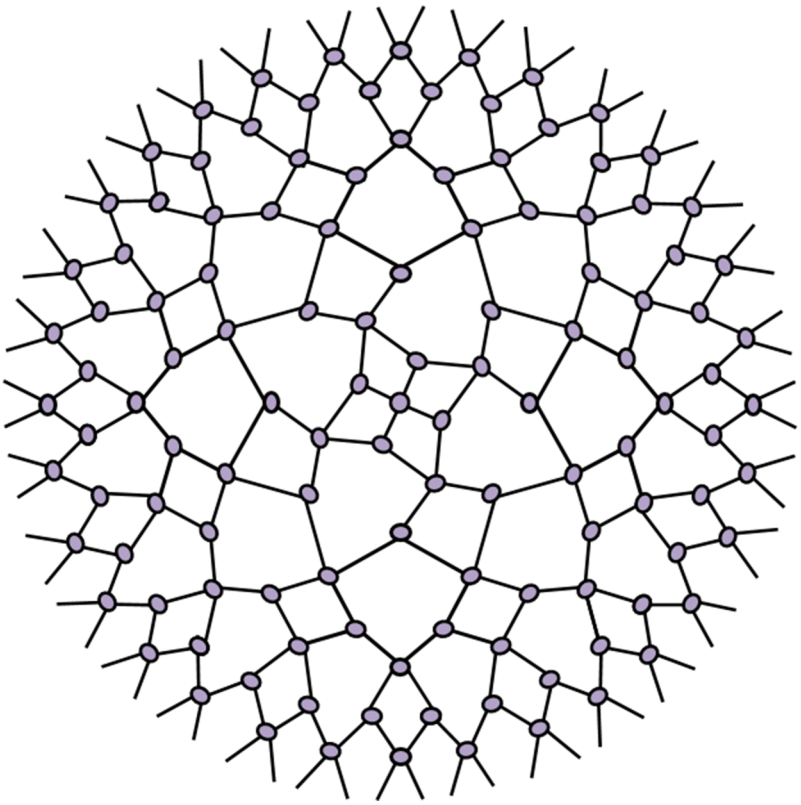
# Tensor Network B-splines

for high-dimensional function approximation

R. Karagöz

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

# Tensor Network B-splines
## for high-dimensional function approximation

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

R. Karagöz

April 21, 2020

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology


# Abstract

B-splines are basis functions for the spline function space and are extensively used in applications requiring function approximation. The generalization of B-splines to multiple dimensions is done through tensor products of their univariate basis functions. The number of basis functions and weights that define a multivariate B-spline surface, therefore, increase exponentially with the number of dimensions, i.e. B-splines suffer from the curse of dimensionality. Tensor network theory provides a mathematical framework to alleviate the curse of dimensionality of B-splines by representing the high-dimensional weight tensor as a low-rank approximation. This thesis presents the Tensor Network B-spline (TNBS) model, along with an optimization algorithm that allows the estimation of the exponentially large weight tensor directly from data, without ever needing to explicitly construct it. P-spline regularization is incorporated to induce additional smoothness and ensure the B-spline hypersurface generalizes well across the high-volume domain. The developed TNBS framework opens doors for the application of B-spline theory in high-dimensional function approximation. This thesis provides an overview of both B-spline and tensor network theory, then uses it to derive the TNBS model. We validate the effectiveness of the model through an application in black-box nonlinear system identification using a NARX approach. An open-source MATLAB implementation of TNBS is made available on GitHub. The work is concluded with some recommendations for further research on this topic.

# Table of Contents

# Acknowledgements

First, I wish to show my gratitude to my supervisor, Kim Batselier, for introducing me to the world of tensor networks. His guidance and detailed feedback have helped me in successfully completing a thesis and paper that I can look back at with pride. Also, special thanks to my friend Oguzhan Kaya for proofreading this work. I would like to thank my family for their encouragement and support throughout my educational career. Last but not least, I want to express my gratitude towards my wife who has stood by me through all my travails and absences. This journey would have been lonesome without her. Thank you.

Delft, University of Technology R. Karagöz
April 21, 2020

# Chapter 1

# Introduction

## 1-1 Thesis Motivation

Function approximation deals with the problem of reconstructing a complex or unknown function, being only provided with a finite set of data points sampled from this function. Figure 1-1 gives an example of a univariate function $g(x)$, which approximates an unknown function $f(x)$ from a set of data points of the form $\{(x, f(x) + \epsilon)\}$, where $\epsilon$ is the sampling error. The best approximation minimises the difference between the original function and the approximation according to some metric, e.g. $\|f(x) - g(x)\|$. Function approximations arises in many branches of science and is fundamental to applications such as pattern recognition, prediction and classification.

In 1885, Karl Weierstrass published a proof of a theorem, nowadays known as the Stone-Weierstrass theorem, which says that any continuous function on the closed interval $[a, b]$ may be arbitrarily closely approximated by polynomials [1]. Using polynomials to approximate more complex functions is a basic building block of many numerical techniques [2]. Polynomials are favorable because of their simple form, well-known properties, and moderate flexibility. They also easily generalize to multivariate functions by means of tensor products of their basis vectors. Polynomial models, therefore, have been widely used in many domains that involve function approximations, such as interpolation and curve fitting [3, 4]. In statistics, polynomial regression is the simplest method to find nonlinear relationships between variables [5]. In machine learning applications, polynomials are often used as a feature vector to map data to a higher dimensional feature space [6], or as a kernel function for algorithms such as support vector machines and Gaussian processes [7, 8]. In nonlinear system identification, models such as Volterra series, NARX and NARMAX use polynomial expansions to model the nonlinear interactions between the delayed input or output variables [9, 10]. There are, however, several disadvantages to using polynomials for function approximation:

- They are poor interpolators. High-degree polynomials are known to oscillate between data points in curve fitting and interpolation. In the special case of interpolating over equidistant interpolation points, this is known as Runge's phenomenon [11].

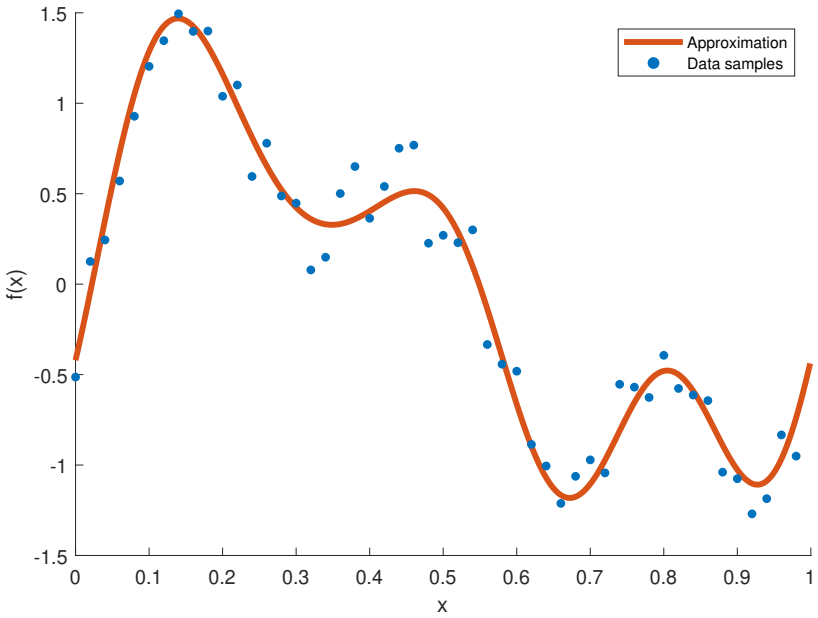



**Figure 1-1:** Function approximation example.

- They are poor extrapolators. Outside the range of data, the fit of the polynomial model often rapidly deteriorates [12].
- The large exponents in high-degree polynomials may cause numerical issues [13].
- The number of monomial terms required to represent multivariate functions grows exponentially with the number of dimensions $d$, i.e. $\mathcal{O}(k^d)$. Storing, fitting and analyzing polynomial models in high-dimensional spaces becomes impractical due to the increase in memory and computational requirements. This phenomenon is referred to as the curse of dimensionality [14].

To address some of the drawbacks of polynomials, while retaining their approximative properties, splines are often regarded as a favorable alternative. Splines refer to a wide class of functions that are defined piece-wise by polynomials. Like polynomials, they are used in applications requiring the interpolation or smoothing of one- or multidimensional data [15] and play an important role in applied mathematics, numerical analysis, geometric modeling, and many other areas due to their useful properties. Many representations of splines exist, including Bézier curves, Hermite splines and Box splines, but it is particularly attractive to represent splines as a linear combination of basis functions [16]. This is the B-spline representation, which has been recognized as an extremely powerful tool in both theory and application.

B-splines have had an impact in countless applications of signal processing [17], control [18, 19], computer graphics [20], computer-aided design [21], partial differential equations [22], finite element analysis [23], medical imaging [24] and atomic and molecular physics [25]. They are used for supervised learning [26], system identification [27–31] and control [32–35]. They have been applied in machine learning as features for clustering methods [36], as kernels for support vector machines (SVMs) [37] or as activation functions for neural networks.

B-splines also generalize to multiple dimensions through tensor products of their univariate basis vectors, and therefore still suffer from the curse of dimensionality. This is a major limitation for many applications of multivariate B-splines with high dimensional inputs [38]. Previous attempts to avoid this limitation involve input feature selection, dimensionality reduction, clustering, generative additive models, Gabor-Kolmogorov polynomial expansions, ANOVA decompositions, and networked or hierarchical structures [39–41]. These methods, however, do not fully overcome the curse of dimensionality, but mostly try to limit the number of input dimensions or basis functions, or sometimes discard many of the, possibly significant, interaction terms between input variables. The most effective method, hierarchical B-splines, relies on sparse grids [42], which reduces the storage complexity from $\mathcal{O}(k^d)$ to $\mathcal{O}(k \log(k)^{d-1})$ elements [43], which is still exponential in the number of dimensions.

## 1-2 Research Objective

A recently emerging way to alleviate the curse of dimensionality is through a concept termed tensor networks. Originally developed in the context of quantum physics, tensor networks efficiently represent high-dimensional tensors as a set of sparsely interconnected low order tensors. The tensor train format, for example, has a storage complexity that grows linearly in the number of dimensions. Furthermore, efficient and numerically reliable algorithms have been developed for the decomposition and reconstruction of tensors, and for operations on them in tensor network formats [44]. Combined with tensor algebra, tensor networks can thus greatly decrease the computational complexity of many applications. They have, for example, been successfully applied in machine learning to compress various types of neural networks [45–48], perform large scale dimensionality reduction [49], and to build high-dimensional polynomial classifiers [50], tensor-based SVMs [51] and other forms of supervised learning methods [52, 53]. Other applications of tensor networks include Kalman filters [54], system identification [55, 56], tensor completion or interpolation [57–60], and Gaussian processes [61].

As B-spline surfaces are multilinear in their parameters, they naturally admit to a tensor network representation. Algorithms for solving linear problems in the tensor network format make it possible to fit multivariate B-spline surfaces onto high-dimensional data without ever constructing the exponentially large weight tensor. Instead, the tensor network representation of the weight tensor can be found directly, thereby overcoming the curse of dimensionality. A similar approach for B-splines has been conducted for the two-dimensional case [62], using the Singular Value Decomposition (SVD) and Adaptive Cross Approximation algorithm, but a generalization to higher dimensions using tensor decompositions lacks in literature. A tensor network representation of multivariate B-splines, which we call the Tensor Network B-splines (TNBS) model, allows the application of B-splines in multivariate function approximation without their computational burdens for higher dimensions. Our research objective is to develop a TNBS framework and evaluate it through numerical experiments.

## 1-3 Thesis Contributions

This thesis gives a solid basis in both B-spline and tensor network theory, then introduces the TNBS model and the optimization algorithm. This model can serve as a general framework for the approximation of continuous high-dimensional functions with B-splines. An open-source MATLAB implementation of TNBS is made available on GitHub.

## 1-4 Thesis Outline

This thesis is structured as follows: Chapter 2 introduces tensors and tensor network theory. The choice for the tensor train decomposition is elaborated and a comparison is made between various optimization methods in the tensor train format. Chapter 3 covers the relevant B-spline theory and illustrates how B-splines can be regularized and applied to function approximation. Chapter 4 introduces the TNBS model, generalizes regularization to the tensor network format and presents an optimization algorithm for fitting the TNBS to data. The proposed methods are validated through numerical experiments. Chapter 5 contains a preprint of our paper which is submitted to the Automatica journal. In this paper, the TNBS model is applied to nonlinear system identification and the results on a benchmark nonlinear system are presented. Chapter 6 concludes this thesis with a summary and lists some recommendations and future work.

# Chapter 2

# Introduction to Tensors

This chapters provides the basic terminology and definitions for understanding tensors and tensor network theory. It treats the notations and operations that will be essential throughout the rest of the thesis. We introduce the tensor train decomposition which will be used as the main structure of our tensor network model. Most of the introduced definitions are based on the papers [44, 63–65].

## 2-1 Basics

A tensor can be represented as a multidimensional array of numerical values, i.e.:

$$\mathcal{A} \in \mathbb{R}^{k_1 \times k_2 \times \cdots \times k_d} \tag{2-1}$$

The order $d$ of the tensor is the number of dimensions or indices. Tensors can thus be seen as generalizations of vectors and matrices. A vector is a first order tensor, whereas a matrix is a tensor of order two. A single element of a tensor will be denoted by indices in the subscripts, e.g.:

$$a = \mathcal{A}_{i_1, i_2, \ldots, i_d}. \tag{2-2}$$

The letters $k_p$, where $p \in \{1, 2, \ldots, d\}$, indicate the size of each dimension, such that $i_p \in \{1, 2, \ldots, k_p\}$. A convenient way of expressing tensors and their operations is through the graphical notation introduced by Roger Penrose in 1972 [63]. Figure 2-1 shows the representation of a scalar, vector, matrix and third order tensor using this notation. Every node represents a tensor. The edges represent the indices of the tensor and the number of edges therefore corresponds to its order. A node with no edges is a zero order tensor, i.e. a scalar. The direction of the edges are irrelevant.

