TECHNISCHE UNIVERSITÄT WIEN

AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

Incorporation of a priori domain knowledge in function approximation using shape-constraint P-splines

DIPLOMARBEIT

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

Diplom-Ingenieur (Dipl.-Ing.)

supervised by

Dr. techn. T. Glück

submitted at the

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

by
Jakob Weber
Matriculation number a01326729
Address line 1
Address line 2
Austria

Vienna, Month Day, YEAR

Acknowledgements

First, I want to thank my supervisors DI DR. Tobias Glück, Dipl.-Ing. Dr.techn. Stephan Strommer and Dipl.-Phys. Dr. Claas Abert, Privatdoz. for giving me the possibility to conduct my master studies as well as supporting and encouraging me throughout the difficult time during the world-wide COVID-19 pandemic. Next, I want to thank my colleagues in the group for Complex Dynamical Systems at AIT Austrian Institute of Technology GmbH, in particular Amadeus Lobe, Markus Gurtner, Pietro Palmesi and David Gruber, for their support. Additionally, I want to thank my fellow students in the computational science master program Charlotte Bode, Lara Ost, Konrad von Kirchbach, Nadja Singer, Lukas Gosch and Dona Novakovic for interesting and weird discussions at lunch and throughout the whole time at the university. Finally, I want to thank my family and my close friends, especially Andre Kolar, Stefan Negovanovic, Patrick Aman, Martin Schnittler and also Peter Gerges, for their non-stop support and encouragement.

Vienna, March 29$^{\text{th}}$, 2021

Abstract

The massive amount of data generated in industrial processes leads to an increasing use of data-driven methods in control applications. A priori domain knowledge about these processes is often available and not utilized. Nevertheless, its incorporation is expected to enhance the quality and robustness of the data-driven model.

In this thesis, we propose an iterative algorithm to incorporate the given a priori domain knowledge into the modeling process using shape-constraint P-splines for one and two dimensional problems. The algorithm is expanded to higher dimensional problems using the concept of additive models. We further evaluate the quality of the proposed method using simulation data as well as real-world examples.

The results suggest that the proposed method enhances the quality of the fit especially in situations of sparse and/or noisy data. We therefore conclude that the use of shape-constraint P-splines to incorporate a priori domain knowledge is a valuable extension for any data-driven modeling tool set.

Kurzzusammenfassung

Die massive Menge an Daten, die in industriellen Prozessen generiert wird, führt zu einem zunehmenden Einsatz von datengetriebenen Methoden in Steuerungsanwendungen. A priori Domänenwissen über diese Prozesse ist oft vorhanden und wird nicht genutzt. Dennoch kann seine Einbeziehung die Qualität und Robustheit des datengetriebenen Modells verbessern.

In dieser Arbeit schlagen wir einen iterativen Algorithmus vor, um das gegebene a priori Domänenwissen in den Modellierungsprozess unter Verwendung von shape-constraint P-Splines für ein- und zweidimensionale Probleme einzubeziehen. Der Algorithmus wird unter Verwendung des Konzepts der additiven Modelle auf höherdimensionale Probleme erweitert. Wir evaluieren die Qualität der vorgeschlagenen Methode anhand von Simulationsdaten und realen Beispielen.

Die Ergebnisse deuten darauf hin, dass die vorgeschlagene Methode die Qualität der Modellierung insbesondere in Situationen mit spärlichen und/oder verrauschten Daten verbessert. Wir kommen daher zu dem Schluss, dass die Verwendung von shape-constraint P-Splines zur Einbindung von a priori Domänenwissen eine wertvolle Erweiterung für jedes datengetriebene Modellierungs-Toolset ist.

Contents

List of Figures

List of Tables

1 Introduction

The digitalization of complex, large-scale industrial plants is leading to a massive increase of process data. This data can be used to enhance the overall understanding of the characterizing physical process inside the plant. Modern observer and control concepts are used to improve the efficiency of the plant. They are based on mathematical models of the underlying process characterized by a high accuracy and low computational effort to enable real-time applications. An exact physical description of the relevant quantities as mathematical model is nevertheless often not feasible due to the complexity of the process as well as computational and measurement limitations.

Data-driven approaches are state-of-the-art in many fields, including e.g. image and speech recognition. The usage of data-driven methods, e.g. artificial neural networks, parametric models, etc., to model process quantities of interest for which measurements are too expensive or not practical, also gains more and more influence and acceptance in the field of process control and optimization. The application of any specific data-driven algorithm depends on issues like data quality, interpretability of the model and computational efficiency.

In most process optimization tasks, specific domain knowledge in form of physical theories and a priori knowledge is available. The combination of the use of physics-based (domain knowledge) and data-driven modeling techniques is called hybrid modeling or grey-box modeling. It lies between the two modeling extrema of white-box models, which are derived from first principles, and black-box models, which are derived from data only [1]. The incorporation of the domain specific knowledge, also known as a priori knowledge, in state-of-the-art data-driven approaches is not trivial and not solved for most algorithms. Nevertheless, its inclusion should improve the interpretability of the model, which itself is of importance in the context of the emerging field of explainable artificial intelligence (XAI). XAI refers to modeling approaches and techniques in which the main goal is that the resulting model is understandable by humans [2].

In this thesis, we present an algorithm for efficient, static, multi-dimensional function approximation using a priori domain knowledge. The algorithm is based on structured additive regression using B-splines [3]. We use user-defined constraints to include a priori domain knowledge in the fitting process, see [4] and [5]. We produce interpretable and efficient models based on the given domain knowledge. The incorporation of domain specific knowledge improves the model quality and robustness as well as the interpolation behavior in situations where measured data is sparse and/or noisy. Furthermore, we evaluate the algorithm using noisy samples from artificial test functions with known behavior as well as real-world data.

1.1 Related Work

We will now discuss commonly used data-driven algorithms. The discussion includes the following model approaches:

- Parametric models: Linear and polynomial regression
- Non-parametric models: Basis function models
- Gaussian process regression
- Artificial neural networks
- Look-up tables

We will focus on the interpretability, computational efficiency and the ability to include domain knowledge of the individual modeling approaches. The list given above is not intended to give a complete overview of the field of data-driven modeling but rather to be an introduction.

The common starting point for the various data-driven modeling approaches is that we have some multi-variate data $\mathcal{D}$, i.e.

$$\mathcal{D} = \left\{ (x_1^{(i)}, \ldots, x_q^{(i)}, y^{(i)}),\ i = 1, \ldots, n \right\}, \tag{1.1}$$

where $x_1, \ldots, x_q$ are the inputs, $y^{(i)}$ is the measured output and n is the number of measurement samples. For ease of presentation, we restrict the following discussion to the single-input setting, i.e. $(x^{(i)}, y^{(i)}),\ i = 1, \ldots, n$. The generalization to multiple input dimensions is given in the respective literature. We then use the given data to estimate a model function $f(x)$ which is used to predict the response or output variable y, i.e.

$$y = f(x). \tag{1.2}$$

Therefore, we are in the setting of supervised learning.

1.1.1 Parametric Models

According to [6], parametric models are defined as models that can describe the true process behavior using a finite number of parameters. An example is given by the linear regression model for one input variable x as

$$y = f(x) = \beta_0 + \beta_1 x. \tag{1.3}$$

Both parameters β_0 and β_1 allow for a direct interpretation as β_0 is the intercept, i.e. the output for the input $x = 0$, and β_1 is the slope, i.e. the constant defining the relationship between the change of the output y with respect to the change of the input x. The interpretability of linear regression models is therefore very high.

Linear regression models are widely used and part of standard software tools. Their parameters can be efficiently computed using the least squares algorithm. One major drawback is that they can only recover linear relationships between input and output variables. They are therefore quite restrictive and do not allow the incorporation of a priori domain knowledge except being increasing or decreasing by the sign of the slope β_1.

An extension of the linear regression model is given by polynomial regression. Here, we try to model the output data y using a polynomial of degree p, i.e.

$$y = f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_p x^p. \tag{1.4}$$

Polynomial regression introduces more flexibility in the fitting process, since the restriction of linear relationship is relaxed to a polynomial relationship of degree p. As for linear models, the interpretability of the parameters is given. We can use the least squares algorithm for parameter estimation. The incorporation of some a priori domain knowledge is possible, e.g. as the degree of the polynomial regression model. The major problem of polynomial regression is that the model function becomes quite wiggly for high polynomial degrees p.

Linear and polynomial regression models are so-called global models. Their parameters act on the complete input space. This property makes the incorporation of specific a priori domain knowledge, e.g. unimodal behavior, difficult and in most cases almost impossible for parametric models.

1.1.2 Non-parametric Models

In [6], non-parametric models are defined as models which require an infinite number of parameters to describe a process exactly. In almost all practical applications this infinite series is approximated by a finite number of parameters using the basis function approach given by

$$y = f(x) = \sum_{i=1}^{d} \Phi_i(x, \theta_i)\beta_i, \tag{1.5}$$

with the basis functions $\Phi_i(\cdot)$, the parameters β_i, the input variable x and the basis function parameters θ_i. The output y is therefore given by a linear combination of d basis functions $\Phi_i(\cdot)$. To model a nonlinear relationship between y and x, the basis functions $\Phi_i(\cdot)$ need to be nonlinear. Commonly used basis functions are, e.g. the *hat function*, the *Gaussian*, the *hinge function* or *splines* [7].

A widely used algorithm utilizing the basis function approach is called Multivariate Adaptive Regression Splines (MARS) [8]. MARS approximates data, as example again for a single input dimension, using the following model

$$y = \sum_{i=1}^{d} \Phi_i(x)\theta_i \tag{1.6}$$

with the constant parameters θ_i. The basis functions $\Phi_i(\cdot)$ are one of the following three alternatives:

1. $\Phi_i(x) = 1$, representing the intercept.
2. $\Phi_i(x) = \max(0, x - c_i)$ or $= \max(0, c_i - x)$, representing the *hinge function* h_i using the constant value c_i.
3. $\Phi_i(x) = h_i h_j$, representing a product of two *hinge functions*.

MARS fits the model by a recursive splitting approach. More information can be found in [7] and [8]. MARS models are more flexible compared to the parametric linear and polynomial regression models. As only hinge functions and products of hinge functions are used, MARS models are efficient and in general simple to understand and interpret. To our knowledge, there is currently no possibility to include a priori domain knowledge in the fitting process when using MARS.

Another popular method utilizing basis functions is the use of *splines*. Splines are defined as piece-wise polynomials on a sequence of knots. Further information can be found in Section 2.3 and [9]. We focus on the so-called B-splines or basis-splines. Using a B-spline consisting of d B-spline basis functions of order l to model some data, we obtain the model formulation as

$$y = f(x) = \sum_{j=1}^{d} B_j^l(x)\beta_j, \tag{1.7}$$

with the B-spline basis functions B_j^l and the parameters β_j. The parameters can be easily calculated by the least squares algorithm. The usage of splines allows a lot of flexibility and computational efficiency. A priori domain knowledge can be incorporated using an iterative variante of the penalized least squares algorithm with a sophisticated choice of mapping and weighting matrices, see Chapter 3, and, e.g. [4] or [5].

The basis function approach in (1.5) may be extended by changing the parameters β_i to more complex forms. An example for this is the so-called local linear neuro-fuzzy model, for which each parameter β_i is changed to be a *local linear model* and each basis function $\Phi_i(\cdot)$ is then called *validity function* determining the region of validity of the local linear model [6]. The validity functions are normalized for any model input x, i.e.

$$\sum_{i=1}^{d} \Phi_i(x) = 1 \tag{1.8}$$

and typically chosen to be *Gaussian* functions, i.e.

$$\Phi_i(x) = a_i \exp\left(\frac{(x - \mu_i)^2}{\sigma_i^2}\right), \tag{1.9}$$

with the normalization constant a_i and the parameters μ_i and σ_i determining the location and scale of the Gaussian function. The output of the local linear neuro-fuzzy model using d local linear models is then given by

$$y = \sum_{i=1}^{d} \Phi_i(x) \left(\beta_{i0} + \beta_{i1} x_1\right). \tag{1.10}$$

The second term in the summation are the *local linear models*. The parameters β_{ij} for $i = 1, \ldots, d$ and $j = 0, 1$ as well as the parameters μ_i and σ_i from the validity functions Φ_i need to be optimized. This is done in the LOLIMOT algorithm, see [6].

Local linear models as extension of linear models possess more flexibility with regards to nonlinear relationships in the data. They can also be efficiently evaluated after the iterative training process. The interpretability is high since each local linear model contributes to the prediction according to its validity function. The ability to include a priori domain knowledge in the fitting process is currently not available.

1.1.3 Gaussian Process Regression

Gaussian process regression is a non-parametric method using a mean function $m(x)$ and a covariance function $k(x, x')$. The covariance function describes the distance between the two inputs x and x'. The commonly used covariance function is the so-called *Squared-Exponential* function, i.e.

$$k_{\mathrm{SE}}(x_m, x_n) = \sigma_f^2 \exp\left(-\frac{\|x_m - x_n\|^2}{2l^2}\right) + \sigma_m^2 \delta_{m,n}, \tag{1.11}$$

where σ_f^2 is the variance of the true function, σ_m^2 is the variance of the measurement noise and $\delta_{m,n}$ is the Kronecker-Delta. The parameter l tunes the smoothness of the *Squared-Exponential*.

We will use the complete data $\mathcal{D}$ with the input $\mathbf{X}^{\mathrm{T}} = [x^{(1)}, \ldots, x^{(n)}]$ and the output $\mathbf{y}^{\mathrm{T}} = [y^{(1)}, \ldots, y^{(n)}]$ to evaluate the Gaussian Process at a new data point x, see [10] and [11]. To do this, with a little abuse of notation, we use the joint distribution between the input data $\mathbf{X}$ and the new data point x, i.e.

$$\begin{bmatrix} \mathbf{y} \\ y \end{bmatrix} \sim \mathcal{N}\left(m(x), \begin{bmatrix} k(\mathbf{X}, \mathbf{X}) & k(\mathbf{X}, x) \\ k(x, \mathbf{X}) & k(x, x) \end{bmatrix}\right), \tag{1.12}$$

using the mean function m and the covariance function k to calculate the mean value and the variance of the data point x as

$$\mathrm{E}(x) = m(x) + k(x, \mathbf{X})(k(\mathbf{X}, \mathbf{X}))^{-1}(\mathbf{y} - m(\mathbf{X})) \tag{1.13}$$

and

$$\mathrm{Var}(x) = k(x,x) - k(x,\mathbf{X})(k(\mathbf{X},\mathbf{X}))^{-1}k(\mathbf{X},x). \tag{1.14}$$

The usage of Gaussian Process regression allows for the incorporation of a priori knowledge through the definition of the covariance function $k(x,x')$. An example can be found in the standard textbook on Gaussian Process regression in [11], in which they fit the Mauna Loa Atmospheric Carbon Dioxide data set using a complex covariance function derived by combining simple, interpretable covariance functions. Nevertheless, as seen in (1.13) and (1.14), we need the information of the whole training data set to evaluate the mean value and the covariance which may lead to a high computational load for large data sets.

1.1.4 Artificial Neural Networks

Artificial neural networks are currently the state-of-the-art solution method for many problems ranging from computer vision over time-series prediction to regression tasks. They are constructed as coarse model of the human brain, consisting of neurons which are connected by some weights. These connections are adapted in the learning process using an algorithm called *backpropagation*. They utilize a high number of parameters to model high-dimensional relationships in the data. Further information can be found in standard textbooks about neural networks, e.g. [12] or [13].

In terms of modeling flexibility, artificial neural networks of sufficient size are proven to be able to represent a wide variety of functions by so-called universal approximation theorems, cf. [14] and [15]. The computational complexity of a neural network depends on its size, aka. the number of parameters. Large networks need many training samples to generate sufficiently accurate predictions. Artificial neural networks are an example of a black-box model. The inclusion of a priori domain knowledge into the learning process of neural networks is possible for specific types of knowledge using the concepts of hints, see [16] and [17].

1.1.5 Look-up Tables

A look-up table is an array of values, which allows to replace computational expensive computations with inexpensive array indexing operations. The values in the look-up table are most often computed and stored beforehand. To gain higher resolution, interpolation techniques such as linear or quadratic interpolation may be applied to look-up tables.

Look-up tables are a standard tool in many fields. They are extremely efficient in terms of computation time. One problem that occurs is the exponential increase in size with the number of dimensions for the look-up table. As example, a 2×2-table needs to save 4 values, while a $2 \times 2 \times 2$ table already needs 8 values without gaining additional accuracy. Another issue is that the values in the look-up table may come from computationally expensive or complex physical models. The creation of the look-up table is then time-consuming.

Lattice regression tackles this problems by jointly estimating all look-up table values by minimizing the regularized interpolation error on training data [18]. They state that using ensembles of look-up tables which combine several *tiny* lattices enables linear scaling in

the number of input dimension even for high dimensions [19]. They further state that lattice regression may be used to incorporate a priori domain knowledge like monotonicity, shape or unimodality into the fitting process, see [20] or [21].

1.2 Outline

The base of this thesis is a literature study about function fitting using a priori domain knowledge focusing on non-parametric techniques and neural networks. We decided to use structured additive regression [3] utilizing B-splines and tensor-product B-splines as non-linear basis functions since these offer a computationally efficient implementation due to their locality feature. This approach enables flexible, multi-dimensional function fitting. We further expanded this method by applying a priori domain knowledge through the use of user-defined constraints. These constraints consist of mapping matrices determined by the type of constraint, and weighting matrices, determining whether the constraint is active, see [4] and [5].

We are able to incorporate the following a priori domain knowledge:

- Jamming, i.e. $f(x^{(p)}) \approx y^{(p)}$ for some point p with high fidelity
- Boundedness, i.e. $f(x) \geq \mathrm{B}$ or $f(x) \leq \mathrm{B}$ for some bound B
- Monotonicity, i.e. $f'(x) \geq 0$ or $f'(x) \leq 0$
- Curvature, i.e. $f''(x) \geq 0$ or $f''(x) \leq 0$
- Unimodality, i.e.
  $f'(x) > 0,\ x < m$ and $f'(x) < 0,\ x > m$ for $m = \arg\max_x f(x)$ or
  $f'(x) < 0,\ x < m$ and $f'(x) > 0,\ x > m$ for $m = \arg\min_x f(x)$

The thesis is divided into 6 chapters: Chapter 2 provides an overview of the fundamental mathematical concepts used. We focus on the description of linear models, model selection and B-splines as well as on the topic of structured additive regression. In Chapter 3, we present the algorithm to incorporate a priori domain knowledge using the concepts given in Chapter 2. In Chapter 4, we show some aspects of the practical application of the algorithm on noisy and sparse data. Chapter 5 provides examples using real-world data. In Chapter 6, we give a summary and outline to future, possible work.

2 Fundamentals

This chapter summarizes the fundamentals of regression. Excellent overviews can be found in the textbooks [3], [7] and [22]. The shown fundamentals are strongly aligned with the presentation given in [3]. Section 2.1 gives an overview of the model assumptions used throughout this work. Furthermore, Section 2.2 outlines methods to evaluate and compare different models against each other in terms of complexity and accuracy. Section 2.3 is devoted to the spline definitions.

2.1 Linear Models

Given the data set $\mathcal{D} = \{(x_1^{(i)}, \dots, x_q^{(i)}, y^{(i)}),\ i = 1, 2, \dots, n\}$ comprising n data points, we aim to model the relation between the q inputs $x_1, \dots, x_q$ and the output y with a deterministic function $f(x_1, \dots, x_q)$. Since we cannot assume that the relationship between the inputs and the output is exact, we will include a random part ϵ, which is used to model e.g. measurement errors. It is typically assumed that this part is additive and thus

$$y = f(x_1, \dots, x_q) + \epsilon. \tag{2.1}$$

We would like to estimate the unknown function $f(x_1, \dots, x_q)$. For this, some assumptions on the model structure are made:

1. *The unknown function f is a linear combination of the inputs*

   The function $f(x_1, \dots, x_q)$ is modeled as a linear combination of inputs, i.e.

   $$f(x_1, \dots, x_q) = \beta_0 + \beta_1 x_1 + \dots + \beta_q x_q, \tag{2.2}$$

   with unknown parameters $\beta_0, \dots, \beta_q$. Note that the model (2.2) is linear in its parameters as well as in its inputs. The parameter β_0 is called intercept or bias in the machine learning community, see [12]. We introduce the input vector $\mathbf{x}^{\mathrm{T}} = [1, x_1, \dots, x_q] \in \mathbb{R}^{1\times q+1}$ and the parameter vector $\boldsymbol{\beta}^{\mathrm{T}} = [\beta_0, \beta_1, \dots, \beta_q] \in \mathbb{R}^{1\times q+1}$ to obtain

   $$f(x_1, \dots, x_q) = \mathbf{x}^{\mathrm{T}}\boldsymbol{\beta}. \tag{2.3}$$

2. *Additive errors*

   An additional assumption of linear models is additivity of errors, which means that

$$y = \mathbf{x}^{\mathrm{T}}\boldsymbol{\beta} + \epsilon. \tag{2.4}$$

   This is reasonable for many practical applications, even though it appears quite restrictive.

To estimate the unknown parameters or coefficients $\boldsymbol{\beta}$, we define the output vector $\mathbf{y}^{\mathrm{T}} = [y^{(1)}, \ldots, y^{(n)}] \in \mathbb{R}^{1\times n}$ and the error vector $\boldsymbol{\epsilon}^{\mathrm{T}} = [\epsilon^{(1)}, \ldots, \epsilon^{(n)}] \in \mathbb{R}^{1\times n}$ as well as the design matrix

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^{(1)} & \ldots & x_q^{(1)} \\ \vdots & \vdots & & \vdots \\ 1 & x_1^{(n)} & \ldots & x_q^{(n)} \end{bmatrix} \in \mathbb{R}^{n\times q+1} \tag{2.5}$$

and generate n equations like (2.4), which can be combined to

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}. \tag{2.6}$$

We assume that the design matrix $\mathbf{X}$ has full column rank, i.e. $\mathrm{rank}(\mathbf{X}) = q + 1$, implying linear independence of the columns of $\mathbf{X}$, which is necessary to obtain a unique estimator for the regression coefficients $\boldsymbol{\beta}$, see [3].

Another necessary requirement is that the number of data points n is larger or equal to the number of regression parameters $p = q + 1$, which is equivalent to the statement that the linear system in (2.6) is not underdetermined.

In addition to the assumptions on the unknown function f, the necessary assumptions on the error vector $\boldsymbol{\epsilon}$ are [3]:

1. *Expectation of the error*
   The errors have a mean value of zero, i.e. $\mathrm{E}(\boldsymbol{\epsilon}) = \mathbf{0}$.

2. *Variances and correlation structure of the errors*
   We assume constant error variance with $\mathrm{Var}(\epsilon^{(i)}) = \sigma^2$ for $i = 1, 2, \ldots, n$. This property is called homoscedasticity. Additionally, we assume that the errors are uncorrelated, which means $\mathrm{Cov}(\epsilon^{(i)}, \epsilon^{(j)}) = 0$ for $i \neq j$. The combination of these assumptions lead to the covariance matrix $\mathrm{Cov}(\boldsymbol{\epsilon}) = \mathrm{E}(\boldsymbol{\epsilon}\boldsymbol{\epsilon}^{\mathrm{T}}) = \sigma^2\mathbf{I}$.

3. *Assumptions on the input and design matrix*
   We have to distinguish two cases where the inputs are deterministic or stochastic. In most cases, the inputs and the output are stochastic and hence all model assumptions are conditioned on the design matrix (2.5). This means that the input $\mathbf{x}^{(i)}$ and the errors $\epsilon^{(i)}$ are not stochastically independent. For notational simplicity, we usually suppress the dependence on the design matrix.

4. *Gaussian errors*
   The errors follow at least approximately a normal distribution. Together with Assumption 1 and 2, we obtain that $\epsilon^{(i)} = \mathcal{N}(0, \sigma^2)$ holds.

Summarizing, we have the following model assumptions:

$$\boldsymbol{\mu} = \mathrm{E}(\mathbf{y}) = \mathbf{X}\boldsymbol{\beta} \tag{2.7}$$

$$\mathrm{Cov}(\mathbf{y}) = \sigma^2\mathbf{I}, \tag{2.8}$$

yielding

$$\mathbf{y} \sim \mathcal{N}(\mathbf{X}\boldsymbol{\beta}, \sigma^2\mathbf{I}). \tag{2.9}$$

A linear model with multiple inputs can therefore be interpreted as a multi-variate normal distribution with its mean vector given by (2.7) and its covariance matrix given by (2.8). To specify the linear model given in (2.9), we need to estimate the regression coefficients $\boldsymbol{\beta}$ and the variance σ^2.

2.1.1 Estimation of the Regression Parameters β

The linear model given in (2.9) features the unknown parameters $\boldsymbol{\beta}$ and σ^2, which need to be estimated using the given data $\mathcal{D}$. In the following, the estimator $\hat{\boldsymbol{\beta}}$ is introduced. The two methods to estimate the regression parameters in the context of linear models are the method of Least Squares (LS) and the method of Maximum Likelihood (ML).

