{\mu}_t , \tag{4.20b}$$

$$\hat{\sigma}_{t+1}^2 = \sigma_n^2 + k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) + \mathbf{K}^{\mathrm{T}}(\mathcal{X}_t, \mathbf{x}_{t+1})(\mathbf{Q}_t\boldsymbol{\Sigma}_t\mathbf{Q}_t - \mathbf{Q}_t)\mathbf{K}(\mathcal{X}_t, \mathbf{x}_{t+1}) . \tag{4.20c}$$

The marginal-likelihood (4.20) is used to predict the latent function at an input $\mathbf{x}_{t+1}$, where $\hat{f}_{t+1}$ is the mean and $\hat{\sigma}_{t+1}^2$ is the variance of the prediction. Finally, the recursive update rule for the posterior distribution, according to (4.16), is Gaussian with mean and covariance defined by

$$p(\mathbf{f}_{t+1} \mid \mathbf{y}_{t+1}) = \mathcal{N}(\boldsymbol{\mu}_{t+1},\, \boldsymbol{\Sigma}_{t+1}) , \tag{4.21a}$$

$$\boldsymbol{\mu}_{t+1} = \begin{bmatrix} \boldsymbol{\mu}_t \\ \hat{f}_{t+1} \end{bmatrix} + \frac{y_{t+1} - \hat{f}_{t+1}}{\hat{\sigma}_{t+1}^2} \begin{bmatrix} \mathbf{h}_{t+1} \\ \hat{\sigma}_{f,t+1}^2 \end{bmatrix} , \tag{4.21b}$$

$$\boldsymbol{\Sigma}_{t+1} = \begin{bmatrix} \boldsymbol{\Sigma}_t & \mathbf{h}_{t+1} \\ \mathbf{h}_{t+1}^{\mathrm{T}} & \hat{\sigma}_{f,t+1}^2 \end{bmatrix} - \frac{1}{\hat{\sigma}_{t+1}^2} \begin{bmatrix} \mathbf{h}_{t+1} \\ \hat{\sigma}_{f,t+1}^2 \end{bmatrix} \begin{bmatrix} \mathbf{h}_{t+1} \\ \hat{\sigma}_{f,t+1}^2 \end{bmatrix}^{\mathrm{T}} , \tag{4.21c}$$

with $\hat{f}_{t+1}$ and $\hat{\sigma}_{t+1}^2$ according to the marginal-likelihood (4.20). The abbreviations

$$\mathbf{h}_{t+1} = \boldsymbol{\Sigma}_t\mathbf{Q}_t\mathbf{K}(\mathcal{X}_t, \mathbf{x}_{t+1}) \qquad \text{and} \qquad \hat{\sigma}_{f,t+1}^2 = \hat{\sigma}_{t+1}^2 - \sigma_n^2 \tag{4.22}$$

are introduced for ease of notation. The inverse kernel matrix $\mathbf{Q}_t$ can be updated iteratively for every new observation via a rank-one update based on the block matrix inversion identity (A.2)

$$\mathbf{Q}_{t+1} = \begin{bmatrix} \mathbf{Q}_t & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & 0 \end{bmatrix} + \frac{1}{\gamma_{t+1}^2} \begin{bmatrix} \mathbf{Q}_t \mathbf{K}(\mathcal{X}_t, \mathbf{x}_{t+1}) \\ -1 \end{bmatrix} \begin{bmatrix} \mathbf{Q}_t \mathbf{K}(\mathcal{X}_t, \mathbf{x}_{t+1}) \\ -1 \end{bmatrix}^{\mathrm{T}} . \tag{4.23}$$

Note, that this operation is numerically unstable if $\gamma_{t+1}^2$ is close to (numerically) zero. If $\gamma_{t+1}^2 = 0$, there is a deterministic relationship between $f_{t+1}$ and $\mathbf{f}_t$, see (4.18). Thus, there is no need to perform the recursive update step.

The dimension of the mean vector and the covariance matrix in the recursive update equation (4.21) increase with every new observation. Thus, similar to the kernel adaptive filter framework, sparsification methods are necessary to limit the size and therefore the computational complexity of the model. From now on suppose that at time $t$, all informative inputs $\tilde{\mathbf{x}}_i$ are stored in the dictionary $\mathcal{D}_t = \{\tilde{\mathbf{x}}_i \mid 1, \ldots, m\}$. Thus, one has to replace $\mathcal{X}_t$ by $\mathcal{D}_t$ in (4.21) and (4.20). In order to limit the size of the dictionary, an input is removed whenever the dictionary size gets larger than a predefined budget $m$ [42]. In the Bayesian framework, the optimal way to remove an element from the dictionary is to marginalize it out from (4.21). For Gaussian distributions the marginalization operation simply corresponds to the removal of the corresponding row and column from the mean vector $\boldsymbol{\mu}_{t+1}$ and the covariance matrix $\boldsymbol{\Sigma}_{t+1}$. Furthermore, an optimal criterion for removal is to select the input, which minimizes the Kullback-Leibler (KL) divergence (see Section A.2.4) between the exact and approximate posterior distribution. For Gaussian distributions, the KL-divergence can be computed analytically according to (A.11). However, the computation the KL-divergence for every element in the dictionary is computational expensive. Instead, the squared-error between the exact posterior mean $\boldsymbol{\mu}_{t+1}$ and its approximation $\tilde{\boldsymbol{\mu}}_{t+1}$ is minimized. According to [40, 42], the index $i^*$, assigned to the input which induces the minimal squared-error during removal, is obtained by the criterion

$$i^* = \arg\min_i \left( \frac{[\mathbf{Q}_{t+1} \boldsymbol{\mu}_{t+1}]_i}{[\mathbf{Q}_{t+1}]_{i,i}} \right)^2 , \tag{4.24}$$

where $[\mathbf{b}]_i$ denotes the $i$-th element of the vector $\mathbf{b}$ and $[\mathbf{A}]_{i,j}$ the $i,j$-th element of the matrix $\mathbf{A}$. The derived algorithm has a computation complexity if $\mathcal{O}(m^2)$ time and space per time step.

The shown algorithm can also be extended to model time varying functions. Following [42], a forgetting factor $\beta \in (0, 1]$ is introduced and the mean and covariance of the Gaussian process are updated each time step according to

$$\boldsymbol{\Sigma}_t = \beta \boldsymbol{\Sigma}_t + (1 - \beta) \mathbf{K}(\mathcal{D}_t, \mathcal{D}_t) , \tag{4.25a}$$

$$\boldsymbol{\mu}_t = \sqrt{\beta} \boldsymbol{\mu}_t . \tag{4.25b}$$

A MATLAB implementation of the algorithm can be found in the Kernel Adaptive Filter Toolbox [39].

### 4.3.2 Recursive Gaussian Process

Huber [51] proposed an online GP algorithm which infers the latent function at a fixed set of $m$ inputs $\mathcal{D} = \{\mathbf{x}_i \mid i = 1 \dots m\}$. This approach differs from the sparsification approaches insofar, that the dictionary $\mathcal{D}$ is not constructed by a sparsification criterion but rather defined a priori. This enables to derive a Kalman filter update for the mean and covariance of the latent function at inputs $\mathcal{D}$.

Let $\mathbf{f}_m$ denote the latent function values at the inputs $\mathcal{D}$. Thus, at time $t = 0$ a GP prior $p(\mathbf{f}_m) = \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_0)$ with zero mean and covariance

$$\boldsymbol{\Sigma}_0 = \mathbf{K}(\mathcal{D}, \mathcal{D}) \tag{4.26}$$

is introduced. Suppose that at time step $t$, the algorithm has processed the inputs $\mathcal{X}_t = \{\mathbf{x}_i \mid i = 1, \dots, t\}$ with corresponding observations $\mathbf{y}_t$. The information, provided by a new sample $(\mathbf{x}_{t+1}, y_{t+1})$, is incorporated by recursive application of Bayes rule, see (4.21). In contrary to the B-KRLS, the growth of the dictionary is circumvented by marginalization of the latent function $f_{t+1}$ from the conditional joint probability $p(\mathbf{f}_m, f_{t+1} \mid \mathbf{y}_{t+1})$ according to

$$p(\mathbf{f}_m \mid \mathbf{y}_{t+1}) = \int p(\mathbf{f}_m, f_{t+1} \mid \mathbf{y}_{t+1}) \, \mathrm{d}f_{t+1} \tag{4.27}$$

The marginalization is crucial to maintain a dictionary of constant size. The conditional joint probability is identical to (4.16) and is given as

$$p(\mathbf{f}_m, f_{t+1} \mid \mathbf{y}_{t+1}) = \frac{p(y_{t+1} \mid f_{t+1}) \, p(f_{t+1} \mid \mathbf{f}_m)}{p(y_{t+1} \mid \mathbf{y}_t)} \, p(\mathbf{f}_m \mid \mathbf{y}_t) \ . \tag{4.28}$$

Again, $p(\mathbf{f}_m \mid \mathbf{y}_t)$ is the posterior of the previous recursion, which gets updated with every new sample. Since marginalization for Gaussian processes corresponds to the removal of the corresponding row and column from the mean vector and the covariance matrix, the solution (4.27) is obtained by performing the update (4.21) without adding the last row and column. This leads to the recursive update rule

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \frac{y_{t+1} - \hat{f}_{t+1}}{\hat{\sigma}^2_{t+1}} \mathbf{h}_{t+1} \tag{4.29a}$$

$$\boldsymbol{\Sigma}_{t+1} = \boldsymbol{\Sigma}_t - \frac{1}{\hat{\sigma}^2_{t+1}} \mathbf{h}_{t+1} \mathbf{h}^{\mathrm{T}}_{t+1} \ , \tag{4.29b}$$

where $\hat{f}_{t+1}$ and $\hat{\sigma}^2_{t+1}$ is the predictive mean and variance as in (4.20) and $\mathbf{h}_{t+1} = \boldsymbol{\Sigma}_t \boldsymbol{\Sigma}_0^{-1} \mathbf{K}(\mathcal{D}, \mathbf{x}_{t+1})$.

Huber [51] derived the update rule (4.29) using a different parametrization which is motivated by the Kalman filter [17]. The Kalman filter recursion performs two steps: inference and update. During the inference step, a prediction at the new input $\mathbf{x}_{t+1}$ is

made. This requires the evaluation of the marginal-likelihood $p(y_{t+1} \mid \mathbf{y}_t)$, according to (4.20), with mean $\hat{f}_{t+1}$ and covariance $\hat{\sigma}_{t+1}$ given as

$$\mathbf{J}_{t+1} = \mathbf{K}(\mathbf{x}_{t+1}, \mathcal{D})\boldsymbol{\Sigma}_0^{-1} , \tag{4.30a}$$

$$\hat{f}_{t+1} = \mathbf{J}_{t+1}\boldsymbol{\mu}_t , \tag{4.30b}$$

$$\hat{\sigma}_{t+1}^2 = k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) + \mathbf{J}_{t+1}(\boldsymbol{\Sigma}_t - \boldsymbol{\Sigma}_0)\mathbf{J}_{t+1}^{\mathrm{T}} , \tag{4.30c}$$

where $\mathbf{J}_{t+1}$ is introduced to simplify the notation. During the update step, the information of the new sample $(\mathbf{x}_{t+1}, y_{t+1})$ is incorporated. Thus, the conditioned joint probability (4.28) is calculated and then the marginalization (4.27) is performed. This leads to the update rule

$$\mathbf{G}_{t+1} = \boldsymbol{\Sigma}_t\mathbf{J}_{t+1}^{\mathrm{T}}\left(\hat{\sigma}_{t+1}^2 + \sigma_n^2\right)^{-1} , \tag{4.31a}$$

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \mathbf{G}_{t+1}(y_{t+1} - \hat{f}_{t+1}) , \tag{4.31b}$$

$$\boldsymbol{\Sigma}_{t+1} = \boldsymbol{\Sigma}_t - \mathbf{G}_{t+1}\mathbf{J}_{t+1}^{\mathrm{T}}\boldsymbol{\Sigma}_t , \tag{4.31c}$$

where $\mathbf{G}_{t+1}$ is known as Kalman gain. Since the dictionary $\mathcal{D}$ is fixed over time, the inverse in (4.30a) can be computed offline. This can be done efficiently by Cholesky factorization, because the covariance matrix $\boldsymbol{\Sigma}_0$ is symmetric and positive definite.

# 5 Evaluation

The online function approximators, introduced in Section 3.3 and Section 4.3, will now be applied to representative data from the electric-drive domain. The data-based approximation of the flux-linkage of a permanent-magnet synchronous-motors (PMSM) is discussed in Section 5.1. The application scenario is motivated in Section 5.1.1. This includes the introduction of a dynamic PMSM model and a common control approach, know as the Maximum Torque Per Ampere (MTPA) strategy. Furthermore, a measurement noise model is derived and the influence of a changing operating condition is discussed. The offline hyperparameter-tuning of the flux-linkage model is discussed in Section 5.1.2. An academic comparison of the online function approximators based on random sampled data is given in Section 5.1.3. A practice-oriented evaluation of the approximators based on artificial operating data follows in Section 5.1.4. Section 5.2 discusses the data-based approximation of a average model of a Voltage Source Inverter (VSI). The section starts by motivating the application scenario. Offline hyperparameter-tuning is discussed in Section 5.2.2. The section closes by evaluating the performance of the approximators based on sequential measurement data.

## 5.1 Permanent Magnet Synchronous Motor

For many industrial applications, PMSMs in combination with a voltage source inverter (VSI) are the standard drive solution [3, 4]. The VSI supplies the three-phase stator windings of PMSM with currents of adjustable magnitude and frequency. The currents generate a rotating magnetic field in the stator which is magnetically linked with the rotor field, generated by the permanent-magnets in the rotor. The magnetic flux-linkage causes the rotor to rotate with the stator field. Thus, the magnetic flux-linkage is an important physical quantity of the PMSM, whose data-based approximation will be discussed in this section.

### 5.1.1 Scenario Description

Usually, the dynamics of the PMSM are described in the rotor-fixed $dq$-reference frame. When dealing with distinct anisotropy in the stator (e. g., single-tooth winding) or rotor (e. g., due to burried magnets), the rotor-fixed flux-linkages $\boldsymbol{\Psi}_{dq}^{\mathrm{T}}(\mathbf{i}_{dq}, \varphi_e) = \begin{bmatrix} \Psi_d & \Psi_q \end{bmatrix}$ are nonlinear functions which depend on the rotor position $\varphi_e$ and the currents $\mathbf{i}_{dq}^{\mathrm{T}} = \begin{bmatrix} i_d & i_q \end{bmatrix}$ in the $dq$-reference frame [3, 4]. However, in this work, the dependency on the rotor position $\varphi_e$ is neglected and only the effect of current dependent magnetic saturation will be further discussed. Thus, the dynamics of the rotor-fixed flux-linkages are described by

the nonlinear differential equations [1, Section 4.3]

$$\frac{\mathrm{d}}{\mathrm{d}t}\Psi_d(i_d, i_q) = -Ri_d + \omega_e \Psi_q(i_d, i_q) + v_d \ , \tag{5.1a}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\Psi_q(i_d, i_q) = -Ri_q - \omega_e \Psi_d(i_d, i_q) + v_q \ . \tag{5.1b}$$

Here, $R$ is the stator resistance, $\omega_e = \frac{\mathrm{d}}{\mathrm{d}t}\varphi_e$ is the electrical speed and $\mathbf{v}_{dq}^{\mathrm{T}} = \begin{bmatrix} v_d & v_q \end{bmatrix}$ are the voltages in the $dq$-reference frame. The generated torque $\tau(i_d, i_q)$ of the PMSM is a nonlinear function of the currents and flux-linkages [1, Section 4.3], given as

$$\tau(i_d, i_q) = \frac{3}{2} n_p (\Psi_d(i_d, i_q) i_q - \Psi_q(i_d, i_q) i_d) \ , \tag{5.2}$$

where $n_p$ is the number of pole paris. The nominal operating range of the motor is characterized by $\|\mathbf{i}_{dq}\|_2 \leq I_n$, where $I_n$ is referred to as nominal current. In nominal operating range, the nonlinear torque equation (5.2) is usually linearized at $\mathbf{i}_{dq} = \mathbf{0}$, which leads to the linearized torque model

$$\tau(i_d, i_q) = \frac{3}{2} n_p (\Psi_m i_q + (L_d - L_q) i_d i_q) \ . \tag{5.3}$$

Here, $\Psi_m = \Psi_d(0, 0)$ is the permanent-magnet flux, $L_d = \frac{\partial}{\partial i_d}\Psi_d(i_d, i_q)|_{\mathbf{i}_{dq}=\mathbf{0}}$ is the nominal inductance in $d$-direction and $L_q = \frac{\partial}{\partial i_q}\Psi_q(i_d, i_q)|_{\mathbf{i}_{dq}=\mathbf{0}}$ is the nominal inductance in $q$-direction. The nominal parameters of the PMSM are assumed to be known and are summarized in Table A.1.

The effect of magnetic saturation is modeled by the nonlinear function $\mathbf{\Psi}_{dq}(\mathbf{i}_{dq})$. From literature and experiments, prior knowledge about the flux-linkage is typically available. In the nominal operating range the flux-linkages $\mathbf{\Psi}_{dq}$ are expected to be in good approximation linear, whereas outside the nominal operating range, the flux-linkages saturate and thus levels off. In order to improve the quality of the dynamic flux-linkage model (5.1) as well as the torque model (5.2), an accurate model of the nonlinear function $\mathbf{\Psi}_{dq}(\mathbf{i}_{dq})$ is desired.

In this study, a Magnetic Equivalent Circuit (MEC) model [3, 4] of a PMSM is used to generate the training data for the function approximators. From the MEC model the nonlinear flux-linkages $\tilde{\mathbf{\Psi}}_{dq}(\mathbf{i}_{dq}, \varphi_e)$ are obtained. The dependency on the rotor position $\varphi_e$ is eliminated by averaging w.r.t. a full electrical revolution, according to

$$\mathbf{\Psi}_{dq}(\mathbf{i}_{dq}) = \frac{1}{2\pi} \int_0^{2\pi} \tilde{\mathbf{\Psi}}_{dq}(\mathbf{i}_{dq}, \varphi_e) \, \mathrm{d}\varphi_e \ . \tag{5.4}$$

The input-space $\mathbb{X}$ of the flux-linkage model $\mathbf{\Psi}_{dq}(\mathbf{i}_{dq})$ is assumed to be constrained by the maximum current $I_{max}$ of the PMSM, defined as

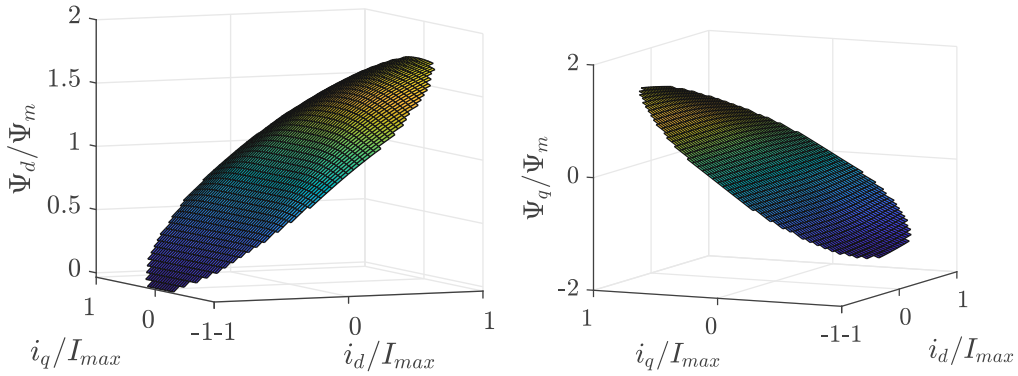
$$\mathbb{X} = \left\{ \mathbf{i}_{dq} \in \mathbb{R}^2 \mid \|\mathbf{i}_{dq}\|_2 \leq I_{max} \right\} \ . \tag{5.5}$$

Furthermore, the MEC model is used to evaluate the performance of the function approximators. All experiments will be performed with normalized[1] data. A plot of the normalized flux-linkages $\Psi_d(i_d, i_q)$ and $\Psi_q(i_d, i_q)$ from the MEC model is shown in Figure 5.1.