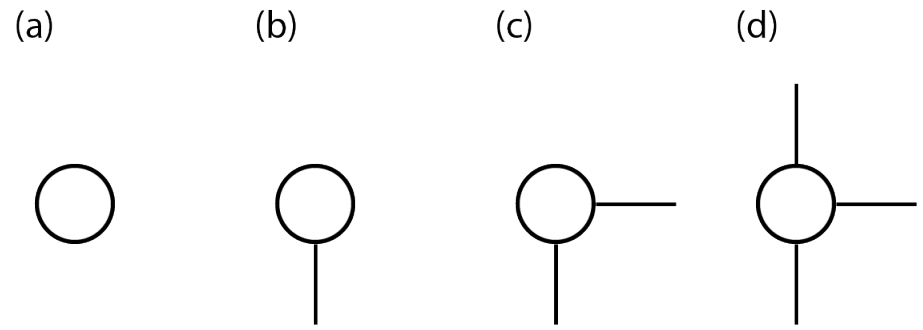



**Figure 2-1:** Graphical notation of a (a) scalar, (b) vector, (c) matrix and (d) third order tensor.

In this work, scalars will be denoted by lowercase letters ($a$), vectors will be denoted by bold lowercase letters ($\boldsymbol{a}$), matrices will be denoted by bold uppercase letters ($\boldsymbol{A}$) and higher order tensors will be denoted by bold calligraphic letters ($\mathcal{A}$). The fibers of a tensor are the generalization of rows and columns of matrices. A mode-$p$ fiber is a vector defined by fixing all indices but the $p$th. For example, the elements of the first mode-3 fiber of the tensor $\mathcal{A} \in \mathbb{R}^{k_1 \times k_2 \times k_3}$ are given by:

$$\boldsymbol{a}_{i_3} = \mathcal{A}_{1,1,i_3}. \tag{2-3}$$

A tensor $\mathcal{T} \in \mathbb{R}^{k_1 \times k_2 \times \cdots \times k_d}$ is of rank one if it can be decomposed into the outer product of $d$ vectors $\boldsymbol{b}^{(p)} \in \mathbb{R}^{k_p}$, e.g:

$$\mathcal{T} = \boldsymbol{b}^{(1)} \circ \boldsymbol{b}^{(2)} \circ \cdots \circ \boldsymbol{b}^{(d)},$$

where $\circ$ denotes the outer product operation. The canonical rank $r$ of a tensor $\mathcal{A}$ is defined as the minimal number of rank one tensors needed to produce $\mathcal{A}$ as their sum.

The vectorization of a tensor $\mathcal{A} \in \mathbb{R}^{k_1 \times k_2 \times \cdots \times k_d}$ is the reordering of its elements into a column vector, denoted by $\boldsymbol{vec}(\mathcal{A}) = \boldsymbol{a} \in \mathbb{R}^{k_1 k_2 \cdots k_d}$. The elements of $\boldsymbol{a}$ are given by:

$$\boldsymbol{a}_{i_1+(i_2-1)k_1+\ldots+(i_d-1)k_1k_2\ldots k_{d-1}} = \mathcal{A}_{i_1,i_2,\ldots,i_d}. \tag{2-4}$$

This can be seen as stacking the mode-1 fibers on top of each other. Vectorization of a third-order tensor is illustrated graphically in 2-2.

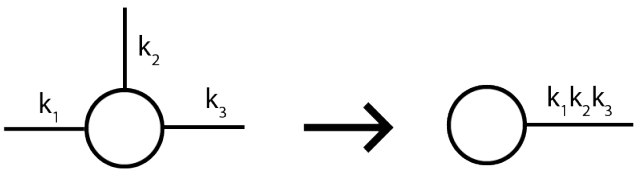



**Figure 2-2:** Vectorization of a third order tensor.

Matricization or unfolding is the reordering the elements of a tensor into a matrix. The mode-$p$ matricification of $\mathcal{A} \in \mathbb{R}^{k_1 \times k_2 \times \cdots \times k_d}$ is given by rearranging all mode-$p$ fibers as column vectors to form a matrix $\boldsymbol{A} \in \mathbb{R}^{k_p \times k_1 k_2 \cdots k_{p-1} k_{p+1} \cdots k_d}$.

More general reshaping operations on tensors can be denoted by adopting the reshape operator from MATLAB. For example, grouping the first two and last two indices of a tensor

$\mathcal{A} \in \mathbb{R}^{k_1 \times k_2 \times k_3 \times k_4}$ results in a tensor $\boldsymbol{reshape}(\mathcal{A}, [k_1 k_2, k_3 k_4]) = \mathcal{B} \in \mathbb{R}^{k_1 k_2 \times k_3 k_4}$. This is illustrated graphically in 2-3.

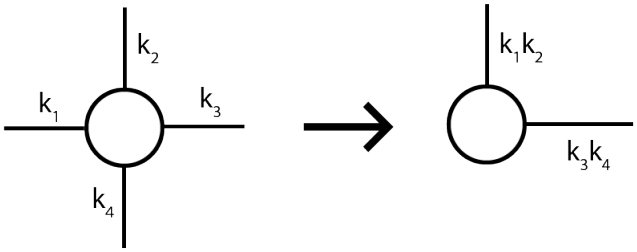



**Figure 2-3:** Reshaping of a fourth order tensor.

## 2-2 Tensor Operations

Two essential operations in tensor algebra are the tensor contraction and the tensor product. The contraction operation is the summing of elements over equal-sized indices and generalizes both of the trace and product operations on matrices. Given the tensors $\mathcal{A} \in \mathbb{R}^{k_1 \times k_2 \times k_3}$ and $\mathcal{B} \in \mathbb{R}^{k_3 \times k_4 \times k_5}$, contracting the index $i_3$ results in a tensor $\mathcal{A} \times_3^1 \mathcal{B} = \mathcal{C} \in \mathbb{R}^{k_1 \times k_2 \times k_4 \times k_5}$ whose elements are given by:

$$\mathcal{C}_{i_1,i_2,i_4,i_5} = \sum_{i_3} \mathcal{A}_{i_1,i_2,i_3} \, \mathcal{B}_{i_3,i_4,i_5}. \tag{2-5}$$

Contraction thereby reduces the total order of the tensors by the number of contracted indices. Contraction is indicated by the left-associative $\binom{q}{p}$-mode product operator [66], where $p$ and $q$ indicate the position of the indices of the first and second tensor respectively. In the graphical notation, contraction is indicated by connecting edges of corresponding indices. This is illustrated for Eq. (2-5) in Figure 2-4.

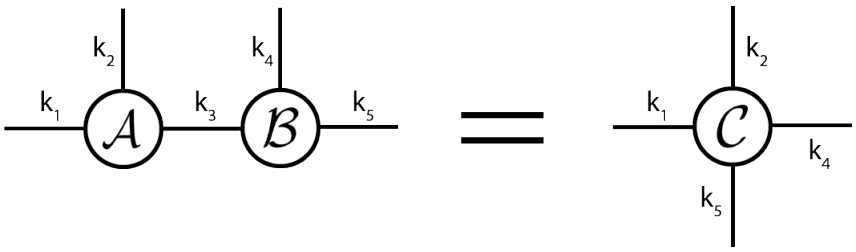



**Figure 2-4:** Tensor contraction in the graphical notation.

The matrix product operation can be regarded as a special case of tensor contraction, performed on a pair of second order tensors. Given two matrices $\boldsymbol{A} \in \mathbb{R}^{k_1 \times k_2}$ and $\boldsymbol{B} \in \mathbb{R}^{k_2 \times k_3}$, the matrix product $\boldsymbol{AB}$ is simply the contraction of the second index of $\boldsymbol{A}$ with the first index of $\boldsymbol{B}$, which results in a matrix $\boldsymbol{A} \times_2^1 \boldsymbol{B} = \boldsymbol{C} \in \mathbb{R}^{k_1 \times k_3}$ whose elements are given by:

$$\boldsymbol{C}_{i_1,i_3} = \sum_{i_2} \boldsymbol{A}_{i_1,i_2} \, \boldsymbol{B}_{i_2,i_3}. \tag{2-6}$$

The tensor product operation is a generalization of the outer product for vectors. Given two tensors $\mathcal{A} \in \mathbb{R}^{k_1 \times k_2 \times k_3}$ and $\mathcal{B} \in \mathbb{R}^{k_4 \times k_5}$, their tensor product $\mathcal{A} \circ \mathcal{B}$ results in a tensor $\mathcal{C} \in \mathbb{R}^{k_1 \times k_2 \times k_3 \times k_4 \times k_5}$ whose elements are given by:

$$\mathcal{C}_{k_1,k_2,k_3,k_4,k_5} = \mathcal{A}_{k_1,k_2,k_3}\, \mathcal{B}_{k_4,k_5}. \tag{2-7}$$

The tensor product operation can also be regarded as a special case of contraction, where the contracted indices have singleton dimensions. The tensor product operation is depicted by a dashed line connecting two nodes, as seen in Figure 2-5 for Eq. (2-7).

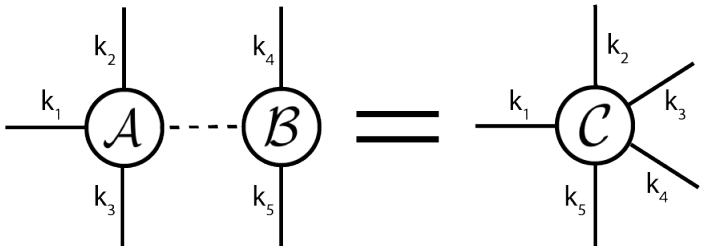



**Figure 2-5:** Tensor product in the graphical notation.

The inner product between two equally dimensioned tensors is the sum of their entry-wise product, resulting in a scalar. This is equivalent to contracting the tensor pair over all indices. Given two tensors $\mathcal{A} \in \mathbb{R}^{k_1 \times k_2 \times k_3}$ and $\mathcal{B} \in \mathbb{R}^{k_1 \times k_2 \times k_3}$, their inner product is given by:

$$\langle \mathcal{A}, \mathcal{B} \rangle = \sum_{k_1,k_2,k_3} \mathcal{A}_{k_1,k_2,k_3} \mathcal{B}_{k_1,k_2,k_3} = \mathcal{A} \times_{1,2,3}^{1,2,3} \mathcal{B} = \boldsymbol{vec}(\mathcal{A})^T \boldsymbol{vec}(\mathcal{B}). \tag{2-8}$$

This is illustrated in graphical notation in Figure 2-6.

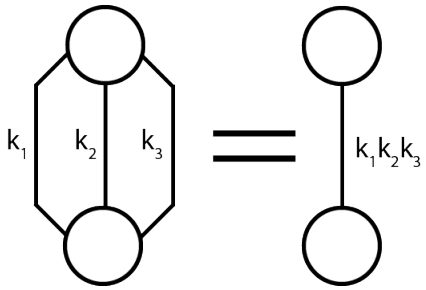



**Figure 2-6:** The inner product of two third-order tensors.

The Frobenius norm of a tensor is defined as the square root of the sum of squares of its entries and can be written as:

$$\|\mathcal{A}\|_2 = \sqrt{\langle \mathcal{A}, \mathcal{A} \rangle}. \tag{2-9}$$

Mervе

An important equation [55] that relates contraction of a $d$-dimensional tensor $\mathcal{A} \in \mathbb{R}^{k_1 \times k_2 \times \cdots \times k_d}$ with $d$ matrices $\boldsymbol{C}^{(p)} \in \mathbb{R}^{k_p \times k_{(p+d)}}$ to a linear operation is the following:

$$\boldsymbol{vec}\left(\mathcal{A} \times_1^2 \boldsymbol{C}^{(1)} \times_2^2 \cdots \times_d^2 \boldsymbol{C}^{(d)}\right) = \left(\boldsymbol{C}^{(d)} \otimes \cdots \otimes \boldsymbol{C}^{(1)}\right) \boldsymbol{vec}(\mathcal{A}) = \boldsymbol{C}\boldsymbol{a}. \tag{2-10}$$

where $\otimes$ denotes the Kronecker product. This is illustrated for a third-order tensor in Figure 2-7.