The Method of Least Squares

The unknown regression parameters $\boldsymbol{\beta} \in \mathbb{R}^p$ are estimated by minimizing the sum of squared error

$$\mathrm{LS}(\mathbf{y}, \boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 = \sum_{i=1}^{n} \epsilon_i^2 = \boldsymbol{\epsilon}^{\mathrm{T}}\boldsymbol{\epsilon}, \tag{2.10}$$

with respect to $\boldsymbol{\beta}$, see, e.g. [7]. Rewriting (2.10) leads to the least squares criterion

$$\begin{aligned} \mathrm{LS}(\mathbf{y}, \boldsymbol{\beta}) &= (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \\ &= \mathbf{y}^{\mathrm{T}}\mathbf{y} - 2\mathbf{y}^{\mathrm{T}}\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\beta}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\boldsymbol{\beta}. \end{aligned} \tag{2.11}$$

The first-order necessary condition for optimality, cf. [23], reads as

$$\frac{\partial \mathrm{LS}(\mathbf{y}, \boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = -2\mathbf{X}^{\mathrm{T}}\mathbf{y} + 2\boldsymbol{\beta}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X} = 0. \tag{2.12}$$

The second-order condition for optimality requires the Hessian, i.e.

$$\frac{\partial^2 \mathrm{LS}(\mathbf{y}, \boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^{\mathrm{T}}} = 2\mathbf{X}^{\mathrm{T}}\mathbf{X}, \tag{2.13}$$

to be positive-definite. Since the design matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$ is assumed to have full rank, the matrix $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ is positive-definite. The least squares estimate $\hat{\boldsymbol{\beta}}_{LS}$ is hence obtained, see (2.12), by solving the so-called *normal equations*

$$\mathbf{X}^{\mathrm{T}}\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^{\mathrm{T}}\mathbf{y}. \tag{2.14}$$

Since $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ is positive-definite, the *normal equations* (2.14) feature a unique solution given by the least squares estimator

$$\hat{\boldsymbol{\beta}}_{\mathrm{LS}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}. \tag{2.15}$$

Maximum Likelihood Estimation

Under the normality assumption and given the data $\mathcal{D}$, the likelihood is defined, see [22], as

$$\mathcal{L}(\boldsymbol{\beta}, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})\right). \tag{2.16}$$

The log-likelihood is then given by taking the logarithm of (2.16) as

$$l(\boldsymbol{\beta}, \sigma^2) = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}). \tag{2.17}$$

Thus, maximizing the log-likelihood $l(\boldsymbol{\beta}, \sigma^2)$ with respect to $\boldsymbol{\beta}$ is equivalent to minimizing the least squares criterion given in (2.10). The maximum likelihood estimator $\hat{\boldsymbol{\beta}}_{\mathrm{ML}}$ is therefore equivalent to the least squares estimator $\hat{\boldsymbol{\beta}}_{\mathrm{LS}}$ in (2.15).

2.1.2 Estimation of the Variance σ^2

The estimation of the variance σ^2 is necessary for the construction of confidence intervals of the regression parameters and for the construction of prediction intervals. It is further used in all kinds of statistical tests as well as in model selection approaches and model assessment criteria [24].

Maximum Likelihood Estimation

The first-order necessary condition for optimality in this case yields

$$\frac{\partial l(\boldsymbol{\beta}, \sigma^2)}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) = 0. \tag{2.18}$$

Substituting the maximum likelihood estimator $\hat{\boldsymbol{\beta}}_{\mathrm{ML}}$, note the equivalence with the least squares estimator $\hat{\boldsymbol{\beta}}_{\mathrm{LS}}$ given in (2.15), for $\boldsymbol{\beta}$ results in the maximum likelihood estimator

$$\hat{\sigma}^2_{ML} = \frac{(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}_{\mathrm{LS}})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}_{\mathrm{LS}})}{n} = \frac{1}{n}\hat{\boldsymbol{\epsilon}}^{\mathrm{T}}\hat{\boldsymbol{\epsilon}} \tag{2.19}$$

where $\hat{\boldsymbol{\epsilon}}$ is the estimate of the error $\boldsymbol{\epsilon}$. This estimator for σ^2 is rarely used since it is biased, i.e. $\mathrm{E}(\hat{\sigma}^2_{\mathrm{ML}}) \neq \sigma^2$, see [3].

Restricted Maximum Likelihood Estimation

The mean value of the sum of squared residuals is $\mathrm{E}(\hat{\boldsymbol{\epsilon}}^{\mathrm{T}}\hat{\boldsymbol{\epsilon}}) = (n-p)\sigma^2$. Hence, another estimator for σ^2 is given by

$$\hat{\sigma}^2_{\mathrm{REML}} = \frac{1}{n-p}\hat{\boldsymbol{\epsilon}}^{\mathrm{T}}\hat{\boldsymbol{\epsilon}}. \tag{2.20}$$

The restricted maximum likelihood estimator (REML) $\hat{\sigma}^2_{\mathrm{REML}}$ is in general less biased [3]. Therefore, it is the commonly used estimator for the variance σ^2.

2.1.3 The Hat Matrix

Using the least squares estimator (2.15), we can estimate the mean of $\mathbf{y}$ by

$$\widehat{\mathrm{E}(\mathbf{y})} = \hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}_{\mathrm{LS}}. \tag{2.21}$$

Substituting $\hat{\boldsymbol{\beta}}_{\mathrm{LS}}$ in (2.21) by (2.15) results in

$$\hat{\mathbf{y}} = \mathbf{H}\mathbf{y}, \tag{2.22}$$

with the matrix

$$\mathbf{H} = \mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}} \in \mathbb{R}^{n\times n}, \tag{2.23}$$

which is called *hat matrix*, cf. [3]. Using the hat matrix $\mathbf{H}$, we can express the residuals $\hat{\epsilon}^{(i)} = y^{(i)} - \hat{y}^{(i)}$ in matrix notation as

$$\hat{\boldsymbol{\epsilon}} = \mathbf{y} - \hat{\mathbf{y}} = (\mathbf{I} - \mathbf{H})\mathbf{y}. \tag{2.24}$$

The hat matrix $\mathbf{H}$ has the following useful properties:

- $\mathbf{H}$ is symmetric.
- $\mathbf{H}$ is idempotent, i.e. $\mathbf{H}^2 = \mathbf{H}$.
- The rank of $\mathbf{H}$ is equal to its trace.

- $\frac{1}{n} \leq h_{ii} \leq 1$, if all data points are different, i.e. $x^{(i)} \neq x^{(j)}$ for $i \neq j$. Here, h_{ii} are the diagonal elements of $\mathbf{H}$.
- The matrix $(\mathbf{I} - \mathbf{H})$ is also idempotent and symmetric, with $\text{rank}(\mathbf{I} - \mathbf{H}) = n - p$.

The hat matrix $\mathbf{H}$ is used in model selection techniques like cross-validation since its trace acts as a measure for the degrees of freedom of the model, as well as in outlier detection and in diagnostic plots for linear models. Note that the trace of the hat matrix $\mathbf{H}$ is equal to the number of parameters for linear models.

2.2 Model Selection

Linear models are widely exploit for regression problems on large data sets ($n \gg 1$), because the solution of the *normal equations* (2.14) can be computed efficiently even for large n. If these data sets also contain a large number of inputs ($q \gg 1$), the situation becomes more complicated since possible interaction effects or correlation of input variables may occur. These interaction terms limit the, otherwise perfect, interpretability of the linear model.

Therefore, we need techniques and criteria to select the *best possible model* out of the variety of different models for a given data set. Model assessment criteria, see Section 2.2.1, are used to compare different models while subset selection techniques, see Section 2.2.2, give an algorithmic approach to model generation. Furthermore, we can influence the estimated coefficients $\boldsymbol{\beta}$ directly via regularization, see Section 2.2.3.

2.2.1 Model Assessment Criteria

One way of comparing various models, i.e. models using different sets of inputs, is the use of model assessment criteria. Generally, model assessment criteria can be split in two components. The first one measures the quality of fit, e.g., using the sum of squared errors, while the second measures the complexity of the model. Most model assessment criteria are based on the sum of the expected squared prediction error (SPSE), which is also known as *generalization error*. Therefore, the derivation of the SPSE is given next.

Sum of Expected Squared Prediction Error

Given independent observations $y^{(i)}$, $i = 1, 2, \ldots, n$ and a subset of q inputs $\{x_0 = 1, x_1, x_2, \ldots, x_q\}$, we want to measure the prediction quality. The specific models are defined by numbers $M \subset \{1, \ldots, p\}$ of used inputs with corresponding design matrix $\mathbf{X}_M$. Moreover, $|M|$ is the cardinal number of M, i.e. the number of inputs included in the model. The least squares estimator for $\boldsymbol{\beta}$, cf. (2.15), is then given by

$$\hat{\boldsymbol{\beta}}_M = (\mathbf{X}_M^{\mathrm{T}} \mathbf{X}_M)^{-1} \mathbf{X}_M^{\mathrm{T}} \mathbf{y}.$$

The data $\mathbf{y}$ can be interpreted as a random variable. We can then define an estimator $\hat{\mathbf{y}}_M$ for the vector $\boldsymbol{\mu}$ of expectations $\mu^{(i)} = \mathrm{E}(y^{(i)})$ as

$$\hat{\mathbf{y}}_M = \mathbf{X}_M \hat{\boldsymbol{\beta}}_M. \tag{2.25}$$

Moreover, it is easy to show that the following properties hold true using the hat matrix $\mathbf{H}_M = \mathbf{X}_M(\mathbf{X}_M^{\mathrm{T}}\mathbf{X}_M)^{-1}\mathbf{X}_M^{\mathrm{T}}$ defined in (2.23):

(i) $\mathrm{E}(\hat{\mathbf{y}}_M) = \mathbf{H}_M \mathrm{E}(\mathbf{y})$

(ii) $\mathrm{Cov}(\hat{\mathbf{y}}_M) = \sigma^2 \mathbf{H}_M$

(iii) $\sum_{i=1}^{n} \mathrm{Var}(\hat{y}_M^{(i)}) = \mathrm{trace}(\mathbf{H}_M)\sigma^2 = |M|\sigma^2$

(iv) Sum of Mean Squared Error

$$\begin{aligned} \mathrm{SMSE} &= \sum_{i=1}^{n} \mathrm{E}\left(\hat{y}_M^{(i)} - \mu^{(i)}\right)^2 \\ &= \sum_{i=1}^{n} \mathrm{E}\left((\hat{y}_M^{(i)} - \mathrm{E}(\hat{y}_M^{(i)})) + (\mathrm{E}(\hat{y}_M^{(i)}) - \mu^{(i)})\right)^2 \\ &= |M|\sigma^2 + \sum_{i=1}^{n} \left(\mathrm{E}(\hat{y}_M^{(i)}) - \mu^{(i)}\right)^2. \end{aligned} \tag{2.26}$$

Note that the estimator (2.25) can be regarded as a prediction for future observations of the form

$$y^{(n+i)} = \mu^{(i)} + \epsilon^{(n+i)} \tag{2.27}$$

for new input data $\{(x_1^{(i)}, \ldots, x_q^{(i)}), i = 1, 2, \ldots, n\}$. Thus, we can derive the SPSE as

$$\begin{aligned} \mathrm{SPSE} &= \sum_{i=1}^{n} \mathrm{E}\left(y^{(n+i)} - \hat{y}_M^{(i)}\right)^2 \\ &= \sum_{i=1}^{n} \mathrm{E}\left((y^{(n+i)} - \mu^{(i)}) - (\hat{y}_M^{(i)} - \mu^{(i)})\right)^2 \\ &= \sum_{i=1}^{n} \mathrm{E}\left(y^{(n+i)} - \mu^{(i)}\right)^2 + 2\mathrm{E}\left((y^{(n+i)} - \mu^{(i)})(\hat{y}_M^{(i)} - \mu^{(i)})\right) + \mathrm{E}\left(\hat{y}_M^{(i)} - \mu^{(i)}\right)^2 \\ &= \sum_{i=1}^{n} \mathrm{E}\left(y^{(n+i)} - \mu^{(i)}\right)^2 + \sum_{i=1}^{n} \mathrm{E}\left(\hat{y}_M^{(i)} - \mu^{(i)}\right)^2 \\ &= n\sigma^2 + \mathrm{SMSE} \\ &= n\sigma^2 + |M|\sigma^2 + \sum_{i=1}^{n} \left(\mathrm{E}(\hat{y}_M^{(i)}) - \mu^{(i)}\right)^2. \end{aligned} \tag{2.28}$$

The SPSE can be split into three parts:

1. *Irreducible Prediction Error Term*: $n\sigma^2$
   This term cannot be reduced through model selection techniques since it only contains the number of data points n and the variance σ^2.

2. *Variance Error Term*: $|M|\sigma^2$
   The second term contains the number of used variables $|M|$ as well as the variance σ^2. It can therefore be reduced by reducing the model complexity, i.e. by using a smaller number of inputs.

3. *Squared Bias Error Term*: $\sum_{i=1}^{n} \left(\mathrm{E}(\hat{y}_M^{(i)}) - \mu^{(i)}\right)^2$
   The last term can be interpreted as bias. It can be reduced by increasing the model complexity, i.e. by using additional inputs.

The SPSE acts as an example of the bias-variance trade-off, which is characteristic for all statistical models. It states that by increasing the model complexity, the bias is reduced but instead the variance is increased. On the other hand, by decreasing model complexity, the variance of the model is reduced, but the bias is increased, see Figure 2.1 and [12].

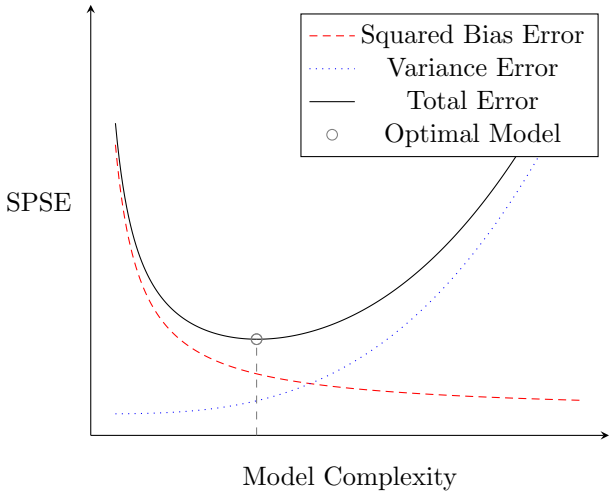

Figure 2.1: Bias-variance trade-off.

In practice, the true value for the SPSE is not accessible since $\mu^{(i)}$ and σ^2 are unknown. Therefore, we need to estimate the SPSE. This can be done by using one of the following two strategies:

1. *Estimate SPSE using new and independent data*
   If new observations are available, the SPSE can be estimated by

$$\widehat{\mathrm{SPSE}} = \sum_{i=1}^{n} \left(y^{(n+i)} - \hat{y}_M^{(i)}\right)^2. \tag{2.29}$$

   These new observations can also be some held-out validation data from a train-validation split of the given data.

2. *Estimate SPSE using existing data*
   When using existing data, the estimate for the SPSE is given by the squared error and an additional term depending on the estimated variance and the model complexity. The estimate is thus given by

$$\widehat{\mathrm{SPSE}} = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}_M^{(i)} \right)^2 + 2|M|\hat{\sigma}^2. \tag{2.30}$$

Typically used model assessment criteria follow the basic idea of the SPSE, see [3].

Mean Squared Error

The mean squared error MSE is one of the standard metrics for regression and defined as

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}_M^{(i)} \right), \tag{2.31}$$

where $y^{(i)}$ is the given data and $\hat{y}_M^{(i)}$ is the model prediction for that point.

Corrected Coefficient of Determination $\mathrm{R}^2_{\mathrm{corr}}$

The corrected coefficient of determination $\mathrm{R}^2_{\mathrm{corr}}$ is an improvement over the coefficient of determination R^2, which is defined as

$$\mathrm{R}^2 = 1 - \frac{\sum_{i=1}^{n} \left(y^{(i)} - \hat{y}_M^{(i)} \right)^2}{\sum_{i=1}^{n} \left(y^{(i)} - \bar{y} \right)^2}, \tag{2.32}$$

where $\bar{y}$ is the mean value of $\mathbf{y}$. The major drawback of R^2 is that it will never decrease when further inputs are included in the model, e.g. the R^2 of a model using $\{x_1, x_2, x_3\}$ is always larger or equal the R^2 of a model using $\{x_1, x_2\}$, even if the variable does not enhance the prediction quality.

The corrected coefficient of determination $\mathrm{R}^2_{\mathrm{corr}}$ reduces this problem by an correction term depending on the number of parameters and is given by

$$\mathrm{R}^2_{\mathrm{corr}} = 1 - \frac{n-1}{n-p}(1 - \mathrm{R}^2). \tag{2.33}$$

The corrected coefficient of determination is a standard output parameter in many statistical programs and may be used to compare even models with different number of used variables [3].

Corrected Coefficient of Determination after McFadden R^2_{McFadden}

The corrected coefficient of determination after McFadden is defined as

$$R^2_{\text{McFadden}} = 1 - \frac{\ln(\mathcal{L}_M) - |M|}{\ln(\mathcal{L}_0)} \tag{2.34}$$

using the likelihood of the model M given by $\mathcal{L}_M$ and the likelihood of the zero model $\mathcal{L}_0$. A standard zero model is given by the mean value $\bar{y}$. Higher values of R^2_{McFadden} correspond to better fits.

Mallow's C_p

Mallow's complexity parameter is based directly on the ideas specified for the estimation of the SPSE and is given by

$$C_p = \frac{\sum_{i=1}^{n} \left(y^{(i)} - \hat{y}_M^{(i)}\right)^2}{\hat{\sigma}^2} - n + 2|M|. \tag{2.35}$$

A lower value of Mallow's C_p corresponds to a better model fit [3].

Akaike Information Criterion

The AIC is among the most used model assessment criteria and defined by

$$\text{AIC} = -2l(\hat{\boldsymbol{\beta}}_{\text{ML}}, \hat{\sigma}^2_{\text{ML}}) + 2(|M| + 1), \tag{2.36}$$

where $l(\hat{\boldsymbol{\beta}}_{\text{ML}}, \hat{\sigma}^2_{\text{ML}})$ is the value of the log-likelihood (2.17) at its maximum, i.e. at $\hat{\boldsymbol{\beta}}_{\text{ML}}$ and $\hat{\sigma}_{\text{ML}}$. It is worth noting that the total number of parameters is $|M| + 1$ because the variance is also counted as parameter. The log-likelihood for a linear model assuming Gaussian errors is given by, cf. (2.17),

$$-2l(\hat{\boldsymbol{\beta}}_{\text{ML}}, \hat{\sigma}^2_{\text{ML}}) = n \log(\hat{\sigma}^2_{\text{ML}}) + n. \tag{2.37}$$

Therefore, neglecting the constant n, the AIC evaluates to

$$\text{AIC} = n \log(\hat{\sigma}^2_{\text{ML}}) + 2(|M| + 1). \tag{2.38}$$

A lower value of the AIC corresponds a to a better model fit [3].

Bayesian Information Criteria

The BIC is similar to the AIC, but it penalizes more complex models much harder than the AIC. In its general form, it is given as

$$\text{BIC} = -2l(\hat{\boldsymbol{\beta}}_{\text{ML}}, \hat{\sigma}^2_{\text{ML}}) + \log(n)(|M| + 1). \tag{2.39}$$

Again, assuming Gaussian errors for a linear model and neglecting the constant term n, the BIC evaluates to

$$\text{BIC} = n \log(\hat{\sigma}^2_{\text{ML}}) + \log(n)(|M| + 1). \tag{2.40}$$

A lower value of the BIC correspond to a better model fit [3].

Cross-validation

The basic idea of cross-validation (CV) is to split the given data set into a training set to estimate the parameters and a validation set to assess the prediction quality. A special case of cross-validation is the "leave-one-out" cross-validation, where all but one data point are used for training and the model is then evaluated on this held-out data point. This seems to be quite expensive, since one needs to estimate one model per data point. However, it can be shown that the cross-validation score can be computed using one model trained on all data $\mathbf{y}$ and the hat matrix $\mathbf{H}_M = \mathbf{X}_M(\mathbf{X}_M^{\text{T}}\mathbf{X}_M)^{-1}\mathbf{X}_M^{\text{T}}$, see Section 2.1.3. The cross-validation score is then given by

$$\text{CV} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y^{(i)} - \hat{y}_M^{(i)}}{1 - h_{ii,M}}\right)^2, \tag{2.41}$$

where $h_{ii,M}$ denote the diagonal elements of the hat matrix $\mathbf{H}_M$ and $\hat{y}_M^{(i)}$ is defined as the prediction of the model M for the input $\{x_1^{(i)}, \ldots, x_q^{(i)}\}$. A lower cross-validation score corresponds to a better model fit [25].

An approximation to the cross-validation score is given by the so-called generalized cross-validation (GCV) score. It is mainly used in the context of non-parametric regression, when the hat matrix $\mathbf{H}_M$ is numerically expensive to compute or when regularization, see Section 2.2.3, is applied. In the GCV score, the diagonal elements of the hat matrix $h_{ii,M}$ are replaced by the mean of the trace of $\mathbf{H}_M$. The GCV score is then given by

$$\text{GCV} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y^{(i)} - \hat{y}_M^{(i)}}{1 - \text{trace}(\mathbf{H}_M)/n}\right)^2. \tag{2.42}$$

The numerical advantage comes from the fact that the trace of a product of matrices is invariant to cyclical permutations, i.e.

$$\begin{aligned} \text{trace}(\mathbf{H}_M) &= \text{trace}\left(\mathbf{X}_M(\mathbf{X}_M^{\mathrm{T}}\mathbf{X}_M)^{-1}\mathbf{X}_M^{\mathrm{T}}\right) \\ &= \text{trace}\left(\mathbf{X}_M^{\mathrm{T}}\mathbf{X}_M(\mathbf{X}_M^{\mathrm{T}}\mathbf{X}_M)^{-1}\right) = |M|. \end{aligned} \tag{2.43}$$

The trace can therefore be computed from the product of two matrices of shape $|M| \times |M|$, see [3]. Note that for a linear model as in (2.6), the trace of the hat matrix $\mathbf{H}$ is equal to the number of parameters p of the linear model. For regularized or non-parametric models, the trace of the hat matrix $\mathbf{H}$ is smaller than p, where p is the number of parameters of the regularized or non-parametric model. Therefore, the trace of the hat matrix $\mathbf{H}$ is also known as *effective degree of freedom* EDoF of the model, see Section 2.2.3.

2.2.2 Subset Selection Methods

To make use of the various model assessment criteria, some algorithmic approach to model selection needs to be given. The most commonly used approaches are forward, backward and stepwise selction [3].

In forward selection, we start with a candidate model, which includes a small number of variables. In each iteration of forward selection, an additional variable is added to the candidate model. The added variable is the one which leads to the largest reduction of a predefined model assessment criteria. The algorithm stops, if no further reduction is achieved.

In backward selection, we start with a candidate model, which includes all variables. In each iteration of backward selection, we eliminate the variable from the model which elimination provides the largest improvement of a predefined model assessment criteria. The algorithm stops, if no further improvement is possible.

In step-wise selection, forward and backward selection are combined to enable the inclusion and deletion of a variable in every operation. The algorithm stops, if no further reduction is possible.

2.2.3 Regularization

Model selection can also be achieved using regularization techniques by directly influencing the parameters $\boldsymbol{\beta}$, which need to be estimated given a data set. In general, regularization restricts the parameter space by adding some penalty term depending on the complexity of the model to the least squares objective function according to (2.10). This leads to the penalized least squares (PLS) criterion

$$\text{PLS}(\mathbf{y}, \boldsymbol{\beta}; \lambda) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \cdot \text{pen}(\boldsymbol{\beta}), \tag{2.44}$$

where λ is the so-called *smoothing parameter* and $\text{pen}(\boldsymbol{\beta})$ is the penalty term describing the regularization technique.

In Ridge regression, the penalty term in the penalized least squares criterion in (2.44) is given by the squared weighted L_2-norm of the parameter vector $\boldsymbol{\beta}$, i.e. $\text{pen}(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_{\mathbf{K}}^2 =$

$\boldsymbol{\beta}^{\mathrm{T}}\mathbf{K}\boldsymbol{\beta}$ with a positive definite penalty matrix $\mathbf{K} \in \mathbb{R}^{p\times p}$. The closed-form solution reads as

$$\hat{\boldsymbol{\beta}}_{\mathrm{PLS}} = \arg\min_{\boldsymbol{\beta}} \left(\mathrm{PLS}(\mathbf{y}, \boldsymbol{\beta}; \lambda)\right) = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{K})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}. \tag{2.45}$$

The additional penalty term in Ridge regression leads to smaller parameter estimates $\hat{\boldsymbol{\beta}}_{\mathrm{PLS}}$ compared to the unpenalized estimate $\hat{\boldsymbol{\beta}}_{\mathrm{LS}}$. For large values of the smoothing parameter λ, the parameter estimates will converge towards, but never reach, zero.

Ridge regression is commonly used when the input dimension q is high, i.e. the number of parameters $\boldsymbol{\beta}$ is large, and also known as Tikhonov regularization [26]. Note that it is also possible to use a penalty matrix $\mathbf{K}(\boldsymbol{\beta})$ resulting in

$$\mathrm{pen}(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|^2_{\mathbf{K}(\boldsymbol{\beta})} = \boldsymbol{\beta}^{\mathrm{T}}\mathbf{K}(\boldsymbol{\beta})\boldsymbol{\beta}. \tag{2.46}$$

However, the resulting penalized least squares problem has no closed-form solution and must be solved by an iterative approach. We start with an initial guess $\boldsymbol{\beta}^{[0]}$ and compute for $k = 0, 1, 2, \ldots$ the iteration

$$\boldsymbol{\beta}^{[k+1]} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{K}(\boldsymbol{\beta}^{[k]})\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}, \tag{2.47}$$

until $\left\|\boldsymbol{\beta}^{[k+1]} - \boldsymbol{\beta}^{[k]}\right\| \leq \mathrm{Tol}$ with Tol being a given tolerance. The derivation of the iteration (2.47) is presented in Appendix A.3.