The flux-linkages $\mathbf{\Psi}_{dq}$ are static functions, hence the order in which the data is presented to the function approximator, is irrelevant. In order to obtain optimal training results, random sampling is commonly used to generate data, since it well covers the input-space $\mathbb{X}$. Therefore, a uniform random sampled set of inputs $\mathcal{X}$ and corresponding observation $\mathbf{y}$ is defined, according to

$$\mathcal{X} = \{\mathbf{i}_{dq,i} \sim \mathcal{U}_{\mathbb{X}} \mid i = 1, \dots, N\} \qquad \text{and} \qquad \mathbf{y} = \{\Psi_d(\mathbf{i}_{dq,i}) + \epsilon \mid \mathbf{i}_{dq,i} \in \mathcal{X}\}\ , \tag{5.6}$$

where $\mathbf{i}_{dq,i} \sim \mathcal{U}_{\mathbb{X}}$ denotes uniform randoms samples from the input-space $\mathbb{X}$, according to (5.5), and $\epsilon$ is a noise model. However, if the flux-linkage is estimated from measurements during online operation, random sampling is not possible since the PMSM shall not be operated at random torques. Furthermore, energy inefficient operating points, where a large amount of dissipative-power is produced, have to be avoided because the resulting thermal load may damage the motor. Thus, in practice training data can only be gathered in a subset $\mathbb{X}_c \subseteq \mathbb{X}$ of the input space. In order to obtain a realistic subset $\mathbb{X}_c$, a typical operating strategy of PMSMs will be discussed next.




(a) Flux-linkage $\Psi_d(i_d, i_q)$. (b) Flux-linkage $\Psi_q(i_d, i_q)$.

Figure 5.1: Normalized flux-linkages $\Psi_d$ and $\Psi_q$ from the MEC model.

### Maximum Torque Per Ampere Strategy

For many practical applications, a common objective is to control the generated torque $\tau(i_d, i_q)$ of the motor [8, 55]. In order to obtain a current vector $\mathbf{i}^*_{dq}$ for a desired torque $\tau^*$, the inverse mapping of the torque equation (5.2) has to be computed. This is not trivial since the inverse map is under-determined, i. e., there are multiple currents vectors which lead to the same torque. A common approach to obtain an unique current vector is to

[1] The flux-linkage $\mathbf{\Psi}_{dq}(\mathbf{i}_{dq})$ is normalized to the permanent-magnet flux $\Psi_m$ and the currents $i_d$ and $i_q$ are normalized to the maximum current $I_{max}$, according to Table A.1.

minimize the ohmic losses in the machine. This is achieved by formalizing the optimization problem

$$\mathbf{i}_{dq}^*(\tau^*) = \arg\min_{\mathbf{i}_{dq}} \quad \frac{1}{2}\|\mathbf{i}_{dq}\|_2^2 \,, \tag{5.7a}$$

$$\text{subject to} \quad \tau(\mathbf{i}_{dq}) = \tau^* \,. \tag{5.7b}$$

This approach is known as the Maximum Torque Per Ampere (MTPA) strategy $\mathbf{i}_{dq}^*(\tau^*)$. The MTPA curve is depicted in Figure 5.2 with the nonlinear (5.2) as well for the linearized (5.3) torque equation. In the nominal operating range of the motor, the nonlinear MTPA curve is well approximated by the linearized MTPA curve. Outside the nominal operating range, the effect of magnetic saturation is not negligible, which causes a deviation between the linearized MTPA curve and the nonlinear MTPA curve.

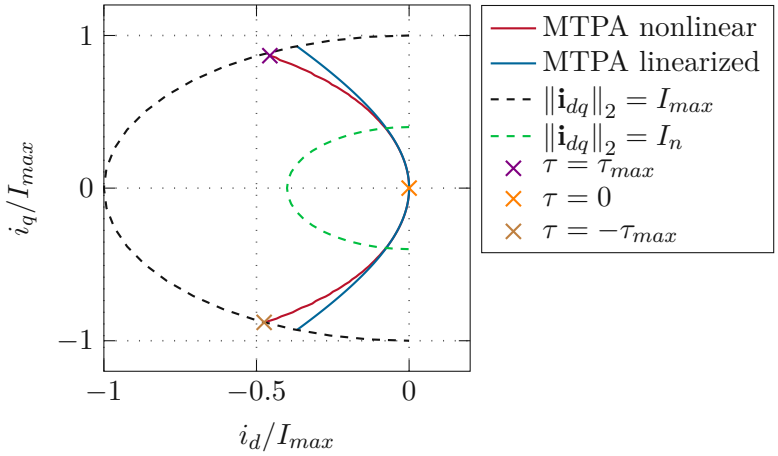



Figure 5.2: MTPA curve for the nonlinear (5.2) and linearized (5.3) torque equation.

The computation of the MTPA strategy (5.7) requires knowledge of the nonlinear flux-linkages, which are often unknown in practice. However, since the nominal parameters are assumed to be known and the linearized MTPA curve is relatively close to the nonlinear MTPA curve, see Figure 5.2, training data can be gathered in a region $\mathbb{X}_c$ around the linearized MTPA curve. It is therefore of special interest how to obtain data in the region $\mathbb{X}_c$ and how the online approximators perform with such training data.

The procedure $\tilde{\mathbf{i}}_{dq} = \mathcal{P}(\tau^*(t))$ to generate training inputs $\tilde{\mathbf{i}}_{dq} \in \mathbb{X}_c$ in the region $\mathbb{X}_c$ proceeds as follows. The procedure takes a desired torque $\tau^*$ as input and calculates the corresponding current vector $\mathbf{i}_{dq}^*(\tau^*)$ according to the MTPA strategy (5.7) using the linearized torque equation (5.3). Samples near the linearized MTPA curve are obtained by adding a random term to the current $i_d^*$, according to

$$\tilde{i}_d = i_d^* + \Delta i_d(\tau^*)\mathcal{U}_{[-1,1]} \,, \tag{5.8}$$

where $\mathcal{U}_{[-1,1]}$ denotes a uniform-distributed random number in the range $[-1, 1]$. Outside the nominal operating range, a larger deviation between the nonlinear and linearized

MTPA curve is expected, see Figure 5.2. Therefore $\Delta i_d(\tau^*)$ is set to small values in the nominal range and to larger values outside to nominal range. The utilized scaling factor is depicted in Figure 5.3. Of course, the pair $(\tilde{i}_d, i_q^*)$, would lead to a deviation in the provided torque $\tau^*$. Therefore, it is necessary to also adjust the set point for $i_q$. This can be done by using the linearized torque equation (5.3), which leads to

$$\tilde{i}_q = \frac{\tau^*}{\frac{3}{2} n_p \Big( \Psi_m + (L_d - L_q) \tilde{i}_d \Big)} \ . \tag{5.9}$$

In order to obtain representative data from the procedure $\mathcal{P}(\tau^*(t))$, a representative torque trajectory $\tau^*(t)$ has to be defined. In practice, the trajectory $\tau^*(t)$ differs according to the considered application scenario. However, independent of the application, a PMSM is usually operated most of in time in the nominal range and rarely at maximum torque. Therefore a representative torque trajectory as depicted in Figure 5.4 is defined in which the torque is continuously varied in the range $\tau^* = [-\tau_{max}, \tau_{max}]$, according to an exponential function

$$\tau^*(t) = \frac{\exp\Big(\frac{t}{\sigma_\tau^2}\Big) - 1}{\exp\Big(\frac{T_e}{4\sigma_\tau^2}\Big) - 1} \qquad \text{with} \qquad t \in \left[0, \frac{1}{4} T_e\right] \ . \tag{5.10}$$

The parameter $\sigma_\tau$ controls the slope of torque trajectory $\tau^*(t)$ and $T_e$ defines the end time. In order to generate $\approx 68\,\%$ of the torque samples in the nominal range, characterized by $|\tau^*| < \tau_n$, the parameter $\sigma_\tau$ in (5.10) is set to $\sigma_\tau = \tau_{max} \tau_n^{-1}$. Thus, the dataset with inputs $\mathcal{X}_\mathcal{P}$ and observations $\mathbf{y}_\mathcal{P}$ is defined as

$$\mathcal{X}_\mathcal{P} = \{ \mathbf{i}_{dq,i} = \mathcal{P}(\tau^*(t_i)) \mid t_i \in \Omega_N[0, T] \} \ , \tag{5.11a}$$

$$\mathbf{y}_\mathcal{P} = \Big\{ \Psi_d(\mathbf{i}_{dq,i}) + \mathcal{N}\Big(0, \sigma_n^2\Big) \mid \mathbf{i}_{dq,i} \in \mathcal{X}_\mathcal{P} \Big\} \ , \tag{5.11b}$$

where $\Omega_n[a, b] = \Big\{ a + i \frac{b-a}{n} \mid i = 0, 1 \ldots n-1 \Big\}$ defines a grid of $n$ equidistant points in the range $[a, b]$. Figure 5.5 shows the samples in the region $\mathbb{X}_\mathcal{P}$ generated by the procedure $\mathcal{P}(\tau^*(t))$ and a torque trajectory as in Figure 5.4. Most of the samples are generated in the nominal operating range. Furthermore, it is visible that the manipulation of set-points according to (5.8) leads to a good coverage of the nonlinear MTPA curve.

### Measurement Model

In practice, the flux-linkages $\mathbf{\Psi}_{dq}$ have to be calculated from measurements during motor operation. In order to generate realistic training data, a measurement model which quantifies the uncertainty in the calculation of $\mathbf{\Psi}_{dq}$ will be derived next. An approach to obtain the flux-linkages from measurements is based on the steady-state of the model (5.1). In the steady-state, i. e., $\frac{\mathrm{d}}{\mathrm{d}t} \mathbf{\Psi}_{dq}(\mathbf{i}_{dq}) = \mathbf{0}$, the flux-linkages can be expressed as

$$\Psi_d = \frac{-R i_q + v_q}{\omega_e} \qquad \text{and} \qquad \Psi_q = \frac{R i_d - v_d}{\omega_e} \ . \tag{5.12}$$

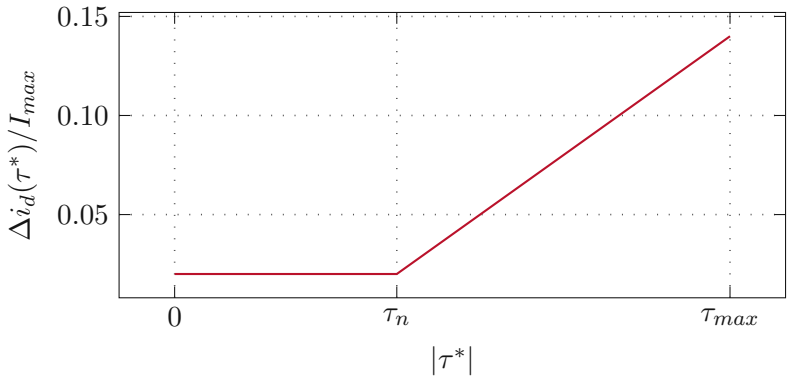



Figure 5.3: Torque dependent scaling factor $\Delta i_d(\tau^*)$ according to (5.8).

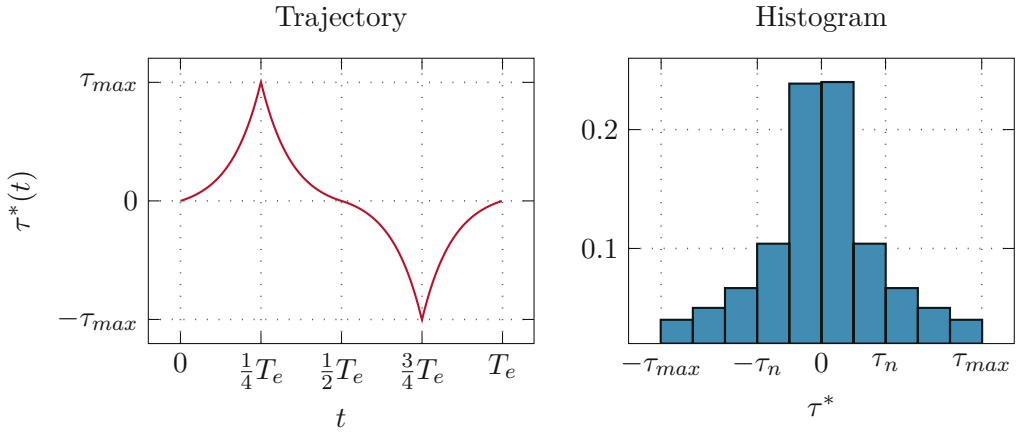



Figure 5.4: Torque trajectory and corresponding histogram of torque samples.

For the sake of compactness, the following discussion will solely focus on the $d$-component of the flux-linkage. The flux-linkage in $d$-direction is a nonlinear function $\Psi_d = g(i_q, R, \omega_e, v_q)$ with arguments $x_i \in \{i_q, R, \omega_e, v_q\}$. Suppose that the quantities $x_i$ can be measured with a variance of $\sigma^2_{x_i}$. The uncertainty in the measurement variables $x_i$ results in an uncertainty in the estimate of the flux-linkage $\Psi_d$. A model of the measurement variance $\sigma^2_{\Psi_d}$ can be obtained with an error propagation analysis [56, Section 3.3] according to

$$\sigma^2_{\Psi_d} = \sum_i \left(\frac{\partial g}{\partial x_i}\right)^2 \sigma^2_{x_i} \ . \tag{5.13}$$

Applying (5.13) to (5.12) yields the propagation of uncertainties for the flux-linkage $\Psi_d$ as

$$\sigma^2_{\Psi_d} = \left(\frac{R}{\omega_e}\right)^2 \sigma^2_{i_q} + \left(\frac{i_q}{\omega_e}\right)^2 \sigma^2_R + \left(\frac{Ri_q - v_q}{\omega_e^2}\right)^2 \sigma^2_{\omega_e} + \left(\frac{1}{\omega_e}\right)^2 \sigma^2_{v_q} \ . \tag{5.14}$$

Obviously, the uncertainties depend on the actual operating point of the PMSM. For a desired operating point $(\tau^*, \omega_e^*)$, the currents $i_d^*$ and $i_q^*$ are obtained according to the

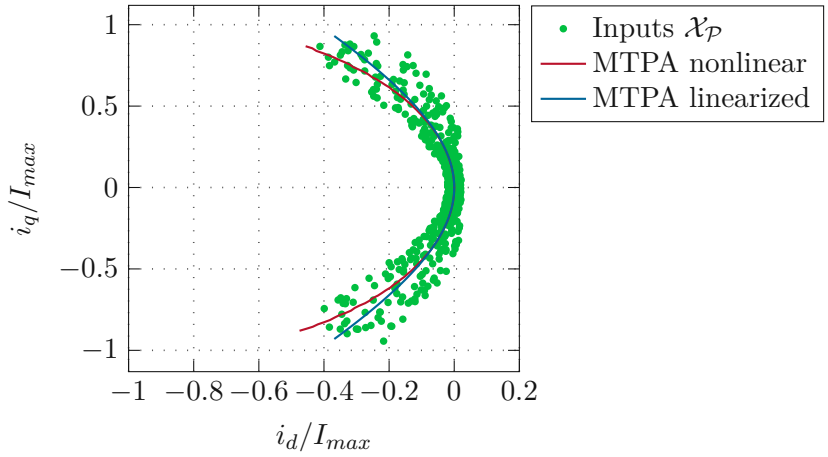



Figure 5.5: Input samples $\mathcal{X}_\mathcal{P}$ in region $\mathbb{X}_c$ near the linearized MTPA curve by the procedure $\mathcal{P}(\tau)$, according to (5.11)

MTPA strategy $\mathbf{i}^*_{dq}(\tau^*)$. The voltage $v_q$ is calculated with the steady-state equation (5.12) which results in

$$\begin{bmatrix} i_d^* & i_q^* \end{bmatrix}^{\mathrm{T}} = \mathbf{i}^*_{dq}(\tau^*) \qquad \text{and} \qquad v_q = \omega_e^* \Psi_d(i_d^*, i_q^*) + R i_q^* \,. \tag{5.15}$$

The modelled standard deviation $\sigma_{\Psi_d}$ as a function of the rotor speed $\omega_e$ is shown in Figure 5.6 for different torque set points. The assumed measurement uncertainties $\sigma^2_{x_i}$ are defined in Table A.2. Figure 5.6 shows, that the uncertainty decreases with increasing rotor speed. Thus, in order to obtain a reliable approximation, the flux-linkages should be determined at the highest possible rotor speed. The slight increase in the modelled uncertainty for higher torques can be tolerated. In statistics [57], noise which changes as function of the inputs, i. e., $\sigma_{\Psi_d} = \sigma_{\Psi_d}(i_d, i_q, \omega_e)$, is referred to as heteroscedastic noise. The noise standard deviation $\sigma_{\Psi_d}(i_d, i_q, \omega_e)$ as function of the currents is depicted in Figure 5.7 for maximum speed $\omega_e = \omega_{e,max}$. The standard deviation increases with increasing magnitude $|i_q|$ and $i_d$.

For all following simulation experiments, it is assumed that for hyperparameter-tuning and model training only noisy observations $y$ of the latent function $\Psi_d(i_d, i_q)$ are available. Thus, the approximators are always trained with data, generated by the MEC model $\Psi_d(i_d, i_q)$, with artificially added Gaussian noise, according to the measurement model (5.14). First, experiments with Gaussian noise of constant standard deviation, according to

$$y = \Psi_d(i_d, i_q) + \mathcal{N}\left(0,\, \sigma_n^2\right) \,, \tag{5.16}$$

will be performed. The standard deviation of the Gaussian noise is set to $\sigma_n = 20 \cdot 10^{-3}$, which corresponds to the modelled value $\sigma_{\Psi_d}$ at maximum rotor speed $\omega_e = \omega_{e,max}$, see Figure 5.6. Second, an experiment with artificially added heteroscedastic Gaussian noise, according to

$$y = \Psi_d(i_d, i_q) + \mathcal{N}\left(0,\, \sigma^2_{\Psi_d}(i_d, i_q, \omega_e)\right) \,, \tag{5.17}$$

will be performed. In order to obtain on average the smallest noise standard deviation $\sigma_{\Psi_d}(i_d, i_q, \omega_e)$, the simulation experiment is performed at maximum normalized rotor speed $\omega_e = \omega_{e,max}$.

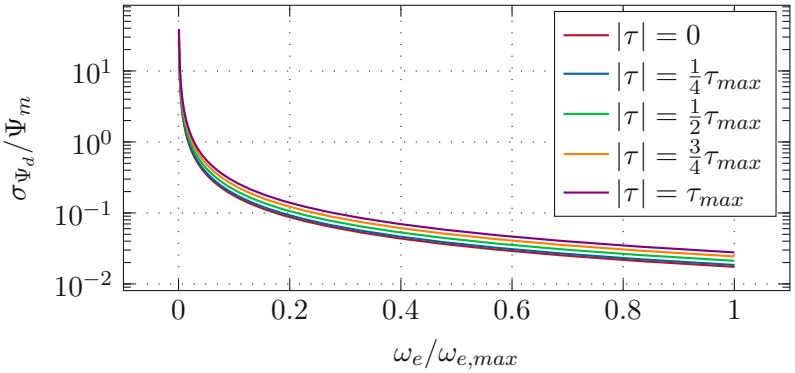



Figure 5.6: Modelled standard deviation $\sigma_{\Psi_d}$ of flux-linage measurement for different torques $\tau$ with varying rotor speed $\omega_e$.