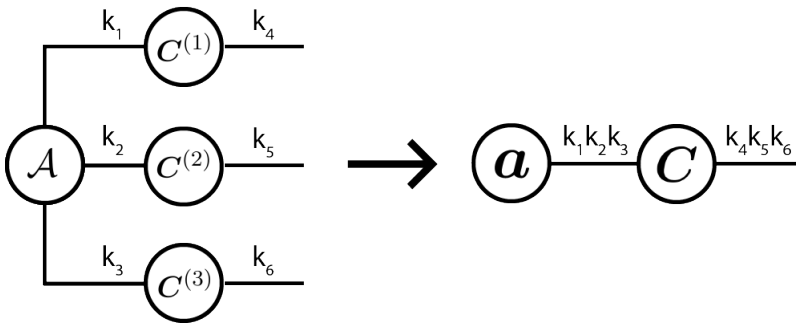



**Figure 2-7:** Conversion from multiple contractions to a linear operation.

## 2-3 Tensor Train Decomposition

Tensor algebra and tensor decompositions are gaining interest in areas of computational mathematics and numerical analysis with high-dimensional problems [67]. Tensors are a natural choice for representing data in many domains, but they suffer from the curse of dimensionality, i.e, the number of elements of a tensor (and therefore the amount of memory to store it and the computational complexity of tensor operations) grows exponentially with the number of dimensions, $\mathcal{O}(k^d)$. Tensor decompositions or tensor networks avoid the curse of dimensionality by accurately reproducing high-order tensors through a series of tensor contractions over low-order tensors [44]. Three of the most popular tensor decomposition methods are the canonical polyadic (CP) decomposition, Tucker decomposition, and Tensor Train (TT) decomposition. The storage complexity of the CP decomposition of a tensor $\mathcal{A} \in \mathbb{R}^{k \times k \times \cdots \times k}$ is $\mathcal{O}(kdr)$, which for high $d$ vastly reduces the storage complexity of the original tensor. There are, however, no straightforward algorithms to determine the canonical rank $r$ of a tensor and finding a low-rank approximation can be an be ill-posed problem, e.g algorithms might fail or converge to a local minimum [68]. The Tucker decomposition is numerically stable, but has an exponential number of parameters in the dimensions $d$, $\mathcal{O}(kdr + r^d)$. It is therefore not suitable for high-dimensional data. The TT decomposition offers the low-parametric representation of the CP decomposition with the numerical stability of the Tucker decomposition [44].

Tensor trains have recently been applied in, among others, big data processing and machine learning [64]. A tensor train, also known as matrix product state (MPS) in quantum physics, expresses a tensor $\mathcal{W} \in \mathbb{R}^{k_1 \times k_2 \times \cdots \times k_d}$ in terms of $d$ third order tensors $\mathcal{G}_{\mathcal{W}}^{(p)} \in \mathbb{R}^{r_{n-1} \times k_n \times r_n}$. Figure 2-8 shows the TT-decomposition of a four-dimensional tensor in graphical notation. The dimensions of the contracted indices, $r_p$, are called TT-ranks. The first and last TT-ranks, $r_0$ and $r_d$, are by definition equal to one, and therefore not depicted in the figure. Keeping in mind that the $\binom{q}{p}$-mode product operator is left-associative, the tensor train in Figure 2-8 can be expressed as:

$$\mathcal{W} = \mathcal{G}_{\mathcal{W}}^{(1)} \times_2^1 \mathcal{G}_{\mathcal{W}}^{(2)} \times_3^1 \mathcal{G}_{\mathcal{W}}^{(3)} \times_4^1 \mathcal{G}_{\mathcal{W}}^{(4)}. \tag{2-11}$$

For a prescribed set of TT-ranks or a prescribed accuracy, the TT-decomposition of a tensor can be computed with the TT-SVD [65] or the TT-Cross [69] algorithm. There exists a set of TT-ranks $r_p = R_p$ for which the decomposition is exact. When $r_p < R_p$, the tensor train represents an approximation of the original tensor. The lower the TT-ranks, the less accurate the decomposition, but the better the compression. When all $r_p$ and dimensions $k_p$ are equal, the storage complexity of the tensor train representation is $\mathcal{O}(kdr^2)$. A TT-decomposition

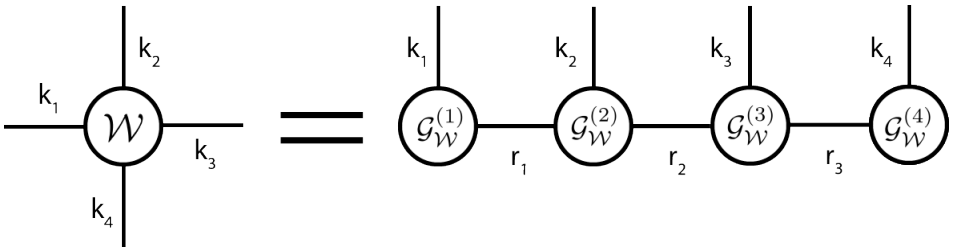



**Figure 2-8:** Graphical notation of the tensor train decomposition for a fourth order tensor.

with low TT-ranks can thus significantly reduce the memory footprint of high-dimensional data. An upper bound for the TT-ranks $R_p$ of an exact decomposition is given by:

$$R_p = \min\left(\prod_{j=1}^{p} k_j\,,\ \prod_{j=p+1}^{d} k_j\right). \tag{2-12}$$

This is best illustrated through the example visualized in Figure 2-9. Consider again a fourth order tensor train. To express an upper bound for $r_2$, we first contract the whole tensor train, except over $r_2$. Grouping the free indices of the resulting cores results in a matrix of size $\mathbb{R}^{k_1k_2 \times k_3k_4}$. The rank of this matrix is now equal to the TT-rank $r_2$. The rank of a matrix is at most equal to its smallest dimension, thus the exact TT-rank $R_2$ is at most:

$$R_2 = \min\left(k_1k_2\,,\ k_3k_4\right). \tag{2-13}$$

Similarly, it can be shown that an upper bound for any TT-rank $R_p$ is given by Eq. (2-12). This means that, in general, the TT-ranks of an efficient TT-decomposition increase towards the middle of the network. The effectiveness of the tensor train representation thus heavily relies on the low TT-rank assumption on the original tensor. If this assumption does not hold, high TT-ranks may be required, potentially resulting in a tensor train which has an even higher storage requirement than the original tensor.

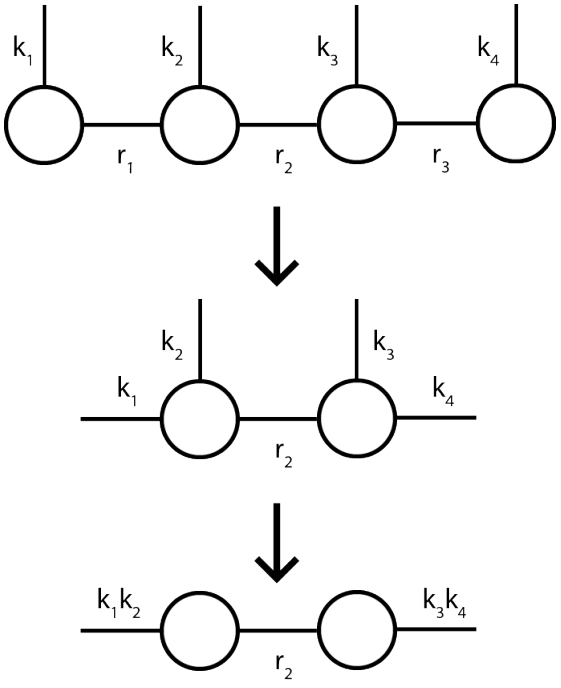



**Figure 2-9:** Illustrating the upper bound on TT-rank $r_2$.

An important notion for TT-cores is orthogonality. A TT-core $\mathcal{G}_{\mathcal{W}}^{(p)}$ is left-orthogonal if it can be reshaped into a matrix $\boldsymbol{G}^{(p)} \in \mathbb{R}^{r_{p-1}k_p \times r_p}$ for which:

$$\boldsymbol{G}^{(p)T}\boldsymbol{G}^{(p)} = \boldsymbol{I}.$$

Likewise, $\mathcal{G}_{\mathcal{W}}^{(p)}$ is right-orthogonal if it can be reshaped into a matrix $\boldsymbol{G}^{(p)} \in \mathbb{R}^{r_{p-1} \times k_p r_p}$ for which:

$$\boldsymbol{G}^{(p)} \boldsymbol{G}^{(p)T} = \boldsymbol{I}.$$

A tensor train is in site-k-mixed-canonical form [70] when for for its TT-cores the following applies:

$$\mathcal{G}_{\mathcal{W}}^{(p)} = \begin{cases} \text{left-orthogonal}, & 1 \leq p \leq k-1 \\ \text{right-orthogonal}, & k+1 \leq p \leq d. \end{cases} \tag{2-14}$$

For a site-k-mixed-canonical tensor train holds that its norm is contained in the $k$-th TT-core, i.e.:

$$\|\mathcal{W}\|_2 = \|\mathcal{G}_{\mathcal{W}}^{(k)}\|_2.$$

There exist efficient algorithms for operations in the tensor train format, such as the addition, multiplication, and rounding of tensor trains, presented in [65]. The multiplication of a tensor by a number in the TT representation is done by simply scaling one of the cores by that number. Addition of two equal-sized tensors, $\mathcal{A}$ and $\mathcal{B}$, in the tensor train format constitutes to merging their cores. TT-rounding is the procedure of recompressing a given tensor train with suboptimal TT-ranks, through a series of QR decompositions and SVD's on the cores. Suboptimal TT-ranks occur due to the rank growth caused by operations in the TT-format e.g, addition or matrix-vector product. For example, adding a tensor train to itself doubles the TT-ranks of the resulting tensor train, while logically the optimal ranks should have stayed the same. TT-rounding is mostly applied after these operations to reduce the ranks while maintaining accuracy. The complexity of this procedure is $\mathcal{O}(kdr^3)$.

Several algorithms have been developed for optimization in the TT-format. The best known method is the alternating linear scheme (ALS) [71]. The TT-ALS algorithm is inspired by the DMRG algorithm of quantum physics and generalizes the alternating least squares (ALS) algorithm to tensor trains. It finds the optimal TT-cores by iteratively optimizing one tensor core at a time while holding the others fixed. At each iteration, a small linear subsystem is solved. The algorithm sweeps back and forth, iterating from the first to the last core and back, until convergence. To ensure numerical stability, each core optimization is followed by an orthogonalization step. Typically all available data is used in each optimization step, thus ALS is not suited for applications requiring online optimization. A modified version of ALS method, MALS [71], updates two cores simultaneously. MALS is computationally more expensive, but is able to adaptively determine the optimal TT-ranks for a specified accuracy, while for ALS the TT-ranks need to be chosen beforehand. Another popular approach to optimization in the TT-format relies on the fact that the set of all $d$-dimensional tensors with fixed TT-ranks form a Riemannian manifold [72]. This allows the use of Riemannian optimization methods such as stochastic Riemannian gradient descent [73], which are also effective in an online fashion. An example of an application for Riemannian optimization is given in [52]. The downside to the Riemannian approach is again that the TT-ranks need to be specified beforehand. An effective method that allows both online optimization and adaptively determines ranks is the tensor network Kalman filter [54]. It enables the adaptive estimation of exponentially large state vectors, without ever needing to explicitly construct them. It can, for example, be used in estimating large scale time varying systems. The equations are tensorized versions of the standard Kalman equations, implemented using

tensor networks. After each update a TT-rounding step is performed to counter the growth of the TT-ranks. This automatically finds the best TT-ranks for a given accuracy. The TT-rounding step can however be computationally intensive for larger TT-ranks. Ultimately, the best suited method for optimizing TT-cores is application dependant. Further on in this thesis, we use ALS as the basis for our optimization method, as it is easy to implement and is guaranteed to converge monotonously.

# Chapter 3

# Introduction to B-splines

This chapter gives an introduction to B-spline theory. First, the basic properties of univariate B-splines are covered. We illustrate how B-splines can be used to approximate functions and treat regularization techniques. Then these methods are generalized to multivariate B-splines. We additionally propose an efficient strategy to evaluate uniform B-splines. A comprehensive treatment of B-splines and their properties is given in the book by de Boor [74].

## 3-1 Splines

A simple univariate spline $S$ maps values from an interval $[a, b]$ to the set of real numbers:

$$S : [a, b] \in \mathbb{R} \rightarrow \mathbb{R}. \tag{3-1}$$

One can partition $[a, b]$ into subintervals $[t_i, t_{i+1}]$ in such a way that $S$ is a polynomial on each subinterval. The points $t_i$ where $S$ changes from one polynomial to another are called knots. A knot sequence $\boldsymbol{t}$ is defined as a non-decreasing and finite sequence of real numbers that define this partition:

$$\begin{aligned} \boldsymbol{t} &= \{t_0, t_1, \ldots, t_{m-1}, t_m\}, \\ a = t_0 \leq t_1 \leq \cdots \leq t_{m-1} &\leq t_m = b. \end{aligned} \tag{3-2}$$

The total number of knots equals $m + 1$. If the knots are equidistantly distributed over the domain of the spline, the spline is called uniform. The degree $\rho \geq 0$ of a spline is equal to the largest degree of its polynomials. When there are no repeating knots, i.e. all knots have distinct values, the spline has $C^{\rho-1}$ continuity, i.e. the first $\rho - 1$ derivatives of the spline are continuous across the domain.

## 3-2 B-splines

### 3-2-1 Basics

The term B-splines first appeared in Schoenberg's paper from 1946 on the approximation of equidistant data by analytic functions [75]. The 'B' stands for basis, referring to their role as basis functions for the spline function space. Any spline of degree $\rho$ on a given knot sequence can be expressed as a unique linear combination of B-splines of the same degree [16]. This expression is given by:

$$S(x) = \sum_{i=0}^{k-1} B_{i,\rho}(x) w_i = \begin{bmatrix} B_{0,\rho}(x) & B_{1,\rho}(x) & \cdots & B_{k-1,\rho}(x) \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{k-1} \end{bmatrix} = \boldsymbol{b}^T \boldsymbol{w}. \tag{3-3}$$

The spline $S(x)$ is also known as a B-spline curve. The B-spline basis functions $B_{i,\rho}(x)$ are contained in the basis vector $\boldsymbol{b}$. The weights $w_i$ are called control points in literature, because they control the shape of the B-spline curve. They are grouped together in the vector $\boldsymbol{w}$. In this thesis the control points are considered to be scalar. The number of control points for a B-spline curve is equal to the number of B-spline basis functions $k$ and relates to the degree of the curve $\rho$ and number of knots $m+1$ by:

$$k = m - \rho. \tag{3-4}$$

**Knot sequence:** The B-spline basis functions $B_{i,\rho}(x)$ are defined by the knot sequence $\boldsymbol{t}$ and their degree $\rho$. If the knot sequence is uniform and also a subset of $\mathbb{Z}$, i.e. a sequence of integers, the spline is referred to as a cardinal spline [76]. This review will put extra focus on cardinal and uniform B-splines, because their form has many advantages such as efficient evaluation and regularization.