Note that by the introduction of the penalty matrix $\mathbf{K} \neq \mathbf{0}$, we reduce the degrees of freedom of the model. This can be seen by comparing the trace of the hat matrix $\mathbf{H}$ of the regularized model, i.e.

$$\mathbf{H}_{\mathrm{pen}} = \mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{K})^{-1}\mathbf{X}^{\mathrm{T}}, \tag{2.48}$$

with the trace of the hat matrix of the unpenalized model, see (2.22). The trace of the hat matrix $\mathbf{H}$ is also called *effective degree of freedom*, i.e.

$$\mathrm{EDoF} = \mathrm{trace}(\mathbf{H}_{\mathrm{pen}}). \tag{2.49}$$

When we use regularization to reduce the degree of freedom of a model, we need to make use of the effective degree of freedom EDoF instead of the number of parameters p in model assessment criteria, see Section 2.2.1.

L-curve method

Another method for choosing the regularization parameter, additional to the model assessment criteria in Section 2.2.1, is the so-called *L-curve method*. The L-curve is

given by the parametric curve $(\rho(\lambda), \eta(\lambda))$, where $\rho(\lambda)$ and $\eta(\lambda)$ measure the size of the regularization $\boldsymbol{\beta}^{\mathrm{T}}\mathbf{K}\boldsymbol{\beta}$ and the residual $\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2$, see [27]. In the setting of Ridge regression, this curve then has a distinct L-shaped corner exactly where the solution $\boldsymbol{\beta}$ changes from being dominated by the regularization errors $\lambda \cdot \mathrm{pen}(\boldsymbol{\beta})$ to being dominated by the mean squared data errors $\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|$. This corner determines the smoothing parameter λ as optimal trade-off between data fidelity and smoothness. The L-curve method can also be used when the data errors are correlated, a situation in which generalized cross-validation, see 2.2.1, fails to produce the optimal regularization parameter. Further information can be found in [27] and [28].

2.3 Splines

A spline is a piecewise polynomial defined on a sequence of knots. This definition is quite general. Therefore, a large variety of splines exists, ranging from regression splines [29], over B-splines [9] to natural cubic splines and many more. We will focus on the definition of B-splines in Section 2.3.1, tensor-product B-splines as the multi-dimensional expansion of B-splines in Section 2.3.1, and P-splines in Section 2.3.2, see for more information [3], [9] and [30].

2.3.1 B-Splines

We put the focus on the definition and use of B-splines $s(x)$, which are constructed using the d B-spline basis functions $B_j^l(x)$ of order l as

$$s(x) = \sum_{j=1}^{d} B_j^l(x)\beta_j \tag{2.50}$$

given the knot sequence

$$K = \{\kappa_{1-l}, \kappa_{1-l+1}, \ldots, \kappa_{d+1}\}. \tag{2.51}$$

The B-spline basis function $B_j^l(x)$ of order l is defined by means of the Cox-de Boor recursion formula as

$$B_j^0(x) = \begin{cases} 1 & \text{for} \quad \kappa_j \leq x < \kappa_{j+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.52}$$

$$B_j^l(x) = \frac{x - \kappa_{j-l}}{\kappa_j - \kappa_{j-l}} B_{j-1}^{l-1}(x) + \frac{\kappa_{j+1} - x}{\kappa_{j+1} - \kappa_{j+1-l}} B_j^{l-1}(x) \tag{2.53}$$

using the knot sequence (2.51). Hence it is composed of $(l+1)$-polynomial pieces of degree l, see [3]. An example of a B-spline basis function of order $l = 0, 1, 2, 3$ is given in Figure 2.2.

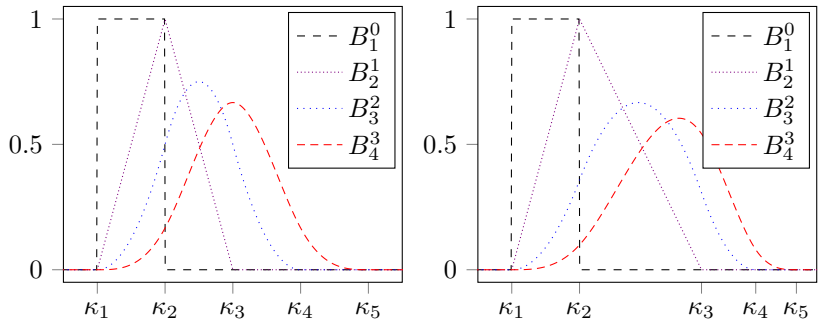

Figure 2.2: B-spline basis function of order $l = 0, 1, 2, 3$ given by the superscript l in B_j^l for equidistant (left) and non-equidistant (right) knots.

The left plot shows the B-spline basis functions based on an equidistant sequence of knots. The B-spline basis function B_1^0 is the zero function, except for $x \in [\kappa_1, \kappa_2]$ where it is equal to 1, see (2.52). The B-spline basis function B_2^1 is the well known *hat function*, being zero except for $x \in [\kappa_1, \kappa_3]$. It consists of two linear pieces, one defined from κ_1 to κ_2, the other from κ_2 to κ_3. This can be seen by expanding the recursive definition (2.53) to

$$B_2^1(x) = a_1(x)B_1^0(x) + a_2(x)B_2^0(x) \tag{2.54a}$$

with

$$a_1(x) = \frac{x - \kappa_1}{\kappa_2 - \kappa_1} \tag{2.54b}$$

$$a_2(x) = \frac{\kappa_3 - x}{\kappa_3 - \kappa_2}. \tag{2.54c}$$

Everywhere else, B_2^1 is equal to zero. At the joining points, the values of the linear pieces are equal. The B-spline basis function B_3^2 consists of three quadratic pieces, joining at the knots κ_2 and κ_3. Expanding the recursive definition shows this as

$$B_3^2(x) = a_1(x)B_1^0(x) + a_2(x)B_2^0(x) + a_3(x)B_3^0(x) \tag{2.55a}$$

with

$$a_1(x) = \frac{x - \kappa_1}{\kappa_3 - \kappa_1}\frac{x - \kappa_1}{\kappa_2 - \kappa_1} \tag{2.55b}$$

$$a_2(x) = \frac{x - \kappa_1}{\kappa_3 - \kappa_1}\frac{\kappa_3 - x}{\kappa_3 - \kappa_2} + \frac{\kappa_4 - x}{\kappa_4 - \kappa_2}\frac{x - \kappa_2}{\kappa_3 - \kappa_2} \tag{2.55c}$$

$$a_3(x) = \frac{\kappa_4 - x}{\kappa_4 - \kappa_2}\frac{\kappa_4 - x}{\kappa_4 - \kappa_3}. \tag{2.55d}$$

At κ_2 and κ_3, the values of the quadratic pieces, as well as their first-order derivatives are equal. Finally, the B-spline basis function B_4^3 consists of 4 cubic pieces with the joining points at κ_2, κ_3 and κ_4 at which respective cubic polynomials possess equal values as

well as equal first-order and second-order derivatives. Expanding the recursive definition shows this as

$$B_4^3(x) = a_1(x)B_1^0(x) + a_2(x)B_2^0(x) + a_3(x)B_3^0(x) + a_4(x)B_4^0(x) \tag{2.56a}$$

with

$$a_1(x) = \frac{x-\kappa_1}{\kappa_4-\kappa_1}\frac{x-\kappa_1}{\kappa_3-\kappa_1}\frac{x-\kappa_1}{\kappa_2-\kappa_1} \tag{2.56b}$$

$$a_2(x) = \frac{x-\kappa_1}{\kappa_4-\kappa_1}\frac{x-\kappa_1}{\kappa_3-\kappa_1}\frac{\kappa_3-x}{\kappa_3-\kappa_2} + \frac{x-\kappa_1}{\kappa_4-\kappa_1}\frac{\kappa_4-x}{\kappa_4-\kappa_2}\frac{x-\kappa_2}{\kappa_3-\kappa_2} + \frac{\kappa_5-x}{\kappa_5-\kappa_2}\frac{x-\kappa_2}{\kappa_4-\kappa_2}\frac{x-\kappa_2}{\kappa_4-\kappa_2} \tag{2.56c}$$

$$a_3(x) = \frac{x-\kappa_1}{\kappa_4-\kappa_1}\frac{\kappa_4-x}{\kappa_4-\kappa_2}\frac{\kappa_4-x}{\kappa_4-\kappa_3} + \frac{\kappa_5-x}{\kappa_5-\kappa_2}\frac{x-\kappa_2}{\kappa_4-\kappa_2}\frac{\kappa_4-x}{\kappa_4-\kappa_3} + \frac{\kappa_5-x}{\kappa_5-\kappa_2}\frac{\kappa_5-x}{\kappa_5-\kappa_3}\frac{x-\kappa_3}{\kappa_4-\kappa_3} \tag{2.56d}$$

$$a_4(x) = \frac{\kappa_5-x}{\kappa_5-\kappa_2}\frac{\kappa_5-x}{\kappa_5-\kappa_3}\frac{\kappa_5-x}{\kappa_5-\kappa_4}. \tag{2.56e}$$

The right plot in Figure 2.2 shows the B-spline basis functions of the same order $l = 0, 1, 2, 3$ defined on a non-equidistant knot sequence. The shown locality, i.e. being nonzero only over a sequence of $l + 2$ knots, is a very attractive feature leading to enhanced numerical properties compared to other types of splines. Some general properties of a B-spline basis function of order l are summarized in the following list. Note that these properties are valid independent of the type of the knot placement.

(i) A B-spline basis function consists of $l + 1$ polynomial pieces of degree l, e.g. a cubic B-spline basis function ($l = 3$) consists of 4 cubic pieces.

(ii) The pieces join at l inner knots.

(iii) At these knots, the derivatives up to order $l - 1$ are continuous.

(iv) The B-spline basis function is positive on the domain spanned by $l + 2$ knots, everywhere else it is zero, e.g. for $l = 2$, a sequence of 4 knots is necessary.

(v) At every given x, only $l + 1$ B-spline basis functions are nonzero.

Using the definition of B-spline basis functions, see (2.52) and (2.53), the first-order derivative of a B-spline basis function of order l is given by

$$\frac{\partial}{\partial x}B_j^l(x) = l\left[\frac{1}{\kappa_j-\kappa_{j-l}}B_{j-1}^{l-1}(x) - \frac{1}{\kappa_{j+1}-\kappa_{j+1-l}}B_j^{l-1}(x)\right] \tag{2.57}$$

using B-spline basis functions of order $l - 1$, see [3]. Higher-order derivatives are obtained by using lower order B-spline basis functions, see [9].

As shown in Figure 2.2, the knots can either be an equidistant sequence, which facilitates the construction and estimation of the coefficients, or a non-equidistant sequence. For equidistant knots, we split the domain $[a, b]$ into $m - 1$ intervals, i.e.

$$h = \frac{b-a}{m-1}, \tag{2.58}$$

where m is given by $m = d - l + 1$ and obtain the sequence

$$\kappa_j = a + h(j-1), \quad j = 1, \ldots, m. \tag{2.59}$$

Non-equidistant knot placement can be obtained using e.g. quantile-based knots, i.e. by using the $(j-1)/(m-1)$-quantiles for $j = 1, \ldots, m$ of the observed inputs $x^{(1)}, \ldots, x^{(n)}$ as knots. Using this approach, more knots are placed in the areas where lots of data are present. The boundary knots, i.e. $\{\kappa_{1-l}, \ldots, \kappa_0\}$ on the left side and $\{\kappa_{d-l+2}, \ldots, \kappa_{d+1}\}$ on the right side, are usually set to be apart from each other by at least the minimal knot distance [3].

The collection of d B-spline basis functions of order l over a sequence of knots $K = \{\kappa_{1-l}, \kappa_{1-l+1},$
$\ldots, \kappa_{d+1}\}$ is called B-spline basis. The basis is created such that it covers the domain $[a, b]$, i.e.

$$\sum_{j=1}^{d} B_j^l(x) = 1 \text{ for } x \in [a, b]. \tag{2.60}$$

A function $f(x)$ can then be represented with a B-spline basis by

$$f(x) = \sum_{j=1}^{d} B_j^l(x)\beta_j = \mathbf{b}^{\mathrm{T}}\boldsymbol{\beta}_b, \tag{2.61}$$

using the B-spline basis functions $B_j^l(x)$ of appropriate order l and the parameter vector $\boldsymbol{\beta}_b^{\mathrm{T}} = [\beta_1, \ldots, \beta_d] \in \mathbb{R}^{1\times d}$. The basis functions can be given in vector notation as $\mathbf{b}^{\mathrm{T}} = [B_1^l(x), \ldots, B_d^l(x)] \in \mathbb{R}^{1\times d}$. Using the data set $\mathcal{D} = \{(x^{(i)}, y^{(i)}),\ i = 1, 2, \ldots, n\}$, the B-spline basis matrix for d basis functions of order l is given by the matrix $\mathbf{B}$ as

$$\mathbf{B} = \begin{bmatrix} B_1^l(x^{(1)}) & \ldots & B_d^l(x^{(1)}) \\ \vdots & & \vdots \\ B_1^l(x^{(n)}) & \ldots & B_d^l(x^{(n)}) \end{bmatrix} \in \mathbb{R}^{n\times d}. \tag{2.62}$$

The n equations (2.61) can then be arranged as a linear model in the form, cf. (2.6),

$$\mathbf{y} = \mathbf{B}\boldsymbol{\beta}_b + \boldsymbol{\epsilon}. \tag{2.63}$$

Once the basis matrix in (2.62) is given, the parameters $\boldsymbol{\beta}_b$ can be estimated using the Least Squares algorithm given in Section 2.1.1 by minimizing the objective function

$$\mathrm{LS}(\mathbf{y}, \boldsymbol{\beta}_b) = \|\mathbf{y} - \mathbf{B}\boldsymbol{\beta}_b\|_2^2. \tag{2.64}$$

Therefore, the estimation is computationally efficient and easy to implement since closed-form solutions exists. Further, the advanced theoretical framework of linear models can be applied to use model selection and regularization approaches as well as to calculate e.g. confidence intervals for the parameters and the prediction.

The derivative of the function $f(x)$ in (2.61) can be calculated by summing over all d B-spline basis functions and including the estimated parameters $\boldsymbol{\beta}_b$ into the B-spline basis function derivative (2.57) as

$$\frac{\partial f(x)}{\partial x} = \frac{\partial}{\partial x} \sum_j^d B_j^l(x)\beta_j = l \sum_j^d \frac{\beta_j - \beta_{j-1}}{\kappa_j - \kappa_{j-l}} B_{j-1}^{l-1}(x). \tag{2.65}$$

Therefore, by estimating the B-spline parameters $\boldsymbol{\beta}_b$, we also generate an estimate for the derivative of the function $f(x)$, see [3].

B-splines of appropriate order $l > 2$ produce smooths curves, i.e. first- and second-order derivatives are continuous, where the smoothness is mostly determined by the number of basis functions d used. By using a low number d, the curve will be quite smooth, but possess a large data error. When using a high number of basis functions d, the data error will be small but the variance of the curve will be large. This is an example of the bias-variance trade-off, a classical problem of regression and supervised learning, see Section 2.2.1 and [9].

Tensor-Product B-Splines

Tensor-product B-splines can be regarded as the multi-dimensional extension of B-splines. We examine an example for two input dimensions x_1 and x_2. Note that tensor-product B-splines can be constructed for arbitrary dimensions using the approach given below. The tensor-product B-spline basis function is constructed by considering the product of two B-spline basis functions of orders l_1 and l_2 of respective dimension, i.e.

$$T_{j,r}(x_1, x_2) = B_j^{l_1}(x_1) B_r^{l_2}(x_2) \tag{2.66}$$

for $j \in \{1, \ldots, d_1\}$ and $r \in \{1, \ldots, d_2\}$. For readability, we omit the order of the tensor-product B-spline basis function since, in principle, B-spline basis functions of arbitrary, even different orders $l_1 \neq l_2$ can be combined. We then obtain the basis function representation of the tensor-product B-spline $t(x_1, x_2)$ by summing all tensor-product B-spline basis functions as

$$t(x_1, x_2) = \sum_{j=1}^{d_1} \sum_{r=1}^{d_2} T_{j,r}(x_1, x_2)\beta_{j,r} \tag{2.67}$$

and in vector notation we obtain

$$t(x_1, x_2) = \mathbf{t}^{\mathrm{T}} \boldsymbol{\beta}_t, \tag{2.68}$$

with $\mathbf{t}^{\mathrm{T}} = [T_{1,1}(x_1, x_2), \ldots, T_{d_1,1}(x_1, x_2), \ldots, T_{1,d_2}(x_1, x_2), \ldots, T_{d_1,d_2}(x_1, x_2)] \in \mathbb{R}^{1 \times d_1 d_2}$ and the corresponding vector of parameters $\boldsymbol{\beta}_t^{\mathrm{T}} = [\beta_{1,1}, \ldots, \beta_{d_1,1}, \ldots, \beta_{1,d_2}, \ldots, \beta_{d_1,d_2}] \in \mathbb{R}^{1 \times d_1 d_2}$. For any set of data

$$\mathcal{D} = \{(x_1^{(i)}, x_2^{(i)}, y^{(i)}),\ i = 1, 2, \ldots, n\}, \tag{2.69}$$

the tensor-product B-spline basis matrix for d_1 and d_2 basis functions in the respective dimensions is given by the matrix $\mathbf{T}$ as

$$\mathbf{T} = \begin{bmatrix} T_{1,1}(x_1^{(1)}, x_2^{(1)}) & \ldots & T_{d_1,d_2}(x_1^{(1)}, x_2^{(1)}) \\ \vdots & & \vdots \\ T_{1,1}(x_1^{(n)}, x_2^{(n)}) & \ldots & T_{d_1,d_2}(x_1^{(n)}, x_2^{(n)}) \end{bmatrix} \in \mathbb{R}^{n \times d_1 d_2}. \tag{2.70}$$

The relationship between the tensor-product B-spline basis matrix $\mathbf{T}$ and the B-spline basis matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ is then given as

$$\mathbf{T} = \mathbf{B}_2 \odot \mathbf{B}_1, \tag{2.71}$$

where $\odot$ indicates the use of the row-wise Kronecker product, see Appendix A.2, $\mathbf{T} \in \mathbb{R}^{n \times d_1 d_2}$ denotes the tensor-product B-spline basis matrix, $\mathbf{B}_1 \in \mathbb{R}^{n \times d_1}$ is the B-spline basis matrix for dimension x_1 and $\mathbf{B}_2 \in \mathbb{R}^{n \times d_2}$ denotes the B-spline basis matrix for dimension x_2 [3].

We can now model a two dimensional function using the data set $\mathcal{D}$ in (2.69) similar to (2.63) as linear model of the form

$$\mathbf{y} = \mathbf{T}\boldsymbol{\beta}_t + \boldsymbol{\epsilon}, \tag{2.72}$$

with the tensor-product B-spline basis matrix $\mathbf{T} \in \mathbb{R}^{n \times d_1 d_2}$ according to (2.70) and the parameter vector $\boldsymbol{\beta}_t^{\mathrm{T}} \in \mathbb{R}^{1 \times d_1 d_2}$. Once the basis matrix in (2.70) is given, the parameters $\boldsymbol{\beta}_t$ can be estimated using the Least Squares algorithm given in Section 2.1.1 by minimizing the objective function

$$\mathrm{LS}(\mathbf{y}, \boldsymbol{\beta}_t) = \|\mathbf{y} - \mathbf{T}\boldsymbol{\beta}_t\|_2^2. \tag{2.73}$$

This approach can in theory be repeated for as many input dimensions as required. In practice, modeling more than two input dimensions using tensor-product B-splines becomes infeasible because of the exponential increase of basis functions and therefore parameters to estimate.

Additive Regression

To circumvent the latter problem, we now assume the restrictive structure of additive models, see [3], given by

$$f(x_1, \ldots, x_q) = f_1(x_1) + \cdots + f_q(x_q). \tag{2.74}$$

Hence, we use one function $f_i(x_i)$ per input dimension x_i. For some given data set $\mathcal{D} = \{(x_1^{(ii)}, \ldots, x_q^{(ii)}, y^{(ii)}),\ ii = 1, 2, \ldots, n\}$, by using a B-spline for each function $f_i(x_i)$ we obtain a linear model

$$f_i(\mathbf{x}_i) = \mathbf{B}_i \boldsymbol{\beta}_{b_i}, \tag{2.75}$$

where $\mathbf{B}_i \in \mathbb{R}^{n \times d_i}$ is the B-spline basis matrix using d_i B-spline basis functions for $i = 1, 2, \ldots, q$, $\mathbf{x}_i^{\mathrm{T}} = [x_i^{(1)}, \ldots, x_i^{(n)}] \in \mathbb{R}^{1 \times n}$ is the data vector of input dimension i and $\boldsymbol{\beta}_{b_i} \in \mathbb{R}^{d_i}$ are the parameters to estimate. This leads to the model structure

$$\mathbf{y} = \mathbf{B}_1 \boldsymbol{\beta}_{b_1} + \cdots + \mathbf{B}_q \boldsymbol{\beta}_{b_q} + \boldsymbol{\epsilon}, \tag{2.76}$$

which can be written as linear model by concatenation of the B-spline basis matrices and parameter vectors as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} = \Big[\mathbf{B}_1, \ldots, \mathbf{B}_q\Big] \begin{bmatrix} \boldsymbol{\beta}_{b_1} \\ \vdots \\ \boldsymbol{\beta}_{b_q} \end{bmatrix} + \boldsymbol{\epsilon}, \tag{2.77}$$

with the matrix $\mathbf{X} \in \mathbb{R}^{n \times d_{total}}$, the parameter vector $\boldsymbol{\beta} \in \mathbb{R}^{d_{total}}$ and $d_{total} = \sum_{i=1}^{q} d_i$ as the total number of parameters. The model (2.77) does not contain interaction terms between inputs. Nevertheless, these can be easily introduced for two dimensions using tensor-product B-splines without an overflowing increase in the number of coefficients. We can then write the additive model with interaction terms as

$$f(x_1, \ldots, x_q) = f_1(x_1) + \cdots + f_q(x_q) + f_{1,2}(x_1, x_2) + \cdots + f_{q-1,q}(x_{q-1}, x_q). \tag{2.78}$$

Hence, we use one function $f_i(x_i)$ per input dimension and per interaction term. Using a tensor-product B-spline $t_{j,r}(x_j, x_r)$ for each interaction term, we obtain the model

$$\mathbf{y} = \mathbf{B}_1 \boldsymbol{\beta}_{b_1} + \cdots + \mathbf{B}_q \boldsymbol{\beta}_{b_q} + \sum_{j=1}^{q-1} \sum_{r>j}^{q} \mathbf{T}_{j,r} \boldsymbol{\beta}_{t_{j,r}} + \boldsymbol{\epsilon}, \tag{2.79}$$

using the tensor-product B-spline basis matrices $\mathbf{T}_{j,r} \in \mathbb{R}^{n \times d_j d_r}$ and the parameter $\boldsymbol{\beta}_{t_{j,r}} \in \mathbb{R}^{d_j d_r}$. Using the notation in (2.79), the theoretical framework of linear models

can be applied to the additive regression model, since (2.79) can be formulated as linear model yielding

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} = \left[\mathbf{B}_1, \ldots, \mathbf{B}_q, \mathbf{T}_{1,2}, \ldots, \mathbf{T}_{q-1,q}\right] \begin{bmatrix} \boldsymbol{\beta}_{b_1} \\ \vdots \\ \boldsymbol{\beta}_{b_q} \\ \boldsymbol{\beta}_{t_{1,2}} \\ \vdots \\ \boldsymbol{\beta}_{t_{q-1,q}} \end{bmatrix} + \boldsymbol{\epsilon}, \tag{2.80}$$

with $\mathbf{X} \in \mathbb{R}^{n \times d_{total}}$ as design matrix, $\boldsymbol{\beta} \in \mathbb{R}^{d_{total}}$ as parameter vector and $d_{total} = \sum_{i=1}^{q} d_i + \sum_{j=1}^{q-1} \sum_{r>j}^{q} d_j d_r$ as the total number of parameters in the model. Hence, the parameters can be calculated efficiently using the Least Squares (LS) algorithm, see Section 2.1.1. Further, the assumptions given in Section 2.1 on the error term, as well as on the model function are used.

Note that additive models are not limited to be used with B-splines. We can also include the linear model given in Section 2.1. For 2 input dimensions x_1 and x_2, we then obtain a model of the form

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{B}_1\boldsymbol{\beta}_{b_1} + \mathbf{B}_2\boldsymbol{\beta}_{b_2} + \mathbf{T}_{1,2}\boldsymbol{\beta}_{t_{1,2}}, \tag{2.81}$$

where $\mathbf{X} \in \mathbb{R}^{n \times 2}$ is the design matrix of the linear model, $\mathbf{B}_1$ and $\mathbf{B}_2$ are the respective B-spline basis matrices and $\mathbf{T}_{1,2}$ is the tensor-product B-spline matrix.