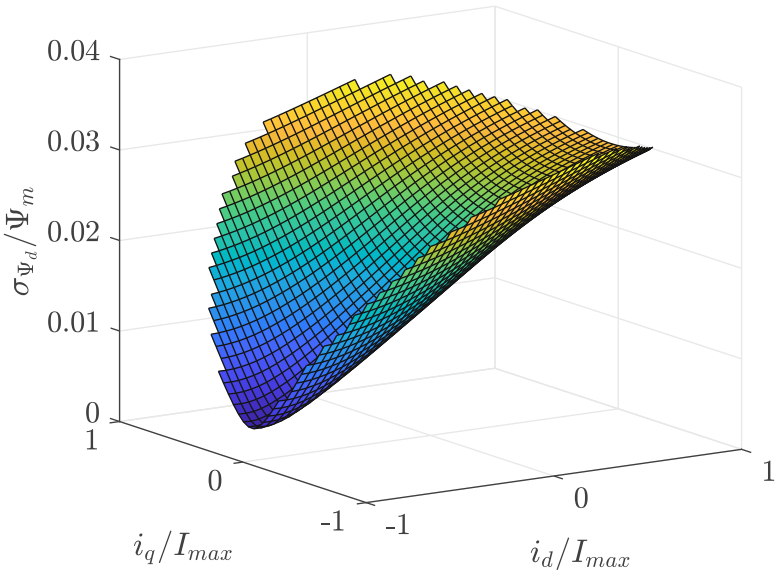



Figure 5.7: Heteroscedastic noise standard deviation $\sigma_{\Psi_d}(i_d, i_q, \omega_e)$, according to (5.14), of the flux-linkage measurement $\Psi_d$ at maximum speed $\omega_e = \omega_{e,max}$.

### Changing Operating Conditions

For use cases, in which the operating condition of the motor is expected to change over time, an online adaption of the flux-linkage model is required. In order to simulate the effect of a changing operating condition, the flux-linkage is scaled by a time-varying factor

$c(t)$, according to

$$\tilde{\Psi}_d(t, i_d, i_q) = c(t)\Psi_d(i_d, i_q) \qquad \text{with} \qquad c(t) \in [0.9, 1.1] \ . \tag{5.18}$$

The scaled flux-linkage is depicted in Figure 5.8 for a constant scaling factor $c(t) = 0.9$. Obviously, scaling affects regions of large $\Psi_d$ values more than region of small $\Psi_d$. Furthermore, scaling slightly changes the curvature of the surface. In order to evaluate test the approximators with a time varying function, an experiment will be performed in which the function approximators are trained with inputs $\mathcal{X}_t$ and corresponding observation $\mathbf{y}_t$, according to

$$\mathcal{X}_t = \{\mathbf{i}_{dq,i} \sim \mathcal{U}_{\mathbb{X}} \mid i = 1, \ldots, N\} \ , \tag{5.19a}$$

$$\mathbf{y}_t = \left\{\tilde{\Psi}_d(t, \mathbf{i}_{dq,i}) + \mathcal{N}\left(0, \sigma_n^2\right) \mid \mathbf{i}_{dq,i} \in \mathcal{X}_t\right\} \ . \tag{5.19b}$$

Here, $\mathbf{i}_{dq,i} \sim \mathcal{U}_{\mathbb{X}}$ denotes uniform random samples from the input-space $\mathbb{X}$.

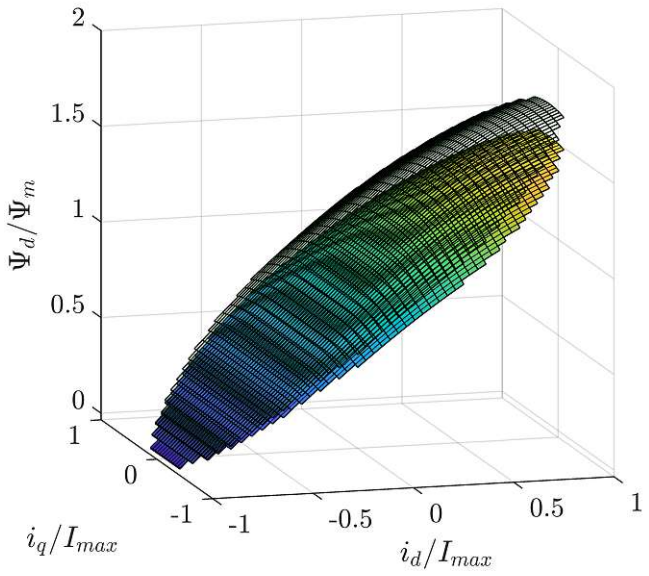



Figure 5.8: Simulated effect of parameter drift according to (5.18). The transparent surface represents the nominal flux-linkage $\Psi_d$. The non-transparent surface represents the flux-linkage $\tilde{\Psi}_d = c(t)\Psi_d$ with a scaling factor $c(t) = 0.9$.

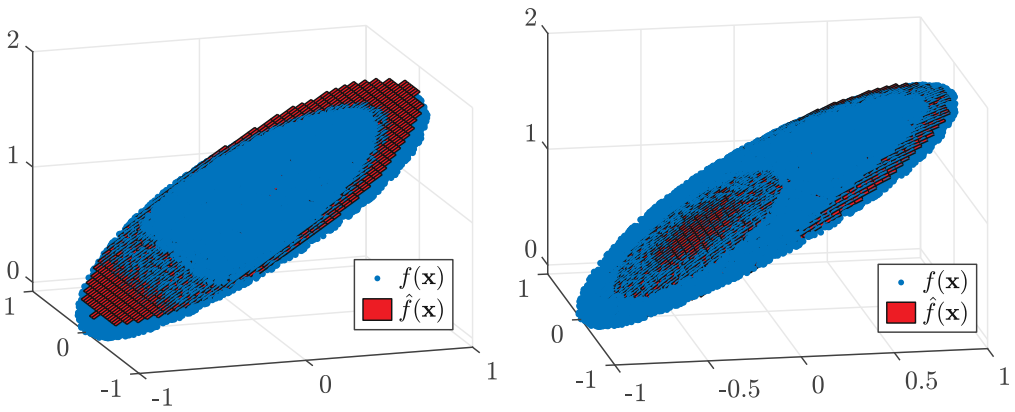
### 5.1.2 Offline Model Selection

The selection of an appropriate kernel function as well as the tuning of the kernel- and noise-parameters will be discussed in this section. Hyperparameter-tuning is performed with uniform random sampled data, according to (5.6) and constant Gaussian noise (5.16). This is possible since the data is obtained from the MEC model.

As discussed in Section 5.1.1, prior knowledge about the flux-linkage is available, which can be used to guide the selection of an appropriate kernel function[2]. Since the flux-linkage $\Psi_d(i_d, i_q)$ is in good approximation linear in $d$-direction for the nominal operating range, a linear kernel, according to (3.22) with $d = 1$, would be an obvious choice. Outside the nominal range, the flux-linkage levels off, therefore a quadratic kernel, according to (3.22) with $d = 2$, may result in a reasonable approximation. However, from a computational point of view, the use of a polynomial kernel is not recommended, since a polynomial basis function model can be trained more efficiently, see Section 3.2. A prediction with the linear and quadratic kernel is shown in Figure 5.9. Although the quadratic kernel already results in a prediction error at noise level, the kernel is not flexible enough to approximate the true underlying function. In order to increase the flexibility of the approximator, a linear kernel is combined with a Gaussian kernel, according to

$$k(\mathbf{x}, \mathbf{x}') = \sigma_{v1}^2 (x_1 x_1' + c) + \sigma_{v2}^2 \exp\left(-\frac{1}{2l^2}\|\mathbf{x} - \mathbf{x}'\|_2^2\right) . \tag{5.20}$$

The linear-kernel captures the linear trend in $d$-direction, whereas the Gaussian-kernel smoothly models the residual errors. Furthermore, the linear kernel improves extrapolation performance of the model in situations where only few data is available. The resulting kernel has four hyperparameters $\boldsymbol{\theta}_k^{\mathrm{T}} = \begin{bmatrix}\sigma_{v1} & c & \sigma_{v2} & l\end{bmatrix}$, which are tuned with cross-validation (CV) and maximum marginal-likelihood (MML).




(a) Linear: $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathrm{T}}\mathbf{x} + 1$. (b) Quadratic: $k(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^{\mathrm{T}}\mathbf{x} + 1\right)^2$.

Figure 5.9: Predictions of a Gaussian Process with polynomial kernels (3.22) on PMSM dataset, where $\hat{f}(\mathbf{x})$ is the prediction and $f(\mathbf{x})$ is the latent function.

### Hyperparameter-tuning with Cross-Validation

As discussed in Section 3.2, CV is a general approach to tune the hyperparameters of a static model, which also could be applied to the online function approximators, introduced in Section 3.3 and Section 4.3. However, this may lead to different hyperparameters for

[2]The kernel function is introduced as $k(\mathbf{x}, \mathbf{x}')$ with inputs $\mathbf{x}^{\mathrm{T}} = [x_1 \quad x_2] = [i_d \quad i_q]$

each online approximator and thus makes a comparison between them difficult. Therefore, the argument in [36, Section 4.8] is followed and hyperparameter-tuning is performed offline with a Gaussian process (GP) according to (2.26). Since all introduced online approximators are approximations of a GP, it is argued that the optimal hyperparameters of the GP are also suitable for the online approximators [36, Section 4.8].

The computational complexity of CV to tune four hyperparameter $\boldsymbol{\theta}_k$ is high. In order to reduce the computational complexity, one hyperparameter can be eliminated by exploiting that the prediction equation (2.15) is invariant against rescaling of the kernel function [42]. Thus, the scale of the linear kernel (5.20) is set to $\sigma_{v1} = 1$. The computational complexity can be further reduced by fixing some of the remaining hyperparameters. In order to obtain a smooth solution with less spatial variations, the length-scale of the Gaussian kernel is set to a large value $l = 1$. Furthermore, the constant of the linear kernel is set to $c = 1$ after some trials. The remaining hyperparameter $\sigma_{v2}$ is then tuned with $K$-fold CV.

$K$-fold CV, as in (3.26) with $K = 10$, is used to obtain an estimate of the root mean squared (RMS) prediction error based on random sampled data, according to (5.6). In order to further reduce the dependency on the choice of dataset, the results of 50 trials with different uniform sampled datasets are averaged. The median and the 16 % and 84 % percentiles [18, Section 2.2.6] are used to obtain an estimate of the mean and a 68 % confidence interval. The likelihood of the GP, defined as (2.25), is set to a rather large value of $\sigma_n = 40 \cdot 10^{-3}$ to obtain a smooth approximation. The hyperparameter $\sigma_{v2}$ is tuned by a grid search of 50 equally distributed values in the range $[0.1, 2]$. The hyperparameters $\sigma_{v1} = 1$, $c = 1$ and $l = 1$ are fixed.

The results of the hyperparameter-tuning for $\sigma_{v2}$ are depicted in Figure 5.10, for a small $N = 50$ and large $N = 700$ dataset size. The tight confidence intervals of the RMS error for $N = 700$ indicate a reliable estimate of the true noise standard deviation. For the small dataset, the RMS error is slightly above the true noise level and its minimum lies in the range $\sigma_{v2} \in [0.3, 0.6]$. For a large dataset, any $\sigma_{v2}$ in the range $[0.3, 2]$ results in an error near to noise level. In order to obtain a good approximation independent of the dataset size, the hyperparameter is set to $\sigma_{v2} = 0.6$. The obtained kernel hyperparameters are summarized in Table 5.1.

#### Hyperparameter-tuning with Maximum Marginal Likelihood

The hyperparameters $\boldsymbol{\theta}_k$ of the kernel (5.20) as well as the noise standard deviation $\sigma_n$ are now tuned, using the Bayesian model selection approach, introduced in Section 4.2. Following the same arguments as for the CV approach, the hyperparameter-tuning is performed offline with a GP. In the Bayesian framework, the hyperparameters of a GP are tuned by maximizing the marginal-likelihood (4.12). The optimization is performed in MATLAB using the Conjugate-Gradient solver of the GPML Toolbox [46]. The GPML Toolbox also provides the required gradients of the marginal-likelihood w.r.t. to the hyperparameters.

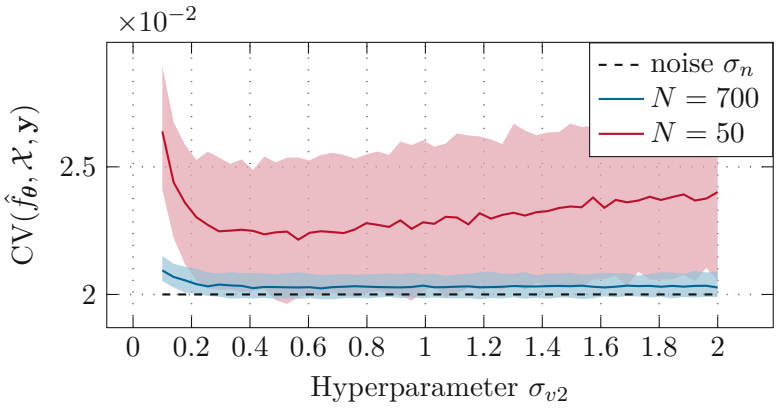



Figure 5.10: Hyperparameter-tuning with 10-fold cross-validation of GP with kernel (5.20).

The results of the Bayesian hyperparameter-tuning with MML are shown in Figure 5.11 for different dataset sizes $N$. In order to reduce the dependency on the choice of the dataset, the results of 50 trials with different uniform sampled datasets are averaged. A histogram plot is used to investigate the distribution of the obtained hyperparameters. The obtained hyperparameter distributions are expected to be not normal distributed. Thus, the median and the 16 % and 84 % percentiles [18, Section 2.2.6] are used to obtain an estimate of the mean and a 68 % confidence interval. The Conjugate Gradient solver is initialized at each trial with $\sigma_n = 1$, $\sigma_{v1} = \sigma_{v2} = 1$ and $l = 2$.

The top graph in Figure 5.11 shows the estimated noise standard deviation $\hat{\sigma}_n$. For a dataset size $N > 400$, the true noise standard deviation $\sigma_n$ is estimated accurately with low variance. The parameter of the linear kernel and the Gaussian kernel are shown in the middle and bottom graph of Figure 5.11, respectively. For a dataset size $N < 400$, the weight $\sigma_{v1}$ of linear kernel is small compared to the weight of the Gaussian kernel and the lengthscale $l$ of the Gaussian kernel is larger. Thus, for $N < 400$ the Gaussian kernel with large lengthscale is dominant. For a dataset size $N > 400$, the weight as well as the lengthscale of the Gaussian kernel decrease and the weight $\sigma_{v1}$ of the linear kernel increases. In this case, the linear kernel is more dominant, indicating that the Gaussian kernel only models the small residual errors.

The confidence intervals of the hyperparameters in Figure 5.11 indicate a reliable estimates for a dataset size $N > 800$.Thus, the hyperparameters are set to median values obtained for $N = 1000$. A histogram of the hyperparameters for $N = 1000$ is shown in Figure 5.12. The distributions of all hyperparameters are fairly peaked, indicating reliable hyperparameter estimates. The MML-tuned hyperparameters are summarized in Table 5.1.

### Comparison

The achievable prediction error of the CV-tuned and MML-tuned kernel is now compared. Therefore a GP is trained with uniform data of different sizes $N$, according to (5.6), and its prediction error is calculated. The true prediction error between the GP prediction $\hat{f}$

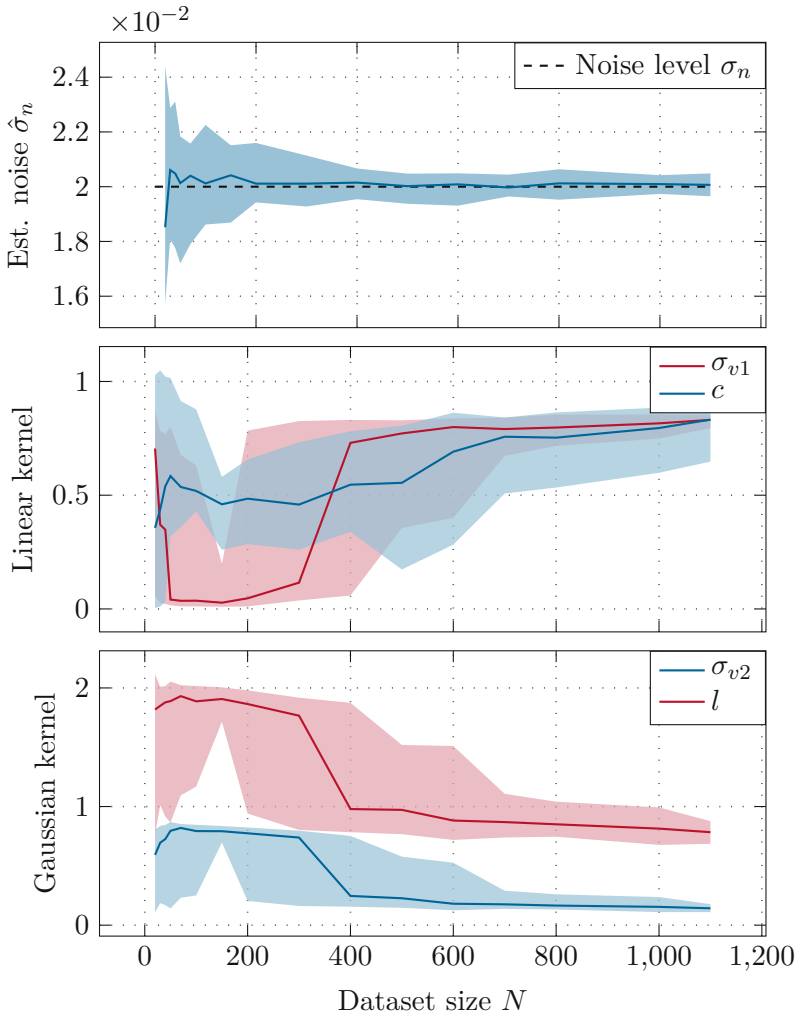



Figure 5.11: Hyperparameter-tuning with MML of GP with kernel (5.20) for different dataset sizes $N$.

and the true flux-linkage $\Psi_d$, according to

$$\mathbf{e} = \hat{\mathbf{f}}(\mathcal{X}_*, \mathcal{X}, \mathbf{y}) - \mathbf{\Psi}_d(\mathcal{X}_*) \ , \tag{5.21}$$

is used as evaluation metric. Here, the notation $\hat{\mathbf{f}}(\mathcal{X}_*, \mathcal{X}, \mathbf{y})$ as in (3.27) is used, where $\mathcal{X}_*$ is a independent uniform sampled test set. The likelihood of the GP is set to the noise level $\sigma_n = 20 \cdot 10^{-3}$ obtained by MML.

As shown in Figure 5.13, for large $N$ both kernels result in a RMS prediction error of $\approx 3 \cdot 10^{-3}$, which is almost a magnitude smaller than noise level $\sigma_n$. Furthermore, the maximum prediction error of both kernels is less than the noise level for $N > 300$. The MML-tuned kernel performs slightly better for all training data sizes. Therefore, the MML kernel is used for all following experiments. The prediction error of the full GP will further serve as a baseline for the online approximators.