**B-spline construction:** For a given a knot sequence $\boldsymbol{t}$, B-spline basis functions of arbitrary degree $\rho$ can be recursively constructed by means of the Cox-de Boor formula [74]:

$$B_{i,0}(x) = \begin{cases} 1 & \text{if} \quad t_i \leq x < t_{i+1} \\ 0 & \text{otherwise} \end{cases}, \tag{3-5}$$

$$B_{i,\rho+1}(x) = \frac{x - t_i}{t_{i+\rho} - t_i} B_{i,\rho}(x) + \frac{t_{i+\rho+1} - x}{t_{i+\rho+1} - t_{i+1}} B_{i+1,\rho}(x). \tag{3-6}$$

Note that the convention of B-splines being left continuous is used. In the case of repeating knots, the denominators can have a value of zero. In these circumstances, $0/0$ is presumed to be zero. Sometimes in literature, a B-spline basis function is also characterized by its order, which is equal to its degree raised by one, such that a first order B-spline equals a zero degree B-spline. In this thesis, the B-splines will only be characterized by their degree $\rho$. The naming

of B-splines by degree follows the naming convention of polynomials, e.g, B-splines of degree one are linear B-splines, degree two are quadratic B-splines, degree three are cubic B-splines, and so on. Figure 3-1 shows the shape of the cardinal B-spline basis functions for degrees 0 to 3. As $\rho$ tends to infinity, the centralized cardinal B-spline converges to the probability density function of the standard normal distribution [77].

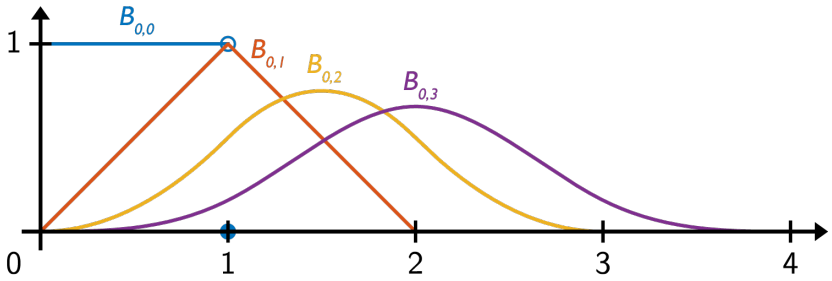



**Figure 3-1:** Cardinal B-splines of degrees 0 to 3 [78].

**Local support:** It is apparent from Figure 3-1 that the degree of a B-spline controls both its smoothness and the size of its support. In general, B-splines have local support, meaning they only map to nonzero values on a small compact subset of the domain of the curve. Furthermore, they are positive within their support.

$$\begin{array}{lcll} B_{i,\rho}(x) & > & 0 & t_i \leq x < t_{i+\rho+1} \\ B_{i,\rho}(x) & = & 0 & \text{otherwise.} \end{array} \tag{3-7}$$

**Natural domain:** Figure 3-2 illustrates an example of a B-spline curve $S$ on a cardinal knot sequence as a linear combination of five B-spline basis functions of degree 2. The control points are here arbitrarily chosen as $\boldsymbol{w} = \begin{bmatrix} 1 & 2 & -1 & 3 & -2 \end{bmatrix}^T$. The dashed lines represent basis functions scaled by their respective weights. Their sum results in the B-spline curve drawn in a thick cyan line. Note that the curve tends to zero at both ends, no matter what values are chosen for the control points. The B-spline curve here is only fully controllable between knots 2 and 5, also known as the natural domain $D_{\mathrm{n}}$. In general, the natural domain of a B-spline curve of degree $\rho$ is given by:

$$D_{\mathrm{n}} = [t_\rho, t_{m-\rho}]. \tag{3-8}$$

The sum of B-splines on any point within this domain equals one. This is the so-called partition of unity property of B-splines:

$$\sum_{i=0}^{k-1} B_{i,\rho}(x) = 1, \qquad t_\rho \leq x \leq t_{m-\rho}. \tag{3-9}$$

A direct consequence of Eq. (3-8) and Eq. (3-9) is that only $\rho + 1$ B-splines $B_{i,\rho}(x)$ map to nonzero values for any $x \in D_{\mathrm{n}}$.

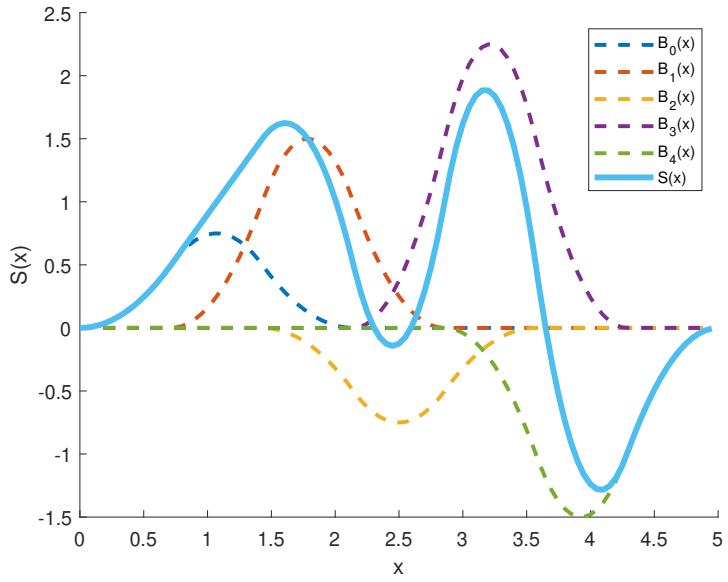



**Figure 3-2:** Cardinal quadratic B-spline curve.

### 3-2-2 Cardinal B-splines

**Construction:** The following will provide an example of the construction of cardinal B-spline basis functions for the integer knot sequence $\boldsymbol{t} = \{0, 1, 2, 3, 4, 5\}$. The degree $\rho = 2$ and $m = 5$, so from Eq. (3-4) it follows that $5-2 = 3$ B-splines need to be constructed. Using the recursion formula, Eq. (3-5), the first one can be determined by substitution:

$$\begin{aligned} B_{0,2}(x) &= \frac{x-0}{2-0}B_{0,1}(x) + \frac{3-x}{3-1}B_{1,1}(x) \\ &= \frac{x-0}{2-0}\left[\frac{x-0}{1-0}B_{0,0}(x) + \frac{2-x}{2-1}B_{1,0}(x)\right] + \frac{3-x}{3-1}\left[\frac{x-1}{2-1}B_{1,0}(x) + \frac{3-x}{3-2}B_{2,0}(x)\right] \\ &= \frac{1}{2}\Big[x^2 B_{0,0}(x) + x(2-x)B_{1,0}(x) + (3-x)(x-1)B_{1,0}(x) + (3-x)^2 B_{2,0}(x)\Big]. \end{aligned}$$

Using Eq. (3-5), this can be decomposed as follows:

$$B_{0,2}(x) = \begin{cases} \frac{1}{2}x^2 & 0 \leq x < 1 \\ \frac{1}{2}(x(2-x) + (x-1)(3-x)) & 1 \leq x < 2 \\ \frac{1}{2}(3-x)^2 & 2 \leq x < 3 \\ 0 & \text{otherwise.} \end{cases} \tag{3-10}$$

It can similarly be shown for the other two B-splines that:

$$B_{1,2}(x) = \begin{cases} \frac{1}{2}(x-1)^2 & 1 \leq x < 2 \\ \frac{1}{2}((x-1)(3-x) + (x-2)(4-x)) & 2 \leq x < 3 \\ \frac{1}{2}(4-x)^2 & 3 \leq x < 4 \\ 0 & \text{otherwise.} \end{cases} \tag{3-11}$$

$$B_{2,2}(x) = \begin{cases} \frac{1}{2}(x-2)^2 & 2 \leq x < 3 \\ \frac{1}{2}((x-2)(4-x) + (x-3)(5-x)) & 3 \leq x < 4 \\ \frac{1}{2}(5-x)^2 & 4 \leq x < 5 \\ 0 & \text{otherwise.} \end{cases} \tag{3-12}$$

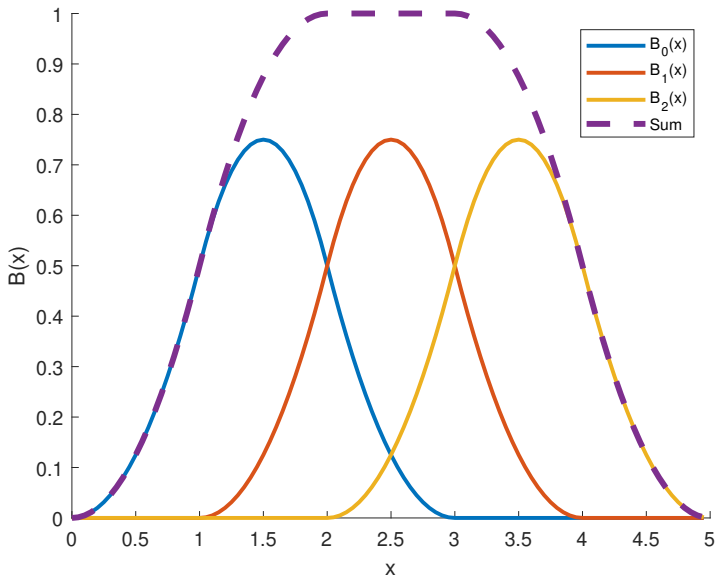



**Figure 3-3:** Cardinal quadratic B-splines and their sum.

Figure 3-3 shows all three B-splines over the knot sequence. The dashed purple line represents their sum. It is easy to verify that the natural domain for this B-spline curve is $D_{\text{n}} = [2, 3]$. Both the plot and the equations illustrate that the basis functions of a B-spline curve on a uniform knot sequence are shifted copies of each other. The relation is given by:

$$B_{i,\rho}(x) = B_{i+1,\rho}(x + \Delta t), \quad \text{where} \quad \Delta t = t_{i+1} - t_i. \tag{3-13}$$

**Matrix form:** Using all three equations, Eq. (3-10), Eq. (3-11) and Eq. (3-12), the curve segment $[2, 3]$ can be described by:

$$\begin{aligned} S(x) &= \sum_{i=0}^{2} B_{i,2}(x) w_i = \begin{bmatrix} B_{0,2}(x) & B_{1,2}(x) & B_{2,2}(x) \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} \\ &= \frac{1}{2} \begin{bmatrix} (3-x)^2 & (x-1)(3-x)+(x-2)(4-x) & (x-2)^2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} \\ &= \begin{bmatrix} (x-2)^2 & (x-2) & 1 \end{bmatrix} \frac{1}{2} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}, \quad 2 \le x < 3. \end{aligned} \tag{3-14}$$

The last step of this derivation is not trivial and involves some algebra. More details of the matrix representation can be found in [79]. In general, a quadratic cardinal B-spline can be expressed as:

$$B_{i,2}(x) = \begin{cases} \frac{1}{2}(x-t_i)^2 & t_i \le x < t_{i+1} \\ \frac{1}{2}((x-t_i)(t_{i+2}-x)+(x-t_{i+1})(t_{i+3}-x)) & t_{i+1} \le x < t_{i+2} \\ \frac{1}{2}(t_{i+3}-x)^2 & t_{i+2} \le x < t_{i+3} \\ 0 & \text{otherwise.} \end{cases} \tag{3-15}$$

From Eq. (3-15) it can be shown that for any given interval $[t_{j+2}, t_{j+3}), \quad 0 \le j \le m-5$ within the natural domain $D_{\mathrm{n}}$, a cardinal quadratic B-spline curve $S_j(x)$ can be expressed in a convenient matrix form:

$$\begin{aligned} S(x) &= \sum_{i=0}^{k-1} B_{i,2}(x) w_i = \sum_{i=j}^{j+\rho} B_{i,2}(x) w_i \\ &= \begin{bmatrix} B_{j,2}(x) & B_{j+1,2}(x) & B_{j+2,2}(x) \end{bmatrix} \begin{bmatrix} w_j \\ w_{j+1} \\ w_{j+2} \end{bmatrix} \\ &= \begin{bmatrix} (x-t_{j+2})^2 & (x-t_{j+2}) & 1 \end{bmatrix} \frac{1}{2} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 2 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} w_j \\ w_{j+1} \\ w_{j+2} \end{bmatrix} = \boldsymbol{\phi}^T \boldsymbol{M_2} \boldsymbol{w}_j, \\ & \qquad\qquad\qquad\qquad\qquad\qquad t_{j+2} \le x < t_{j+3}. \end{aligned} \tag{3-16}$$

It is apparent that the function value at any point within the natural domain of the curve is influenced by only $\rho+1$ control points, because this is the number of basis functions that are nonzero within this interval. The matrix $\boldsymbol{M}_2$ is referred to as a basis matrix of degree 2. A general method to find a basis matrix $\boldsymbol{M}_\rho \in \mathbb{R}^{(\rho+1)\times(\rho+1)}$ for cardinal B-splines of arbitrary degree $\rho$ is given in [80].