2.3.2 P-Splines

P-splines combine the concepts of B-spline basis functions and regularization to produce sufficiently smooth function estimations. A function is said to be smooth if its second-order derivative is continuous. Therefore, a penalty of the form

$$\lambda \cdot \int \left(f''(x)\right)^2 \, \mathrm{d}x \tag{2.82}$$

is typically introduced to penalize the curvature of a function which is measured by its second-order derivative [31]. The penalty is weighted by the so-called *smoothing parameter* λ. This yields the penalized least squares objective function, cf. Section 2.2.3, as

$$\mathrm{PLS}(\mathbf{y}, \boldsymbol{\beta}; \lambda) = \|\mathbf{y} - \mathbf{B}\boldsymbol{\beta}_b\|_2^2 + \lambda \int \left(f''(x)\right)^2 \, \mathrm{d}x \tag{2.83}$$

using the B-spline basis matrix $\mathbf{B} \in \mathbb{R}^{n \times d}$ for some data $\mathcal{D} = \{(x^{(i)}, y^{(i)}),\ i = 1, 2, \ldots, n\}$. Inserting the B-spline basis function formulation (2.61), using order l and d basis functions, into (2.82) results in

$$\begin{aligned}\int \left(f''(x)\right)^2 \,\mathrm{d}x &= \int \left(\sum_{j=1}^{d} B_j''(x)\beta_j\right)^2 \,\mathrm{d}x \\ &= \int \sum_{j=1}^{d}\sum_{r=1}^{d} \beta_j\beta_r B_j''(x)B_r''(x)\,\mathrm{d}x \\ &= \boldsymbol{\beta}^{\mathrm{T}}\mathbf{K}\boldsymbol{\beta},\end{aligned} \tag{2.84}$$

with the penalty matrix $\mathbf{K}[j,r] = \int B_j''(x)B_r''(x)\,\mathrm{d}x$ as a matrix of dimension $\mathbb{R}^{d\times d}$. The entries of $\mathbf{K}$ are given by the integrated products of the second-order derivatives of the B-spline basis functions $B_j(x)$ and $B_r(x)$. For readability, the order l is omitted. These second-order derivatives can be obtained by using the derivative properties of B-spline basis functions, see [3].

Eilers and Marx proposed in [30] to base the penalty on finite differences of higher order of the parameters of adjacent B-spline basis functions which circumvents the direct calculation of the second-order derivative and the integral. Hence, the complexity is reduced from n, the number of data points to evaluate the integral on, to d, the number of parameters [30]. The squared second-order finite difference gives a good discrete approximation of the integral of the squared second-order derivative in (2.82), i.e.

$$\sum_{j=3}^{d}\left(\Delta^2\beta_j\right)^2 \propto \int \left(f''(x)\right)^2 \,\mathrm{d}x \tag{2.85}$$

where $\Delta^2\beta_j$ is defined as

$$\begin{aligned}\Delta^2\beta_j &= \Delta(\Delta\beta_j) \\ &= \Delta(\beta_j - \beta_{j-1}) \\ &= \beta_j - 2\beta_{j-1} + \beta_{j-2}.\end{aligned} \tag{2.86}$$

In matrix form, (2.86) may be given as

$$\mathbf{D}_2\boldsymbol{\beta}_b = \begin{bmatrix} 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_d \end{bmatrix}, \tag{2.87}$$

with $\mathbf{D}_2 \in \mathbb{R}^{(d-2)\times d}$. Applying the matrix form of the second-order finite difference operator (2.87) into (2.85) yields

$$\lambda\sum_{j=3}^{d}\left(\Delta^2\beta_j\right)^2 = \lambda\boldsymbol{\beta}_b^{\mathrm{T}}\mathbf{D}_2^{\mathrm{T}}\mathbf{D}_2\boldsymbol{\beta}_b = \lambda\boldsymbol{\beta}_b^{\mathrm{T}}\mathbf{K}\boldsymbol{\beta}_b. \tag{2.88}$$

We then obtain the objective function to minimize as

$$\mathrm{PLS}(\mathbf{y}, \boldsymbol{\beta}; \lambda) = \|\mathbf{y} - \mathbf{B}\boldsymbol{\beta}_b\|_2^2 + \lambda \boldsymbol{\beta}_b^{\mathrm{T}} \mathbf{K} \boldsymbol{\beta}_b, \tag{2.89}$$

which is equivalent to the objective function of Ridge regression, cf. Section 2.2.3, for the special choice of $\mathbf{K}$ given by $\mathbf{K} = \mathbf{D}_2^{\mathrm{T}} \mathbf{D}_2 \in \mathbb{R}^{d \times d}$. As in Ridge regression, the smoothness parameter λ plays a critical role. For $\lambda \to 0$, the P-spline approaches the underlying B-spline since the penalty term in (2.89) goes to 0. For $\lambda \to \infty$, the P-spline approaches a polynomial model. The order of the polynomial is given by the order of the finite difference penalty, e.g. for second-order finite difference penalty, we penalize the discrete approximation of the second-order derivative leading to a linear function, because for these, the second-order derivative is equal to zero. Note that higher-order difference penalties are also possible.

The main advantage of P-splines is their easy set up by replacing the integral of the squared second-order derivative of the B-spline basis functions with the squared second-order finite differences of parameters of adjacent B-spline basis functions. This reduces the computational complexity and allows faster training and evaluation. Hence, P-splines are widely used in practice [30].

A similar penalty term for tensor-product B-splines can be constructed using the Kronecker product. Recall the definition of a tensor-product B-spline given in (2.67) and (2.72).

The spatial alignment of the B-spline basis functions and the corresponding parameters of the two-dimensional tensor-product B-spline needs to be incorporated by the definition of the term *adjacent parameters*. An example for these adjacent parameters, also called spatial neighborhood, is taken from [3] and given in Figure 2.3. Here, we choose the parameters left and right, i.e. $\beta_{j,r-1}$ and $\beta_{j,r+1}$, as well as above and below, i.e. $\beta_{j-1,r}$ and $\beta_{j+1,r}$, as spatial neighborhood for $\beta_{j,r}$.

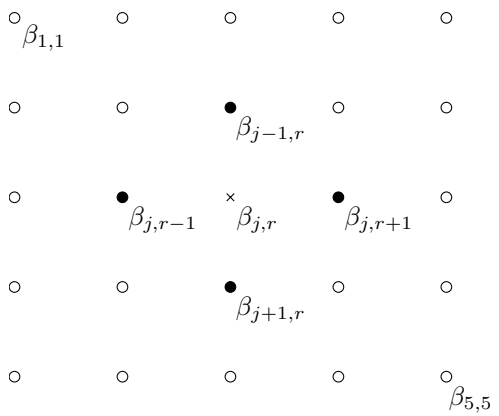

Figure 2.3: Spatial neighborhood or adjacent parameters for a tensor-product B-spline.

We now use the concepts of P-splines in both dimensions and penalize the integral of the squared Hessian of the tensor-product B-spline, see (2.82), by a higher-order finite

difference approximation in both dimensions. Using second-order finite differences as in (2.85) leads to the following definition of the penalty term

$$\left[\sum_{j=3}^{d_1}\sum_{r=1}^{d_2}(\Delta_1^2\beta_{j,r})^2+\sum_{j=1}^{d_1}\sum_{r=3}^{d_2}(\Delta_2^2\beta_{j,r})^2\right]\propto\iint\left(f''(x_1,x_2)\right)^2\,\mathrm{d}x_1\,\mathrm{d}x_2 \tag{2.90}$$

as discrete approximation of the integral of the squared Hessian of the tensor-product B-spline. The first term on the left side calculates the "row-wise" squared second-order differences, i.e. in direction x_1, using

$$\Delta_1^2\beta_{j,r}=\beta_{j,r}-2\beta_{j-1,r}+\beta_{j-2,r} \tag{2.91}$$

and the second term on the left side calculates the "column-wise" squared second-order differences, i.e. in direction x_2, using

$$\Delta_2^2\beta_{j,r}=\beta_{j,r}-2\beta_{j,r-1}+\beta_{j,r-2}. \tag{2.92}$$

The subscript for Δ in (2.91) and (2.92) indicates the direction of the finite differences. Using the matrix form of the second-order finite difference operator and the Kronecker product, see Appendix A.1, as well as the parameter vector $\boldsymbol{\beta}_t\in\mathbb{R}^{d_1d_2\times 1}$, we can then write the "row-wise" penalty as

$$\boldsymbol{\beta}_t^{\mathrm{T}}(\mathbf{I}_{d_2}\otimes\mathbf{D}_{1,2})^{\mathrm{T}}(\mathbf{I}_{d_2}\otimes\mathbf{D}_{1,2})\boldsymbol{\beta}_t=\sum_{j=3}^{d_1}\sum_{r=1}^{d_2}\left(\Delta_1^2\beta_{j,r}\right)^2 \tag{2.93}$$

and the "column-wise" penalty as

$$\boldsymbol{\beta}_t^{\mathrm{T}}(\mathbf{D}_{2,2}\otimes\mathbf{I}_{d_1})^{\mathrm{T}}(\mathbf{D}_{2,2}\otimes\mathbf{I}_{d_1})\boldsymbol{\beta}_t=\sum_{j=1}^{d_1}\sum_{r=3}^{d_2}\left(\Delta_2^2\beta_{j,r}\right)^2 \tag{2.94}$$

using the identity matrices $\mathbf{I}_{d_1}\in\mathbb{R}^{d_1\times d_1}$ and $\mathbf{I}_{d_2}\in\mathbb{R}^{d_2\times d_2}$ and the second-order difference matrices $\mathbf{D}_{1,2}\in\mathbb{R}^{(d_1-2)\times d_1}$ and $\mathbf{D}_{2,2}\in\mathbb{R}^{(d_2-2)\times d_2}$, cf. (2.87). The first subscript of $\mathbf{D}$ indicates the direction of the finite differences and the second subscript indicates the use of the second-order finite differences. Summing up both penalties leads to a formulation similar to (2.89) given by

$$\mathrm{PLS}(\mathbf{y},\boldsymbol{\beta}_t;\lambda)=\|\mathbf{y}-\mathbf{T}\boldsymbol{\beta}_t\|_2^2+\lambda\boldsymbol{\beta}_t^{\mathrm{T}}\mathbf{K}\boldsymbol{\beta}_t, \tag{2.95}$$

with the tensor-product B-spline basis matrix $\mathbf{T}\in\mathbb{R}^{n\times d_1d_2}$, the smoothing parameter λ and the penalty matrix $\mathbf{K}$ given by

$$\mathbf{K}=\left[(\mathbf{I}_{d_2}\otimes\mathbf{D}_{1,2})^{\mathrm{T}}(\mathbf{I}_{d_2}\otimes\mathbf{D}_{1,2})+(\mathbf{D}_{2,2}\otimes\mathbf{I}_{d_1})^{\mathrm{T}}(\mathbf{D}_{2,2}\otimes\mathbf{I}_{d_1})\right]\in\mathbb{R}^{d_1d_2\times d_1d_2}. \tag{2.96}$$

3 Solution Approach

The main goal of this thesis is the development and implementation using Matlab and Python of an algorithm to include a priori domain knowledge like monotonicity, curvature, unimodality, etc. into the data fitting process. Using this additional information should improve the generalization capabilities of the model in situations of sparse, noisy or even partly wrong data. In this chapter, we use the theory discussed in Chapter 2, i.e. B-splines in Section 2.3.1 and P-splines in Section 2.3.2, and extend it by adding a shape-constraint penalty term of the form

$$\lambda_c \cdot \mathrm{con}(\boldsymbol{\beta}) \tag{3.1}$$

depending on the user-defined a priori domain knowledge leading to the so-called *shape-constraint P-splines* (SCP-splines). The various types of a priori domain knowledge that can be included are listed in Table 3.1.

Constraint	**Description**		**Section**
Jamming		$f(x^{(p)}) \approx y^{(p)}$	3.1.7
Boundedness	lower	$f(x) \geq B$	3.1.8
	upper	$f(x) \leq B$	3.1.8
Monotonicity	increasing	$f'(x) \geq 0$	3.1.1
	decreasing	$f'(x) \leq 0$	3.1.2
Curvature	convex	$f''(x) \geq 0$	3.1.3
	concave	$f''(x) \leq 0$	3.1.4
Unimodality	peak	$m = \arg\max_x f(x)$ $f'(x) \geq 0 \quad \text{if } x < m$ $f'(x) \leq 0 \quad \text{if } x > m$	3.1.5
	valley	$m = \arg\min_x f(x)$ $f'(x) \leq 0 \quad \text{if } x < m$ $f'(x) \geq 0 \quad \text{if } x > m$	3.1.6

Table 3.1: Overview of the considered constraints.

The focus of this chapter is the definition of shape-constraint P-splines, see Section 3.1, which are characterized by their parameters $\boldsymbol{\beta}$ given by solving the optimization problem

$$\mathrm{PLS}_{\mathrm{SC}}(\mathbf{y}, \boldsymbol{\beta}_b; \lambda, \lambda_c) = \|\mathbf{y} - \mathbf{B}\boldsymbol{\beta}_b\|_2^2 + \lambda \cdot \mathrm{pen}(\boldsymbol{\beta}_b) + \lambda_c \cdot \mathrm{con}(\boldsymbol{\beta}_b), \tag{3.2}$$

where $\mathrm{pen}(\boldsymbol{\beta}_b)$ is the smoothness penalty term for P-splines, see Section 2.3.2, and $\mathrm{con}(\boldsymbol{\beta}_b)$ specifies the user-defined shape-constraint penalty term to incorporate a priori domain

knowledge, see [4] and [5]. We further extend the concept proposed in literature of shape-constraint P-splines to two dimensions and discuss shape-constraint tensor-product P-splines, see Section 3.2.

3.1 Shape-constraint P-splines

In Section 2.3.2, we enforce smoothness by penalizing the second-order derivative of the underlying B-spline using finite differences of adjacent parameters over the whole input space to create the so-called P-splines. We will now utilize the same idea to create the shape-constrained penalty term $\text{con}(\boldsymbol{\beta}_b)$ in (3.2). We motivate the approach using the example of monotonic increasing behavior. The descriptions for the other constraints listed in Table 3.1 follow afterwards. In the following discussion, we use d B-spline basis functions of order l and the data set $\mathcal{D} = \{(x^{(i)}, y^{(i)}),\ i = 1, 2, \ldots, n\}$ resulting in the B-spline basis matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$.

3.1.1 Monotonic increasing constraint

A function is monotonic increasing if its first-order derivative is larger than or equal to zero for the whole input space. We therefore introduce a penalty of the form

$$\lambda_c \int \left(f(x)'\right)^2 \, \mathrm{d}x \quad \text{if } f'(x) < 0, \tag{3.3}$$

to penalize negative first-order derivatives of the estimated function. The penalty is weighted by the *constraint parameter* λ_c. Once again, we make use of the finite difference approximation, see Section 2.3.2. Now, the first-order derivative leads to a penalty of the form

$$\lambda_c \cdot \text{con}(\boldsymbol{\beta}_b) = \lambda_c \boldsymbol{\beta}_b^{\mathrm{T}} \mathbf{K}_c \boldsymbol{\beta}_b, \tag{3.4}$$

with the shape-constraint penalty matrix $\mathbf{K}_c = \mathbf{D}_1^{\mathrm{T}} \mathbf{V}_c \mathbf{D}_1 \in \mathbb{R}^{d \times d}$. The first-order derivative is approximated using the matrix form of the first-order finite difference operator $\Delta^1 \beta_j = \beta_j - \beta_{j-1}$ given by

$$\mathbf{D}_1 \boldsymbol{\beta}_b = \begin{bmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_d \end{bmatrix}, \tag{3.5}$$

with the difference matrix $\mathbf{D}_1 \in \mathbb{R}^{(d-1) \times d}$. A weighting matrix $\mathbf{V}_c \in \mathbb{R}^{(d-1) \times (d-1)}$ is introduced to handle the if-condition in (3.3). It is a diagonal matrix with the diagonal elements v_j defined as

$$v_{j-1}(\boldsymbol{\beta}_b) = \begin{cases} 0, & \text{if } \Delta^1 \beta_j \geq 0 \\ 1, & \text{if } \Delta^1 \beta_j < 0 \end{cases} \quad \text{for } j = 2, 3, \ldots, d. \tag{3.6}$$

Therefore, the weighting matrix $\mathbf{V}_c := \mathbf{V}_c(\boldsymbol{\beta}_b)$ depends on the parameters $\boldsymbol{\beta}_b$ and we arrive at a formulation similar to Ridge regression with a parameter-dependent, non-linear penalty matrix $\mathbf{K} := \mathbf{K}(\boldsymbol{\beta}_b)$, see Section 2.2.3. The objective function is finally of the form

$$\text{PLS}_{\text{SCP}}(\mathbf{y}, \boldsymbol{\beta}_b; \lambda, \lambda_c) = \|\mathbf{y} - \mathbf{B}\boldsymbol{\beta}_b\|_2^2 + \lambda \boldsymbol{\beta}_b^{\mathrm{T}} \mathbf{K} \boldsymbol{\beta}_b + \lambda_c \boldsymbol{\beta}_b^{\mathrm{T}} \mathbf{K}_c \boldsymbol{\beta}_b. \tag{3.7}$$

We use the iterative approach given in Algorithm 1 to estimate the optimal parameters $\hat{\boldsymbol{\beta}}_{SC}$ under the user-defined shape constraint of monotonic increasing behavior.

Algorithm 1: Estimation of the shape-constraint P-spline coefficients.

Result: $\hat{\boldsymbol{\beta}}_{\text{SC}}$
$i \leftarrow 1$;
$\hat{\boldsymbol{\beta}}_i \leftarrow$ Solution from (3.7) for $\lambda_c = 0$;
$\mathbf{V}_c^0 \leftarrow \mathbf{0}$;
$\mathbf{V}_c^1 \leftarrow \mathbf{V}_c(\hat{\boldsymbol{\beta}}_i)$;
while $\mathbf{V}_c^i \neq \mathbf{V}_c^{i-1}$ **do**
  $\hat{\boldsymbol{\beta}}_{i+1} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda \mathbf{D}_2^{\mathrm{T}}\mathbf{D}_2 + \lambda_c \mathbf{D}_c^{\mathrm{T}} \mathbf{V}_c^i \mathbf{D}_c)^{-1} \mathbf{X}^{\mathrm{T}}\mathbf{y}$;
  $\mathbf{V}_c^{i+1} \leftarrow \mathbf{V}_c(\hat{\boldsymbol{\beta}}_{i+1})$;
  $i \leftarrow i + 1$;
end
$\hat{\boldsymbol{\beta}}_{\text{SC}} \leftarrow \hat{\boldsymbol{\beta}}_i$;

In Algorithm 1, we make use of a Newton-Raphson scheme for the estimation of $\hat{\boldsymbol{\beta}}_{i+1}$. For more information, see Appendix A.3 and [5]. The approach described above incorporates the shape-constraint as soft constraint depending on the constraint parameter λ_c with the limits of no constraint for $\lambda_c \to 0$ and hard constraint for $\lambda_c \to \infty$. Therefore, λ_c should be set by hand reflecting the user's confidence in the a priori domain knowledge. To incorporate the other constraints listed in Table 3.1, we need to specify the shape-constraint penalty matrix $\mathbf{K}_c$ depending on the respective constraint, i.e. we define the constraint specific mapping matrix $\mathbf{D}_c$ and weighting matrix $\mathbf{V}_c$.

3.1.2 Monotonic decreasing constraint

Monotonic decreasing behavior can be introduced by penalizing positive first-order derivatives. Therefore, we use the matrix form of the first-order finite difference operator given in (3.5) as mapping matrix $\mathbf{D}_1 \in \mathbb{R}^{(d-1)\times d}$ and define the diagonal elements of the weighting matrix $\mathbf{V}_c \in \mathbb{R}^{(d-1)\times(d-1)}$ as

$$v_{j-1}(\boldsymbol{\beta}_b) = \begin{cases} 0, & \text{if } \Delta^1 \beta_j \leq 0 \\ 1, & \text{if } \Delta^1 \beta_j > 0 \end{cases}, \text{ for } j = 2, 3, \ldots, d. \tag{3.8}$$

3.1.3 Convex constraint

Convex behavior can be introduced by penalizing negative second-order derivatives. Therefore, we use the matrix form of the second-order finite difference operator $\Delta^2 \beta_j =$

$\beta_j - 2\beta_{j-1} + \beta_{j-2}$ given by

$$\mathbf{D}_2\boldsymbol{\beta}_b = \begin{bmatrix} 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_d \end{bmatrix}, \tag{3.9}$$

with the mapping matrix $\mathbf{D}_2 \in \mathbb{R}^{(d-2)\times d}$ and define the diagonal elements of the weighting matrix $\mathbf{V}_c \in \mathbb{R}^{(d-2)\times(d-2)}$ as

$$v_{j-2}(\boldsymbol{\beta}_b) = \begin{cases} 0, & \text{if } \Delta^2\beta_j \geq 0 \\ 1, & \text{if } \Delta^2\beta_j < 0 \end{cases}, \text{ for } j = 3, 4, \ldots, d. \tag{3.10}$$

3.1.4 Concave constraint

Concave behavior can be introduced by penalizing positive second-order derivatives. Therefore, we use the matrix form of the second-order finite difference operator, see (3.9), as mapping matrix $\mathbf{D}_2 \in \mathbb{R}^{(d-2)\times d}$ and define the diagonal elements of the weighting matrix $\mathbf{V}_c \in \mathbb{R}^{(d-2)\times(d-2)}$ as

$$v_{j-2}(\boldsymbol{\beta}_b) = \begin{cases} 0, & \text{if } \Delta^2\beta_j \leq 0 \\ 1, & \text{if } \Delta^2\beta_j > 0 \end{cases}, \text{ for } j = 3, 4, \ldots, d. \tag{3.11}$$

3.1.5 Peak constraint

Peak behavior can be introduced by penalizing negative first-order derivatives for the increasing part and positive first-order derivatives for the decreasing part of the function. Therefore, we use the matrix form of the first-order finite difference operator, see (3.5), as mapping matrix $\mathbf{D}_1 \in \mathbb{R}^{(d-1)\times d}$. The weighting matrix $\mathbf{V}_c \in \mathbb{R}^{(d-1)\times(d-1)}$ now has a special structure. First, we find the data point $x^{(\max)}$ corresponding to the peak value in the data by finding its data index max, i.e. $\max = \arg\max_i y^{(i)}$ for $i = 1, 2, \ldots, n$. Next, we automatically identify the dominant B-spline basis function $B_p^l(x)$ around $x^{(\max)}$, i.e. the B-spline basis function with the maximal value at $x^{(\max)}$, such that

$$B_p^l(x^{(\max)}) \geq B_j^l(x^{(\max)}), \quad \text{for } j = 1, 2, \ldots, d. \tag{3.12}$$

We now use the index p of the dominant B-spline basis function in the definition of the diagonal elements of the weighting matrix $\mathbf{V}_c \in \mathbb{R}^{(d-1)\times(d-1)}$ as

$$v_{j-1}(\boldsymbol{\beta}_b) = \begin{cases} 0, & \text{if } \Delta^1\beta_j \geq 0 \\ 1, & \text{if } \Delta^1\beta_j < 0 \end{cases}, \quad \text{for } j = 2, 3, \ldots, p \tag{3.13}$$

and

$$v_{j-1}(\boldsymbol{\beta}_b) = \begin{cases} 0, & \text{if } \Delta^1\beta_j \leq 0 \\ 1, & \text{if } \Delta^1\beta_j > 0 \end{cases}, \quad \text{for } j = p+1, p+2, \ldots, d. \tag{3.14}$$

3.1.6 Valley constraint

Valley behavior can be introduced by the same approach as above by multiplying the data with -1 or by always doing the inverse operation. Therefore, we use the matrix form of the first-order finite difference operator, see (3.5), as mapping matrix $\mathbf{D}_1 \in \mathbb{R}^{(d-1)\times d}$ and define the diagonal elements of the weighting matrix $\mathbf{V}_c \in \mathbb{R}^{(d-1)\times(d-1)}$ as

$$v_{j-1}(\boldsymbol{\beta}_b) = \begin{cases} 0, & \text{if } \Delta^1\beta_j \leq 0 \\ 1, & \text{if } \Delta^1\beta_j > 0 \end{cases}, \quad \text{for } j = 2, 3, \ldots, p \tag{3.15}$$

and

$$v_{j-1}(\boldsymbol{\beta}_b) = \begin{cases} 0, & \text{if } \Delta^1\beta_j \geq 0 \\ 1, & \text{if } \Delta^1\beta_j < 0 \end{cases}, \quad \text{for } j = p+1, p+2, \ldots, d, \tag{3.16}$$

for p being the index of the dominant B-spline basis function $B_p^l(x)$ around $x^{(\min)}$ with $\min = \arg\min_i y^{(i)}$ for $i = 1, 2, \ldots, n$ as data index of the valley value.