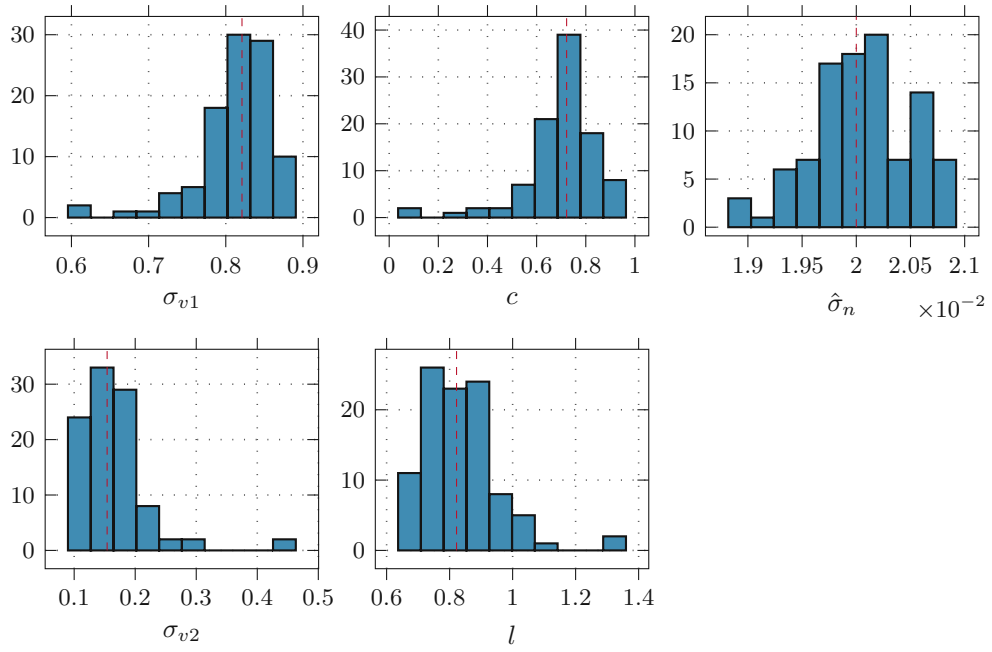



Figure 5.12: Histogram of MML-tuned kernel hyperparameter (5.20) and estimated likelihood $\hat{\sigma}_n$ for a random dataset of size $N = 1000$.

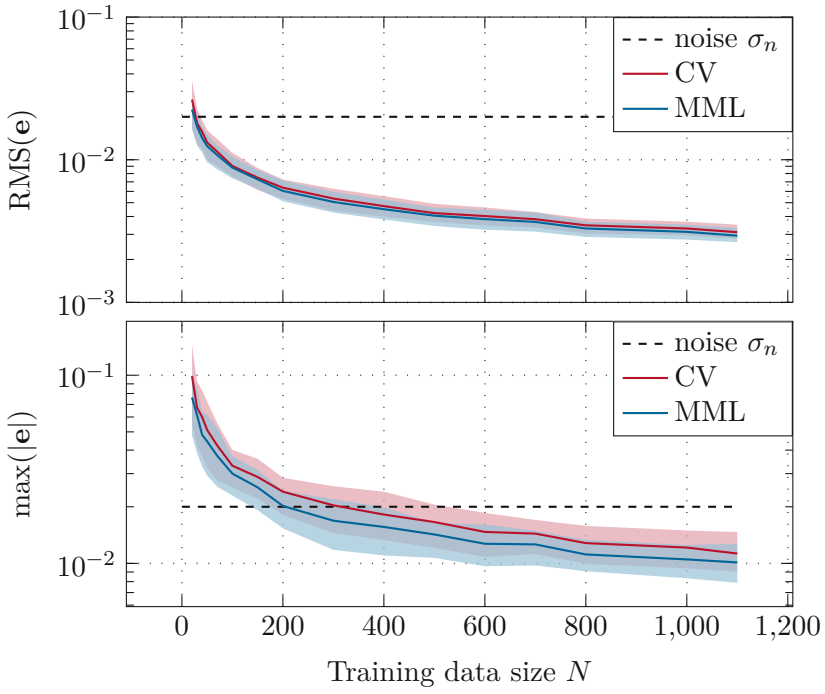



Figure 5.13: True prediction error of a GP with CV-tuned kernel and MML-tuned kernel.

| Method | Kernel Hyperparameter | | | |
|---|---|---|---|---|
| CV | $\sigma_{v1} = 1.000$ | $c = 1.000$ | $\sigma_{v2} = 0.600$ | $l = 1.000$ |
| MML | $\sigma_{v1} = 0.808$ | $c = 0.766$ | $\sigma_{v2} = 0.166$ | $l = 0.820$ |

Table 5.1: Hyperparameters of kernel (5.20) for PMSM dataset.

### 5.1.3 Evaluation on Random Sampled Data

In this section, the accuracy of the online function approximators, introduced in Section 3.3 and Section 4.3, is evaluated based on uniform random sampled data, according to (5.6). Furthermore, the constant noise model, defined in (5.16), is used.

The accuracy of the approximators is evaluated by means of a learning curve. A learning curve shows the prediction error RMS($\mathbf{e}$) of an approximator as a function of the processed training samples $t$. The true prediction error, as defined in (5.21), is used as metric. The prediction error is calculated based on large random sampled test set of size $N_T = 2000$. In order to reduce the influence of the training dataset, the learning curves of 100 trials with different uniform sampled training datasets are averaged. The mean learning curve is obtained by an arithmetic mean whereas a 68 % confidence interval is calculated using the 16 % and 84 % percentiles [18, Section 2.2.6]. The parameters of the approximators are chosen in a way, that the maximum dictionary size of all approximators is $|\mathcal{D}| \approx 16$. The parameter settings of the approximators are summarized in Table 5.2. The learning curve together with the dictionary size $|\mathcal{D}|$ is shown in Figure 5.14 as function of the training samples, whereas the learning curve of the first 200 samples is depicted in Figure 5.15.

The kernel recursive least squares (KRLS) approximator, introduced in Section 3.3.2, results in the smallest RMS prediction error of $\approx 3 \cdot 10^{-3}$ after $t = 5000$ samples. This is close to the accuracy of a full GP, see Figure 5.13. However, only $\approx 1.5\,\%$ of the GPs memory is used. The dictionary size of the KRLS can be tuned with the parameter $\nu$, according to (3.28). For a small $\nu$, the accuracy improves at the cost of a larger dictionary, for large $\nu$ the contrary holds. For a very small $\nu \approx 1 \cdot 10^{-9}$ the coefficient updates KRLS are numerically unstable, see (3.46). In this case large prediction errors are obtained. Although the KRLS results in the smallest mean prediction error after $t = 5000$ samples, the error of the first 150 samples is larger compared to the Bayesian-KRLS (B-KRLS) and the recursive-Gaussian-Process (rec-GP), see Figure 5.15. The larger error in the initial phase may be caused by the smaller dictionary size. The prediction error reaches noise level after $\approx 30$ trainings samples.

The rec-GP approximator, introduced in Section 4.3.2, uses a predefined dictionary of size $m$, defined in (4.26). For each trial, the dictionary is initialized with $m = 16$ uniform random sampled inputs $\mathbf{x} \in \mathbb{X}$. Furthermore, the likelihood (4.17) is set to the estimated value $\hat{\sigma}_n = 20 \cdot 10^{-3}$ of MML, see Figure 5.11. After $t = 5000$ training samples, a mean RMS error of $\approx 4 \cdot 10^{-3}$ is reached. The error is slightly above the full GP and the KRLS. As shown in Figure 5.15, the prediction error reaches noise level after $\approx 20$ trainings samples, which is roughly 10 samples earlier than the KRLS.

The B-KRLS, introduced in Section 4.3.1, uses a sparsification criterion which allows to predefine the maximum size of the dictionary. Whenever the maximum size is reached, the least significant element, according to criterion (4.24), is removed from the dictionary. The likelihood (4.17) is again set to the MML estimated value $\hat{\sigma}_n = 20 \cdot 10^{-3}$. The B-KRLS reaches mean RMS error of $\approx 7 \cdot 10^{-3}$ after $t = 5000$ training samples. As shown in Figure 5.15, noise level is reached after $\approx 20$ trainings samples.

The kernel affine projection (KAP) approximator, introduced in Section 3.3.3, shows the worst prediction accuracy with a minimum mean RMS error slightly below noise level. The KAP updates its coefficients by a stochastic approximation based on $q$ recent training samples. A value of $q = 20$ is used, since larger values did not significantly improved the prediction error. The learning rate $\eta$, according to (3.55), is used to control the learning speed of the KAP. For a large $\eta$, the algorithm learns faster, but also becomes more prune to noise. The coherence parameters $\mu_0$, defined in (3.31), controls the size of the dictionary. For small $\mu_0$, the dictionary remains small but the prediction error is large. For larger $\mu_0$, the dictionary size increases. For a too large $\mu_0$, the dictionary size increases drastically, often without significantly improving the prediction performance. As shown in Figure 5.15 noise level is reached after $\approx 540$ trainings samples, which is significantly slower compared to the other approximators.

| Approximator | Section | Parameters |
|---|---|---|
| KRLS | 3.3.2 | $\nu = 3 \cdot 10^{-4}$ |
| KAP | 3.3.3 | $\mu_0 = 0.5275$ $q = 20$<br>$\epsilon = 1 \cdot 10^{-6}$ $\eta = 0.05$ |
| B-KRLS | 4.3.1 | $m = 16$ $\sigma_n = 20 \cdot 10^{-3}$ |
| rec-GP | 4.3.2 | $m = 16$ $\sigma_n = 20 \cdot 10^{-3}$ |

Table 5.2: Parameter settings of online function approximators for the PMSM dataset.

### 5.1.4 Evaluation on Artificial Operating Data

In this section, the performance of the approximators is evaluated according to the application specific scenarios described in Section 5.1.1. To begin with, the approximators are evaluated with training data in the neighborhood of the MTPA curve. Then, an experiment with heteroscedastic noise is performed and the influence of slowly changing operating conditions is analyzed. Furthermore, the prediction uncertainty provided by the Bayesian-based approximators is discussed. For all experiments, the MML-tuned kernel, with hyperparameter according to Table 5.1 and the same approximator setting as for random data, see Table 5.2 are used.

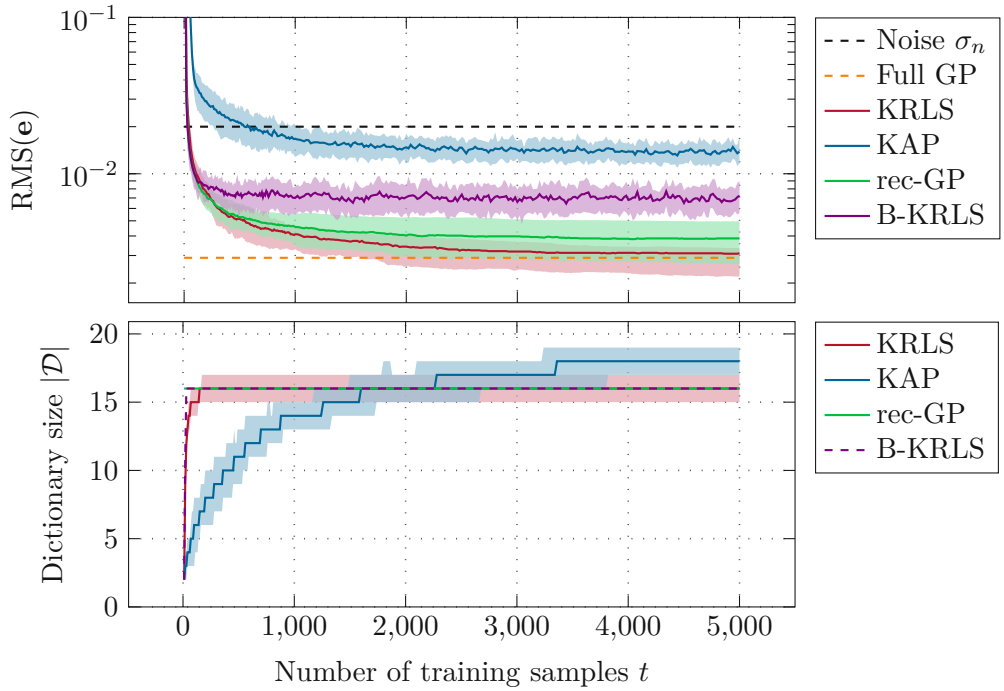



Figure 5.14: Learning curves for uniform random sampled data.

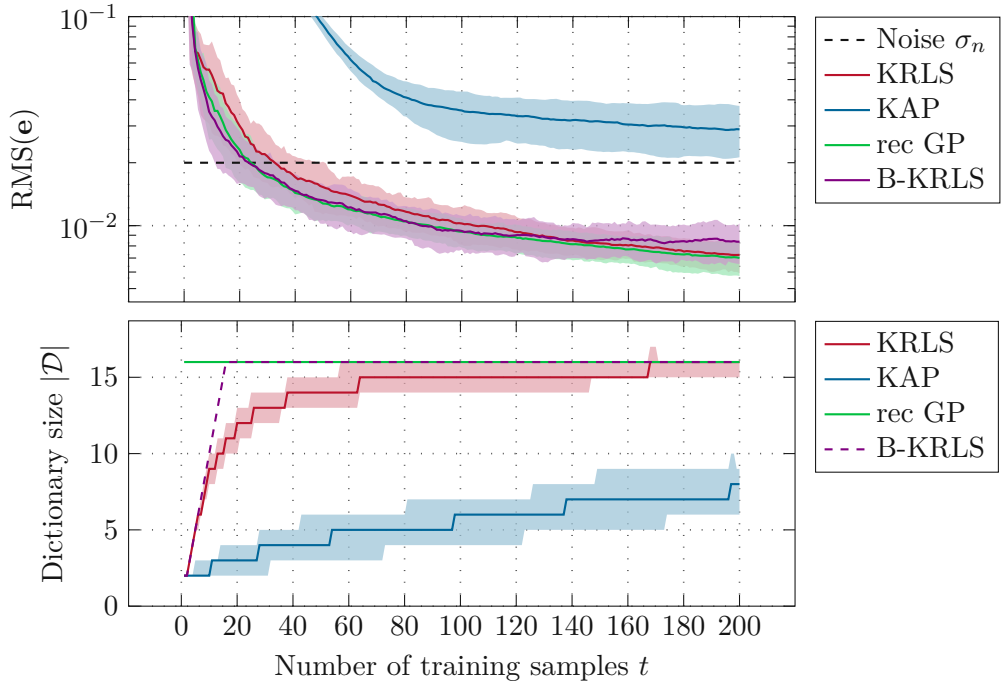



Figure 5.15: Initial learning curves for uniform random sampled data.

### MTPA Curve

The performance of the online approximators is first evaluated with data near the MTPA curve according to (5.11). The training data is generated to obtain a torque trajectory as depicted in Figure 5.4. Since the torque is continuously varied, successive training samples are expected to be relatively close in the input-space. The effect of such training data on the performance of the approximators will be studied next.

The performance of the approximators is evaluated with a learning curve, where the prediction error is defined as

$$\mathbf{e} = \hat{\mathbf{f}}(\mathcal{X}_\mathcal{N}, \mathcal{X}_\mathcal{P}, \mathbf{y}_\mathcal{P}) - \mathbf{\Psi}_d(\mathcal{X}_\mathcal{N}) \ . \tag{5.22}$$

Here, $\mathcal{X}_\mathcal{N}$ is an independent set of test inputs, which are Gaussian-distributed in $d$-direction around the nonlinear MTPA curve, according to

$$\mathcal{X}_\mathcal{N} = \left\{ \mathbf{i}^*_{dq}(\tau^*) + \begin{bmatrix} \mathcal{N}(0,\, \sigma_m^2) \\ 0 \end{bmatrix} \mid \tau_i^* \in \Omega_n[-\tau_{max}, \tau_{max}] \right\} . \tag{5.23}$$

The notation $\Omega_n[-\tau_{max}, \tau_{max}]$ refers to a grid of $n$ equidistant points in the range $[-\tau_{max}, \tau_{max}]$. The parameter $\sigma_m$ in (5.23) is used to control the spread of the samples around the MTPA curve. In order to reduce the dependency of noise, the learning curves of 100 trials with the same dataset but different noisy observations are averaged. The learning curve for $N = 5000$ is shown in Figure 5.16 whereas the learning curve of the first 400 samples is depicted in Figure 5.17.

The KRLS reaches a mean RMS error of $\approx 3 \cdot 10^{-3}$ after 5000 samples. Compared to the experiment with random data, shown in Figure 5.15, the dictionary of the KRLS grows much slower and only reaches a maximum value of $|\mathcal{D}| = 11$. This is a consequence of the ALD sparsification criterion (3.28), which only adds elements to the dictionary whenever the minimum kernel-induced distance between the new input and the present dictionary exceeds $\nu$. Since the training data covers only a subspace $\mathbb{X}_c$ of the input-space $\mathbb{X}$ and successive sample are relatively close, the ALD criterion only slowly increases the dictionary. Furthermore, it is observed that each time the KRLS adds an element to the dictionary, the RMS error increases. Roughly 100 samples are needed till the error is decayed to the previous value. This behavior can be fixed by increasing of $\nu$. However, increasing $\nu$ comes at the cost of a larger final prediction error. Although the RMS error of the KRLS is slightly larger than the B-KRLS and the rec-GP, the KRLS learns faster, see Figure 5.17. Noise level is reached after $\approx 200$ samples, which is roughly 150 samples faster than the B-KRLS and the rec-GP.

The rec-GP and the B-KRLS perform best with a mean RMS error of $\approx 2 \cdot 10^{-3}$ after 5000 samples, see Figure 5.16. For both approximators, the learning curves shows a peak after $\approx 500$ samples. Besides of the peak, the error of both approximators tends to decrease as more data is observed. Both approximators reach noise level after $\approx 350$ samples, see Figure 5.17.

The KAP shows the worst performance with a prediction error above noise level. The prediction error increases and decreases over time and does not converge as more data is observed. The KAP updates its coefficients by a stochastic approximation based on the $q$ most recent training samples. Since successive training samples are relatively close in the input-space, the $q$ most recent training samples are only a poor approximation of the function in range $\mathbb{X}_c$. Therefore, the KAP is not suited for this scenario. Also for a large value of $q = 200$ the performance did not significantly improve.

The absolute prediction error $|e(i_d, i_q)|$ over the input-space $\mathbb{X}$ after training with the MTPA data is shown in Figure 5.18. The KRLS approximates the function in the region of the training data with errors smaller than noise level $\sigma_n = 2 \cdot 10^{-2}$. Due to the linear kernel, the region $i_d \approx -I_{max}$ gets well extrapolated with a maximal error of $\approx 6 \cdot 10^{-2}$. However, the linear kernel is not able to model the saturation of the flux-linkage in the region $i_d \approx I_{max}$. Thus, the error is large in this region. Compared to the KRLS, the rec-GP results in a even better approximation in the region $i_d \approx -I_{max}$ and $i_d \approx I_{max}$. This may be explained since the dictionary elements of the rec-GP are uniformly distributed over the whole input-space and not concentrated in the region of the training data like in the case of the KRLS. The B-KRLS performs slightly worse than the rec-GP with larger errors in the region around $(i_d, i_q) \approx (-I_{max}, 0)$. However, the B-KRLS extrapolates better than the KRLS with a maximum error of $\sigma_n = 4 \cdot 10^{-2}$ in the region $i_d \approx -I_{max}$. The superior performance of the B-KRLS may be a result of the sparsification criterion which removes the least significant element from the dictionary whenever its size exceeds a predefined budget. Therefore, the location of the dictionary elements can be adjusted over time. The KAP performs worst with large errors over the whole input-space.