Similarly, it can be shown that the $j$th curve segment $[t_{j+\rho}, t_{j+\rho+1})$, $0 \leq j \leq m-2\rho-1$ within the natural domain of a cardinal B-spline curve of degree $\rho$ can be written as:

$$S(x) = \begin{bmatrix} (x-t_{j+\rho})^{\rho} & (x-t_{j+\rho})^{\rho-1} & \cdots & (x-t_{j+\rho}) & 1 \end{bmatrix} \boldsymbol{M}_{\rho} \begin{bmatrix} w_j \\ w_{j+1} \\ \vdots \\ w_{j+\rho-1} \\ w_{j+\rho} \end{bmatrix}, \quad t_{j+\rho} \leq x < t_{j+\rho+1}. \tag{3-17}$$

Because the B-splines are left continuous, this expression in general only holds for $t_{j+\rho} \leq x < t_{j+\rho+1}$. For degrees $\rho \geq 1$, however, it also holds for $x = t_{m-\rho}$.

### 3-2-3 Uniform B-splines

**Cardinal to uniform:** The usefulness of the matrix form for cardinal knot sequences is manifest when realizing that any uniform B-spline can be formed from a cardinal B-spline of the same degree, by shifting and scaling the knots linearly. This means that Eq. (3-17) can be used to evaluate any uniform B-spline curve, by first mapping the input variable to a cardinal domain. Given are a uniform B-spline curve $S^U(x)$ defined on the knot sequence $\boldsymbol{t^U} = \{t_0^U, t_1^U, \ldots, t_m^U\}$ and a cardinal B-spline curve $S^C(z)$ on the integer knot sequence $\boldsymbol{t^C} = \{t_0^C, t_1^C, \ldots, t_m^C\}$. If both the uniform and cardinal B-spline share the same weight vector $\boldsymbol{w}$, the following equality holds:

$$S^U(x) = S^C\left(\frac{x - t_0^U}{\Delta t^U} + t_0^C\right), \qquad \text{where} \quad \Delta t^U = t_{i+1}^U - t_i^U. \tag{3-18}$$

**Matrix form:** In many applications, one is only interested in the natural domain of a B-spline curve. In curve fitting, for example, it is desirable for the B-spline curve to be fully controllable over the whole range of the input samples. We therefore derive the following expression to represent the natural domain $x \in D_{\text{n}}$ of any uniform B-spline curve in an computationally efficient manner:

$$S^U(x) = \begin{bmatrix} u^{\rho} & u^{\rho-1} & \cdots & u & 1 \end{bmatrix} \boldsymbol{M}_{\rho} \begin{bmatrix} w_j \\ w_{j+1} \\ \vdots \\ w_{j+\rho-1} \\ w_{j+\rho} \end{bmatrix} = \boldsymbol{u}^T \boldsymbol{M}_{\rho}\, \boldsymbol{w}_j, \tag{3-19}$$

$$u = \frac{x - t_0^U}{\Delta t^U} - j - \rho, \tag{3-20}$$

$$j = \begin{cases} \left\lfloor \frac{x-t_0^U}{\Delta t^U} \right\rfloor - \rho, & \text{if} \quad t_{\rho}^U \leq x < t_{m-\rho}^U \\ m - 2\rho - 1, & \text{if} \quad x = t_{m-\rho}^U \end{cases}. \tag{3-21}$$

*Proof.* Combining Eq. (3-17) and Eq. (3-18) yields the following relation:

$$S_j^U(x) = \begin{bmatrix} \left(z - t_{j+\rho}^C\right)^\rho & \left(z - t_{j+\rho}^C\right)^{\rho-1} & \cdots & \left(z - t_{j+\rho}^C\right) & 1 \end{bmatrix} \boldsymbol{M}_\rho \boldsymbol{w}_j,$$
$$z = \frac{x - t_0^U}{\Delta t^U} + t_0^C, \qquad t_{j+\rho}^U \leq x < t_{j+\rho+1}^U. \tag{3-22}$$

For integer knots, the following relation holds:

$$t_0^C - t_{j+n}^C = -(j+n). \tag{3-23}$$

We define the auxiliary variable $u$ as:

$$u = z - t_{j+\rho}^C = \frac{x - t_0^U}{\Delta t^U} + t_0^C - t_{j+\rho}^C = \frac{x - t_0^U}{\Delta t^U} - j - \rho. \tag{3-24}$$

Then, using that $t_{j+\rho}^C = \lfloor z \rfloor$ when $\quad t_{j+\rho}^C \leq z < t_{j+\rho+1}^C$, an expression for the index $j$ can be derived as follows:

$$\begin{aligned} t_0^C - t_{j+\rho}^C &= t_0^C - \left\lfloor \frac{x - t_0^U}{\Delta t^U} + t_0^C \right\rfloor = t_0^C - \left\lfloor \frac{x - t_0^U}{\Delta t^U} \right\rfloor - t_0^C \\ &= - \left\lfloor \frac{x - t_0^U}{\Delta t^U} \right\rfloor = -(j+\rho). \\ \therefore j + \rho &= \left\lfloor \frac{x - t_0^U}{\Delta t^U} \right\rfloor \qquad (3\text{-}25) \\ \therefore j &= \left\lfloor \frac{x - t_0^U}{\Delta t^U} \right\rfloor - \rho. \qquad (3\text{-}26) \end{aligned}$$

Lastly, when $x = t_{m-\rho}^U$ the index $j$ is defined as:

$$j = \left\lfloor \frac{t_{m-\rho}^U - t_0^U}{\Delta t^U} \right\rfloor - \rho - 1 = m - 2\rho - 1. \tag{3-27}$$

□

In the remainder of this thesis, we will assume that knot sequences are chosen such that $D_\mathrm{n}$ coincides with the unit interval $[0, 1]$. Any set of input data $\{(x)\}_i^N$ can always be normalized to this interval, such that Eq. (3-19) describes the spline $S(x)$ for every point.

**Constructing the basis vector:** For the task of curve fitting and interpolation, one must construct the B-spline basis vector $\boldsymbol{b}$ for every data point. As noted before, only $\rho+1$ of the B-splines are nonzero for an input $x$. An efficient approach for uniform knot sequences is to first calculate the nonzero B-splines using Eq. (3-19) as follows:

$$\begin{bmatrix} B_{j,\rho}(x) & B_{j+1,\rho}(x) & \cdots & B_{j+\rho-1,\rho}(x) & B_{j+\rho,\rho}(x) \end{bmatrix} = \begin{bmatrix} u^{\rho} & u^{\rho-1} & \cdots & u & 1 \end{bmatrix} \boldsymbol{M}_{\rho}. \quad (3\text{-}28)$$

Subsequently, these B-splines can be inserted into $\boldsymbol{b}$, such that its $i$th element is given by:

$$\boldsymbol{b}_i = \begin{cases} B_{i,\rho}(x) & j \leq i \leq j+\rho \\ 0 & \text{otherwise.} \end{cases} \quad (3\text{-}29)$$

The computational efficiency of constructing $\boldsymbol{b}$ using Eq. (3-28) and Eq. (3-29), compared to using the recursive formula in Eq. (3-5) or even the triangular scheme in [81], is one of the reasons to prefer uniform knot sequences for B-splines in tasks involving many functions evaluations.

### 3-2-4 Function Approximation

Function approximation with B-splines, also referred to as curve fitting, constitutes to finding the weights $w$ for which the B-spline curve best describes the relationship between a set of input-output data $\{(x_i, y_i)\}_{i=1}^{N}$, $x_i \leq x_{i+1}$. Curve fitting is often performed as a means of data analysis or to make predictions about new data. Typical use-cases occur in pattern recognition, signal processing and computer aided design. The degree of the B-splines and the number of knots are chosen based on the desired smoothness and complexity of the resulting curve.

**Cost function:** The weights that provide the best fit to the data are determined by minimizing the distance between the data points and the curve. The most commonly used distance metric is the least squared error between the data points and the function values. This results in the following cost function:

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix},$$

$$\boldsymbol{s} = \begin{bmatrix} S(x_1) \\ S(x_2) \\ \vdots \\ S(x_N) \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}^{(1)T} \\ \boldsymbol{b}^{(2)T} \\ \vdots \\ \boldsymbol{b}^{(N)T} \end{bmatrix} \boldsymbol{w} = \begin{bmatrix} B_{0,\rho}(x_1) & B_{1,\rho}(x_1) & \dots & B_{k-1,\rho}(x_1) \\ B_{0,\rho}(x_2) & B_{1,\rho}(x_2) & \dots & B_{k-1,\rho}(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ B_{0,\rho}(x_N) & B_{1,\rho}(x_N) & \dots & B_{k-1,\rho}(x_N) \end{bmatrix} \boldsymbol{w} = \boldsymbol{B}\boldsymbol{w},$$

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - S(x_i)\right)^2 = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{s}\|_2^2 = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{B}\boldsymbol{w}\|_2^2. \tag{3-30}$$

The factor of a half is included to simplify the derivative.

**Optimization:** Minimizing the cost function from Eq. (3-30) with respect to the weights can be done with ordinary least squares (OLS). The matrix containing the B-splines, $\boldsymbol{B} \in \mathbb{R}^{N\times k}$, is called the collocation matrix [74]. When the degrees of freedom of the curve are equal to, or larger than, the number of data points, i.e. $k \geq N$, it is possible to find weights for which the function passes exactly through every given data point. This process is known as interpolation. In case the degrees of freedom of the curve are lower than the number of data points, i.e. $k < N$, the curve fitting process is known as smoothing. The convexity of the cost function, Eq. (3-30), can be shown by computing the Hessian, i.e. the second-order partial derivative w.r.t. the weights:

$$\begin{aligned} E &= \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{B}\boldsymbol{w}\|_2^2 = \frac{1}{2}(\boldsymbol{y} - \boldsymbol{B}\boldsymbol{w})^T(\boldsymbol{y} - \boldsymbol{B}\boldsymbol{w}) \\ &= \frac{1}{2}\left(\boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{w}^T\boldsymbol{B}^T\boldsymbol{y} + \boldsymbol{w}^T\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{w}\right) && (3\text{-}31) \\ \nabla E &= \frac{1}{2}\left(-2\boldsymbol{B}^T\boldsymbol{y} + 2\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{w}\right) \\ &= \boldsymbol{B}^T(\boldsymbol{B}\boldsymbol{w} - \boldsymbol{y}) && (3\text{-}32) \\ \nabla^2 E &= \boldsymbol{B}^T\boldsymbol{B}. && (3\text{-}33) \end{aligned}$$

The Hessian is positive definite if $\boldsymbol{B}$ has full column rank, and positive semi-definite otherwise. Hence, the cost function is convex and any extremum is a global minimum. The optimal control points $\boldsymbol{w}$ can be found by setting the gradient of the error, Eq. (3-32), equal to zero. This yields the following normal equation:

$$(\boldsymbol{B}^T\boldsymbol{B})\boldsymbol{w} = \boldsymbol{B}^T\boldsymbol{y}. \tag{3-34}$$

To solve Eq. (3-34) for $\boldsymbol{w}$, the matrix $\boldsymbol{B}^T\boldsymbol{B}$ needs to be invertible. Thus the knot sequence needs to be chosen such that the collocation matrix $\boldsymbol{B}$ is full rank. Assuming this is the case, solving the normal equation for $\boldsymbol{w}$ with an LU factorization has a complexity of $\mathcal{O}(N(k)^2)$.

$$\boldsymbol{w} = (\boldsymbol{B}^T\boldsymbol{B})^{-1}\boldsymbol{B}^T\boldsymbol{y}. \tag{3-35}$$

For very large datasets, or for applications that require online optimization, iterative methods such as recursive least squares (RLS) and least mean squares (LMS) can provide an alternative to adaptively find the optimal control points.

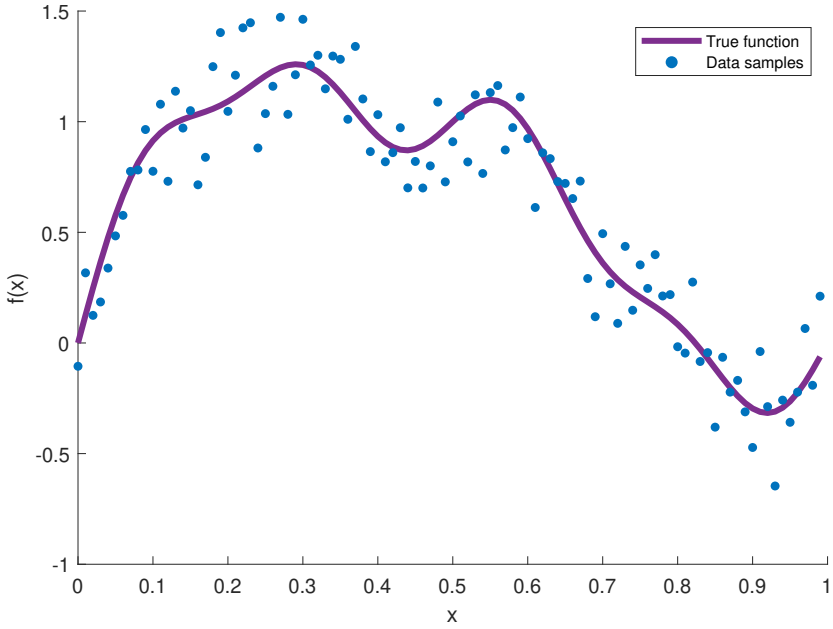



**Figure 3-4:** Data points (blue) sampled from the true function (purple).

**Curve fitting experiment:** To illustrate function approximation with uniform B-splines, we consider the function $f(x)$ which is constructed as a sum of sinusoids:

$$f(x) = \sin(\pi x) + \frac{1}{2}\sin(2\pi x) + \frac{1}{4}\sin(4\pi x) + \frac{1}{8}\sin(8\pi x). \tag{3-36}$$

This function is shown in Figure 3-4 for the domain $x \in [0, 1]$. From this function, we are provided with 100 equidistantly sampled data points $\{(x_i, y_i)\}_{i=1}^{100}$ with additive zero mean Gaussian white noise:

$$y_i = f(x_i) + \varepsilon_i. \tag{3-37}$$

These data samples are shown in the figure as blue dots. The signal to noise ratio (SNR) of the samples is 10 dB. These samples will be used to approximate the original function $f(x)$ using uniform B-splines. As a performance metric, we use the root mean squared error (RMSE) between the outputs of the original function $f(x)$ and the outputs predicted by the approximation $S(x)$ for 1000 inputs $\{(x_i^{\text{test}})\}_{i=1}^{1000}$ equidistantly distributed over $[0, 1]$:

$$e_{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(f(x_i^{\text{test}}) - S(x_i^{\text{test}})\right)^2}. \tag{3-38}$$

Figure 3-5 illustrates uniform B-spline curves fitted on the dataset with OLS for several different choices of the degree $\rho$ and number of knots $m$. The RMSE scores are given in the titles. We observe that higher degrees increase the smoothness, while the number of knots influences how close the curve fits the data. The best performance here is obtained for $\rho = 2$ and $m = 9$. In practice, these parameters are tuned using methods such as cross-validation or AIC [82].