3.1.7 Jamming constraint

Jamming the function $f(x)$ by some point $p = (x^{(p)}, y^{(p)})$ means that the estimated function $f(x^{(p)}) \approx y^{(p)}$. This can be incorporated using the B-spline basis matrix $\mathbf{B} \in \mathbb{R}^{n\times d}$ as mapping matrix $\mathbf{D}_c \in \mathbb{R}^{n\times d}$ and a weighting matrix $\mathbf{V}_c \in \mathbb{R}^{n\times n}$ with diagonal elements v_j given by

$$v_j(\boldsymbol{\beta}_b) = \begin{cases} 0, & \text{if } x^{(j)} \neq x^{(p)} \\ 1, & \text{if } x^{(j)} = x^{(p)} \end{cases}, \text{for } j = 1, 2, \ldots, n. \tag{3.17}$$

3.1.8 Boundedness constraint

The user-defined constraint for boundedness from below by, e.g. $B = 0$ uses as mapping matrix $\mathbf{D}_c \in \mathbb{R}^{n\times d}$ the B-spline basis matrix $\mathbf{B} \in \mathbb{R}^{n\times k}$. For the weighting matrix $\mathbf{V}_c \in \mathbb{R}^{n\times n}$, the diagonal weights v_j are defined as

$$v_j(\boldsymbol{\beta}_b) = \begin{cases} 0, & \text{if } f(x^{(j)}) \geq B \\ 1, & \text{if } f(x^{(j)}) < B \end{cases}, \text{for } j = 1, 2, \ldots, n. \tag{3.18}$$

Using different values of B allows us to bound from below from any number B. Swapping the comparison operators in (3.18) enables us to bound functions from above.

3.2 Shape-constraint tensor-product P-splines

In Section 2.3.1, we extended the univariate approach of B-splines to bivariate tensor-product B-splines by using the row-wise Kronecker product, see Appendix A.2. We will now extend the tensor-product P-splines, see Section 2.3.2, to incorporate a priori domain knowledge as in Section 3.1 into the fitting process. We start by showing how to include a priori domain knowledge in one dimension, see Section 3.2.1, and describe the various constraints listed in Table 3.2. Afterwards, we show how to include a constraint based on both dimensions, see Section 3.2.5. In the following discussion, we use d_1 and d_2 B-spline basis functions of order l for the dimensions 1 and 2 respectively and the data set $D = \{(x_1^{(i)}, x_2^{(i)}, y^{(i)}),\ i = 1, 2, \ldots, n\}$ resulting in the tensor-product B-spline basis matrix $\mathbf{T} \in \mathbb{R}^{d_1 d_2 \times n}$.

Constraint	Description		Section
Monotonicity	increasing	$\frac{\partial f(x_1,x_2)}{\partial x_1} \geq 0$	3.2.1
	decreasing	$\frac{\partial f(x_1,x_2)}{\partial x_1} \leq 0$	3.2.2
Curvature	convex	$\frac{\partial^2 f(x_1,x_2)}{\partial x_1^2} \geq 0$	3.2.3
	concave	$\frac{\partial^2 f(x_1,x_2)}{\partial x_1^2} \leq 0$	3.2.4
Multiple			3.2.5

Table 3.2: Overview of the constraints for shape-constraint tensor-product P-splines.

Note that the constraints *Jamming* and *Boundedness* in Table 3.1 are dimension-independent and therefore not listed in Table 3.2. Nevertheless, they can also be applied to tensor-product B-splines using the descriptions in Section 3.1.7 and 3.1.8. We will present the scheme for input dimension 1. For dimension 2, minor changes need to be done, e.g. some reordering in the weighting matrix $\mathbf{V}_c$.

3.2.1 Monotonic increasing constraint in dimension 1

A function is monotonic increasing in dimension 1 if its first-order derivative in dimension 1 is larger than or equal to zero for the whole input space. Therefore, we introduce a penalty of the form

$$\lambda_c \iint \left(\frac{\partial f(x_1, x_2)}{\partial x_1} \right)^2 \mathrm{d}x_1 \, \mathrm{d}x_2 \quad \text{if } \frac{\partial f(x_1, x_2)}{\partial x_1} < 0, \tag{3.19}$$

to penalize negative first-order partial derivatives of the estimated function. The penalty term is again weighted by the *constraint parameter* λ_c. Approximating the first-order derivative by finite-differences of order 1, similar to (2.93) and (2.94), leads to a penalty of the known form

$$\lambda_c \cdot \mathrm{con}(\boldsymbol{\beta}_t) = \lambda_c \boldsymbol{\beta}_t^{\mathrm{T}} \mathbf{K}_c \boldsymbol{\beta}_t, \tag{3.20}$$

with the shape-constraint penalty matrix $\mathbf{K}_c = \mathbf{D}_c^{\mathrm{T}}\mathbf{V}_c\mathbf{D}_c \in \mathbb{R}^{d_1d_2 \times d_1d_2}$. The first-order derivative is approximated by the matrix form of the first-order finite difference operator, see (3.5), and by using the Kronecker product, cf. Appendix A.1, as

$$\mathbf{D}_c\boldsymbol{\beta}_t = (\mathbf{I}_{d_2} \otimes \mathbf{D}_1)\boldsymbol{\beta}_t, \tag{3.21}$$

with the mapping matrix $\mathbf{D}_c \in \mathbb{R}^{(d_1-1)d_2 \times d_1d_2}$ using the identity matrix $\mathbf{I}_{d_2} \in \mathbb{R}^{d_2 \times d_2}$ and the finite-difference matrix for the first dimension $\mathbf{D}_1 \in \mathbb{R}^{(d_1-1)\times d_1}$. The diagonal elements v_j of the weighting matrix $\mathbf{V}_c \in \mathbb{R}^{(d_1-1)d_2 \times (d_1-1)d_2}$ are defined as

$$v_{j+(i-1)(d_1-1)}(\boldsymbol{\beta}_t) = \begin{cases} 0, & \text{if } \Delta^1_{d_1}\beta_{j+(i-1)d_1+1} \geq 0 \\ 1, & \text{if } \Delta^1_{d_1}\beta_{j+(i-1)d_1+1} < 0 \end{cases}, \\ \text{for } j = 1, 2, \ldots, d_1 - 1 \text{ and } i = 1, 2, \ldots, d_2, \tag{3.22}$$

using the first-order finite-difference operator for dimension 1 $\Delta^1_{d_1}$ defined as

$$\Delta^1_{d_1}\beta_j = \beta_j - \beta_{j-1}. \tag{3.23}$$

Hence, we obtain an objective function similar to (3.2) as

$$\mathrm{PLS}_{\text{SC-TP}}(\mathbf{y}, \boldsymbol{\beta}_t; \lambda, \lambda_c) = \|\mathbf{y} - \mathbf{T}\boldsymbol{\beta}_t\|_2^2 + \lambda\boldsymbol{\beta}_t^{\mathrm{T}}\mathbf{K}\boldsymbol{\beta}_t + \lambda_c\boldsymbol{\beta}_t^{\mathrm{T}}\mathbf{K}_c\boldsymbol{\beta}_t, \tag{3.24}$$

with the smoothness penalty matrix $\mathbf{K} \in \mathbb{R}^{d_1d_2 \times d_1d_2}$, see Section 2.3.2, and the shape-constraint penalty matrix $\mathbf{K} \in \mathbb{R}^{d_1d_2 \times d_1d_2}$. The optimal parameters under the shape constraint $\hat{\boldsymbol{\beta}}_{\text{SC-TP}}$ can be estimated using the iterative scheme given in Algorithm 1.

3.2.2 Monotonic decreasing constraint for dimension 1

Monotonic decreasing behavior in dimension 1 can be introduced by penalizing positive first-order partial derivatives. Therefore, we use the matrix form of the first-order finite difference operator given in (3.5), i.e. $\mathbf{D}_1 \in \mathbb{R}^{(d_1-1)\times d_1}$, in the set up of the mapping matrix $\mathbf{D}_c \in \mathbb{R}^{(d_1-1)d_2 \times d_1d_2}$, see (3.21), and define the diagonal elements of the weighting matrix $\mathbf{V}_c \in \mathbb{R}^{(d_1-1)d_2 \times (d_1-1)d_2}$ as

$$v_{j+(i-1)(d_1-1)}(\boldsymbol{\beta}_t) = \begin{cases} 0, & \text{if } \Delta^1_{d_1}\beta_{j+(i-1)d_1+1} \leq 0 \\ 1, & \text{if } \Delta^1_{d_1}\beta_{j+(i-1)d_1+1} > 0 \end{cases}, \\ \text{for } j = 1, 2, \ldots, d_1 - 1 \text{ and } i = 1, 2, \ldots, d_2, \tag{3.25}$$

using the first-order finite difference operator for dimension 1 in (3.23).

3.2.3 Convex constraint for dimension 1

Convex behavior in dimension 1 can be introduced by penalizing negative second-order partial derivatives. Therefore, we use the matrix form of the second-order finite difference operator in (3.9), i.e. $\mathbf{D}_2 \in \mathbb{R}^{(d_1-2)\times d_1}$, in the set up of the mapping matrix $\mathbf{D}_c \in \mathbb{R}^{(d_1-2)d_2\times d_1 d_2}$, see (3.21), and define the diagonal elements of the weighting matrix $\mathbf{V}_c \in \mathbb{R}^{(d_1-2)d_2\times(d_1-2)d_2}$ as

$$v_{j+(i-1)(d_1-2)}(\boldsymbol{\beta}_t) = \begin{cases} 0, & \text{if } \Delta^2_{d_1}\beta_{j+(i-1)d_1+2} \geq 0 \\ 1, & \text{if } \Delta^2_{d_1}\beta_{j+(i-1)d_1+2} < 0 \end{cases} \tag{3.26}$$
$$\text{for } j = 1, 2, \ldots, d_1 - 2 \text{ and } i = 1, 2, \ldots, d_2,$$

using the second-order finite difference operator for dimension 1 defined as

$$\Delta^2_{d_1}\beta_j = \beta_j - 2\beta_{j-1} + \beta_{j-2}. \tag{3.27}$$

3.2.4 Concave constraint for dimension 1

Concave behavior in dimension 1 can be introduced by penalizing positive second-order partial derivatives. Therefore, we use the matrix form of the second-order finite difference operator in (3.9), i.e. $\mathbf{D}_2 \in \mathbb{R}^{(d_1-2)\times d_1}$, in the set up of the mapping matrix $\mathbf{D}_c \in \mathbb{R}^{(d_1-2)d_2\times d_1 d_2}$, see (3.21), and define the diagonal elements of the weighting matrix $\mathbf{V}_c \in \mathbb{R}^{(d_1-2)d_2\times(d_1-2)d_2}$ as

$$v_{j+(i-1)(d_1-2)}(\boldsymbol{\beta}_t) = \begin{cases} 0, & \text{if } \Delta^2_{d_1}\beta_{j+(i-1)d_1+2} \leq 0 \\ 1, & \text{if } \Delta^2_{d_1}\beta_{j+(i-1)d_1+2} > 0 \end{cases} \tag{3.28}$$
$$\text{for } j = 1, 2, \ldots, d_1 - 2 \text{ and } i = 1, 2, \ldots, d_2,$$

using the second-order finite difference operator for dimension 1 defined in (3.27).f

3.2.5 Shape constraints in two dimensions

To enforce shape constraints in two dimensions, e.g monotonic increasing behavior in dimension 1 (c_1) and monotonic decreasing behavior in dimension 2 (c_2), we use the same idea as given in 3.2.1 for both dimensions. Hence, we obtain an objective function similar to (3.2) as

$$\text{PLS}_{\text{SC-TP}}(\mathbf{y}, \boldsymbol{\beta}_t; \lambda, \lambda_{c_1}, \lambda_{c_2}) = \|\mathbf{y} - \mathbf{T}\boldsymbol{\beta}_t\|_2^2 + \lambda\boldsymbol{\beta}_t^{\mathrm{T}}\mathbf{K}\boldsymbol{\beta}_t + \lambda_{c_1}\boldsymbol{\beta}_t^{\mathrm{T}}\mathbf{K}_{c_1}\boldsymbol{\beta}_t + \lambda_{c_2}\boldsymbol{\beta}_t^{\mathrm{T}}\mathbf{K}_{c_2}\boldsymbol{\beta}_t, \tag{3.29}$$

with the smoothness penalty matrix $\mathbf{K} \in \mathbb{R}^{d_1 d_2 \times d_1 d_2}$, see Section 2.3.2, and two *constraint parameter* λ_{c_1} and λ_{c_2} reflecting the users trust in the a priori domain knowledge. The optimal parameters under the shape constraint $\hat{\boldsymbol{\beta}}_{\text{SC-TP}}$ can then be estimated again using the iterative scheme given in Algorithm 1.

4 Illustrative Simulation Examples

In Chapter 3, we discussed the theoretical basis for shape-constraint P-splines and their ability to incorporate a priori domain knowledge into the fitting process by choice of the constraint term described by the mapping matrix $\mathbf{D}_c$ and the weighting matrix $\mathbf{V}_c$. We will now consider the practical application of these, as well as their limits in terms of data fitting and constraint fidelity based on the example of peak behavior. We will evaluate the performance of shape-constraint P-splines using noisy data, see Section 4.1, and sparse data, see Section 4.2, and compare it to B-splines and P-splines. If not further stated, we will use equidistant knot placement throughout this section. Furthermore, we will discuss the effect of the constraint parameter λ_c, see Section 4.3. In Section 4.4, we show that the presented algorithm incorporates the a priori domain knowledge via soft constraints, hence it can not be guaranteed that the estimate holds the given constraint in every possible situation. In Section 4.5, we apply shape-constraint P-splines to discontinuous data generated by the *Heaviside Function*.

Throughout this chapter, we will mostly use the function $f(x)$, i.e.

$$f(x) = \exp\left(-\frac{(x-0.35)^2}{0.1}\right) \tag{4.1}$$

as underlying true function. The function is monotonically increasing up to the peak value at $x = 0.35$ and then monotonically decreasing.

It is important to notice that the addition of the constraint term in (3.7) further reduces the effective degree of freedom of the model, similar as for P-splines, resulting in a less flexible model. We therefore expect that the measured metric on the training data will be worse compared to a pure B-spline fit. Nevertheless, the metric of interest is the mean squared error MSE, see (2.31), on the validation data, i.e. the held out data that the model has not seen before. Supposing that the a priori domain knowledge reflects the true, underlying function behavior, we expect the measured error on the validation data to be lower than or equal to the error given by an optimal P-spline fit. Here, optimality is based on the optimal smoothing parameter λ given by generalized cross-validation, see Section 2.2.1. This can be seen by recognizing the equivalence of the objective functions for P-splines, see (2.89), and shape-constraint P-splines, see (3.7), when the underlying B-spline fit does not violate the user-defined constraint, i.e. all $v_j = 0$. Note this feature is one of the limits of shape-constraint P-splines, i.e. we cannot influence the model using this approach if no constraint violations are present. On the other hand, if the a priori domain knowledge reflects the underlying function, we can expect a far better generalization capability of the model compared to the B-spline fits, especially in situations of noisy or sparse data.

4.1 Peak Constraint in Practice

We will now utilize shape-constraint P-splines and the a priori domain knowledge of peak behavior to fit the data $\mathcal{D} = \{(x^{(i)}, y^{(i)}),\ i = 1, 2, \ldots, n\}$. The data is artificially generated by random sampling of $n = 200$ points $x^{(i)}$ in the interval $[0.1, 0.8]$. These are then used as input for the function $f(x)$ in (4.1) to generate the $y^{(i)}$ as

$$y^{(i)} = f(x^{(i)}) + \epsilon^{(i)}, \tag{4.2}$$

with the Gaussian noise $\epsilon^{(i)}$ characterized by the mean $\mu = 0$ and the variance $\sigma^2 = 0.01$. We randomly split the data in a training set $\mathcal{D}_{\text{train}}$ with 150 samples and a validation set $\mathcal{D}_{\text{val}}$ with 50 samples. For the validation set, we further generate the true validation set $\mathcal{D}_{\text{val,true}}$ for which we omit the Gaussian noise ϵ and therefore obtain the true function values given by the function in (4.1).

At first, we use $d = 45$ B-spline basis functions of order $l = 3$ to fit a B-spline to the training data $\mathcal{D}_{\text{train}}$. Then, we fit a P-spline using the same configuration $d = 45$ and $l = 3$ with an optimal smoothness parameter $\lambda_{\text{opt}} = 7.74$ determined by generalized cross-validation. Finally, we enforce the peak behavior with a shape-constraint P-spline, see Section 3.1, with $d = 45$ and $l = 3$ as well as the optimal smoothness parameter $\lambda_{\text{opt}} = 7.74$ and the constraint parameter $\lambda_c = 6000$ reflecting high trust in the a priori domain knowledge. The various fits and the full data $\mathcal{D}$ are shown in Figure 4.1. The mean squared errors MSE, see (2.31), on the validation data, as well as on the true, underlying function in (4.1), are given in Table 4.1.

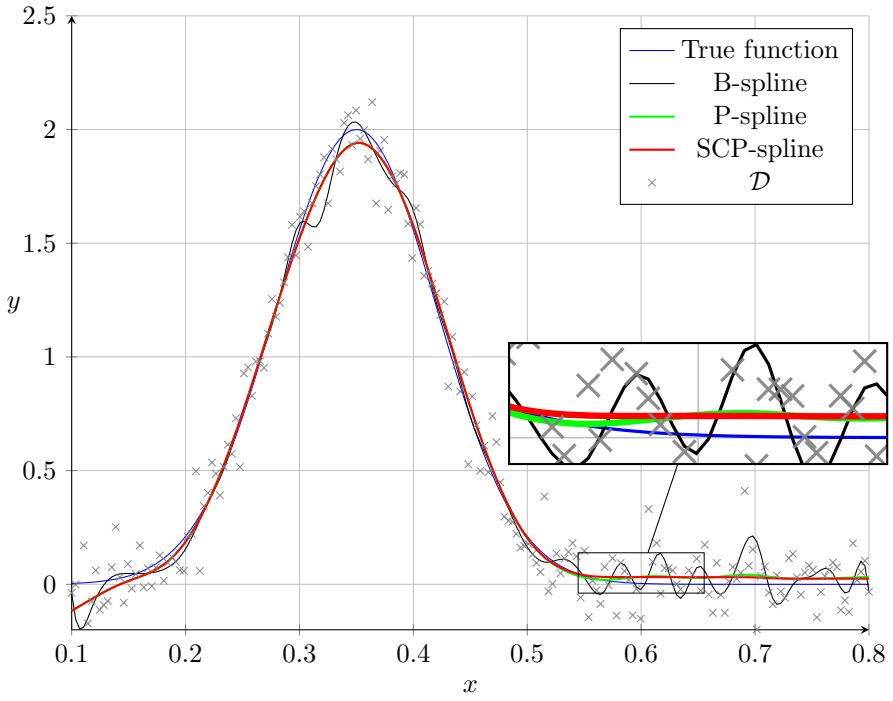

Figure 4.1: B-spline, P-spline and SCP-spline fit for the data $\mathcal{D}$.

The B-spline, as black curve in Figure 4.1, captures the basic shape of the true function, but the flexibility of the B-spline, due to the high number of B-spline basis functions, leads to a wiggly estimate especially for the almost constant part. This violates the peak behavior of the true function which is being monotonically decreasing for $x \in [0.35, 0.8]$. For the P-spline (green), this problem relaxes due to the smoothing aspect of the penalty term but does not completely vanish, as seen in the magnified part in Figure 4.1. Note that the additional smoothness penalty given by the P-spline already fits the data almost perfectly with respect to a smooth result. The SCP-spline estimate gives almost identical results as the P-spline as it adjusts only the parts of the P-spline, which violate the peak constraint. It may be seen as "fine-tuning" of the fit using the a priori domain knowledge. Hence, it is the best solution here based on a visual interpretation of the a priori knowledge.

Model	$\text{MSE}_{\mathcal{D}_{\text{val}}}$	$\text{MSE}_{\mathcal{D}_{\text{val,true}}}$
B-spline	$1.9 \cdot 10^{-2}$	$7.04 \cdot 10^{-3}$
P-spline	$1.22 \cdot 10^{-2}$	$1.52 \cdot 10^{-3}$
SCP-spline	$1.22 \cdot 10^{-2}$	$1.48 \cdot 10^{-3}$

Table 4.1: Mean squared errors on the validation set $\mathcal{D}_{\text{val}}$ and the true validation set $\mathcal{D}_{\text{val,true}}$

The mean square errors $\text{MSE}_{\mathcal{D}_{\text{val}}}$, see (2.31), on the noisy validation data $\mathcal{D}_{\text{val}}$ in Table 4.1 for P-spline and SCP-spline are almost identical and do not show a favorable model. Comparing the various models with the true, underlying function by calculating $\text{MSE}_{D_{\text{val,true}}}$, i.e. the mean squared error on the true function values, leads to the assessment that the SCP-spline is the more accurate model with regard to the a priori knowledge. This also coincides with the graphs in Figure 4.1. Hence, the incorporation of a priori domain knowledge via shape-constraints improves the generalization capability measured by the mean squared error on the true function values in our example.

4.2 Peak Constraint and Sparse Data

We will now examine the behavior of B-, P- and SCP-splines for sparse data, i.e. few data and unevenly distributed, sampled from the function in (4.1). The data set $\mathcal{D}$ now contains 70 data points distributed in a way that there is only few data in the peak region, i.e. for $x \in [0.2, 0.5]$. In this region, we have only 10 data points. We perform an random train-validation split of the data in $\mathcal{D}$, i.e. the training data $\mathcal{D}_{\text{train}}$ consists of 52 points and the validation data $\mathcal{D}_{\text{val}}$ consists of 18 points. For the validation set, we again generate the true validation set $\mathcal{D}_{\text{val,true}}$ for which we omit the Gaussian noise ϵ.

We follow the same approach as above, i.e. fit a B-spline, perform generalized cross-validation to determine the optimal P-spline and finally apply the shape-constraint to enforce the peak behavior. The small data set indicates that a different, not equidistant

knot placement may be helpful. Hence, we carry out 2 experiments, one with equidistant knot placement and the other with quantile-based knot placement, see Section 2.3.1 for further information on quantile-based knot placement. We use $d = 25$ B-spline basis functions of order $l = 3$. The optimal smoothness parameter was given as $\lambda_{\text{opt}} = 0.00657$ for the equidistant fit and $\lambda_{\text{opt}} = 0.00215$ for the quantile-based fit determined by generalized cross-validation. The constraint parameter λ_c was set to $\lambda_c = 1000$ for both fits, reflecting high trust in the a priori domain knowledge. Figure 4.2 shows the resulting fits with equidistant knot placement and Figure 4.3 with quantile-based knot placement.

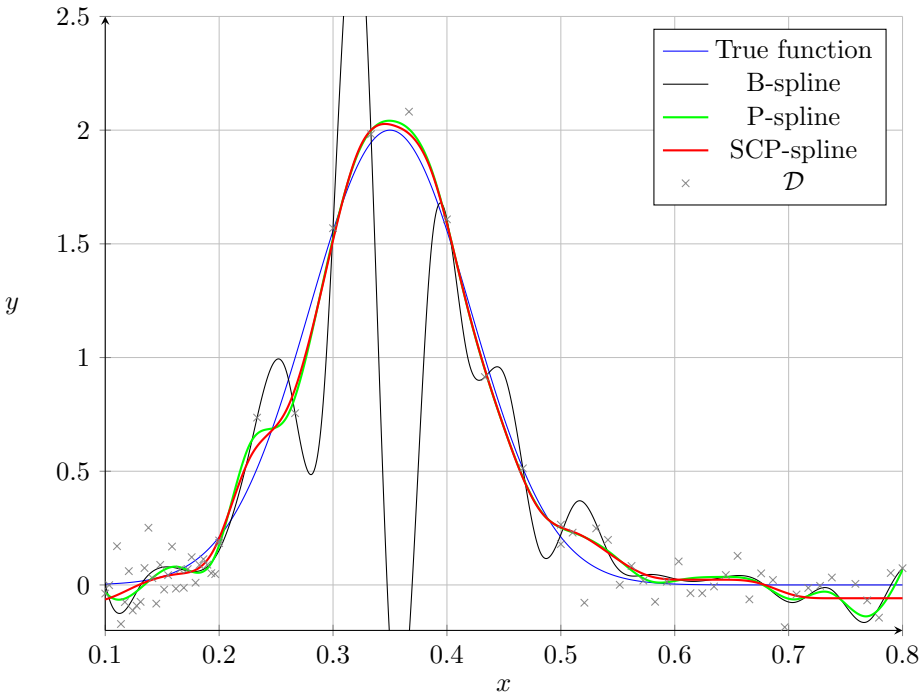

Figure 4.2: Equidistant B-spline, P-spline and SCP-spline fit for sparse data $\mathcal{D}$.

The B-spline (black) in Figure 4.2 shows the problem of equidistant knot placement in sparse data situations. The estimate becomes very wiggly in the region of few data, similar to polynomial fits with a high polynomial degree. Nevertheless, utilizing the regularization through the additional smoothness penalty in P-splines, the estimate (green) becomes smoother and reflects the true function quite well. The shape-constraint P-spline (red) further improves the quality of the fit, which can be seen by a comparison of the mean squared errors MSE, see (2.31), in Table 4.2.

Model	$\text{MSE}_{\mathcal{D}_{\text{val}}}$	$\text{MSE}_{\mathcal{D}_{\text{val,true}}}$
B-spline	0.32	0.27
P-spline	$1.44 \cdot 10^{-2}$	$3.94 \cdot 10^{-3}$
SCP-spline	$1.36 \cdot 10^{-2}$	$2.32 \cdot 10^{-3}$

Table 4.2: Mean squared errors on the validation set $\mathcal{D}_{\text{val}}$ and the true validation set $\mathcal{D}_{\text{val,true}}$ for equidistant knot placement.