### Heteroscedastic Noise

Next, the effect of input dependent noise $\sigma_{\Psi_d}(i_d, i_q, \omega_e)$ on the prediction accuracy of the online function approximators is investigated. For model training, uniform random sampled data, according to (5.6), with heteroscedastic Gaussian noise, according to (5.17), is used. The learning curve is obtained by calculating the true prediction error, defined in (5.21), based on an random sampled test set. The learning curves of the online approximators are shown in Figure 5.19.

The KRLS and the rec-GP approximator result in almost the same prediction error as for constant noise, see Section 5.1.3. However, learning is slightly slower for heteroscedastic noise, i. e., both approximators reach a mean RMS error of $20 \cdot 10^{-3}$ approximately 15 samples later, compared to the constant noise case, depicted in Figure 5.15.

The performance of the KAP and B-KRLS is more affected by heteroscedastic noise. Although the mean RMS error of the KAP is only slightly increased, the learning speed decreased drastically. The KAP reaches mean RMS error of $20 \cdot 10^{-3}$ approximately 450 samples later. The learning speed of the B-KRLS is less affected by heteroscedastic noise but the achievable mean RMS error decreased from $7 \cdot 10^{-3}$ to $9 \cdot 10^{-3}$.

As discussed in Chapter 4, the Bayesian framework allows to account for heteroscedastic

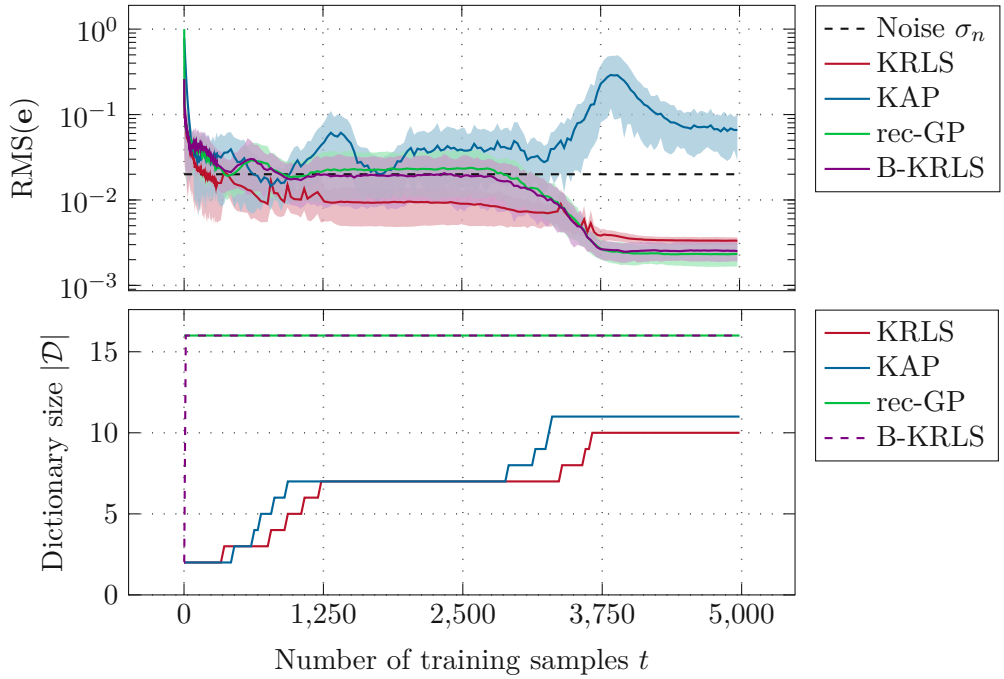



Figure 5.16: Learning curves for data in the region $\mathbb{X}_c$ near the MTPA curve. The prediction is obtained according to (5.22).

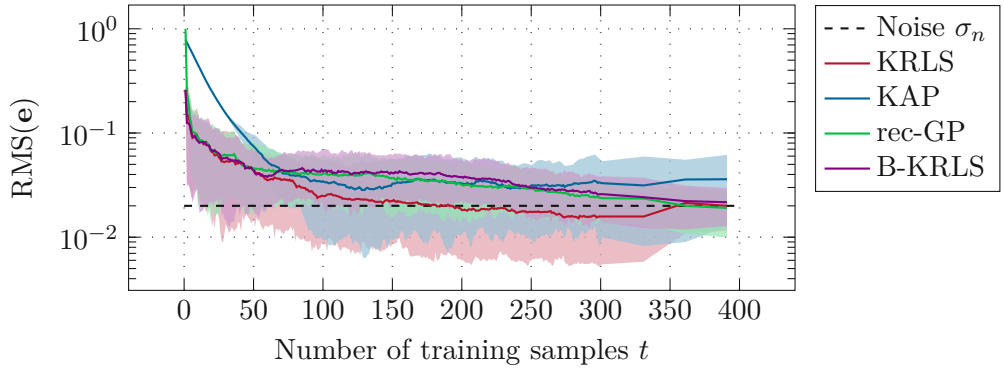



Figure 5.17: Initial learning curves for data in the region $\mathbb{X}_c$ near the MTPA curve.

noise. Assuming that the input depended noise term $\sigma_{\Psi_d}(i_d, i_q, \omega_e)$ is known, it can be included in the derivation of the B-KRLS as well as the rec-GP by replacing $\sigma_n$ in (4.21) with heteroscedastic noise term $\sigma_{\Psi_d}(i_d, i_q, \omega_e)$. Furthermore, when concerning the prediction variances (4.20), the inclusion of the heteroscedastic noise term is important since otherwise unreasonable prediction variances are obtained. A comparison of the B-KRLS and rec-GP with constant noise and B-KRLS-HN and rec-GP-HN with heteroscedastic noise is shown in Figure 5.20. The accuracy of the B-KRLS is improved with heteroscedastic noise term, while the accuracy of the rec-GP gets slightly worse.

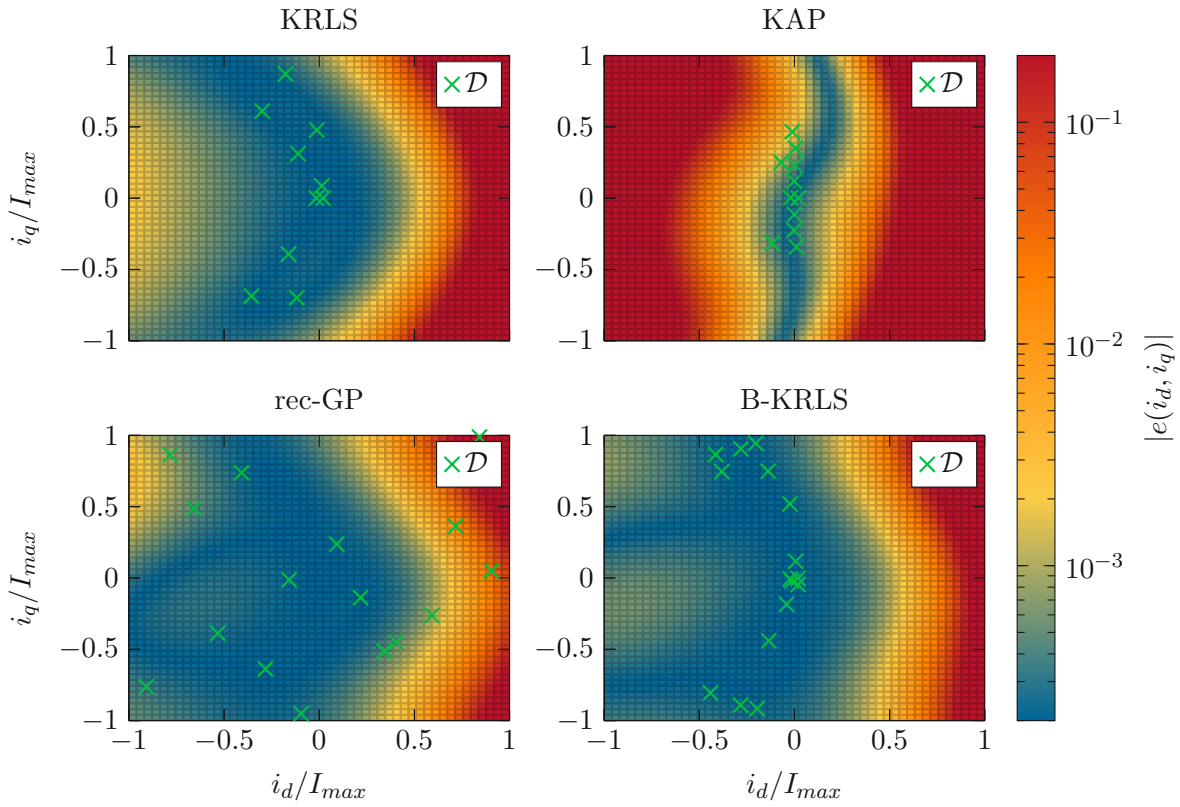



Figure 5.18: Absolute prediction error $|e(i_d, i_q)|$ after training with data in the region $\mathbb{X}_c$ near the MTPA curve.

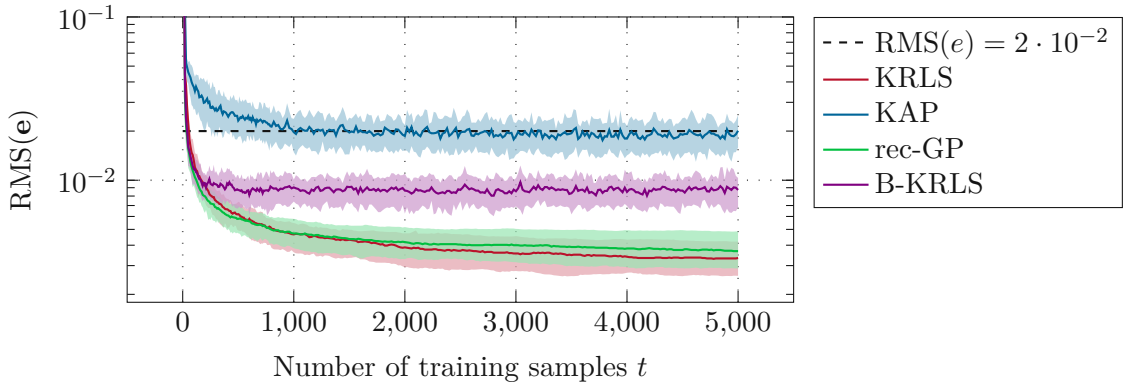



Figure 5.19: Learning curves for heteroscedastic noise, according to (5.14), for $\omega_e = \omega_{e,max}$ with uniform random sampled data.

### Changing Operating Conditions

In this section, the effect of a changing operating condition on the performance of the approximators is analyzed. The changing operating condition is simulated using (5.18),

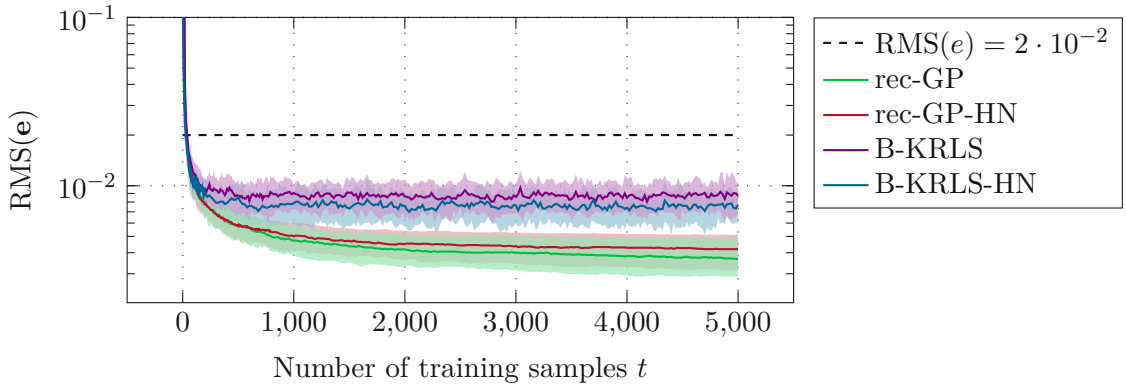



Figure 5.20: Learning curves of B-KRLS and rec-GP with constant likelihood and B-KRLS-HN and rec-GP-HN with heteroscedastic likelihood, with random sampled data.

where a time varying scaling factor $c(t)$, as depicted in Figure 5.21, is used. A learning curve is used for evaluation. In the time-varying scenario, the prediction error is defined according to

$$\mathbf{e} = \hat{\mathbf{f}}(\mathcal{X}_*, \mathcal{X}_t, \mathbf{y}_t) - \tilde{\mathbf{\Psi}}_d(t, \mathcal{X}_*) \ , \tag{5.24}$$

where $\tilde{\mathbf{\Psi}}_d(t, \mathcal{X}_*)$ is the scaled flux-linkage, according to (5.18). The approximators are trained with random sampled inputs $\mathcal{X}_t$ and observations $\mathbf{y}_t$, defined as in (5.19), and constant Gaussian noise (5.16). The learning curves are shown in Figure 5.22.

As expected, the KRLS and the rec-GP are not able to approximate time varying functions, since both are using a fixed dictionary. Therefore, large amount of data is needed to adapt their coefficients. If the latent function changes too fast, the coefficients can not be updated accordingly, thus resulting in a large prediction error.

The B-KRLS and KAP on the other hand, can adapt to the changing operating condition. The B-KRLS changes its dictionary over time, which in turn enables to adapt to a changing function. Furthermore, a forgetting mechanism, according to (4.25), with a forgetting factor $\beta = 0.999$ is used. The forgetting mechanism further improves the learning speed of the B-KRLS. The KAP is able to approximate time varying functions since its coefficients are updated based on the recent $q$ samples. Both approximators result in an RMS error $\approx 20 \cdot 10^{-3}$, which is close to noise level $\sigma_n$. Although the B-KRLS learns initially faster, the variance of its RMS error is larger compared to the KAP.

Furthermore, its worth mentioning that for scenarios in which the latent function changes drastically over time, an online adaption of the kernel hyperparameters may be required for a proper approximation. However, this is not the case for the considered scenario since the curvature of the flux-linkage changes only slightly through rescaling, see Figure 5.8.

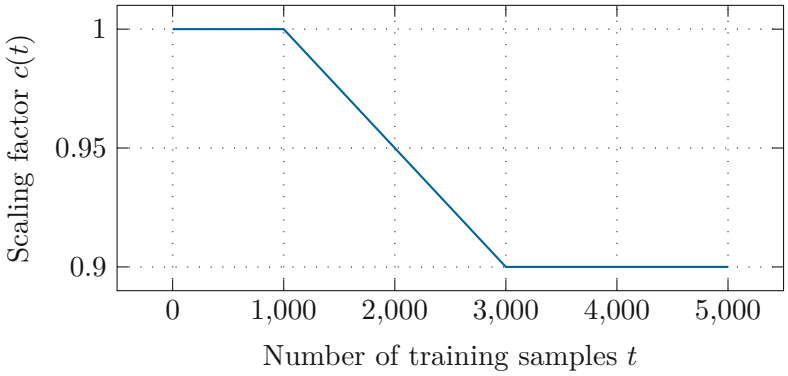



Figure 5.21: Time varying scaling factor $c(t)$, according to (5.18).

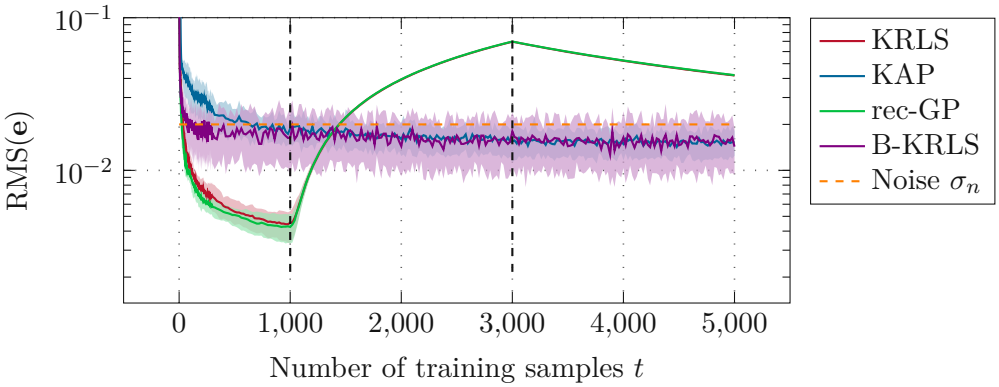



Figure 5.22: Learning curves for changing operating condition, according to (5.18) and random sampled data.

### Prediction Uncertainty

As discussed in Section 4.3, the Bayesian framework provides an estimate of the variance $\hat{\sigma}^2(\mathbf{x})$ of the prediction $\hat{f}(\mathbf{x})$. When concerning the prediction variance, the proper scaling of the kernel function becomes important [42]. In order to obtain a statistically sound predictive variance, the scale of the kernel has to be calibrated. The calibration of the kernel may be performed with cross-validation [58]. Although the uncalibrated prediction variance may not correspond to the true statistical distribution of the prediction, it can still be used as a relative measure to distinguish between predictions of high and low uncertainty. As illustrative example, Figure 5.23 shows the predictive standard deviation of a GP for different numbers of training samples $\mathcal{X}$. The predictive standard deviation is small in regions where a lot of training data is observed, whereas it is larger in regions of sparse training data. About 20 random samples are needed to obtain a predictive standard deviation in the range of noise $\hat{\sigma}(\mathbf{x}) \approx \sigma_n$ over the whole input-space $\mathbb{X}$. Thus, $\hat{\sigma}(\mathbf{x})$ can be used to evaluate the coverage of the input-space. Furthermore, the predictive variance can be used to actively explore the input-space in a sample-efficient manner [59].

Since the nonlinear MTPA curve is of special interest for practical applications, the predictive standard deviation $\hat{\sigma}(\mathbf{x})$ is evaluated at a set of test inputs

$$\mathcal{X}_\mathcal{S} = \left\{ \mathbf{i}^*_{dq}(\tau^*) \mid \tau^*_i \in \Omega_N[-\tau_{max}, \tau_{max}] \right\} \tag{5.25}$$

along the MTPA curve $\mathbf{i}^*_{dq}(\tau^*)$. The standard deviation evaluated at inputs $\mathcal{X}_\mathcal{S}$ along the MTPA curve is then defined as a vector

$$\hat{\boldsymbol{\sigma}}(\mathcal{X}_\mathcal{S}) = \left[ \hat{\sigma}(\mathbf{i}_{dq,i}) \right] \quad \text{with} \quad \mathbf{i}_{dq,i} \in \mathcal{X}_\mathcal{S} \ . \tag{5.26}$$

For the experiment, the MML-tuned kernel is used, and the function approximators are trained with uniform sampled data, according to (5.6), with constant Gaussian noise (5.16). The predictive standard deviation $\hat{\sigma}(\mathbf{x})$ of the B-KRLS and rec-GP, according to (4.20) and (4.31) respectively, is compared with the prediction standard deviation of a full GP, according to (2.26). The mean of the predictive standard deviation mean$(\hat{\boldsymbol{\sigma}}(\mathcal{X}_\mathcal{S}))$ as function of the number of processed training samples is shown in Figure 5.24. Both approximators result in a similar predictive standard deviation as the full GP. Although the B-KRLS approximates the standard deviation of the GP initially better than the rec-GP, the rec-GP results in a better approximation after $\approx 40$ samples. Noise levels is reached after $\approx 100$ random samples.