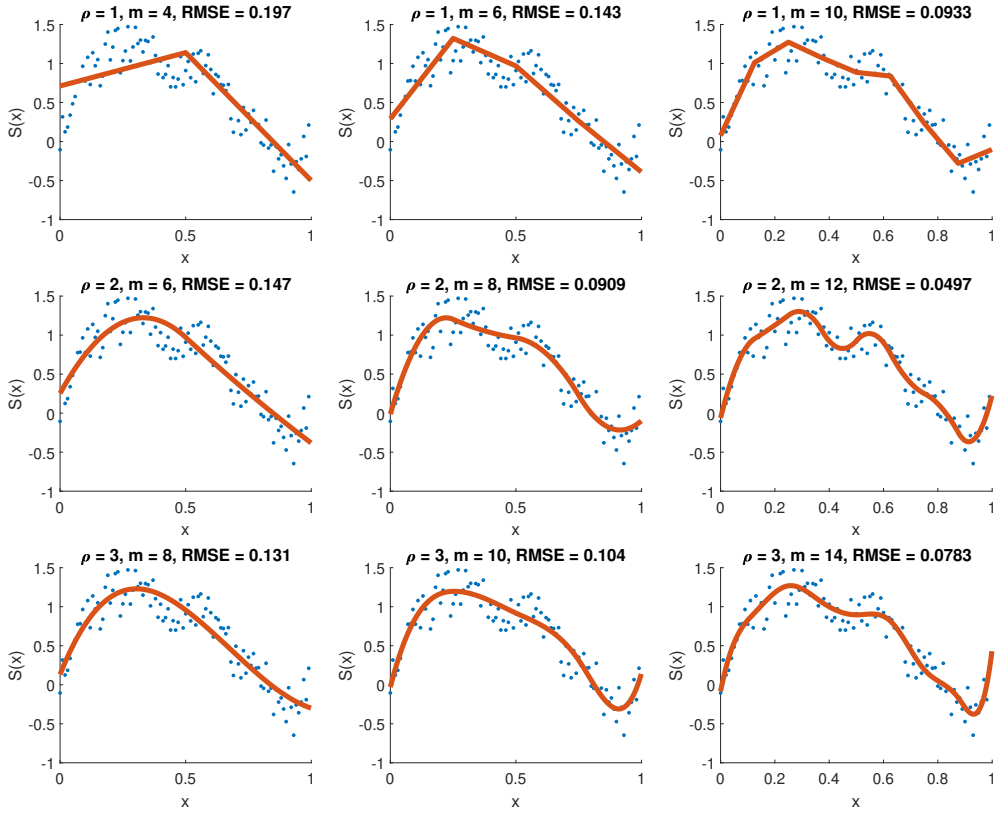



**Figure 3-5:** Uniform B-spline curves fitted on the dataset for different choices of $\rho$ and $m$.

### 3-2-5 Regularization

Most data from real life measurements are scattered across the function domain and contain noise. The challenge in function approximation is to capture relevant input/output relations of the data while avoiding modeling random noise. This is known as the bias-variance tradeoff in statistics. B-spline curves inherently possess the ability to tune the smoothness by varying the number of knots and their degree. In general, too many knots lead to overfitting on the data, too few knots lead to underfitting. One necessary and obvious condition on the knot sequence given by de Boor [74] for $\boldsymbol{B}$ to be full rank is that every B-spline needs data within its support to be fit on:

$$t_j \leq x_i < t_{j+\rho+1} \quad \forall j. \tag{3-39}$$

As we restrict ourselves to uniform B-splines, this cannot be guaranteed for a scattered dataset unless we severely limit the number of knots $m$ or use very high degrees $\rho$. This would in both cases lead to underfitting. Even for non-uniform B-splines, the choice of knot placement has been a subject of much research, but an attractive all-purpose scheme lacks. To tackle both the bias-variance tradeoff and the problem of scattered data, much interest is shown in literature to non-parametric smoothing [83].

**Non-parametric smoothing:** The general approach to non-parametric smoothing with B-splines is to use a large number of uniform knots, but regularize the control points to ensure the desired smoothness. The first to propose this strategy was O'Sullivan (1986). He penalized the roughness of a spline by adding a penalty term $R(\boldsymbol{w})$ to the cost function. The penalty is based on the integrated squared second derivative of the curve in the cost function:

$$J(\boldsymbol{w}) = E(\boldsymbol{w}) + \lambda R(\boldsymbol{w}) = \sum_{i=1}^{N} \left(y_i - S(x_i)\right)^2 + \lambda \int_a^b (S^{''}(x_i))^2. \tag{3-40}$$

Using the second derivative in the penalty is common [84], but lower or higher derivatives are possible. The regularization parameter $\lambda > 0$ controls the tradeoff between smoothness and fitting to the data. As $\lambda \to 0$, no penalty is imposed and the resulting curve will follow the noisy data very closely. As $\lambda \to \infty$, the second derivative of the resulting curve converges to zero, thus the resulting spline is a straight line. For B-splines, Eq. (3-40) can be written in quadratic form [84]:

$$J(\boldsymbol{w}) = \|\boldsymbol{y} - \boldsymbol{B}\boldsymbol{w}\|_2^2 + \lambda\, \boldsymbol{w}^T \boldsymbol{\Omega} \boldsymbol{w}. \tag{3-41}$$

The penalty matrix $\boldsymbol{\Omega} \in \mathbb{R}^{k \times k}$ is banded, and its elements are given by:

$$\boldsymbol{\Omega}_{i_1,i_2} = \int_a^b (B^{''}_{i_1,\rho}(x)\, B^{''}_{i_2,\rho}(x)), \quad i_1, i_2 \in [0, 1, \ldots, k-1]. \tag{3-42}$$

The gradient of the penalized least squares cost function is:

$$\nabla J = 2\left(\boldsymbol{B}^T\boldsymbol{B} + \lambda\boldsymbol{\Omega}\right)\boldsymbol{w} - 2\boldsymbol{B}^T\boldsymbol{y}. \tag{3-43}$$

Hence, the normal equation becomes:

$$\left(\boldsymbol{B}^T\boldsymbol{B} + \lambda\boldsymbol{\Omega}\right)\boldsymbol{w} = \boldsymbol{B}^T\boldsymbol{y}. \tag{3-44}$$

The cost function is strongly convex and can be solved using the mentioned direct or iterative methods. The construction of $\boldsymbol{\Omega}$ can, however, be tedious and computationally expensive.

**P-splines:** P-splines [82] circumvent the construction of $\boldsymbol{\Omega}$ by penalizing the finite differences of the weights of adjacent B-splines. P-splines, which stand for penalized splines, are defined on uniform knot sequences. The P-spline penalty is given by:

$$R(\boldsymbol{w}) = \|\boldsymbol{D}_\alpha \boldsymbol{w}\|_2^2, \tag{3-45}$$

where $\boldsymbol{D}_\alpha \in \mathcal{R}^{(k-\alpha)\times(k)}$ is the $\alpha$-th order difference matrix such that $\boldsymbol{D}_\alpha\boldsymbol{w} = \Delta^\alpha\boldsymbol{w}$ results in a vector of $\alpha$-th order differences of $\boldsymbol{w}$. The cost function can be conveniently written as:

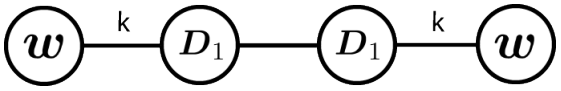



**Figure 3-6:** The P-spline penalty in graphical notation.

$$\begin{aligned} J(\boldsymbol{w}) &= E(\boldsymbol{w}) + \lambda R(\boldsymbol{w}) \\ &= \|\boldsymbol{y} - \boldsymbol{B}\boldsymbol{w}\|_2^2 + \lambda\, \boldsymbol{w}^T \boldsymbol{D}_\alpha^T \boldsymbol{D}_\alpha \boldsymbol{w}. \end{aligned} \tag{3-46}$$

The penalty matrix $\boldsymbol{D}_\alpha^T \boldsymbol{D}_\alpha$ is a discrete approximation to $\boldsymbol{\Omega}$. The matrix $\boldsymbol{D}_\alpha$ can be constructed by using the difference operator $\alpha$ times consecutively on the identity matrix. For example, given are a weight vector and the first order difference matrix:

$$\boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}, \qquad \boldsymbol{D}_1 = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix}.$$

The vector of differences is given by:

$$\boldsymbol{D}_1 \boldsymbol{w} = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} (w_0 - w_1) \\ (w_1 - w_2) \end{bmatrix}.$$

The penalty term then equals:

$$R(\boldsymbol{w}) = \|\boldsymbol{D}_1 \boldsymbol{w}\|_2^2 = (\boldsymbol{D}_1 \boldsymbol{w})^T (\boldsymbol{D}_1 \boldsymbol{w}) \tag{3-47}$$

$$= \begin{bmatrix} (w_0 - w_1) & (w_1 - w_2) \end{bmatrix} \begin{bmatrix} (w_0 - w_1) \\ (w_1 - w_2) \end{bmatrix} = (w_0 - w_1)^2 + (w_1 - w_2)^2. \tag{3-48}$$

This penalty is shown in the graphical tensor notation in Figure 3-6. The cost function with P-spline regularization is again quadratic and the resulting normal equation is similar to Eq. (3-44):

$$\left(\boldsymbol{B}^T \boldsymbol{B} + \lambda \boldsymbol{D}_\alpha^T \boldsymbol{D}_\alpha\right) \boldsymbol{w} = \boldsymbol{B}^T \boldsymbol{y}. \tag{3-49}$$

Other popular regularization methods, often used in statistics, are L1 and L2 regularization [85]. The key assumption of both methods is that smaller weights generate simpler functions. Although this somewhat holds for B-spline curves, the weights also directly and locally influence the height of the curve. Large values of $\lambda$ will bias the curve towards zero. L2 regularization, also known as Tikhonov regularization, is in fact equal to the discrete P-spline penalty with zeroed order differences, i.e. the penalty matrix $\boldsymbol{D}_0$ is simply the identity matrix. For a P-spline penalty with $\alpha = 1$, we get what is known in signal processing as Total Variation regularization [86].

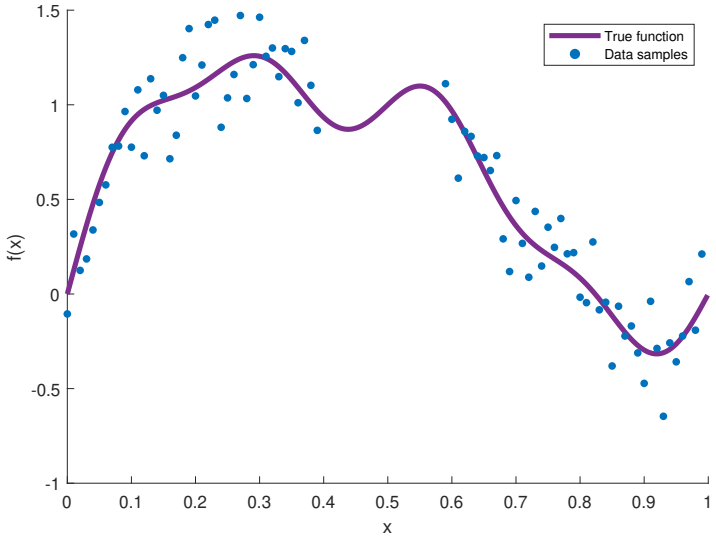



**Figure 3-7:** Data points (blue) sampled from the function (purple).

**Regularization experiment:** We consider again the function in Eq. (3-36). This time, we assume to have no samples from the interval [0.4, 0.6] in our dataset, as shown in Figure 3-7. To illustrate the effect of regularization, we try to approximate this function using a penalized uniform B-spline curve of degree $\rho = 3$ with $m = 26$ knots. Eq. (3-49) is solved for different values of $\alpha$ and regularization parameter $\lambda$.

We again use the RMSE between the predicted outputs $S(x_i^{\text{test}})$ and real outputs of the original function $f(x_i^{\text{test}})$ to compare the performance of the approximations. The results are shown in figure 3-8. Without regularization, finding the weights would be an ill-posed problem because no unique solution exists. For $\alpha = 0$, i.e. Tiknonov regularization, the curve does not generalize well between the known data samples and introduces bias because the curve is induced to approach zero. For $\alpha = 1$ and $\alpha = 2$ the curve is adequately regularized and approximates the original function well. Higher values of $\lambda$ result in smoother curves. The best performance is achieved here when $\alpha = 1$ and $\lambda = 0.1$.