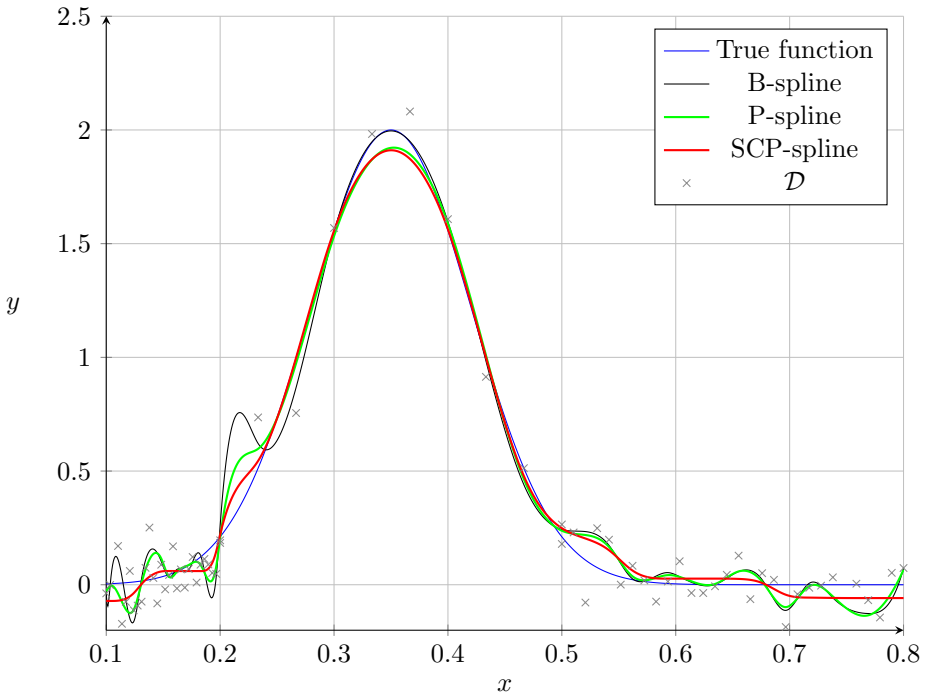

Figure 4.3: Quantile-based B-spline, P-spline and SCP-spline fit for sparse data $\mathcal{D}$.

The quantile-based B-spline (black) in Figure 4.3 shows some interesting features. At first, it fits the peak surprisingly well despite the small sample size in this area. This is due to the inherent smoothing aspect of quantile-based knot placement in regions of small sample size, see 2.3.1. Further, we see the flexibility of the B-spline in the regions with more data, i.e. $x \in [0.1, 0.2]$ and $x \in [0.5, 0.8]$, where it clearly reproduces the noise part instead of the true function. Hence, quantile-based knot placement may lead to partial overfitting, i.e. overfitting of the estimate only on a part of the input space. The P-spline (green) does not resolve this problem as the estimates (black and green) overlap almost everywhere. The reason for this effect is based on the approximation of the smoothness penalty term (2.84) by means of the second-order finite difference (2.85), which is valid for equidistant knot placement. However, a weighting arises for quantile-based knot placement.

Nevertheless, including additional regularization through the shape-constraint leads to an estimate which is smooth, follows the a priori domain knowledge and generalizes better, also demonstrated by the comparison of $\text{MSE}_{D_{\text{val}}}$ and $\text{MSE}_{D_{\text{val,true}}}$ in Table 4.3.

Model	$\text{MSE}_{\mathcal{D}_{\text{val}}}$	$\text{MSE}_{\mathcal{D}_{\text{val,true}}}$
B-spline	$2.52 \cdot 10^{-2}$	$9.24 \cdot 10^{-3}$
P-spline	$2.07 \cdot 10^{-2}$	$6.45 \cdot 10^{-3}$
SCP-spline	$1.59 \cdot 10^{-2}$	$3.56 \cdot 10^{-3}$

Table 4.3: Mean squared errors on the validation set $\mathcal{D}_{\text{val}}$ and the true validation set $\mathcal{D}_{\text{val,true}}$ for quantile-based knot placement.

To summarize, we can conclude that equidistant knot placement is superior to quantile-based knot placement in this example as it yields lower mean squared errors MSE on the validation set $\mathcal{D}_{\text{val}}$ and on the true validation set $\mathcal{D}_{\text{val,true}}$, seen in Table 4.2 and Table 4.3.

4.3 The Effect of the Constraint Parameter λ_c

We will now discuss the effect of the constraint parameter λ_c. As noted in Section 4.1, the shape-constraint acts as "fine-tuning" of the estimate according to the a priori domain knowledge, i.e. it only changes the estimate at locations where it violates the user-defined shape constraint. We use the constraint parameter λ_c as a measure of trust in the a priori domain knowledge, such that high trust is reflected by high values of λ_c. To show this in practice, we use the data $\mathcal{D}$ given in Section 4.1 and fit shape-constraint P-splines using various values of the parameter λ_c. The results are show in Figure 4.4. Since most constraint violations are present for larger $x > 0.55$, we focus on this part of the data.

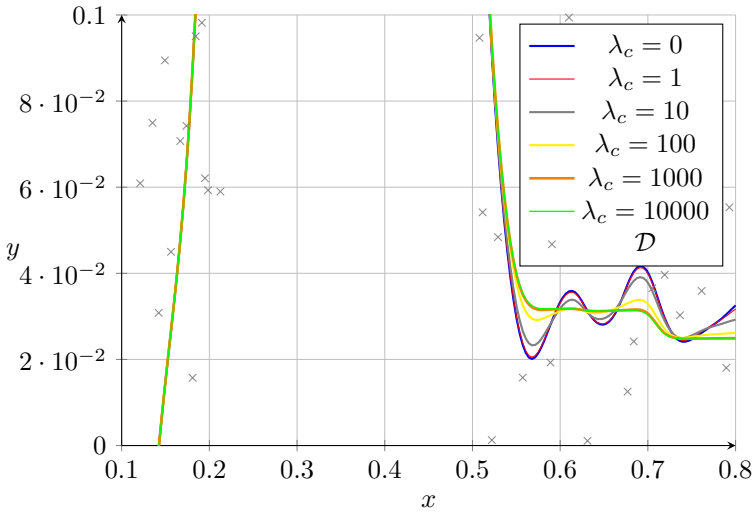

Figure 4.4: SCP-splines for different λ_c for data $\mathcal{D}$.

In Figure 4.4, we clearly see that with increasing λ_c we enforce the a priori domain knowledge. For small $\lambda_c \leq 10$, the estimates (blue, red and gray) violate the constraint qualitatively quite strong. Medium values of $\lambda_c = 100$ already produce an estimate (yellow) that follows the constraint far better, but not optimally. High values of $\lambda_c \geq 1000$ enforce the a priori domain knowledge, as seen by the orange and the green estimate. Nevertheless, quantitatively small violations of the a priori domain knowledge are possible even for this values of the constraint parameters.

To further investigate the effect of the constraint parameter λ_c, we will use the so-called *L-curve* method, see Section 2.2.3 and [27], which is a well known graphical heuristic for choosing regularization parameters. The main idea behind the L-curve is to plot the residual of the model, i.e. $\|\mathbf{y} - \mathbf{B}\boldsymbol{\beta}\|_2^2$, against the penalty term $\boldsymbol{\beta}^{\mathrm{T}}\mathbf{K}\boldsymbol{\beta}$ for various values of the constraint parameter λ_c, i.e. to plot the L-curve which is parameterized by λ_c. The heuristic is then to choose the value according to the maximal curvature of the plotted curve. The L-curve for the example at hand is shown in Figure 4.5. To ensure an enhanced visualization of the shape, we plot a finer grid of λ_c values compared to Figure 4.4. The colorbar indicates the level of λ_c.

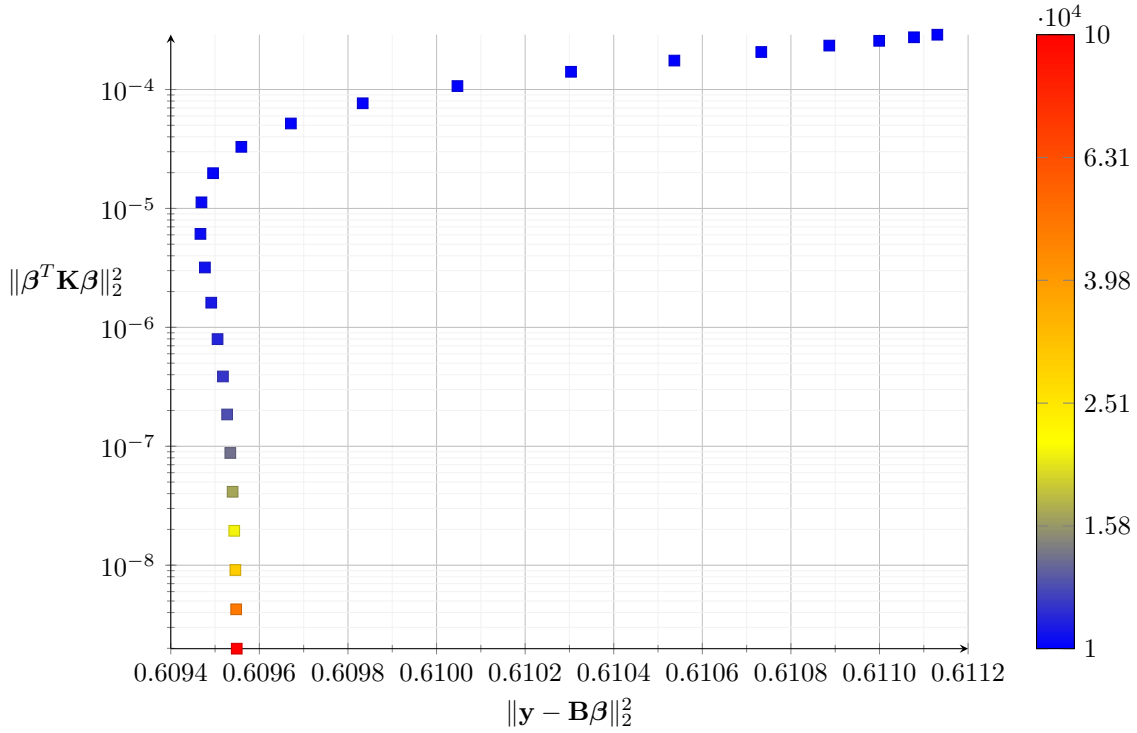

Figure 4.5: L-curve evaluated on the validation data $\mathcal{D}_{\text{val}}$.

According to the L-curve in Figure 4.5, an increase in λ_c leads to a decrease in the residual on the validation data $\mathcal{D}_{\text{val}}$ up to a critical $\lambda_c \approx 148$, which would be the optimal regularization parameter as it is also in the region of maximal curvature, i.e. the corner of the curve. Nevertheless, please note that a constraint parameter λ_c selected by data-driven methods like the L-curve may not be the best parameter for situations in which we want to preserve the a priori domain knowledge. Consider for example Figure 4.4. The optimal λ_c according to the L-curve is between the yellow and the orange curve and may violate the constraint for $x \geq 0.5$.

The L-curve evaluated on the true validation data $\mathcal{D}_{\text{val,true}}$ is given in Figure 4.6. Here we see that for increasing λ_c, as given in the colorbar, the residual on the true validation data $\mathcal{D}_{\text{val,true}}$ decreases. The optimal regularization parameter here would be the maximum $\lambda_c = 10^4$, which is given as green curve in Figure 4.4.

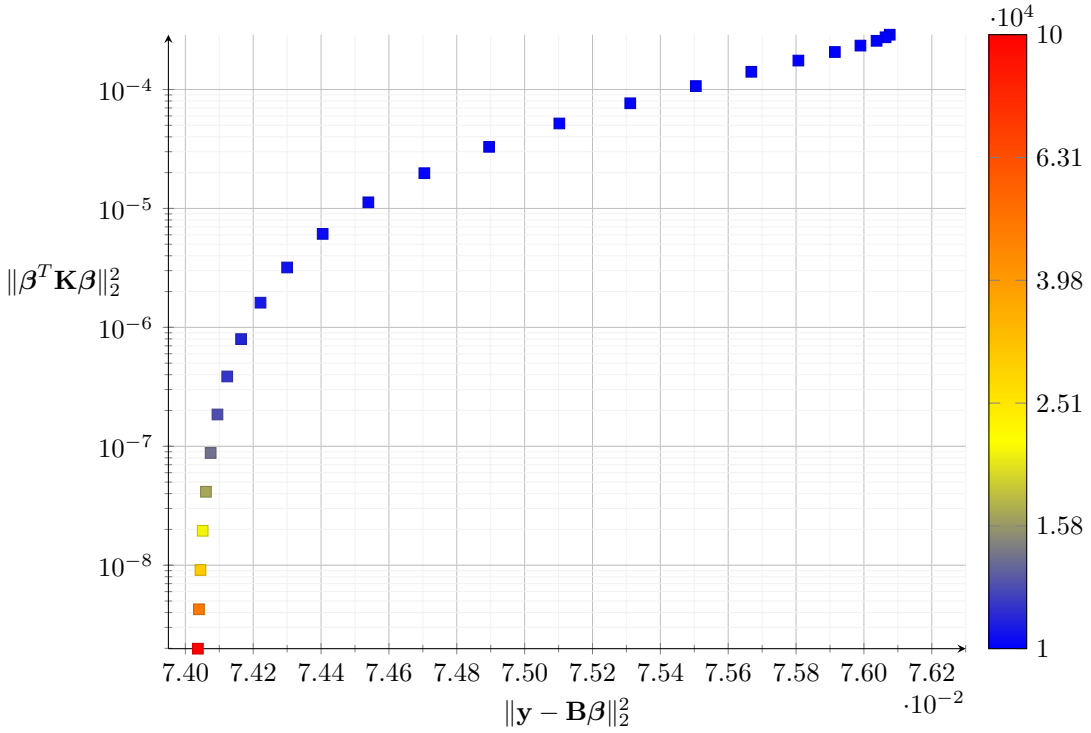

Figure 4.6: L-curve evaluated on the true validation data $\mathcal{D}_{\text{val,true}}$.

4.4 Limits of the a priori Domain Knowledge

We will now consider an example in which the a priori domain knowledge does not comply with the data. For this, we only use the increasing part of the function in (4.1), i.e. for $x \in [0.1, 0.38]$, and the monotonic decreasing constraint. We again use $d = 45$ basis functions of order $l = 3$ to fit shape-constraint P-splines with various λ_c. The results for $\lambda_c = [10, 10^3, 10^5, 10^{14}, 10^{17}]$ are shown in Figure 4.7.

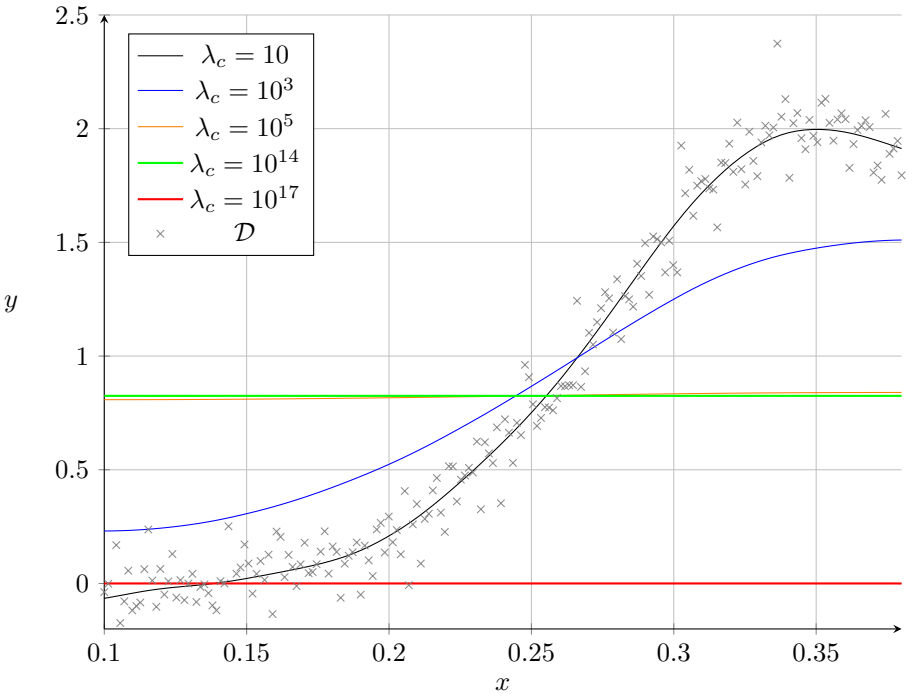

Figure 4.7: Decreasing constraint for increasing data and various λ_c.

For small constraint parameter $\lambda_c = 10$ (black), the effect of the constraint is not visible in the fit even though it is active almost everywhere. The data error $\|\mathbf{y} - \mathbf{B}\boldsymbol{\beta}\|_2^2$ is the dominant term in the cost function (3.7). The importance of the constraint term in the cost function is increased for $\lambda_c = 1000$ (blue). The algorithm tries to find the optimal trade-off between data fidelity and a priori domain knowledge. This is further enhanced for $\lambda_c = 10^5$ (orange). Here, the algorithm tries to minimize the effect of the constraint by minimizing the differences between adjacent parameters resulting in an almost straight line estimate. Further increasing λ_c to large values up to $\lambda_c = 10^{14}$ (green) does not change the estimate. Nevertheless, increasing the constraint parameter even further, i.e. $\lambda_c = 10^{17}$, leads to a fit (red) that is almost 0 everywhere. Therefore, we conclude that the constraint of decreasing behavior cannot be guaranteed even for large $\lambda_c \to \infty$.

4.5 Discontinuous Data

We will now try to fit data with a discontinuous step. Such data is often encountered in control applications and represents a challenge for basis function based data-driven models. For this, we generate data based on the so-called *Heaviside Function*, i.e.

$$H(x) = \begin{cases} 0 & , \ x < 0 \\ 1 & , \ x \geq 0 \end{cases} , \tag{4.3}$$

by random sampling of $n = 200$ points $x^{(i)}$ in the interval $[-0.1, 0.1]$ to get the target values $y^{(i)}$ as

$$y^{(i)} = 4H(x^{(i)}) + \epsilon^{(i)}, \tag{4.4}$$

with the Gaussian noise $\epsilon^{(i)}$ characterized by the mean $\mu = 0$ and the variance $\sigma^2 = 0.01$.

At first, we use $d = 100$ B-spline basis functions of order $l = 3$ to fit a B-spline to the training data $\mathcal{D}_{\text{train}}$. Then, we fit a P-spline using the same configuration $d = 100$ and $l = 3$ with an optimal smoothness parameter $\lambda_{\text{opt}} = 0.02$ determined by generalized cross-validation. Finally, we enforce the monotonic increasing behavior with a shape-constraint P-spline, see Section 3.1, with $d = 100$ and $l = 3$ as well as the optimal smoothness parameter $\lambda_{\text{opt}} = 0.02$ and the constraint parameter $\lambda_c = 6000$ reflecting high trust in the a priori domain knowledge. The various fits and the full data $\mathcal{D}$ are shown in Figure 4.8.

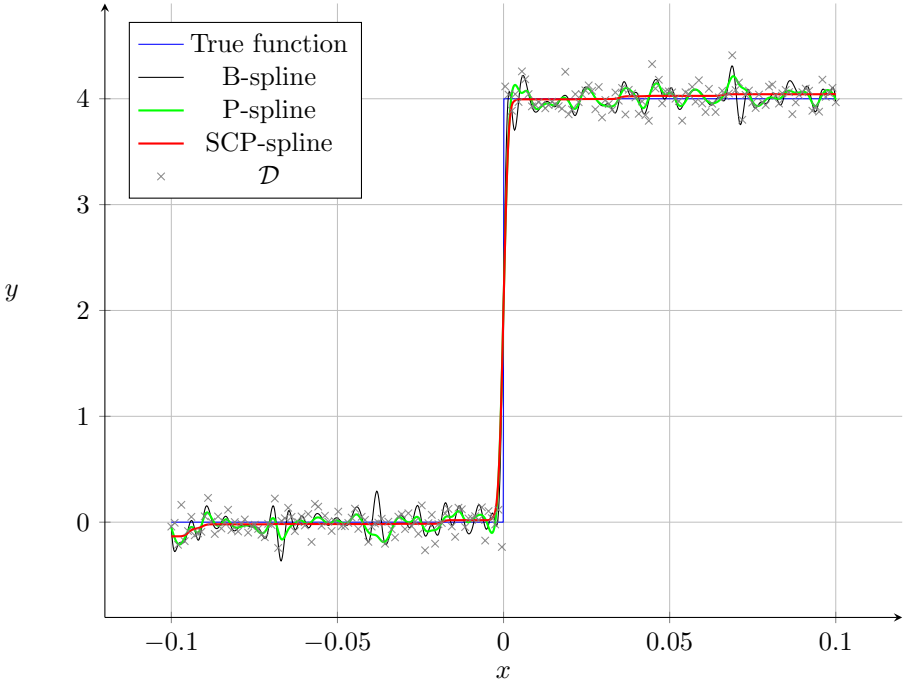

Figure 4.8: B-spline, P-spline and SCP-spline fit for data $\mathcal{D}$.

The flexible B-spline (black) and P-spline (green) are not able to recover the true nature of the *Heaviside Function* whereas the shape-constraint P-spline estimate (red) is surprisingly accurate. Especially for the constant parts, the additional regularization through the shape constraint improves the quality of the fit.

5 Practical Applications

We will now apply the concept of shape-constraint P-splines at real-world data to incorporated a priori domain knowledge into the modeling process. In the first example, see Section 5.1, we use the domain knowledge to enhance the estimation of the heat transfer coefficient in a heat-treatment process. The main challenge here is the sparse and noisy data situation combined with a highly non-linear process. In the second example, see Section 5.2, we investigate the behavior of a servo compensation which possesses a discontinuity as main challenge.

5.1 Parameter Estimation

The data is generated from a heat-treatment process of aluminum strips. The aluminum strip is heated up to a specific temperature, hold at this temperature for some predefined time and then rapidly cooled down to room temperature by means of water jets. The controlled heating and cooling enhances the temperature evolution and thus the specific mechanical properties of the aluminum strip.

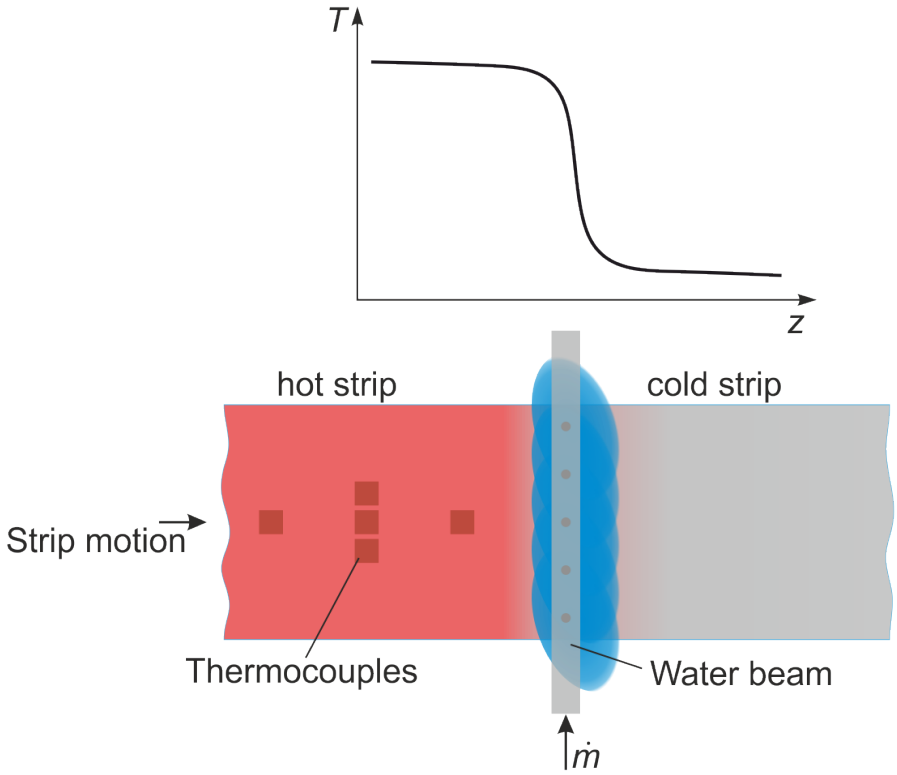

Figure 5.1: Schematic representation of the aluminum strip and the water beam.

The area of interest is the cooling phase of the strip, in which some highly non-linear processes take place due to phase transitions of the cooling medium water, see Figure 5.1. The temperature evolution T of the strip mainly depends on the heat transfer coefficient α characterizing the interaction between the aluminum strip and water. A complete description of the physical effects during water cooling is still research. Therefore, modeling of the heat transfer coefficient using first-principle methods is not expedient. Nevertheless, we can use the first-principle ideas in the form of a priori domain knowledge. To describe the temperature evolution during the cooling phase, we try to estimate the heat transfer coefficient, i.e.

$$\alpha := \alpha(T, \dot{m}), \tag{5.1}$$

as a function of the temperature T of the aluminum strip and the mass flow $\dot{m}$ of the cooling medium. We know beforehand, that the heat transfer coefficient α may only increase with increasing mass flow $\dot{m}$ and that it shows unimodal peak behavior for increasing temperature T, motivated by the so-called Nukiyama curve given in Figure 5.2, see [32] and [33].