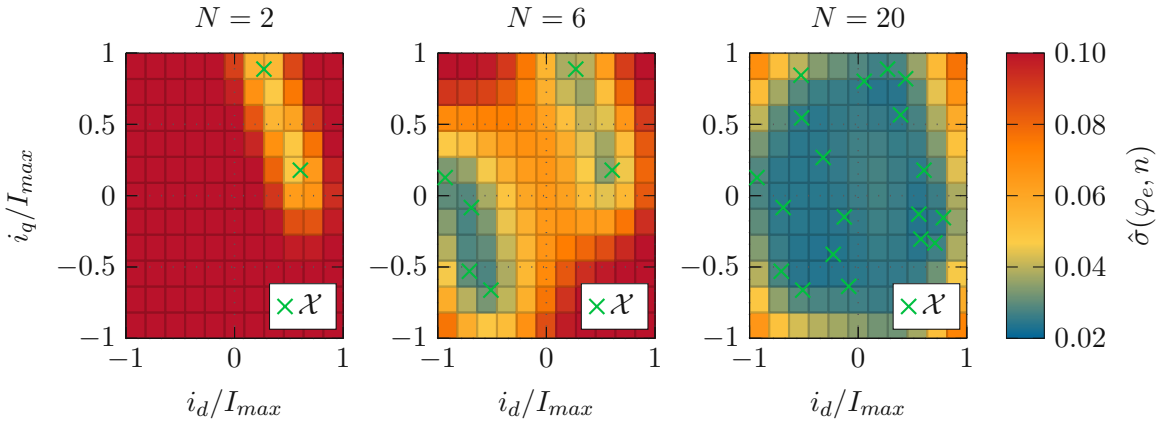



Figure 5.23: Predictive standard deviation $\hat{\sigma}(i_d, i_q)$ of a GP for different number $N$ of training inputs $\mathcal{X}$.

## 5.2 Inverter

The second application scenario from the electric drive domain, considers a voltage source inverter (VSI). A VSI converts a DC input-voltage to phase-voltages of adjustable magnitude and frequency. Each of the three phases of the VSI consists of two controllable electrical switches [1, Chapter 12]. The phase-voltages of the VSI can be controlled by the duty-cycle, i. e., the fraction of time in which a switch is active. Thus, the PMSM is controlled by adjusting the duty-cycle of the VSI. In order to improve the performance of

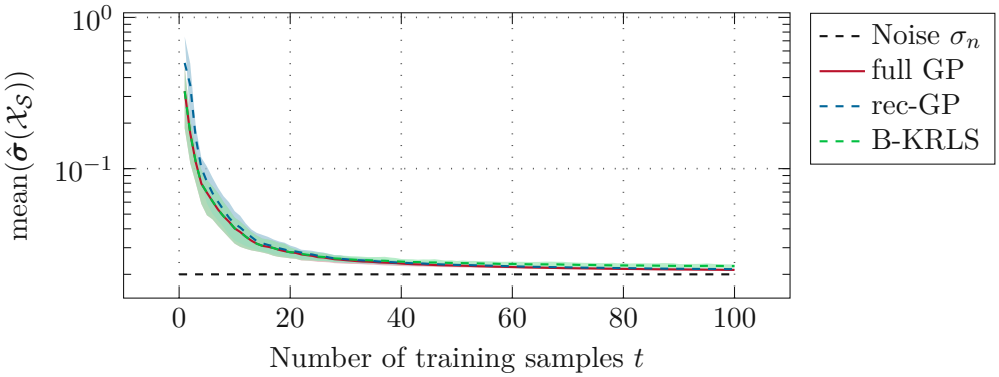



Figure 5.24: Mean prediction standard deviation mean($\hat{\boldsymbol{\sigma}}(\mathcal{X}_\mathcal{S})$) at MTPA curve.

the whole drive system, the nonlinear dependency of the duty-cycle on the motor operating point has to be modeled. Therefore, the data-based approximation of the duty-cycle nonlinearity will be discussed next.

### 5.2.1 Scenario Description

A common approach to control an inverter-fed PMSM is a model-based two degrees-of-freedom (2DoF) controller (e. g. [8]). The aim of the controller is to set the duty-cycle $\delta$ so, that a desired phase-current $\mathbf{i}^*_{abc}$ and rotor speed $\omega^*_e$ is reached. Thus, the duty-cycle is a function of the set-point $(\mathbf{i}^*_{abc}, \omega^*_e)$. If the PMSM shows distinct anisotropies in the stator or rotor, the duty-cycle is assumed to further depend on the rotor position $\varphi_e$. Thus, $\delta = \delta(\varphi_e, \omega_e, \mathbf{i}^*_{abc})$ holds. The 2DoF-controller, depicted in Figure 5.25, utilizes a model of the PMSM and VSI to calculate a feed-forward duty-cycle $\delta_{FF}$. Due to model uncertainties, external disturbances, measurement noise, etc., these models never exactly capture the behavior of the PMSM and VSI. Therefore, a feedback controller is used to stabilize the error between the desired $(\mathbf{i}^*_{abc}, \omega^*_e)$ and actual $(\mathbf{i}_{abc}, \omega_e)$ set-point via the feedback term $\Delta\delta$. Thus, a single phase of the VSI is controlled with a duty-cycle according to

$$\delta = \delta_{FF} + \Delta\delta \ . \tag{5.27}$$

During motor operation, online function approximation can then be used to train a model $\delta(\varphi_e, \omega_e, \mathbf{i}^*_{abc})$ for all operating points of interest. The mechanical and electrical characteristics of a PMSM are periodic after a full mechanical revolution of the rotor. Thus, the duty-cycle $\delta$ is expected to be periodic w.r.t. the rotor position $\varphi_e$, according to $\delta(\varphi_e) = \delta(\varphi_e + 2\pi)$. The feed-forward $\delta_{FF}$ and feedback $\Delta\delta$ duty-cycle of a single phase are shown in Figure 5.26 for a set-point $\mathbf{i}^*_{abc} = \mathbf{0}$. Both terms vary periodically as function of the rotor position $\varphi_e$ and increase in magnitude with increasing rotor speed $n = \omega_e/n_p$. The feed-forward duty-cycle can be precalculated offline since no feedback from the real PMSM is needed. Therefore, only the approximation of the feedback term $\Delta\delta(\varphi^*_e, \omega^*_e)$ will be studied further.

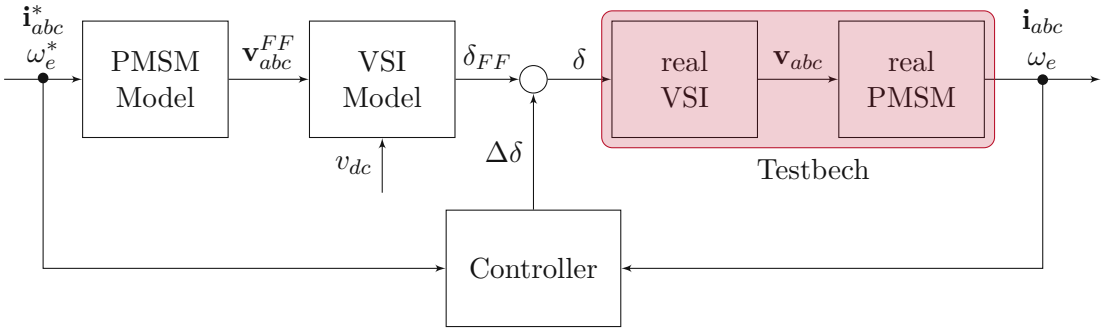
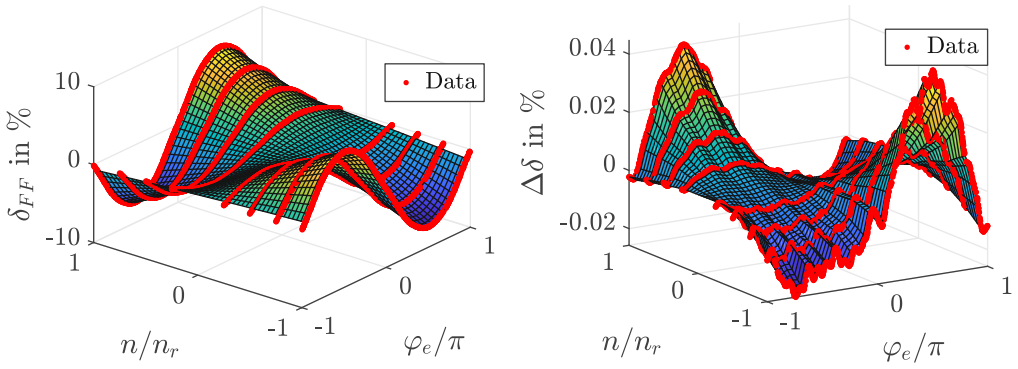



Figure 5.25: Block diagram of the two degrees-of-freedom controller.




(a) Feed-forward $\delta_{FF}$.

(b) Feedback $\Delta\delta$.

Figure 5.26: Feed-forward $\delta_{FF}$ and feedback $\Delta\delta$ duty-cycle of single VSI phase for a desired set-point $\mathbf{i}_{abc} = \mathbf{0}$.

In this application scenario only noisy measurements $y(\mathbf{x}) = \Delta\delta(\mathbf{x})$ at inputs $\mathbf{x}$ are available to train and test the function approximators. Thus, the latent function $f(\mathbf{x})$ as well as the measurement noise model $\epsilon$, according to (2.3), are unknown. Furthermore, the dataset is obtained from measurements at $N_n = 8$ equidistant rotor speeds $n_i$, see Figure 5.26, further denoted as

$$\mathbf{N} = \{n_i \in \Omega_{N_n}[-1, 1]\} \,, \tag{5.28}$$

where $\Omega_{N_n}[-1, 1]$ defines an equidistant grid in the range $[-1, 1]$. Thus, a temporal-sequence of $N_t$ samples $\varphi_e(t_j)$ corresponds to each measurement at a certain rotor speed. A simulation experiment will be performed in which the online approximators are trained with sequential data, defined according to

$$\mathcal{X}_s = \{(n_i, \varphi_e(t_j)) \mid n_i \in \mathbf{N} \;, t_j \in \Omega_{N_t}[0, T]\} \quad \text{and} \quad \mathbf{y}_s = \{y(\mathbf{x}_i) \mid \mathbf{x}_i \in \mathcal{X}_s\} \,, \tag{5.29}$$

where $t_j$ are equidistant time steps and $T$ is the total measurement time. For hyperparameter-tuning, it is assumed that the whole dataset is available from which random samples can be drawn. Thus, a dataset with random samples is defined, according to

$$\mathcal{X}_r = \left\{(n_i, \varphi_e(t_j)) \mid n_i \in \mathbf{N} \;, t_j \sim \mathcal{U}_{[0,T]}\right\} \quad \text{and} \quad \mathbf{y}_r = \{y(\mathbf{x}_i) \mid \mathbf{x}_i \in \mathcal{X}_r\} \,, \tag{5.30}$$

where $\mathcal{U}_{[0,T]}$ denotes uniform random time samples in the range $[0, T]$. In order to evaluate the prediction error of the approximators, a random test set with inputs $\mathcal{X}_*$ and outputs $\mathbf{y}_*$, according to (5.30), is used.

### 5.2.2 Offline Model Selection

This section discusses the selection of an appropriate kernel function as well as the tuning of the kernel- and noise-parameters. Hyperparameter-tuning is performed with random sampled data, according to (5.30).

Again, the kernel function is selected with respect to the available prior knowledge described in Section 5.2.1. The only available knowledge is the periodicity of $\Delta\delta$ w.r.t. the rotor position $\varphi_e$. Periodicity is enforced by mapping the rotor position $\varphi_e$ to the interval $(-1, 1]$. Furthermore, the mapping preserves continuity at the interval limits. In order to obtain a smooth approximation, a Gaussian kernel (3.11) with different lengthscales for each dimension according to

$$k(\mathbf{x}, \mathbf{x}') = \sigma_v^2 \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^{\mathrm{T}} \mathbf{\Sigma} (\mathbf{x} - \mathbf{x}')\right), \tag{5.31}$$

is used[3]. Here, $\mathbf{\Sigma}$ is a diagonal matrix according to $\mathbf{\Sigma} = \operatorname{diag}\left(\begin{bmatrix} l_1^{-2} & l_2^{-2} \end{bmatrix}\right)$. Since only measurements at discrete speeds are available, the kernel parameters have to be chosen in a way that the data is properly interpolated. The resulting kernel has three hyperparameters $\boldsymbol{\theta}_k^{\mathrm{T}} = \begin{bmatrix} \sigma_v & l_1 & l_2 \end{bmatrix}$. As discussed in Section 5.1.2, the hyperparameter-tuning with cross-validation (CV) and maximum marginal-likelihood (MML) is performed offline with an Gaussian Process according to (2.26).

#### Hyperparameter-tuning with Cross-Validation

In order to reduce the computational complexity of the hyperparameter-tuning with CV, some of the hyperparameters are fixed a priori. Since only a single kernel is used, the weight is set to $\sigma_v = 1$. In oder to ensure that the region between the measurement is well interpolated, the lengthscale $l_2$ of the Gaussian kernel is set to a large value $l_2 = 1$. The remaining hyperparameter $l_1$ is then tuned with CV.

$K$-fold CV, according to (3.26) with $K = 10$, is used to obtain an estimate of the RMS prediction error based on random sampled data, as in (5.30). The dependency on the choice of dataset is reduced by averaging the results of 50 trials with different uniform sampled datasets which are defined by (5.30). The median and the 16 % and 84 % percentiles [18, Section 2.2.6] are used to obtain an estimate of the mean and a 68 % confidence interval. The hyperparameter $l_1$ is tuned by a grid search of 50 equally distributed values in the range $[0.01, 0.5]$. Furthermore, the likelihood of the GP, defined as (2.25), is set to a rather large value of $\sigma_n = 1 \cdot 10^{-3}$ to obtain a smooth approximation.

[3] The kernel function is introduced as $k(\mathbf{x}, \mathbf{x}')$ with inputs $\mathbf{x}^{\mathrm{T}} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} = \begin{bmatrix} \varphi_e & n \end{bmatrix}$

The RMS error as function of the hyperparameter $l_1$ is shown in Figure 5.27 for different dataset sizes $N$. The lengthscale $l_1$ tends to decrease for larger dataset sizes. As more data is observed, the small variations along the first dimension are not treated as noise anymore. Thus, the lengthscale decreases to capture the these small variations. However, for a to small lengthscale the online approximators will need a large dictionary for a proper approximation. Thus, a trade-off between computational complexity and accuracy has to be made. The selected kernel hyperparameters are summarized in Table 5.3.

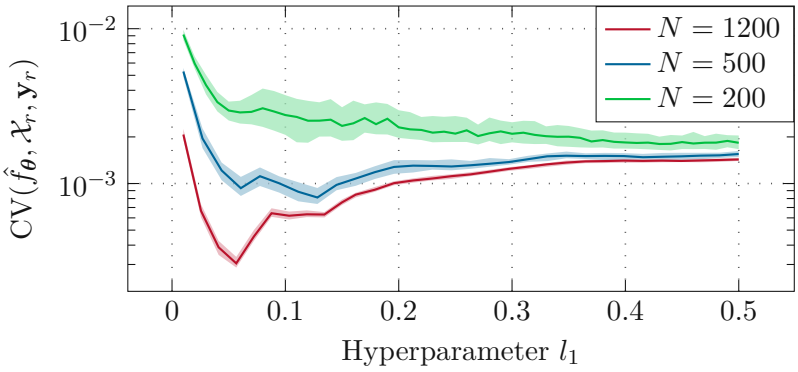



Figure 5.27: Hyperparameter tuning with 10-fold cross-validation using a GP with kernel (5.31).

### Hyperparameter-tuning with Maximum Marginal Likelihood

As alternative, hyperparameter-tuning is now performed offline by maximization of the marginal-likelihood (4.12) of a GP. The optimization is performed in MATLAB using the Conjugate-Gradient solver of the GPML-Toolbox [46]. Again, a histogram plot is used to investigate the distribution of the obtained hyperparameters. Since the hyperparameter distributions are not normal distributed in general, the median and the 16 % and 84 % percentiles [18, Section 2.2.6] are used to obtain an estimate of the mean and a 68 % confidence interval.

The lengthscale $l_1$ of the kernel should not be estimated too small since this would require a large dictionary. Therefore, the dependency on the dataset size is investigated first. For a large dataset $N > 1000$, MML tends to result in a rather small $l_1$. Finally, a dataset size of $N = 800$ is used. A histogram plot of the hyperparameter distribution is depicted in Figure 5.28. All distributions are well peaked, indicating a reliable estimate. Furthermore, a not to small lengthscale $l_1$ is obtained. The estimated noise standard deviation $\hat{\sigma}_n$ lies in a similar range as the RMS error obtained by CV, see Figure 5.27. The hyperparameters are set to the median values in Figure 5.28 and are summarized in Table 5.3. Since the obtained kernel hyperparameters are very similar to the CV obtained hyperparameters, the MML-tuned kernel is used for all following experiments.

As baseline for the online approximators serves a GP with the hyperparameters obtained

by MML. Cross-validation with a random sampled dataset, according to (5.30) with $N = 2000$, is used to calculate the prediction RMS error. The GP results in a mean RMS error of $\approx 600 \cdot 10^{-6}$. The value corresponds to the MML estimated noise standard deviation in Figure 5.28.

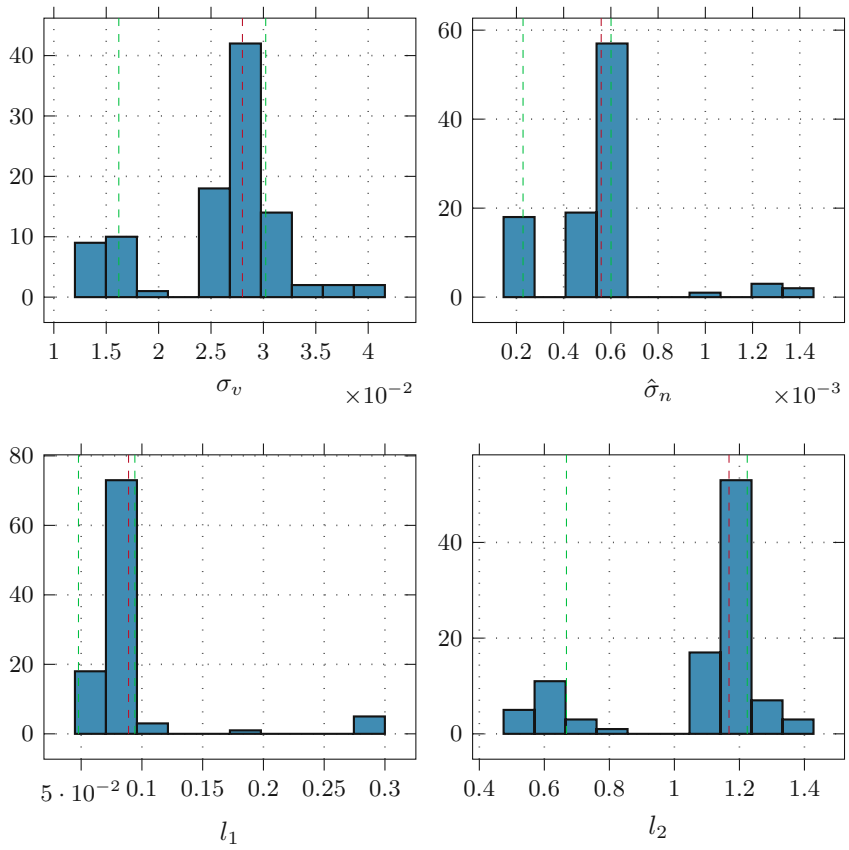



Figure 5.28: Histogram of the MML-tuned hyperparameters for different random datasets of size $N = 800$.

| Method | Kernel Hyperparameter | | |
|---|---|---|---|
| CV | $\sigma_v = 1.000$ | $l_1 = 0.100$ | $l_2 = 1.000$ |
| MML | $\sigma_v = 0.028$ | $l_1 = 0.089$ | $l_2 = 1.168$ |

Table 5.3: Hyperparameters of kernel (5.31) for VSI dataset.