## 3-3 Multivariate B-splines

All of the previous definitions treated only univariate B-spline curves, i.e. B-spline curves with knots that lie in a single dimension. Generalizing B-splines to multiple input dimensions is done through tensor products of the univariate basis. This section will first illustrate the bivariate case, also known as a B-spline surface or tensor product surface, then generalize to an arbitrary number of dimensions $d$.

### 3-3-1 B-spline Surfaces

B-spline surfaces are piecewise polynomials on rectangular domains. Given are two univariate B-spline basis vectors $\boldsymbol{b}^{(1)}$ and $\boldsymbol{b}^{(2)}$, defined on knot sequences $\boldsymbol{t}^{(1)}$ and $\boldsymbol{t}^{(2)}$ respectively:

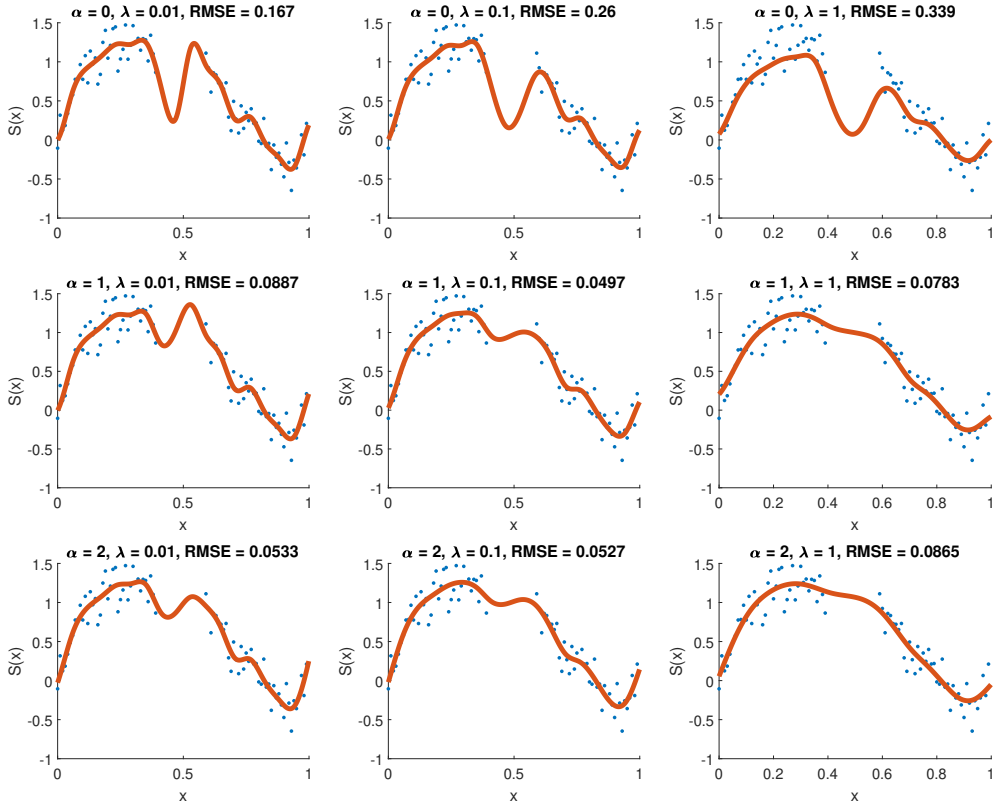



**Figure 3-8:** P-spline curves fitted on noisy samples for different choices of $\alpha$ and $\lambda$.

$$\boldsymbol{b}^{(1)} = \begin{bmatrix} B_{0,\rho}(x_1) & B_{1,\rho}(x_1) & \cdots & B_{k_1-1,\rho}(x_1) \end{bmatrix}^T,$$
$$\boldsymbol{b}^{(2)} = \begin{bmatrix} B_{0,\rho}(x_2) & B_{1,\rho}(x_2) & \cdots & B_{k_2-1,\rho}(x_2) \end{bmatrix}^T.$$

Also given is a matrix of control points $\boldsymbol{W}$. One can construct the tensor product surface that lies on the rectangle spanned by $\boldsymbol{t}^{(1)}$ and $\boldsymbol{t}^{(2)}$ with:

$$S(x_1, x_2) = \sum_{i_1=0}^{k_1-1} \sum_{i_2=0}^{k_2-1} B_{i_1,\rho}(x_1) B_{i_2,\rho}(x_2) \boldsymbol{W}_{i_1,i_2}. \tag{3-50}$$

The B-spline basis matrix $\boldsymbol{B}$ is defined as the tensor product of $\boldsymbol{b}^{(1)}$ and $\boldsymbol{b}^{(2)}$:

$$\boldsymbol{B} = \boldsymbol{b}^{(1)} \circ \boldsymbol{b}^{(2)}.$$

$$= \begin{bmatrix} B_{0,\rho}(x_1)B_{0,\rho}(x_2) & B_{0,\rho}(x_1)B_{1,\rho}(x_2) & \dots & B_{0,\rho}(x_1)B_{k_2-1,\rho}(x_2) \\ B_{1,\rho}(x_1)B_{0,\rho}(x_2) & B_{1,\rho}(x_1)B_{1,\rho}(x_2) & \dots & B_{1,\rho}(x_1)B_{k_2-1,\rho}(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ B_{k_1-1,\rho}(x_1)B_{0,\rho}(x_2) & B_{k_1-1,\rho}(x_1)B_{1,\rho}(x_2) & \dots & B_{k_1-1,\rho}(x_1)B_{k_2-1,\rho}(x_2) \end{bmatrix}. \tag{3-51}$$

The spline surface can then be written as an inner product:

$$S(x_1, x_2) = \langle \boldsymbol{B}, \boldsymbol{W} \rangle . \tag{3-52}$$

Figure 3-9 illustrates a quadratic tensor product B-spline in two dimensions.

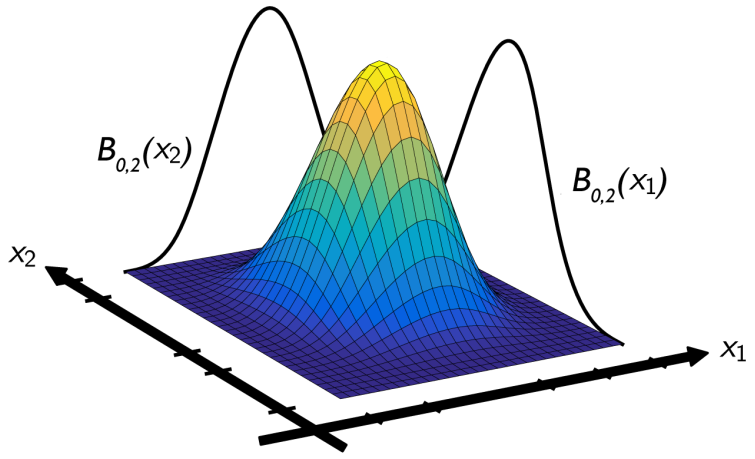



**Figure 3-9:** Two-dimensional quadratic tensor product B-spline [78].

### 3-3-2 B-spline Hypersurfaces

More generally, one can construct a $d$-dimensional hypersurface from B-splines using:

$$S(x_1, x_2, \dots, x_d) = \sum_{i_1=0}^{k_1-1} \sum_{i_2=0}^{k_2-1} \cdots \sum_{i_d=0}^{k_{d-1}} B_{i_1,\rho}(x_1)B_{i_2,\rho}(x_2)\cdots B_{i_d,\rho}(x_d)\mathcal{W}_{i_1 i_2 \cdots i_d}. \tag{3-53}$$

The B-spline basis tensor $\mathcal{B}$ is defined as:

$$\mathcal{B} = \boldsymbol{b}^{(1)} \circ \boldsymbol{b}^{(2)} \circ \cdots \circ \boldsymbol{b}^{(d)}.$$

The spline hypersurface can be written as an inner product:

$$S(x_1, x_2, \dots, x_d) = \langle \mathcal{B}, \mathcal{W} \rangle . \tag{3-54}$$

### 3-3-3 Multivariate Function Approximation

Techniques for curve fitting easily generalize to multivariate B-splines by writing Eq. (3-54) in vectorized form:

$$S(x_1, x_2, \ldots, x_d) = \langle \mathcal{B}, \mathcal{W} \rangle = \boldsymbol{vec}(\mathcal{B})^T \boldsymbol{vec}(\mathcal{W}). \tag{3-55}$$

For example, given a data $d$-dimensional dataset $\{(x_1^i, x_2^i, \ldots, x_d^i, y^i)\}_{i=1}^N$, fitting a hypersurface to the data is done by finding the weight tensor that minimizes the following least squares error function:

$$\boldsymbol{y} = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^N \end{bmatrix},$$

$$\boldsymbol{s} = \begin{bmatrix} S(x_1^1, x_2^2, \ldots, x_d^1) \\ S(x_1^2, x_2^2, \ldots, x_d^2) \\ \vdots \\ S(x_1^N, x_2^N, \ldots, x_d^N) \end{bmatrix} = \begin{bmatrix} \boldsymbol{vec}(\mathcal{B}^1)^T \\ \boldsymbol{vec}(\mathcal{B}^2)^T \\ \vdots \\ \boldsymbol{vec}(\mathcal{B}^N)^T \end{bmatrix} \boldsymbol{vec}(\mathcal{W}) = \boldsymbol{B}\, \boldsymbol{vec}(\mathcal{W}), \tag{3-56}$$

$$E(\mathcal{W}) = \frac{1}{2} \sum_{i=1}^{N} \left( y^i - S(x_1^i, x_2^i, \ldots, x_d^i) \right)^2 = \frac{1}{2} \|\boldsymbol{y} - \boldsymbol{s}\|_2^2. \tag{3-57}$$

The weights that minimize this error can be found using the discussed optimization algorithms. Using for example the pseudoinverse one can find:

$$\boldsymbol{vec}(\mathcal{W}) = (\boldsymbol{B}^T \boldsymbol{B})^{-1} \boldsymbol{B}^T \boldsymbol{y}. \tag{3-58}$$

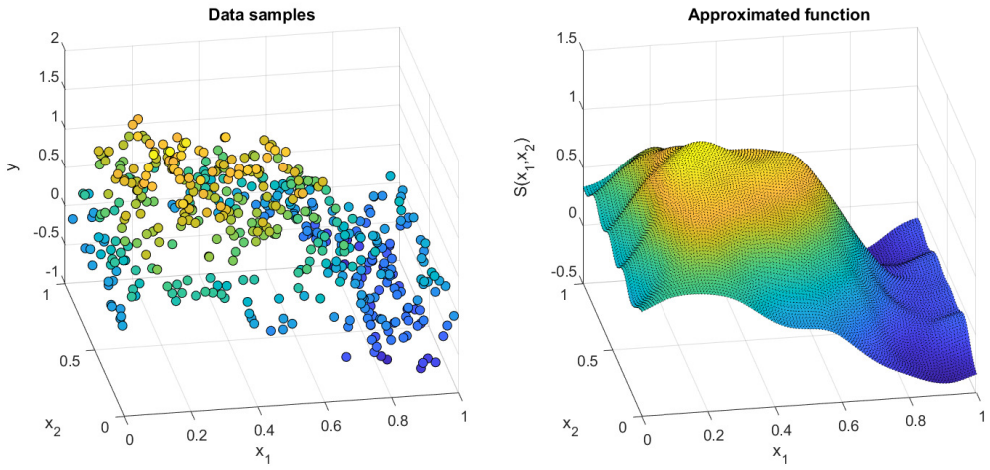



**Figure 3-10:** Cubic B-spline surface (right) fitted on data samples (left).

Reshaping $\boldsymbol{vec}(\mathcal{W})$ back into an order $d$ tensor gives the desired $\mathcal{W}$. Figure 3-10 illustrates an example of a cubic B-spline surface used to smooth out a noisy two-dimensional dataset.