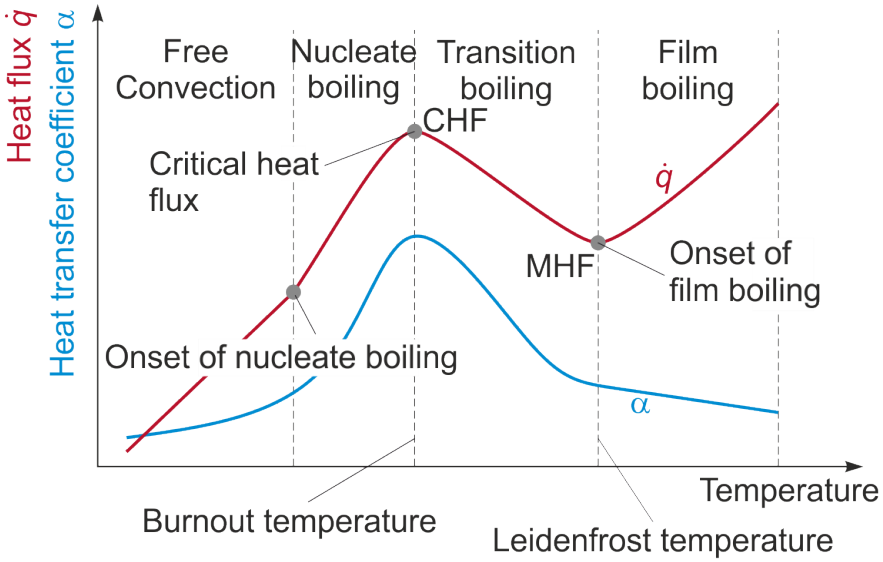

Figure 5.2: Nukiyama curve.

The data is determined by elaborate experiments performed on a prototype of Ebner Industrieofenbau. The distribution of data is visualized in Figure 5.3. Here we show how many data points are given in a square area of approximately 50 K and 0.9 kg/s, which represent approximately 1% of the whole input space. We have some regions, where no data is available, while the majority of data points is located in small areas, see Section 4.2. This problem, i.e. of irregular data distribution, is often encountered in real-world situations.

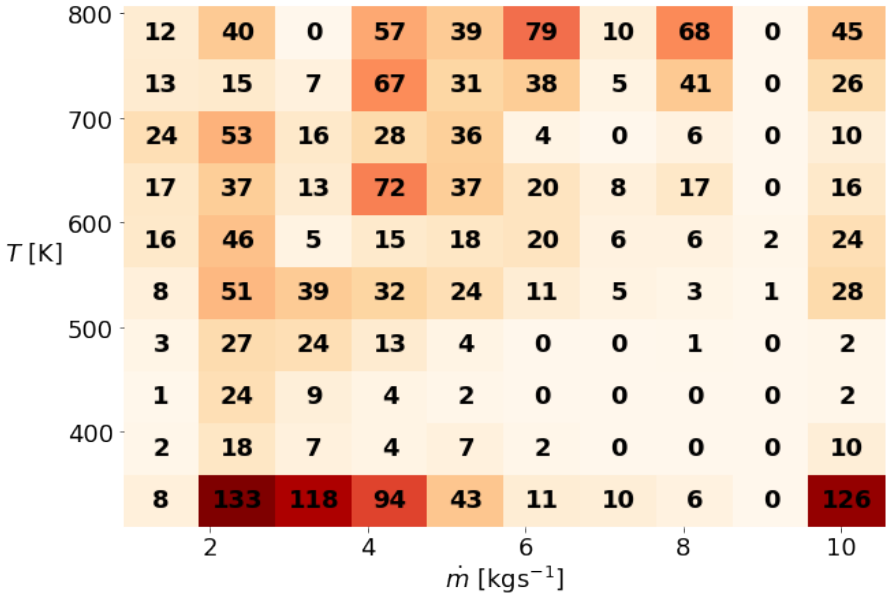

Figure 5.3: Data distribution.

The data distribution may lead to some difficulties when using B-splines and tensor-product B-splines, as they do not handle sparse data situations well, cf. Chapter 4. Nevertheless, using regularization via smoothness and constraint penalties should help to cope with the data situation.

We will now use various models based on B-spline bases, tensor-product B-spline bases and their shape-constraint alternatives to recover a model for the heat transfer coefficient given the data. The following list, in which $s(x_i)$ denotes the use of a B-spline basis for input $x_i \in \{\dot{m}, T\}$ and $t(\dot{m}, T)$ denotes the use of a tensor-product B-spline basis, describes the models that are analyzed.

(i) M1 $= s(\dot{m}) + s(T)$ without constraints

(ii) M2 $= s(\dot{m}) + s(T)$, with monotonicity for $s(\dot{m})$ and unimodal peak behavior for $s(T)$

(iii) M3 $= t(\dot{m}, T)$ without constraints

(iv) M4 $= t(\dot{m}, T)$, with monotonicity for input $\dot{m}$

(v) M5 $= s(\dot{m}) + s(T) + t(\dot{m}, T)$, as additive model without constraints

(vi) M6 $= s(\dot{m}) + s(T) + t(\dot{m}, T)$, as additive model with constraints defined in M2 and M4

Note that we use a smoothness penalty optimized via generalized cross-validation for each individual basis. The unconstraint models M1, M3 and M5 are therefore P-spline models rather than pure B-spline models. The shape-constraint models M2, M4 and M6 are SCP-spline models.

To fit the models listed above, we perform a randomized train-validation split on the data, i.e. we split the data into a training set $\mathcal{D}_{\text{train}}$ and a validation set $\mathcal{D}_{\text{val}}$, fit the models to the resulting training set and evaluate its performance by calculating the mean squared error MSE, see (2.31), on the validation set $\mathcal{D}_{\text{val}}$ and the effective degree of freedom EDoF of the model, see (2.49), as well as by visual inspection. We choose to split the data into sets of the same size to generate a more stable estimation of the prediction error for previously unseen data. A visual inspection of the of the data distribution given by the train-validation split is given in Figure 5.4.

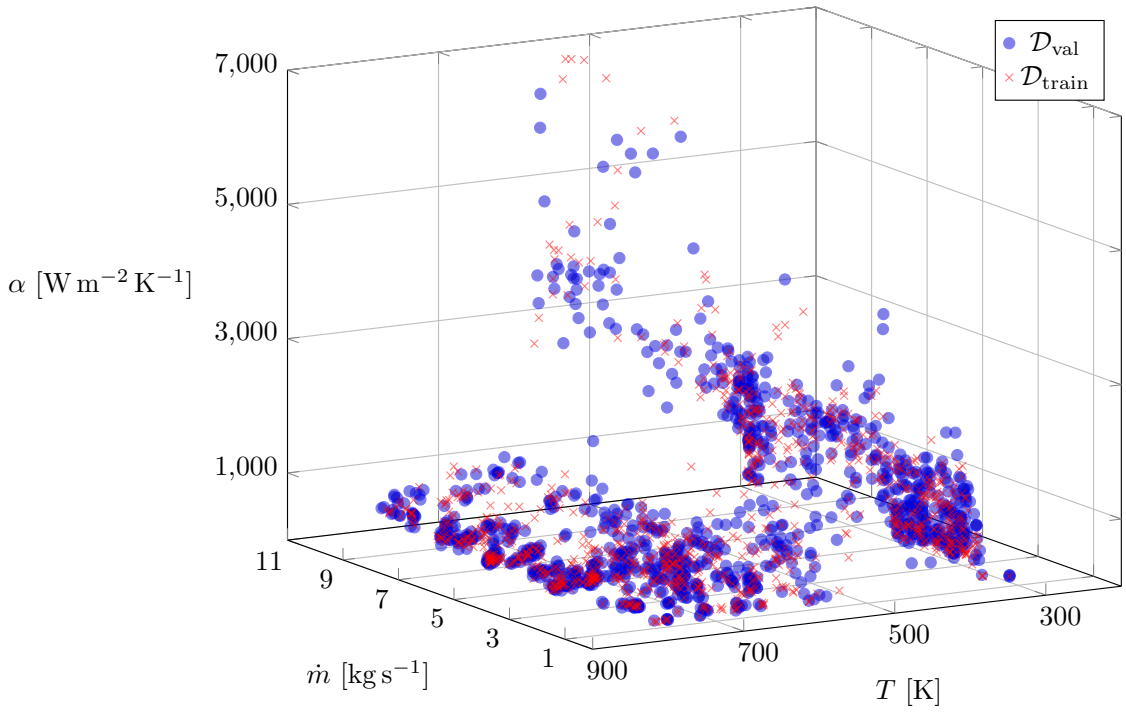

Figure 5.4: Validation set $\mathcal{D}_{\text{val}}$ and training set $\mathcal{D}_{\text{train}}$.

The mean squared errors MSE_{val} evaluated on the validations set $\mathcal{D}_{\text{val}}$ as well as the effective degree of freedom EDoF of the models are given in Table 5.1. According to these, the superior model is M4, i.e. the shape-constraint tensor-product B-spline with a monotonicity constraint in the mass flow dimension. The models M3, i.e. the tensor-product P-spline, and M6, i.e. the additive model using shape-constraints, perform nearly as well as M4 according to the mean squared error on the validation set but both possess a significantly higher EDoF which allows for some overfitting.

Model	$\text{MSE}_{\text{val}}[\text{W}^2/(\text{m}^4\,\text{K}^2)]$	EDoF
M1	$3.22 \cdot 10^5$	108.02
M2	$3.27 \cdot 10^5$	30.5
M3	$2.14 \cdot 10^5$	120.04
M4	$2.05 \cdot 10^5$	41.72
M5	$2.25 \cdot 10^5$	192.33
M6	$2.13 \cdot 10^5$	82.68

Table 5.1: Mean squared errors on the validation set $\mathcal{D}_{\text{val}}$ and effective degree of freedom EDoF of the models.

The predictions for model M4 are shown in Figure 5.5. The peak behavior in the temperature dimension is clearly visible, as well as an increasing trend within the mass flow dimension. Therefore, we conclude that model M4 is the superior model with regards to the domain knowledge and data fidelity.

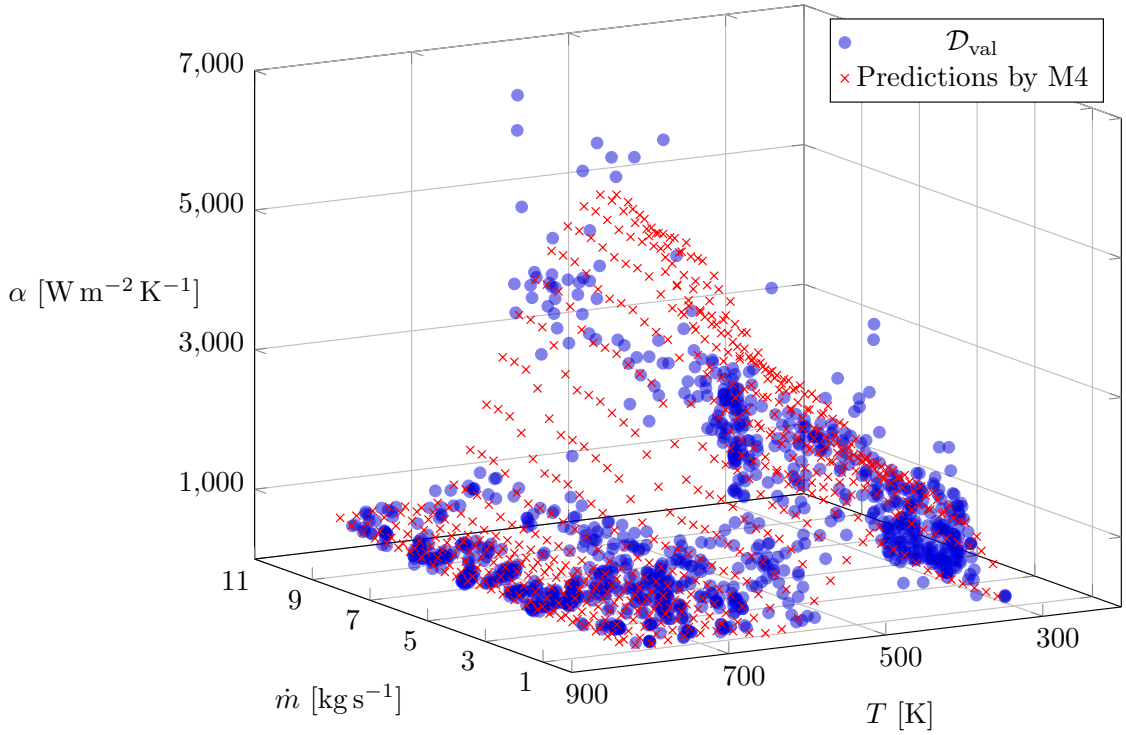

Figure 5.5: Validation set $\mathcal{D}_{\text{val}}$ and predictions by model M4.

The predictions for model M6 are shown in Figure 5.6. Here, the peak behavior can also be identified, but in a weaker fashion as for model M4 in Figure 5.5. We also obtain an increasing trend in the mass flow dimension.

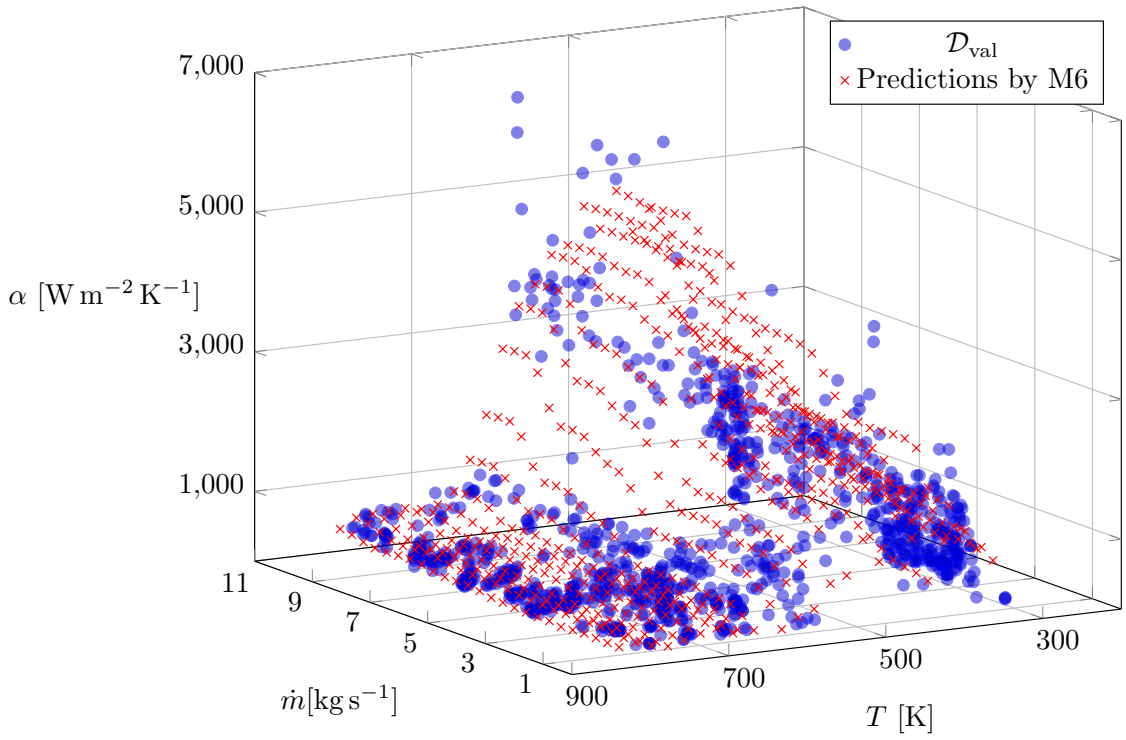

Figure 5.6: Validation set $\mathcal{D}_{\text{val}}$ and predictions by model M6.

The visual inspection of the predictions of model M3 in Figure 5.7 indicates that there is massive overfitting present. Neither smooth, nor the a priori known behavior (increasing in the mass flow dimension and a peak behavior in the temperature dimension) is identifiable.

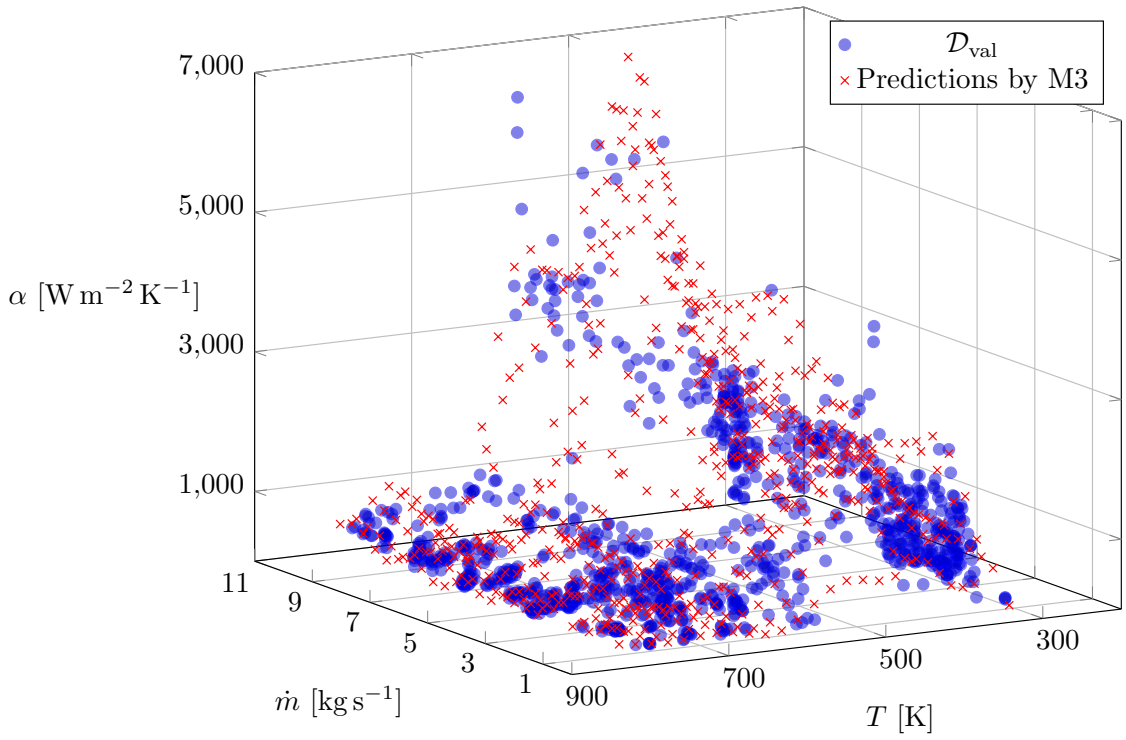

Figure 5.7: Validation set $\mathcal{D}_{\text{val}}$ and predictions by model M3.

We omit the visual presentation of the predictions by the models M1, M2 and M5, since for all of these the metrics, mean squared errors on the validation set $\mathcal{D}_{\text{val}}$ and the effective degree of freedom EDoF, indicate even worse models.

To summarized, we see that the incorporation of a priori domain knowledge improves the quality of the fit in all of the above models, e.g. M2 outperforms M1, M4 outperforms M3 and M6 outperforms M5 according to the mean squared errors on the validation set $\mathcal{D}_{\text{val}}$, respectively. Therefore, we conclude that the incorporation of a priori domain knowledge through shape-constraints improves the model quality for our real-world data example.

5.2 Hybrid Modeling and Control Example

As a second example, we consider a hybrid modeling and control concept applied to a hydraulic 4/3 two-stage directional control valve. A mathematical model and a model-based control concept was derived and validated by means of measurements on a test bench in a former project at the AIT Austrian Institute of Technology GmbH. Since the pilot valve is of overlapping design, the system has a large dead zone. This dead zone aggravates high precision positioning and places high demands on the controller design. A reduced-order model Σ was derived using singular perturbation techniques. It can be formulated as a Hammerstein system which consists of a linear system and a static input nonlinearity. Here, the input is given by the voltage u and outputs are the main valve

piston position s_h and the current i_p, respectively.

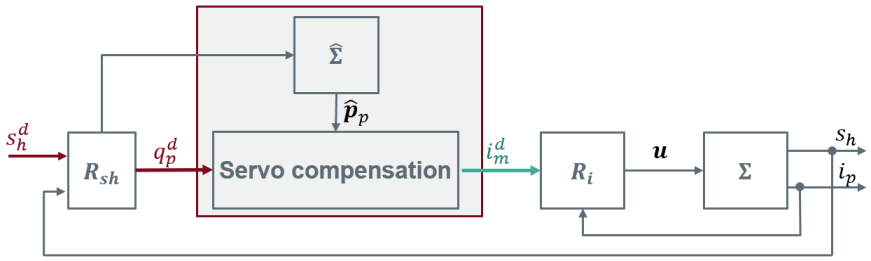

Figure 5.8: Block diagram of the control concept.

The proposed (cascaded) control concept, which is depicted in Figure 5.8, consists of a current controller R_i, a compensation (marked in red) and a position controller R_{sh} for the main valve piston position. The compensation is composed of a pressure observer $\hat{\Sigma}$ and a servo compensation, and can be described by a function of the form

$$i_m^d = f(q_p^d, s_h^d), \tag{5.2}$$

where s_h^d denotes the desired main valve piston position and q_p^d is the desired differential volume flow. The function (5.2) represents the most complex part (in terms of the amount of operations and the number of parameters) of the designed control concept. Therefore, we aim to replace it by a function approximation.

The data $\mathcal{D}$ consisting of 1000 points is generated using a validated first-principle model and is visualized in Figure 5.9. The figure shows two distinct model regions with a discontinuity at the desired main valve piston position $s_h^d = 0$ m. The main challenge here is to model the discontinuity in a smooth way without introducing modeling artifacts, i.e. over- or undershooting behavior. We know beforehand that the differential current i_m^d is monotonic increasing with the desired differential volume flow q_p^d. We further constrain the models to be monotonic increasing with the desired main valve piston position s_h^d to enforce a smooth transition at the discontinuity at $s_h^d = 0$ m.

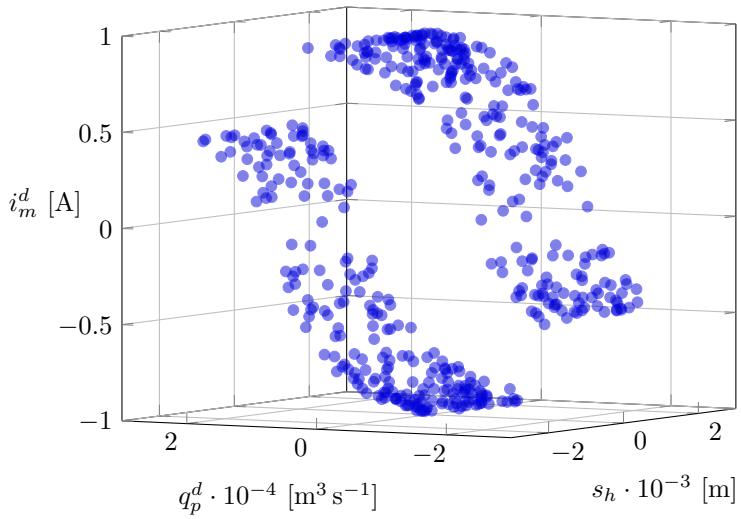

Figure 5.9: Data Situation

We will now use various models based on B-spline bases, tensor-product B-spline bases and their shape-constraint alternatives to recover a model for the differential current i_m^d. The following list, in which $s(x_i)$ denotes the use of a B-spline basis for input $x_i \in \{q_p^d, s_h^d\}$ and $t(q_p^d, s_h^d)$ denotes the use of a tensor-product B-spline basis, describes the models that are analyzed.

(i) M1 $= s(q_p^d) + s(s_h^d)$, without constraints

(ii) M2 $= s(q_p^d) + s(s_h^d)$, with monotonicity for $s(s_h^d)$

(iii) M3 $= t(q_p^d, s_h^d)$, without constraints

(iv) M4 $= t(q_p^d, s_h^d)$, with monotonicity for input $s(q_p^d)$ and $s(s_h^d)$

(v) M5 $= s(s_h^d) + t(q_p^d, s_h^d)$, as additive model without constraints

(vi) M6 $= s(s_h^d) + t(q_p^d, s_h^d)$, as additive model with the constraints defined in M2 and M4

Note that we use a smoothness penalty optimized via generalized cross-validation for each individual basis. The unconstraint models M1, M3 and M5 are therefore P-spline models rather than pure B-spline models. The shape-constraint models M2, M4 and M6 are SCP-spline models.

To fit the models listed above, we perform a randomized train-validation split on the data. The training set $\mathcal{D}_{\text{train}}$ then consists of 750 data points and a validation set $\mathcal{D}_{\text{val}}$ of 250 data points. We fit the models to the resulting training set and evaluate its performance by calculating the mean squared error on the validation set $\mathcal{D}_{\text{val}}$ and the effective degree of freedom of the models EDoF, as well as by visual inspection.

The mean squared errors MSE_{val} evaluated on the validations set $\mathcal{D}_{\text{val}}$ are given in Table 5.2 as well as the effective degree of freedom of the models as a measure of model complexity.

Model	$\text{MSE}_{\text{val}}[\text{A}^2]$	EDoF
M1	$3.45 \cdot 10^{-2}$	20.49
M2	$3.46 \cdot 10^{-2}$	15.48
M3	$5.83 \cdot 10^{-3}$	262.33
M4	$5.08 \cdot 10^{-3}$	57.22
M5	$5.7 \cdot 10^{-3}$	265.21
M6	$4.72 \cdot 10^{-3}$	53.59

Table 5.2: Mean squared errors on the validation set $\mathcal{D}_{\text{val}}$ and the effective degree of freedom EDoF of the models.