### 5.2.3 Evaluation on Measurement Data

The accuracy of the online function approximators, introduced in Section 3.3 and Section 4.3, is now evaluate with sequential measurement data as defined by (5.29). Since the VSI data is less smooth compared to PMSM data, a larger dictionary size is required

for a proper approximation. Thus, the parameters of the approximators are chosen in a way, that the maximum dictionary size of all approximators is $|\mathcal{D}| \approx 100$. The parameter settings of the approximators are defined in Table 5.4. The dictionary of rec-GP is initialized with random points from the dataset.

The learning curve of the approximators is shown in Figure 5.29. The B-KRLS performs best and reaches an error of $\approx 1.1 \cdot 10^{-3}$ after 5700 samples. The KRLS and the rec-GP perform slightly worse, with RMS error of $\approx 1.4 \cdot 10^{-3}$. The KAP shows the worst performance with an error of $\approx 27 \cdot 10^{-3}$.

A plot of the absolute prediction error $|e(\varphi_e, n)|$ as function of the rotor position $\varphi_e$ and the speed $n$ is shown in Figure 5.30. The B-KRLS resulted in the best approximation with only a few region where $|e(\varphi_e, n)| > 1 \cdot 10^{-3}$. The KRLS performed slightly worse. For both approximators, the maximum approximation error is $\approx 5 \cdot 10^{-3}$. The rec-GP performed similar to the KRLS but obtained large errors of $\approx 9 \cdot 10^{-3}$ in some regions. The KAP shows the worst performance with the largest errors over the whole space and a maximum error of $\approx 59 \cdot 10^{-3}$.

Figure 5.31 shows the predictive standard deviation $\hat{\sigma}(\varphi_e, n)$ of the rec-GP and B-KRLS. For both approximators, $\hat{\sigma}(\varphi_e, n)$ is high in regions where also the prediction error, according to see Figure 5.30, is high.

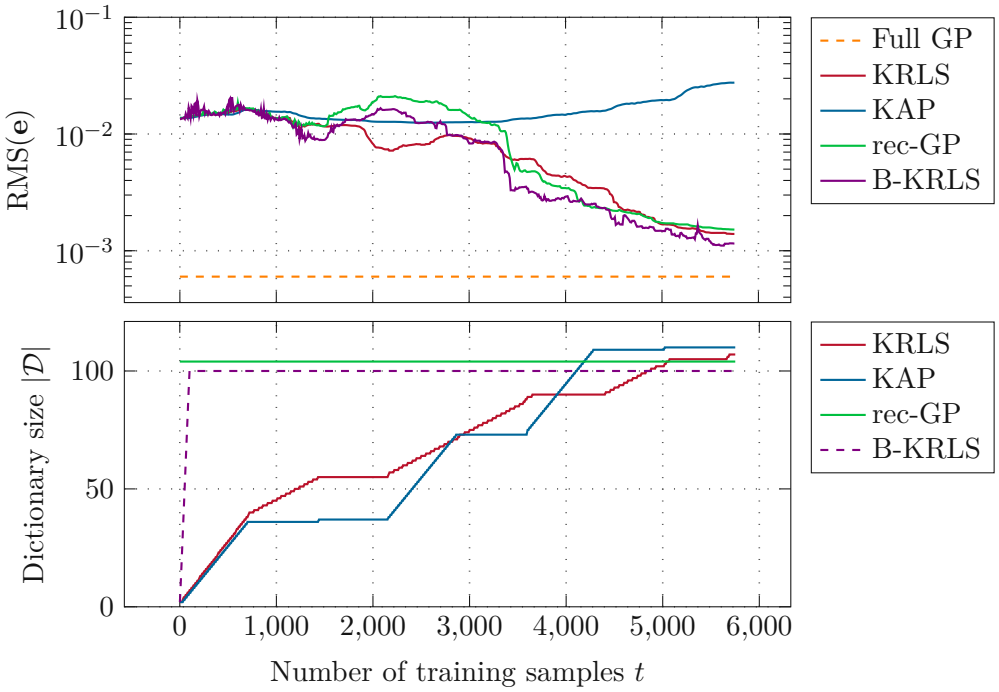



Figure 5.29: Learning curves of online function approximators on sequential measurement data, according to (5.29).

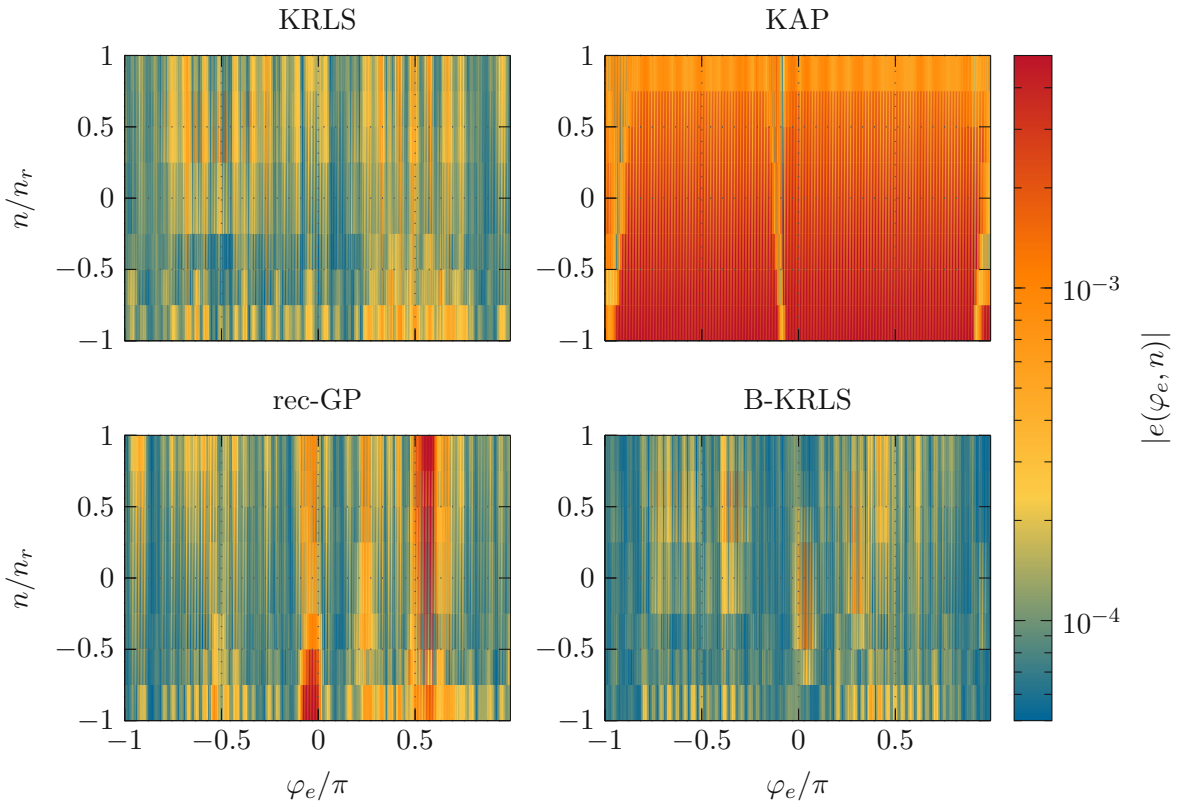



Figure 5.30: Absolute prediction error $|e(\varphi_e, n)|$ after training with sequential data, according to (5.29).

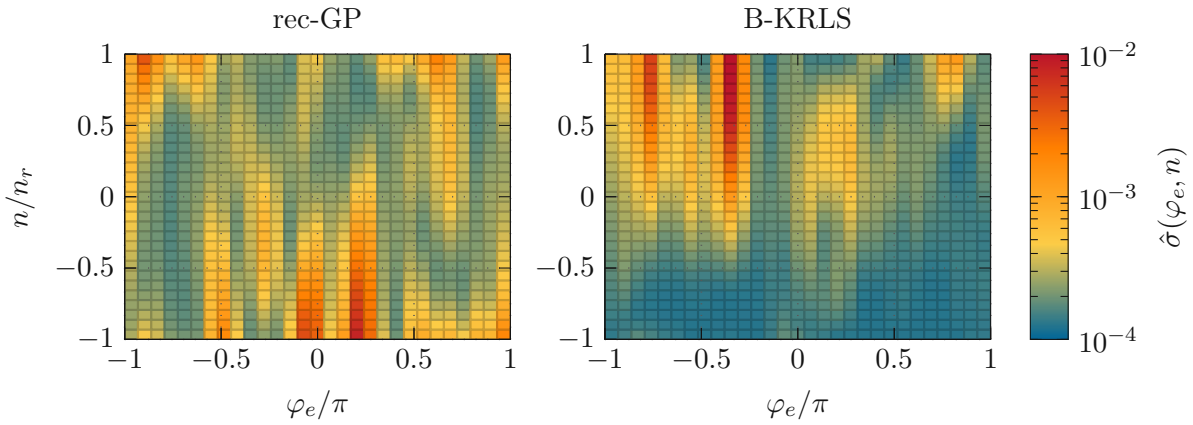



Figure 5.31: Predictive standard deviation error $\hat{\sigma}(\varphi_e, n)$ after training with sequential data, according to (5.29).

| Approximator | Section | Parameters |
|---|---|---|
| KRLS | 3.3.2 | $\nu = 5 \cdot 10^{-5}$ |
| KAP | 3.3.3 | $\mu_0 = 650 \cdot 10^{-6}$ $q = 100$<br>$\epsilon = 1 \cdot 10^{-6}$ $\eta = 0.02$ |
| B-KRLS | 4.3.1 | $m = 104$ $\sigma_n = 559 \cdot 10^{-6}$ |
| rec-GP | 4.3.2 | $m = 104$ $\sigma_n = 559 \cdot 10^{-6}$ |

Table 5.4: Parameter settings of online function approximators for the VSI dataset.

## 5.3 Discussion

This section summarizes the main results of previous two sections and discusses which of the proposed online functions approximators is best suited for the described PMSM and VSI application scenario.

For random sampled PMSM data with constant noise, the kernel recursive least squares (KRLS) obtained a prediction error a magnitude smaller than noise. This is close to the error of a full GP. However, the KRLS uses only a fraction of the GPs computational resources. In the presence of heteroscedastic noise, the learning speed of the KRLS decreased slightly, but the prediction error remained almost identical as for constant noise. Furthermore, the KRLS shows a fast learning speed for random as well as for non-random data. However, choosing the sparsification parameter $\nu$, according to (3.28), is not trivial and requires to perform offline trials. For the use in real-time control applications, the maximum computational complexity per time step has to be defined a priori. Thus, the KRLS is not suited for such scenarios since the maximum size of its dictionary can not be fixed a priori. It is concluded, that the KRLS is best suited for scenarios in which a static function has to be approximated based on a stream of arbitrary sampled data and where it is possible to perform offline experiments to adjust $\nu$.

The kernel affine projection (KAP) reaches a prediction error at noise level for the random sampled PMSM data. However, the performance drastically decreased for data near the MTPA curve and VSI measurement data, since the most recent samples do not provide information about the function over the whole input-space. Thus, for non-random data, the KAP is only suited to make local predictions in the neighborhood of the recent samples (e. g. prediction for time-series). The prediction error of the KAP slightly decreased for heteroscedastic noise, while the learning speed drastically decreased with heteroscedastic noise. Furthermore, the tuning of the coherence parameter $\mu_0$, according to (3.31), requires offline trials which hinders the usage in a real-time system. Therefore it is concluded that the KAP is not suited for the PMSM and VSI scenario.

The recursive-GP (rec-GP) performed almost as well as the KRLS for random PMSM data. However, the rec-GP allows to predefine the size of the dictionary and as a consequence also the maximum computational complexity. Thus, the algorithm is suitable for

real-time applications. The rec-GP has only one tuning-parameter, namely the likelihood $\sigma_n$ in (4.31), which can be estimated by CV and MML. Thus, the rec-GP can be tuned easily. Furthermore, heteroscedastic noise did not affect the prediction error significantly. Although the rec-GP performed slightly better than the KRLS for data near the MTPA curve, the KRLS resulted in a better approximation for the VSI measurement data. The rec-GP additionally provides a prediction uncertainty which could be used the evaluate the reliability of the prediction and the coverage of the input space. The rec-GP can further be extended to learn the hyperparameters online from data [51]. This is interesting in scenarios in which the latent function changes drastically over time.

The Bayesian-KRLS (B-KRLS) showed the best performance for data near the MTPA curve and for the VSI measurement data. For random PMSM data, the prediction error of the B-KRLS is slightly larger compared to the KRLS and rec-GP. However, this is not a drawback, since random data rarely occurs in practice. Furthermore, the B-KRLS is able to approximate drifting functions with a fast learning speed and a prediction error at noise level. The Bayesian framework further allows to estimate the forgetting factor $\beta$ in (4.25) from data [60]. This is especially useful when dynamics of the time-varying function are unknown. Similar to the rec-GP, the maximum dictionary size of the B-KRLS can be predefined, which allows to trade accuracy against computational complexity a priori. The B-KRLS can be easily tuned with the likelihood $\sigma_n$ in (4.20). For heteroscedastic noise the accuracy of the B-KRLS slightly decreased. The B-KRLS additionally provides a prediction uncertainty. Thus, it is concluded that the B-KRLS is best suited for the PMSM and VSI application scenario.

# 6 Summary and Outlook

In this thesis, data-driven methods for online nonlinear function approximation are investigated and evaluated with focus on representative application scenarios from the electric drive domain.

A literature review of state-of-the-art function approximators is given in Chapter 2. Kernel methods and Gaussian Processes are introduced as powerful non-parametric modelling tools. Both approaches allow to incorporate prior knowledge by means of a kernel function. The Bayesian framework additionally provides a model for the prediction uncertainty and a systematic framework for model selection. Chapter 3 deals with the mathematical background for frequentist function approximation. Here, two online kernel methods from the kernel adaptive filter framework, namely the kernel recursive least squares (KRLS) and the kernel affine projection (KAP) algorithm are introduced. Both algorithms have a computational complexity of $\mathcal{O}(m^2)$, where $m \ll N$ is a constant which is independent of the data size $N$. In Chapter 4, the function approximation problem is stated within a Bayesian framework. Furthermore, two Bayesian online algorithms, namely the Bayesian-KRLS (B-KRLS) and the recursive-GP (rec-GP), with a computational complexity of $\mathcal{O}(m^2)$ are introduced.

An evaluation of the online function approximators based on two application scenarios from the electric drive domain follows in Chapter 5. The data-based approximation of the flux-linkage of a Permanent-Magnet Synchronous-Motor (PMSM) was studied in Section 5.1. For all experiments, a model of the flux-linkage is available. After describing the specific application scenario, a common control strategy, known as Maximum Torque per Ampere (MTPA) strategy, is summarized. The MTPA strategy is utilized to artificially generate training data in a realistic operating range of the motor. Based on stationary operating points of a dynamic PMSM model, a measurement model, which quantifies the expected uncertainties in the flux-linkages, is derived. The measurement model serves as basis to evaluate the robustness of the approximators against constant, as well as heteroscedastic Gaussian noise. Furthermore, the influence of changing operating conditions is discussed. Next, an appropriate kernel function is selected based on available prior knowledge. Since the flux-linkage is a smooth function with linear trend, a Gaussian kernel in combination with a linear kernel is used. The hyperparameters of the kernel are tuned offline with cross-validation (CV) and maximum marginal likelihood (MML). In order to reduce the computational complexity of CV, some hyperparameters are fixed a priori. For comparison, MML is performed without the use of prior knowledge. The reliability of the obtained hyperparameters is evaluated with histograms and confidence intervals. A comparison of the CV- and MML-tuned kernel showed that the MML-tuned kernel performs slightly better. As a baseline for the online approximators serves an

experiment with random sampled data and constant Gaussian noise. Thereby, the KRLS and rec-GP perform best, with a prediction error a magnitude smaller than noise level. This is close to the prediction error of a GP. However, both approximators require only a fraction of the GPs computational complexity. This is followed by an evaluation of the approximators with artificially generated data according the described application scenarios. For representative operating data, all approximators, except the KAP, resulted in an prediction error a magnitude smaller than noise level in the region of the training data. The rec-GP and the B-KRLS performed best and also extrapolated the data properly. In presence of heteroscedastic noise, the learning speed of the approximators decreased slightly in general. The accuracy of the KRLS and the rec-GP with heteroscedastic noise remained almost identical as for constant noise. The KAP and the B-KRLS are able to approximate drifting functions with an prediction error below noise level. Thereby, the B-KRLS learned faster and the error is slightly smaller compared to the KAP. The B-KRLS and the rec-GP further provide a model of the prediction uncertainties.

The data-based approximation of an average model of an voltage source inverter (VSI), is discussed in Section 5.2. In this application scenario, the data was obtained from online measurements during motor operation. Thus, the latent function as well as the noise model are unknown. Since the measurements are performed at discrete rotor speeds, the kernel hyperparameters are tuned to interpolate the measurement properly. Therefore, a Gaussian kernel with different lengthscales for each direction is used. The size of the dataset affected the hyperparameter tuning significantly. Again histograms and confidence intervals are used to evaluate the reliability of the results. An evaluation based on streaming measurement data was given in Section 5.2.3. The KRLS and the B-KRLS result in the best approximation. However, the KRLS requires to perform offline trails in order to find suitable parameter settings, which may hinder its usage in a real-time system.

It is concluded, that the B-KRLS is best suited for both application scenario since it results in a good approximations for random and representative operating data, allows to model drifting functions and provides a predictive variance. Furthermore, its computational complexity can be defined a priori which is often required for the use in a real-time system control system.

With regards to future work, it would be interesting to implement the online algorithms in a motor control system and investigate how the use of online function approximation affects the control performance of the whole drive system.

# A Appendix

## A.1 Parameters

| Name | Parameter | Value |
|---|---|---|
| Stator resistance | $R$ | 26.6 mΩ |
| Inductance | $L_d$ | 60.9 µH |
| Inductance | $L_q$ | 89.6 µH |
| Per. magnet flux | $\Psi_m$ | 4.82 mWb |
| Pole pairs | $n_p$ | 4 |
| Maximal current | $I_{max}$ | 84.9 A |
| Nominal current | $I_n$ | 34 A |
| Mechanical speed | $n_n$ | 1400/min |
| Maximal torque | $\tau_{max}$ | 2.4 N m |
| Nominal torque | $\tau_n$ | 1 N m |

Table A.1: Parameters of the considered PMSM.

| Parameter | Value |
|---|---|
| $\sigma_R$ | 1 mΩ |
| $\sigma_i$ | 10 mA |
| $\sigma_v$ | 10 mV |
| $\sigma_\omega$ | 10 rad/s |

Table A.2: Assumed standard deviation of measurement variables for error propagation analysis (5.13).

## A.2 Mathematical Background

### A.2.1 Matrix Inversion Lemma

If $\mathbf{A}$, $\mathbf{C}$ and $\mathbf{A}+\mathbf{BCD}$ are regular quadratic matrices, then the matrix inversion lemma [36] states

$$(\mathbf{A}+\mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}\Big(\mathbf{C}^{-1}+\mathbf{DA}^{-1}\mathbf{B}\Big)^{-1}\mathbf{DA}^{-1} \,. \tag{A.1}$$

### A.2.2 Block Matrix Inversion Identity

If $\mathbf{A}$ and $\mathbf{D}$ and are regular quadratic matrices, then the block matrix inversion identity [36, Appendix A.5] states

$$\begin{bmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D}\end{bmatrix}^{-1} = \begin{bmatrix}\left(\mathbf{A}-\mathbf{B}\mathbf{D}^{-1}\mathbf{C}\right)^{-1} & -\mathbf{A}^{-1}\mathbf{B}\left(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\\ -\mathbf{D}^{-1}\mathbf{C}\left(\mathbf{A}-\mathbf{B}\mathbf{D}^{-1}\mathbf{C}\right)^{-1} & \left(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\end{bmatrix}. \tag{A.2}$$

### A.2.3 Gaussian Identities

An $m$-dimensional multivariate Gaussian distribution with mean $\boldsymbol{\mu} \in \mathbb{R}^m$ and covariance $\boldsymbol{\Sigma} \in \mathbb{R}^{m\times m}$ is defined as

$$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi \det(\boldsymbol{\Sigma}))^{m/2} \exp\left(\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right). \tag{A.3}$$

The definition of a Gaussian process [25, 45] implies that, if the two arbitrary finite sets $\mathbf{f}_1$ and $\mathbf{f}_2$ are jointly Gaussian random vectors

$$p(\mathbf{f}_1, \mathbf{f}_2) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_1\\ \boldsymbol{\mu}_2\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12}\\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{bmatrix}\right) \tag{A.4a}$$

$$p(\mathbf{f}_1) = \int p(\mathbf{f}_1, \mathbf{f}_2)\,\mathrm{d}\mathbf{f}_2 = \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}) \tag{A.4b}$$

$$p(\mathbf{f}_2) = \int p(\mathbf{f}_1, \mathbf{f}_2)\,\mathrm{d}\mathbf{f}_1 = \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22}) \tag{A.4c}$$

then the conditional probability $p(\mathbf{f}_2 \mid \mathbf{f}_1)$ of $\mathbf{f}_2$ given $\mathbf{f}_1$ is also Gaussian

$$p(\mathbf{f}_2 \mid \mathbf{f}_1) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{A.5a}$$

$$\boldsymbol{\mu} = \boldsymbol{\mu}_2 - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\mathbf{f}_1 - \boldsymbol{\mu}_1) \tag{A.5b}$$

$$\boldsymbol{\Sigma} = \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}\,. \tag{A.5c}$$

The product of two Gaussian distributions is also Gaussian

$$\mathcal{N}(\mathbf{x} \mid \mathbf{a}, \mathbf{A})\mathcal{N}(\mathbf{P}\mathbf{x} \mid \mathbf{b}, \mathbf{B}) = z_c\,\mathcal{N}(\mathbf{x} \mid \mathbf{c}, \mathbf{C}) \tag{A.6}$$

where $\mathbf{P}\mathbf{x}$ is a linear projection and

$$\mathbf{c} = \mathbf{C}\left(\mathbf{A}^{-1}\mathbf{a} + \mathbf{P}^{\mathrm{T}}\mathbf{B}^{-1}\mathbf{b}\right) \tag{A.7}$$

$$\mathbf{C} = \left(\mathbf{A}^{-1} + \mathbf{P}^{\mathrm{T}}\mathbf{B}^{-1}\mathbf{P}\right)^{-1} \tag{A.8}$$

$$z_c = (2\pi)^{-\frac{m}{2}} \det\left(\mathbf{B}+\mathbf{P}\mathbf{A}\mathbf{P}^{\mathrm{T}}\right)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mathbf{b}-\mathbf{P}\mathbf{a})^{\mathrm{T}}\left(\mathbf{B}+\mathbf{P}\mathbf{A}\mathbf{P}^{\mathrm{T}}\right)^{-1}(\mathbf{b}-\mathbf{P}\mathbf{a})\right) \tag{A.9}$$

### A.2.4 Kullback-Leibler Divergence

The Kullback-Leibler (KL) divergence or relative entropie is a measure how similar two probability density functions (PDF) are. The KL-divergence between two PDF is defined as

$$\mathrm{KL}(p(x)\,,\,q(x)) = \int p(x) \ln\left(\frac{p(x)}{q(x)}\right) \mathrm{d}x \geq 0\,. \tag{A.10}$$

The KL-divergence is not symmetric $\mathrm{KL}(p(x)\,,\,q(x)) \neq \mathrm{KL}(q(x)\,,\,p(x))$ and positive. Its zero if and only if $p(x) = q(x)$. The KL-divergence between two multivariate Gaussian distributions (A.3) can be obtained in closed form. Suppose $p(\mathbf{x}) = \mathcal{N}\left(\boldsymbol{\mu}_p, \mathbf{P}\right)$ and $q(\mathbf{x}) = \mathcal{N}\left(\boldsymbol{\mu}_q, \mathbf{Q}\right)$ are $m$-dimensional Gaussian PDFs, then

$$\mathrm{KL}(p(\mathbf{x})\,,\,q(\mathbf{x})) = \frac{1}{2}\left(\ln\frac{\det(\mathbf{Q})}{\det(\mathbf{P})} + \mathrm{Tr}\left(\mathbf{Q}^{-1}\mathbf{P}\right) + \left(\boldsymbol{\mu}_p - \boldsymbol{\mu}_q\right)^{\mathrm{T}}\mathbf{Q}^{-1}\left(\boldsymbol{\mu}_p - \boldsymbol{\mu}_q\right) - m\right)\,. \tag{A.11}$$

## A.3 Gaussian Process with Heteroscedastic Noise

Heteroscedastic noise is noise which depends on the input locations, i.e. $\sigma_n = \sigma_n(\mathbf{x})$. The predictive equation of a Gaussian process with heteroscedastic noise [15] is given as

$$p(y_* \mid \mathbf{x}_*, \mathbf{y}) = \mathcal{N}\left(\hat{f}(\mathbf{x}_*),\, \hat{\sigma}^2(\mathbf{x}_*)\right) \tag{A.12a}$$

$$\hat{f}(\mathbf{x}_*) = \mathbf{K}(\mathbf{x}_*, \mathbf{X})(\mathbf{K}(\mathbf{X}, \mathbf{X}) + \boldsymbol{\Sigma}_n)^{-1}\mathbf{y} \tag{A.12b}$$

$$\hat{\sigma}^2(\mathbf{x}_*) = k(\mathbf{x}_*, \mathbf{x}_*) + \sigma_n^2(\mathbf{x}_*) - \mathbf{K}(\mathbf{x}_*, \mathbf{X})(\mathbf{K}(\mathbf{X}, \mathbf{X}) + \boldsymbol{\Sigma}_n)^{-1}\mathbf{K}(\mathbf{X}, \mathbf{x}_*)\,. \tag{A.12c}$$

where $\boldsymbol{\Sigma}_n = \mathrm{diag}\left(\left[\sigma_n^2(\mathbf{x}_1) \ldots \sigma_n^2(\mathbf{x}_N)\right]\right)$ is a diagonal matrix.

# Bibliography


[1] P. Krause, O. Wasynczuk, S. Sudhoff, and S. Pekarek, *Analysis of Electric Machinery*, 3rd Edition. Wiley, 2013, vol. 15. DOI: 10.1109/MPER.1995.365078.

[2] A. E. Fitzgerald, C. Kingsley, and S. D. Umans, *Electric machinery*, 6. Edition, ser. McGraw-Hill series in electrical and computer engineering. McGraw-Hill, 2005.

[3] W. Kemmetmüller, D. Faustner, and A. Kugi, "Modeling of a permanent magnet synchronous machine with internal magnets using magnetic equivalent circuits," *IEEE Transactions on Magnetics*, vol. 50, no. 6, 2014. DOI: 10.1109/TMAG.2014.2299238.

[4] D. Faustner, W. Kemmetmüller, and A. Kugi, "Magnetic equivalent circuit modeling of a saturated surface-mounted permanent magnet synchronous machine," in *Proceedings of the 8th Vienna International Conference on Mathematical Modelling (MATHMOD)*, Vienna, Austria, Feb. 2015, pp. 360–365. DOI: 10.1016/j.ifacol.2015.05.033.

[5] F. Mink, N. Kubasiak, B. Ritter, and A. Binder, "Parametric model and identification of pmsm considering the influence of magnetic saturation," in *13th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*, 2012, pp. 444–452. DOI: 10.1109/OPTIM.2012.6231768.

[6] N. Bedetti, S. Calligaro, and R. Petrella, "Stand-Still Self-Identification of Flux Characteristics for Synchronous Reluctance Machines Using Novel Saturation Approximating Function and Multiple Linear Regression," *IEEE Transactions on Industry Applications*, vol. 52, no. 4, pp. 3083–3092, 2016. DOI: 10.1109/TIA.2016.2535413.

[7] V. Hagenmeyer and M. Zeitz, "Flachheitsbasierter Entwurf von linearen und nichtlinearen Vorsteuerungen," *at-Automatisierungstechnik*, vol. 52, no. 1, pp. 3–12, 2004.

[8] D. Faustner, W. Kemmetmüller, and A. Kugi, "Flatness-based torque control of saturated surface-mounted permanent magnet synchronous machines," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 4, pp. 1201–1213, 2016. DOI: 10.1109/TCST.2015.2501345.

[9] G. Shen, W. Yao, B. Chen, K. Wang, K. Lee, and Z. Lu, "Automeasurement of the inverter output voltage delay curve to compensate for inverter nonlinearity in sensorless motor drives," *IEEE Transactions on Power Electronics*, vol. 29, no. 10, pp. 5542–5553, 2014. DOI: 10.1109/TPEL.2013.2293134.

[10] S. Zhang, "Artificial Intelligence in Electric Machine Drives: Advances and Trends," 2021. DOI: 10.36227/techrxiv.16782748.

[11] M. Stender, O. Wallscheid, and J. Bocker, "Comparison of Gray-Box and Black-Box Two-Level Three-Phase Inverter Models for Electrical Drives," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 9, pp. 8646–8656, 2021. DOI: 10.1109/TIE.2020.3018060.

[12] W. Kirchgässner, O. Wallscheid, and J. Böcker, "Data-Driven Permanent Magnet Temperature Estimation in Synchronous Motors with Supervised Machine Learning: A Benchmark," *IEEE Transactions on Energy Conversion*, vol. 36, no. 3, pp. 2059–2067, 2021. DOI: 10.1109/TEC.2021.3052546.

[13] O. Nelles, *Nonlinear System Identification*. Springer Berlin, Heidelberg, 2001. DOI: 10.1007/978-3-662-04323-3.

[14] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY: Springer New York, 2009. DOI: 10.1007/978-0-387-84858-7.

[15] C. M. Bishop, *Pattern Recognition and Machine Learning*. Berlin, Heidelberg: Springer-Verlag, 2006.

[16] V. N. Vapnik, "An overview of statistical learning theory," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 988–999, 1999. DOI: 10.1109/72.788640.

[17] S. Theodoridis, *Machine Learning: A Bayesian and Optimization Perspective*, 1st. USA: Academic Press, Inc., 2020.

[18] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

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

[20] T. Evgeniou, M. Pontil, and T. Poggio, "Regularization Networks and Support Vector Machines," *Advances in computational mathematics*, vol. 13, pp. 1–53, 2000.

[21] B. Schoelkopf and A. J. Smola, *Learning With Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 2003, vol. 98. DOI: 10.1198/jasa.2003.s269.

[22] G. Wahba and Y. Wang, "Representer Theorem," *Wiley StatsRef: Statistics Reference Online*, vol. 1155–1178, pp. 1–11, 2014. DOI: 10.1002/9781118445112.stat08200.

[23] N. Shawe-Taylor, John and Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004. DOI: 10.1017/CBO9780511809682.

[24] M. Papageorgiou, *Optimierung: Statische, Dynamische, Stochastische Verfahren*. Springer Berlin, Heidelberg, 2015. DOI: 10.1007/978-3-540-34013-3.

[25] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, 2006. DOI: 10.7551/mitpress/3206.001.0001.

[26] D. J. MacKay, "Introduction to Gaussian Processes," *NATO ASI series F: computer and systems sciences*, vol. 168, pp. 133–166, 1998.

[27] S. Särkkä, *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013. DOI: 10.1017/CBO9781139344203.

[28] D. Simon, *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. USA: Wiley-Interscience, 2006.

[29] C. Williams and M. Seeger, "Using the nyström method to speed up kernel machines," in *Advances in Neural Information Processing Systems*, vol. 13, MIT Press, 2000.

[30] A. Ranganathan, M. H. Yang, and J. Ho, "Online sparse Gaussian Process Regression and its applications," *IEEE Transactions on Image Processing*, vol. 20, no. 2, pp. 391–404, Feb. 2011. DOI: 10.1109/TIP.2010.2066984.

[31] J. Kivinen, A. J. Smola, and R. C. Williamson, "Online learning with kernels," *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2165–2176, 2004. DOI: 10.1109/TSP.2004.830991.

[32] V. Vapnik, S. E. Golowich, and A. Smola, "Support vector method for function approximation, regression estimation, and signal processing," *Advances in Neural Information Processing Systems*, pp. 281–287, 1997.

[33] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods - Support Vector Learning*, The MIT Press, Jan. 1998.

[34] O. Chapelle, "Training a support vector machine in the primal," *Neural Computation*, vol. 19, no. 5, pp. 1155–1178, 2007. DOI: 10.1162/neco.2007.19.5.1155.

[35] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter, "Pegasos: Primal estimated sub-gradient solver for SVM," *Mathematical Programming*, vol. 127, no. 1, pp. 3–30, 2011. DOI: 10.1007/s10107-010-0420-4.

[36] W. Liu, J. C. Príncipe, and S. Haykin, *Kernel Adaptive Filtering: A Comprehensive Introduction*. John Wiley and Sons, Mar. 2010. DOI: 10.1002/9780470608593.

[37] P. Honeine, "Analyzing Sparse Dictionaries for Online Learning with Kernels," *IEEE Transactions on Signal Processing*, vol. 63, no. 23, pp. 6343–6353, 2015. DOI: 10.1109/TSP.2015.2457396.

[38] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least-squares algorithm," *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2275–2285, 2004. DOI: 10.1109/TSP.2004.830985.

[39] S. Van Vaerenbergh and I. Santamaría, "A comparative study of kernel adaptive filtering algorithms," in *2013 IEEE Digital Signal Processing and Signal Processing Education Meeting (DSP/SPE)*, Software available at https://github.com/steven2358/kafbox/, 2013, pp. 181–186. DOI: 10.1109/DSP-SPE.2013.6642587.

[40] L. Csato and M. Opper, "Sparse On-Line Gaussian Processes," *Neural Computation*, vol. 14, no. 3, pp. 641–668, 2002. DOI: 10.1162/089976602317250933.

[41] A. J. Smola · and B. Schölkopf, "Nonlinear Component Analysis as a Kernel Eigenvalue Problem," *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, 1998. DOI: 10.1162/089976698300017467.

[42] S. Van Vaerenbergh, M. Lazaro-Gredilla, and I. Santamaria, "Kernel recursive least-squares tracker for time-varying regression," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 8, pp. 1313–1326, 2012. DOI: 10.1109/TNNLS.2012.2200500.

[43] C. Richard, J. C. M. Bermudez, and P. Honeine, "Online prediction of time series data with kernels," *IEEE Transactions on Signal Processing*, vol. 57, no. 3, pp. 1058–1067, 2009. DOI: 10.1109/TSP.2008.2009895.

[44] W. Liu and J. C. Príncipe, "Kernel affine projection algorithms," *Eurasip Journal on Advances in Signal Processing*, vol. 2008, no. 1, 2008. DOI: 10.1155/2008/784292.

[45] Joaquin Quinonero-Candela and Carl Edward Rasmussen, "A unifying view of sparse approximate gaussian process regression," *Journal of Machine Learning Research*, vol. 6, pp. 1939–1959, 2005.

[46] C. E. Rasmussen and H. Nickisch, "Gaussian processes for machine learning (gpml) toolbox," *The Journal of Machine Learning Research*, vol. 11, pp. 3011–3015, 2010, Software available at https://gitlab.com/hnickisch/gpml-matlab.

[47] H. Bijl, J. W. van Wingerden, T. B. Schön, and M. Verhaegen, "Online sparse Gaussian process regression using FITC and PITC approximations," *IFAC-PapersOnLine*, vol. 48, no. 28, pp. 703–708, 2015. DOI: 10.1016/j.ifacol.2015.12.212.

[48] M. K. Titsias, "Variational Learning of Inducing Variables in Sparse Gaussian Processes," *Proceedings of Machine Learning Research*, vol. 5, no. 9, pp. 567–574, 2009.

[49] T. D. Bui, J. Yan, and R. E. Turner, "A unifying framework for Gaussian Process pseudo-point approximations using power expectation propagation," *Journal of Machine Learning Research*, vol. 18, pp. 1–72, 2017.

[50] J. Hensman, N. Fusi, and N. D. Lawrence, "Gaussian processes for big data," in *Uncertainty in Artificial Intelligence*, vol. 29, AUAI Press, 2013. DOI: 10.1016/S0074-7696(01)08005-6.

[51] M. F. Huber, "Recursive gaussian process: On-line regression and learning," *Pattern Recognition Letters*, vol. 45, pp. 85–91, 2014. DOI: 10.1016/j.patrec.2014.03.004.

[52] C. A. Micchelli, Y. Xu, and H. Zhang, "Universal kernels," *Journal of Machine Learning Research*, vol. 7, pp. 2651–2667, 2006.

[53] D. Duvenaud, J. Lloyd, R. Grosse, J. Tenenbaum, and G. Zoubin, "Structure discovery in nonparametric regression through compositional kernel search," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 28, Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1166–1174.

[54] G. Meanti, L. Carratino, E. De Vito, and L. Rosasco, "Efficient hyperparameter tuning for large scale kernel ridge regression," in *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 151, PMLR, 28–30 Mar 2022, pp. 6554–6572.

[55] W. Kemmetmüller, D. Faustner, and A. Kugi, "Optimal torque control of permanent magnet synchronous machines using magnetic equivalent circuits," *Mechatronics*, vol. 32, pp. 22–33, 2015. DOI: 10.1016/j.mechatronics.2015.10.007.

[56] H. W. Coleman and W. G. Steele, *Experimentation, Validation, and Uncertainty Analysis for Engineers*, Fourth edition. John Wiley and Sons, 2018.

[57] J. S. Long and L. H. Ervin, "Correcting for heteroscedasticity with heteroscedasticity consistent standard errors in the linear regression model: Small sample considerations," *Indiana University, Bloomington, IN*, vol. 47405, pp. 1–33, 1998.

[58] N. Acharki, A. Bertoncello, and J. Garnier, "Robust prediction interval estimation for Gaussian processes by cross-validation method," *Computational Statistics and Data Analysis*, vol. 178, p. 107 597, 2022. DOI: 10.1016/j.csda.2022.107597.

[59] E. Schulz, M. Speekenbrink, and A. Krause, "A Tutorial on Gaussian Process Exploration-Exploitation," *Journal of Mathematical Psychology*, vol. 85, no. 1, pp. 1–16, 2018.

[60] S. Van Vaerenbergh, I. Santamaría, and M. Lázaro-Gredilla, "Estimation of the forgetting factor in kernel recursive least squares," *IEEE International Workshop on Machine Learning for Signal Processing, MLSP*, 2012. DOI: 10.1109/MLSP.2012.6349749.

# Eidesstattliche Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct - Regeln zur Sicherung guter wissenschaftlicher Praxis (in der aktuellen Fassung des jeweiligen Mitteilungsblattes der TU Wien), insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Vienna, April 2023

Aaron Raffeiner, BSc