The purely B-spline based models M1 and M2 show low degrees of freedom but higher values of the mean squared error on the validation set $\mathcal{D}_{\text{val}}$. Therefore, they lack some flexibility to model the data accurately. The predictions of these models are visualized in Figure 5.10 and Figure 5.11. Note the left plot is the projection of the three dimensional data onto the s_h^d axis for better visualization of the model predictions. The purely tensor-product B-spline models M3 and M4 have a significantly lower mean squared error on the validation set $\mathcal{D}_{\text{val}}$. Both models are visualized in Figure 5.12 and Figure 5.13. With regards to the effective degree of freedom, we see that the regularization through the shape-constraints clearly decreases the model flexibility which leads to a better generalization behavior seen by the lower MSE_{val} of M4 compared to M3. This can also be seen in the left plot of Figure 5.12. Note, for example, the prediction point (red) at $q_p^d \approx 0.7 \cdot 10^{-4}$ m^3s^{-1}. Such a prediction is only possible for a very wiggly and non-smooth function. The additive models using B-splines and tensor-product B-splines, i.e. M5 and M6, both show low mean squared errors on the validation set MSE_{val}. We again see the effect of the shape-constraint in the effective degree of freedom as well as in the visualizations in Figure 5.14 and Figure 5.15.

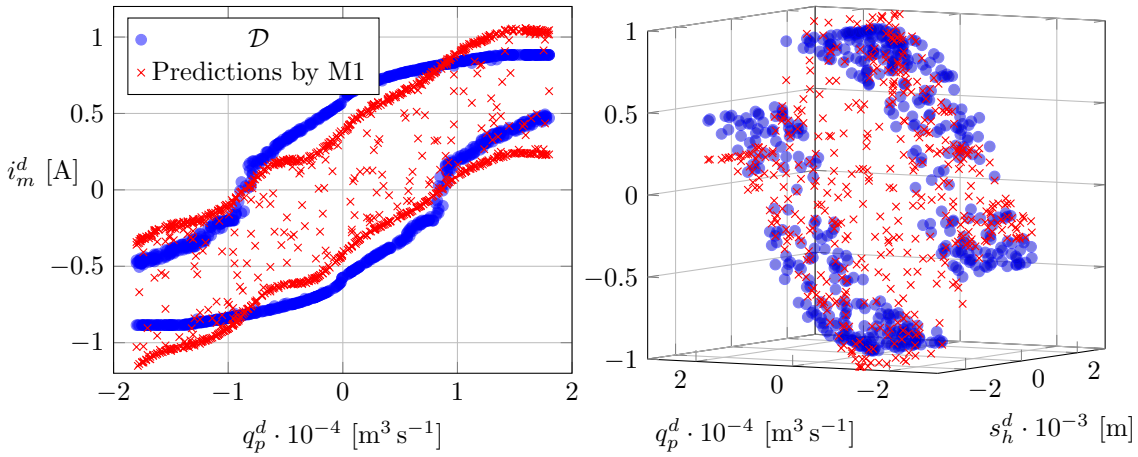

Figure 5.10: Data $\mathcal{D}$ and predictions by model M1.

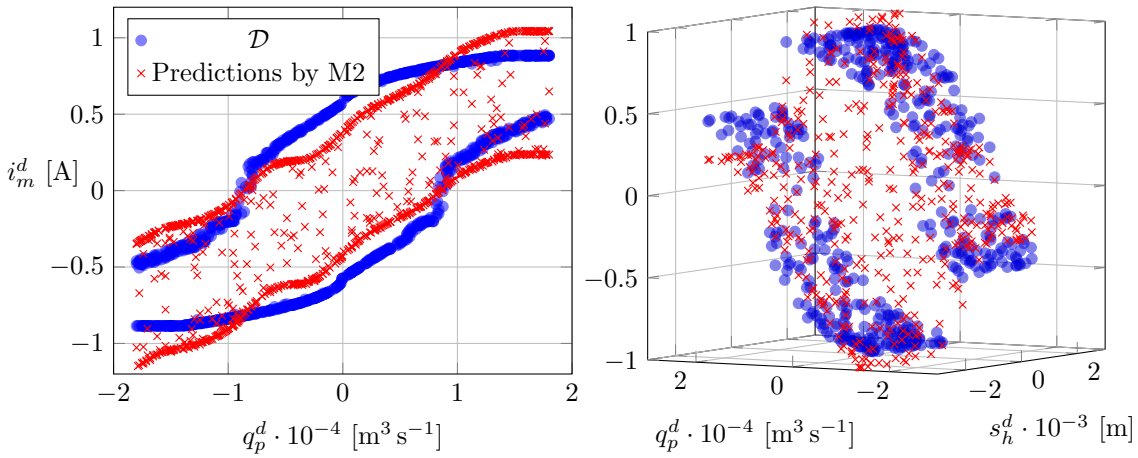

Figure 5.11: Data $\mathcal{D}$ and predictions by model M2.

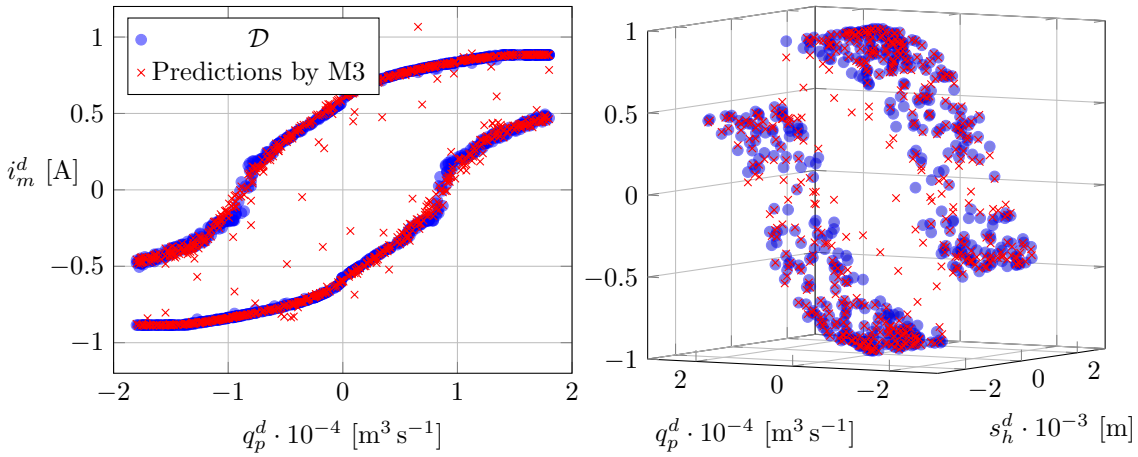

Figure 5.12: Data $\mathcal{D}$ and predictions by model M3.

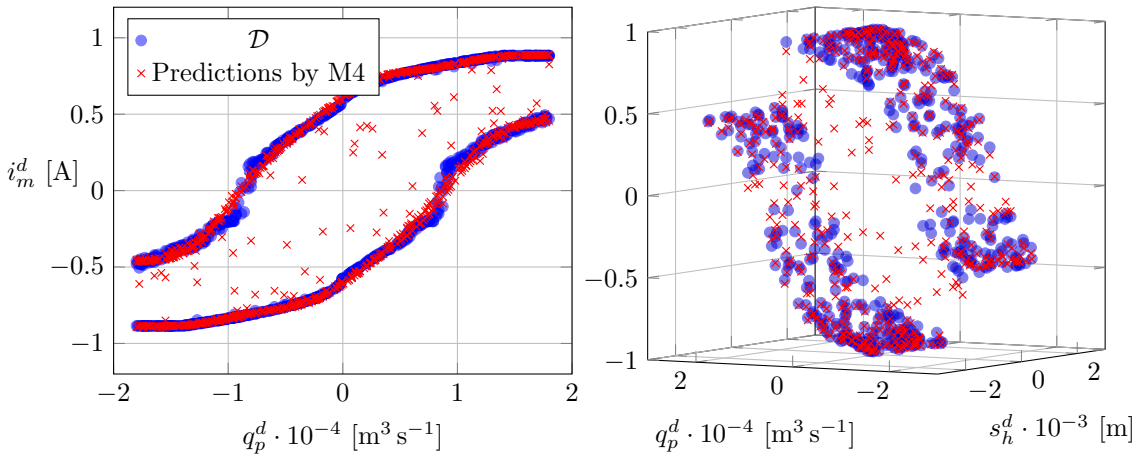

Figure 5.13: Data $\mathcal{D}$ and predictions by model M4.

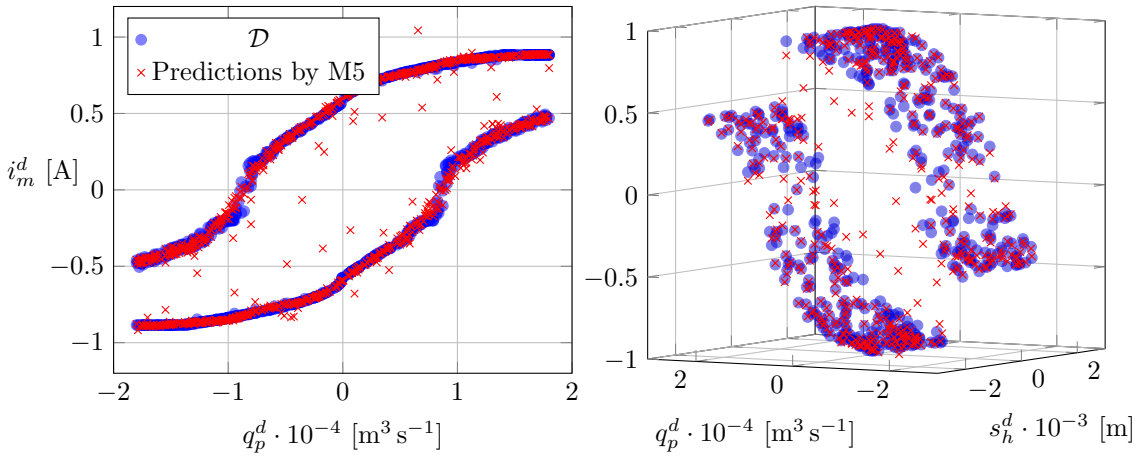

Figure 5.14: Data $\mathcal{D}$ and predictions by model M5.

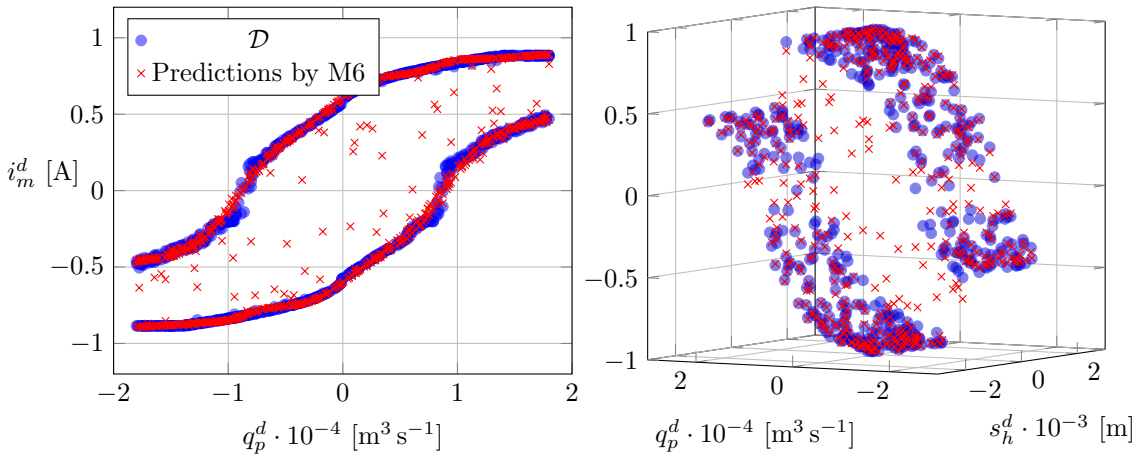

Figure 5.15: Data $\mathcal{D}$ and predictions by model M6.

To summarize, we see that the incorporation of a priori domain knowledge improves the quality of the fit for this example too. The constrained models show lower mean squared errors on the validation data as well as lower effective degrees of freedom. The superior model is clearly M6, since it has a significantly lower MSE_{val} as well as EDoF.

6 Summary and Outlook

The main goal of this thesis was to find a way to include a priori domain knowledge into the fitting process of a data-driven modeling approach. An overview of the related literature regarding the topics of linear models, model selection and B-splines was given in Chapter 2. In Chapter 4, we have presented an algorithm to incorporate a priori domain knowledge in the fitting process using shape-constraint P-splines and additive models. The performance of the developed algorithm has been demonstrated by artificial problems, see Chapter 5, and two real-world examples, see Chapter 5.

With regards to future work, it would be interesting to create an algorithm that automatically finds the best possible combination of B-splines and tensor-product B-splines for a multi-dimensional problem with some a priori domain knowledge. For example, using 3 inputs, there are various combinations possible to create an additive model, e.g. a B-spline for dimension 1 and a tensor-product B-spline for dimension 2 and 3. An algorithm that automatically chooses the optimal combination with respect to some predefined criterion would help to enhance the usability of this approach. Another interesting aspect is to investigate further possible constraints that can be implemented using mapping and weighting matrices in the penalized least squares approach. An example of interest may be some kind of multi-peak constraint for B-splines. Other open topics are listed below.

- Enforce a priori domain knowledge only on specific parts of the input space.
- Improve the available constraint, e.g. peak or valley constraint, to enhance robustness against noise.
- Extension of the available constraints for tensor-product B-splines.
- Runtime optimizations with regards to the number of B-splines for time critical applications.
- Extension of the algorithm to online learning.

A Appendix

A.1 Kronecker product

The Kronecker product $\mathbf{A} \otimes \mathbf{B}$ of the $n \times p$-matrix $\mathbf{A}$ and the $r \times q$-matrix $\mathbf{B}$ is defined as $nr \times pq$-matrix

$$\mathbf{C} = \mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \dots & a_{1p}\mathbf{B} \\ \vdots & & \vdots \\ a_{n1}\mathbf{B} & \dots & a_{np}\mathbf{B} \end{bmatrix}. \tag{A.1}$$

As example, we define $\mathbf{A} \in \mathbb{R}^{3\times 2}$ and $\mathbf{B} \in \mathbb{R}^{2\times 2}$ as

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}, \tag{A.2}$$

resulting in the matrix $\mathbf{C} \in \mathbb{R}^{6\times 4}$ given by

$$\mathbf{C} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 2 & 4 \\ 1 & 2 & 2 & 4 \\ 3 & 6 & 4 & 8 \\ 3 & 6 & 4 & 8 \\ 5 & 10 & 6 & 12 \\ 5 & 10 & 6 & 12 \end{bmatrix}. \tag{A.3}$$

A.2 Row-wise Kronecker product

Given the $n \times p$-matrix$\mathbf{A}$ and the $n \times q$-matrix $\mathbf{B}$, the row-wise Kronecker product is defined as $n \times pq$-matrix $\mathbf{C}$ given by

$$\mathbf{C} = \mathbf{A} \odot \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{b}_1 & \dots & a_{1p}\mathbf{b}_1 \\ \vdots & & \vdots \\ a_{n1}\mathbf{b}_n & \dots & a_{np}\mathbf{b}_n \end{bmatrix}, \tag{A.4}$$

where $\mathbf{b}_i$ denotes the i-th row of the matrix $\mathbf{B}$. As example, we define $\mathbf{A} \in \mathbb{R}^{3\times 3}$ and $\mathbf{B} \in \mathbb{R}^{3\times 2}$ as

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 1 & 2 \\ 1 & 2 \end{bmatrix}, \tag{A.5}$$

where $\mathbf{a}_i$ denotes the i-th row of the matrix $\mathbf{A}$ resulting in the matrix $\mathbf{C} \in \mathbb{R}^{3\times 6}$ given by

$$\mathbf{C} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 \\ \mathbf{a}_2 \otimes \mathbf{b}_2 \\ \mathbf{a}_3 \otimes \mathbf{b}_3 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 2 & 4 & 3 & 6 \\ 4 & 8 & 5 & 10 & 6 & 12 \\ 7 & 14 & 8 & 16 & 9 & 19 \end{bmatrix}. \tag{A.6}$$

The row-wise Kronecker product is also known as *face-splitting product* or as *transposed Khatri-Rao product* [34].

A.3 Derivation of the iterative scheme

The following derivation is take from [5]. To find an optimal solution for the penalized least squares objective function for shape-constraint P-splines given by

$$\mathrm{L}(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\boldsymbol{\beta}^{\mathrm{T}}\mathbf{K}\boldsymbol{\beta} + \lambda_c\boldsymbol{\beta}^{\mathrm{T}}\mathbf{K}_c\boldsymbol{\beta}, \tag{A.7}$$

we use a Newton-Raphson scheme. At each iteration i, the new estimate $\hat{\boldsymbol{\beta}}_{i+1}$ is computed such that

$$\mathbf{g}(\boldsymbol{\beta}_i) + \mathbf{H}(\boldsymbol{\beta}_i)(\boldsymbol{\beta}_{i+1} - \boldsymbol{\beta}_i) = 0, \tag{A.8}$$

with $\mathbf{g}$ being the gradient and $\mathbf{H}$ being the Hessian of the objective function $\mathrm{L}(\boldsymbol{\beta})$. The gradient of $\mathrm{L}(\boldsymbol{\beta})$ equals to

$$\mathbf{g}(\boldsymbol{\beta}) = -2\mathbf{X}^{\mathrm{T}}\mathbf{y} + 2\Big[\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{D}_2^{\mathrm{T}}\mathbf{D}_2 + \lambda_c\mathbf{D}_c^{\mathrm{T}}\mathbf{V}_c(\boldsymbol{\beta})\mathbf{D}_c\Big]\boldsymbol{\beta} + \lambda_c\boldsymbol{\beta}^{\mathrm{T}}\mathbf{D}_c^{\mathrm{T}}\frac{\partial\mathbf{V}_c(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}}\mathbf{D}_c\boldsymbol{\beta}, \tag{A.9}$$

which can be simplified, see [5], to

$$\mathbf{g}(\boldsymbol{\beta}) = -2\mathbf{X}^{\mathrm{T}}\mathbf{y} + 2\Big[\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{D}_2^{\mathrm{T}}\mathbf{D}_2 + \lambda_c\mathbf{D}_c^{\mathrm{T}}\mathbf{V}_c(\boldsymbol{\beta})\mathbf{D}_c\Big]\boldsymbol{\beta}. \tag{A.10}$$

The gradient is therefore a piecewise linear function of $\boldsymbol{\beta}$ since $\mathbf{V}(\boldsymbol{\beta})$ is a diagonal matrix with 0s and 1s as diagonal elements, see (A.10). This implies that the Hessian is a step function of $\boldsymbol{\beta}$ and equal to

$$\mathbf{H}(\boldsymbol{\beta}) = 2\mathbf{X}^{\mathrm{T}}\mathbf{X} + 2\lambda\mathbf{D}_2^{\mathrm{T}}\mathbf{D}_2 + 2\lambda_c\mathbf{D}_c^{\mathrm{T}}\mathbf{V}_c(\boldsymbol{\beta})\mathbf{D}_c + 2\lambda_c\mathbf{D}_c^{\mathrm{T}}\frac{\partial\mathbf{V}_c(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}}\mathbf{D}_c\boldsymbol{\beta}. \tag{A.11}$$

The last part can be neglected again, see [5]. Substituting (A.10) and (A.11) into (A.8) leads to

$$\begin{aligned} 0 = & -2\mathbf{X}^{\mathrm{T}}\mathbf{y} \\ & + 2\Big[\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{D}_2^{\mathrm{T}}\mathbf{D}_2 + \lambda_c\mathbf{D}_c^{\mathrm{T}}\mathbf{V}_c(\boldsymbol{\beta}_i)\mathbf{D}_c\Big]\boldsymbol{\beta}_i \\ & + 2\Big[\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{D}_2^{\mathrm{T}}\mathbf{D}_2 + \lambda_c\mathbf{D}_c^{\mathrm{T}}\mathbf{V}_c(\boldsymbol{\beta}_i)\mathbf{D}_c\Big](\boldsymbol{\beta}_{i+1} - \boldsymbol{\beta}_i), \end{aligned} \tag{A.12}$$

which can be reformulated as

$$-\mathbf{X}^{\mathrm{T}}\mathbf{y} + \Big[\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{D}_2^{\mathrm{T}}\mathbf{D}_2 + \lambda_c\mathbf{D}_c^{\mathrm{T}}\mathbf{V}_c(\boldsymbol{\beta}_i)\mathbf{D}_c\Big]\boldsymbol{\beta}_{i+1} = 0, \tag{A.13}$$

and, hence, we obtain

$$\boldsymbol{\beta}_{i+1} = \Big[\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{D}_2^{\mathrm{T}}\mathbf{D}_2 + \lambda_c\mathbf{D}_c^{\mathrm{T}}\mathbf{V}_c(\boldsymbol{\beta}_i)\mathbf{D}_c\Big]^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}. \tag{A.14}$$

Bibliography

[1] W. R. Ashby, *An Introduction to Cybernetics*. London, UK : Chapman & Hall Ltd, 1957.

[2] F. K. Došilović, "Explainable artificial intelligence: A survey," in *International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2018, pp. 210–215.

[3] L. Fahrmeir, T. Kneib, S. Lang, and B. Marx, *Regression*. Berlin-Heidelberg, GER : Springer, 2013.

[4] B. Hofner, "Monotonicity-constrained species distribution models," *Ecology*, vol. 92, no. 10, pp. 1895–1901, 2011.

[5] K. Bollaerts, "Simple and multiple p-splines regression with shape constraints," *British Journal of Mathematical and Statistical Psychology*, vol. 59, no. 2, pp. 451–469, 2006.

[6] O. Nelles, *Nonlinear System Identification*. Berlin-Heidelberg, GER : Springer, 2001.

[7] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. New York, USA: Springer, 2001.

[8] J. H. Friedman, "Multivariate adaptive regression splines," *The Annals of Statistics*, vol. 19, no. 1, pp. 1–67, 1991.

[9] C. De Boor, *A practical Guide to Splines*. New York, USA : Springer, 1978.

[10] D. Bergmann, "Gaußprozessregression zur modellierung zeitvarianter systeme," *at-Automatisierungstechnik*, vol. 67, no. 8, pp. 637–647, 2019.

[11] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Massachusetts, USA : The MIT Press, 2006.

[12] C. Bishop, *Pattern Recognition and Machine Learning*. New York, USA : Springer, 2006.

[13] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*. Massachusetts, USA : The MIT Press, 2016, http://www.deeplearningbook.org.

[14] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, 1989.

[15] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[16] Y. S. Abu-Mostafa, "Learning from hints in neural networks," *Journal of Complexity*, vol. 6, no. 2, pp. 192–198, 1990.

[17] J. Sill, "Monotonicity hints," in *Advances in Neural Information Processing Systems 9*, 1996, pp. 634–640.

[18] E. Garcia, "Lattice regression," in *Advances in Neural Information Processing Systems 22*, 2009, pp. 594–602.

[19] K. Canini, "Fast and flexible monotonic functions with ensembles of lattices," in *Advances in Neural Information Processing Systems 30*, 2016, pp. 2927–2927.

[20] M. Gupta, "Monotonic calibrated interpolated look-up tables," *Journal of Machine Learning Research*, vol. 17, no. 109, pp. 1–47, 2016.

[21] S. You, "Deep lattice networks and partial monotonic functions," in *Advances in Neural Information Processing Systems 31*, 2017, pp. 2985–2993.

[22] S. N. Wood, *Generalized Additive Models*. CRC Press, 2017.

[23] D. G. Luenberger and Y. Ye, *Linear and Nonlinear Programming*. Springer International Publishing, 2016.

[24] V. Blobel, *Statistische und numerische Methoden der Datenanalyse*. Wiesbaden, GER : Vieweg+Teubner Verlag, 1998.

[25] G. H. Golub, "Generalized cross-validation as a method for choosing a good ridge parameter," *Technometrics*, vol. 21, no. 2, pp. 215–223, 1979.

[26] A. E. Hoerl, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[27] P. C. Hansen and D. P. O'Leary, "The use of the l-curve in the regularization of discrete ill-posed problems," *SIAM*, vol. 14, no. 6, pp. 1487–1503, 1993.

[28] P. C. Hansen, "Analysis of discrete ill-posed problems by means of the l-curve," *SIAM*, vol. 34, no. 4, pp. 561–580, 1992.

[29] R. L. Eubank, "Testing the goodness of fit of a linear model via nonparametric regression techniques," *Journal of the American Statistical Association*, vol. 85, no. 410, pp. 387–392, 1990.

[30] P. H. C. Eilers and B. D. Marx, "Flexible smoothing with b-splines and penalties," *Statistical Science*, vol. 11, pp. 89–102, 1996.

[31] F. O'Sullivan, "A statistical perspective on ill-posed inverse problems," *Statistical Science*, vol. 1, no. 4, pp. 502–518, 1986.

[32] F. Mayinger, *Strömung und Wärmeübergang in Gas-Flüssigkeits-Gemischen*. Wien, AUT : Springer, 1982.

[33] H. Baehr and S. K., *Heat and Mass Transfer*. Berlin-Heidelberg, GER : Springer, 2006, vol. 2.Auflage.

[34] V. I. Slyusar, "Analytical model of the digital antenna array on a basis of face-splitting matrixs product," in *Proceedings of International Conference on Antenna Theory and Techniques*, 1997, pp. 108–109.