### 3-3-4 Multivariate Regularization

The same reasoning for regularization in the univariate case applies to multivariate B-splines. For a high-dimensional B-spline surface, the volume of the domain of the function increases exponentially. This makes available data very sparse and scattered, which can lead to an ill-posed optimization problem. For multivariate functions, the P-spline penalty allows anisotropy, e.g. different penalties on the roughness for each dimension. To illustrate, we first consider the two-dimensional case. Given is the following B-spline surface:

$$S(x_1, x_2) = \langle \boldsymbol{B}, \boldsymbol{W} \rangle . \tag{3-59}$$

Assuming we have three B-spines per dimension, the weights are stored in a matrix:

$$\boldsymbol{W} = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix}.$$

The differences in weights have to be penalized along each dimension individually. The differences of the first dimension are given by:

$$\boldsymbol{D}_1\boldsymbol{W} = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} (w_{01} - w_{10}) & (w_{01} - w_{11}) & (w_{02} - w_{12}) \\ (w_{10} - w_{20}) & (w_{11} - w_{21}) & (w_{12} - w_{22}) \end{bmatrix}.$$

The penalty term is then given by:

$$R_1(\boldsymbol{W}) = \|\boldsymbol{D}_1\boldsymbol{W}\|_2^2 = trace\Big(\boldsymbol{W}^T\boldsymbol{D}_1^T\boldsymbol{D}_1\boldsymbol{W}\Big).$$

This penalty is shown in the graphical tensor notation in Figure 3-11a. Similarly, the penalty along the second dimension of the weights is given by:

$$R_2(\boldsymbol{W}) = \|\boldsymbol{D}_1\boldsymbol{W}^{\boldsymbol{T}}\|_2^2 = trace\Big(\boldsymbol{W}\boldsymbol{D}_1^T\boldsymbol{D}_1\boldsymbol{W}^T\Big).$$

This penalty is shown in Figure 3-11b. Incorporating these penalties in the cost function results in:

$$J(\boldsymbol{W}) = \|\boldsymbol{y} - \langle \boldsymbol{B}, \boldsymbol{W} \rangle\|_2^2 + \lambda_1\|\boldsymbol{D}_1\boldsymbol{W}\|_2^2 + \lambda_2\|\boldsymbol{D}_1\boldsymbol{W}^{\boldsymbol{T}}\|_2^2. \tag{3-60}$$

We can generalize this to arbitrary number of inputs. This is done by contracting the second index of the difference matrix $\boldsymbol{D}_\alpha$ with the dimension of the weight tensor $\mathcal{W}$ along which the penalty is applied, then taking the norm of the result. For a B-spline curve with $d$ inputs, the penalty on the $\alpha$-th order differences along the $j$-th dimension is given by:

$$R_j(\mathcal{W}) = \|\mathcal{W} \times_j^2 \boldsymbol{D}_\alpha\|_2^2 = \Big\langle (\mathcal{W} \times_j^2 \boldsymbol{D}_\alpha),\ (\mathcal{W} \times_j^2 \boldsymbol{D}_\alpha) \Big\rangle .$$

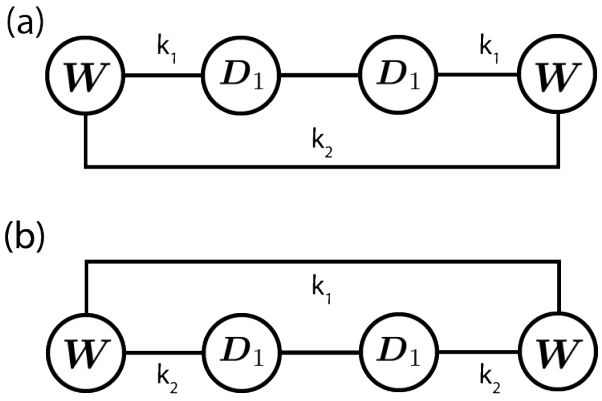



**Figure 3-11:** P-spline penalty for the (a) first and (b) second dimension of a two-dimensional B-spline surface.

The smoothing parameter $\lambda_p \geq 0$ controls the penalization of the roughness along dimension $j$. After adding the penalties, the cost function for fitting a multivariate B-spline surface becomes:

$$J(\mathcal{W}) = \|\boldsymbol{y} - \langle \mathcal{B}, \mathcal{W} \rangle\|_2^2 + \sum_{j=1}^{d} \lambda_j \|\mathcal{W} \times_j^2 \boldsymbol{D}_\alpha\|_2^2. \tag{3-61}$$

# Chapter 4

# Tensor Network B-splines

For multivariate B-splines, the B-spline tensor $\mathcal{B}$ is defined by a tensor product of univariate B-spline vectors $\boldsymbol{b}$. Therefore, the number of basis functions and control points increase exponentially with the number of dimensions, $\mathcal{O}((k)^d)$, i.e., multivariate B-spline surfaces suffer from the curse of dimensionality. For high dimensional data, it can quickly become computationally infeasible to store or operate on the tensors $\mathcal{B}$ and $\mathcal{W}$. Using tensor network theory, the multivariate B-spline surface can be represented in a low-parametric format. We first derive the TNBS model, then discuss how it can be used to approximate high-dimensional functions. Finally, we validate the proposed method though numerical experiments.

## 4-1 Model Structure

Without loss of generality, we derive the Tensor Network B-spline model structure using a three-dimensional B-spline surface as an example. Figure 4-1 will be used as a visual reference to walk through the derivation steps. Given are a weight tensor $\mathcal{W} \in \mathbb{R}^{k_1 \times k_2 \times k_3}$ and B-spline tensor $\mathcal{B} \in \mathbb{R}^{k_1 \times k_2 \times k_3}$. The three-dimensional B-spline hypersurface $S(x_1, x_2, x_3)$ can be represented as an inner product:

$$S(x_1, x_2, x_3) = \langle \mathcal{B}, \mathcal{W} \rangle . \tag{4-1}$$

The inner product is equal to the contraction over all pairs of indices. This is shown graphically in Figure 4-1a. As $\mathcal{B}$ is a rank one tensor, it can be decomposed into the tensor product of three B-spline vectors $\boldsymbol{b}^{(p)}$ as done in Figure 4-1b. Tensor products that close a loop in a tensor network are redundant and hence omitted in Figure 4-1c. Now $S(x_1, x_2, x_3)$ is simply the contraction of $\mathcal{W}$ with the B-spline basis vectors:

$$S(x_1, x_2, x_3) = \mathcal{W} \times_1^1 \boldsymbol{b}^{(1)} \times_2^1 \boldsymbol{b}^{(2)} \times_3^1 \boldsymbol{b}^{(3)}. \tag{4-2}$$

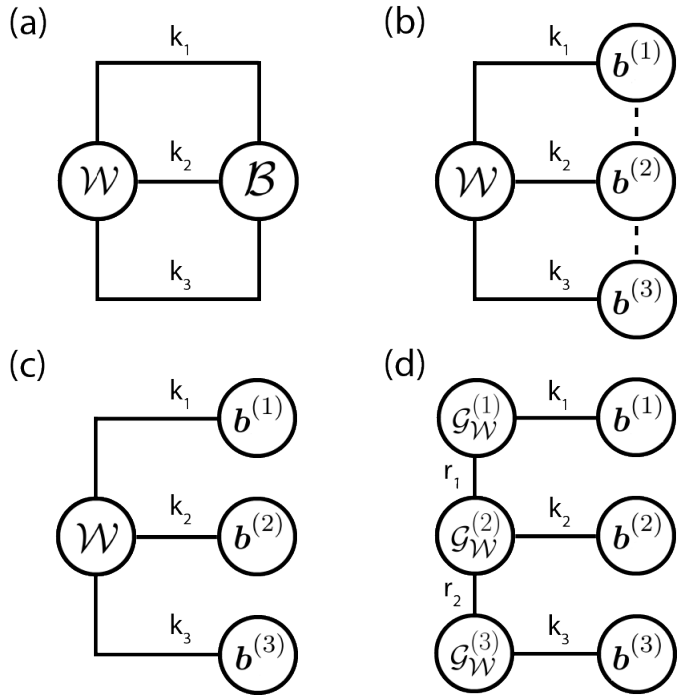



**Figure 4-1:** Derivation of the Tensor Network B-splines model of order 3.

Finally, $\mathcal{W}$ is decomposed into a tensor train, which gives us the tensor network representation of our hypersurface:

$$S(x_1, x_2, x_3) = (\mathcal{G}_{\mathcal{W}}^{(1)} \times_2^1 \boldsymbol{b}^{(1)})(\mathcal{G}_{\mathcal{W}}^{(1)} \times_2^1 \boldsymbol{b}^{(2)})(\mathcal{G}_{\mathcal{W}}^{(1)} \times_2^1 \boldsymbol{b}^{(3)}). \tag{4-3}$$

This is shown graphically Figure 4-1d. For a given input $(x_1, x_2, x_3)$, Eq. (4-3) is evaluated by constructing the B-spline vectors $\boldsymbol{b}^{(p)}$, contracting them with the corresponding tensor train cores and finally multiplying the sequence of resulting matrices. Due to the constraints, $r_0 = r_d = 1$, this results in a scalar output. Extending the TNBS model to $l$ outputs can be realized by removing one of these constraints, e.g. $r_0 = l$. In general, a $d$-dimensional TNBS surface is represented by:

$$S(x_1, x_2, \ldots, x_d) = \prod_{p=1}^{d} (\mathcal{G}_{\mathcal{W}}^{(p)} \times_2^1 \boldsymbol{b}^{(p)}). \tag{4-4}$$

## 4-2 Function Approximation

We can use the TNBS model to approximate high-dimensional functions in a computational efficient manner, by directly optimizing the TT-cores $\mathcal{G}_{\mathcal{W}}^{(p)}$. We first consider the unregularized case and incorporate P-spline regularization in the next section.

### 4-2-1 Fitting with ALS

We illustrate, without loss of generality, how to approximate functions with TNBS by means of the following example. Suppose we have the following three dimensional function:

$$y^i = f(x_1^i, x_2^i, x_3^i) + \varepsilon_i, \tag{4-5}$$

where $\varepsilon_i$ is Gaussian white noise. We are given a set of $N$ datasamples $\{(x_1^i, x_2^i, x_3^i, y^i)\}_{i=1}^N$. We approximate the function $f$ with the three-dimensional TNBS from Eq. (4-3), by minimizing the least-squared cost function:

$$\begin{aligned} &\min_{\mathcal{W}} \ \|\boldsymbol{y} - \boldsymbol{s}\|_2^2 \\ &\text{s.t. TT-rank}(\mathcal{W}) = (r_1, r_2), \end{aligned} \tag{4-6}$$

where

$$\boldsymbol{y} = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^N \end{bmatrix}, \qquad \boldsymbol{s} = \begin{bmatrix} S(x_1^1, x_2^1, x_3^1) \\ S(x_1^2, x_2^2, x_3^2) \\ \vdots \\ S(x_1^N, x_2^N, x_3^N) \end{bmatrix}.$$

We solve Eq. (4-6) directly for the TT-cores using ALS. The TT-ranks are chosen beforehand and the TT-cores are initialized randomly. Our strategy of choosing TT-ranks is by determining the upper bounds on $R_p$ with Eq. (2-12), then truncating them uniformly to a value $r$ such that:

$$r_p = \min(R_p, r). \tag{4-7}$$

This limits the number of hyperparameters that we need to tune. After initializing the TT-cores, ALS iteratively optimizes one tensor core at a time while holding the others fixed. Optimizing one core is equal to solving a small linear subsystem. Suppose that we wish to update the second core from Figure 4-1d. The idea is to contract everything in the network up until the nodes adjacent to $\mathcal{G}_{\mathcal{W}}^{(2)}$ (Figure 4-2a), whereupon Eq. (2-10) is used to rewrite the network as an inner product of two vectors (Figure 4-2b):

$$\begin{aligned} y &= \mathcal{G}_{\mathcal{W}}^{(2)} \times_1^1 \boldsymbol{v}_<^{(2)} \times_2^1 \boldsymbol{b}^{(2)} \times_3^2 \boldsymbol{v}_>^{(2)} \\ &= \mathcal{G}_{\mathcal{W}}^{(2)} \times_1^2 \boldsymbol{v}_<^{(2)T} \times_2^2 \boldsymbol{b}^{(2)T} \times_3^2 \boldsymbol{v}_>^{(2)} \\ &= \left(\boldsymbol{v}_>^{(2)T} \otimes \boldsymbol{b}^{(2)T} \otimes \boldsymbol{v}_<^{(2)}\right) \boldsymbol{vec}\left(\mathcal{G}_{\mathcal{W}}^{(2)}\right) \\ &= \boldsymbol{a}^{(2)T} \boldsymbol{g}^{(2)}. \end{aligned} \tag{4-8}$$

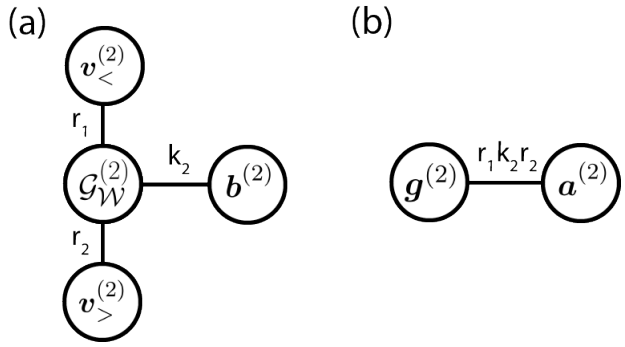



**Figure 4-2:** The tensor network written as a vector inner product.

More generally, rewriting Eq. (4-4) for the $i$-th data sample as a linear function of the elements of the $p$-th core gives:

$$y^i = \left(\boldsymbol{v}_{>,i}^{(p)T} \otimes \boldsymbol{b}_i^{(p)T} \otimes \boldsymbol{v}_{<,i}^{(p)}\right) \boldsymbol{vec}\left(\mathcal{G}_{\mathcal{W}}^{(p)}\right), \tag{4-9}$$

where

$$\boldsymbol{v}_{<,i}^{(p)} = \prod_{j=1}^{p-1} (\mathcal{G}_{\mathcal{W}}^{(j)} \times_2^1 \boldsymbol{b}_i^{(j)}) \in \mathbb{R}^{1 \times r_{p-1}},$$
$$\boldsymbol{v}_{>,i}^{(p)} = \prod_{j=p+1}^{d} (\mathcal{G}_{\mathcal{W}}^{(j)} \times_2^1 \boldsymbol{b}_i^{(j)}) \in \mathbb{R}^{r_p}.$$

for $2 \leq p \leq d-1$, and $\boldsymbol{v}_{<,i}^{(1)} = \boldsymbol{v}_{>,i}^{(d)} = 1$.

Computing Eq. (4-9) for all $N$ data samples results in a system of linear equations. The subproblem for updating the $p$-th core thus becomes:

$$\min_{\boldsymbol{g}^{(p)}} \left\| \boldsymbol{y} - \boldsymbol{A}^{(p)} \boldsymbol{g}^{(p)} \right\|_2^2, \tag{4-10}$$

where

$$\boldsymbol{A}^{(p)} = \begin{bmatrix} \boldsymbol{v}_{>,1}^{(p)T} \otimes \boldsymbol{b}_1^{(p)T} \otimes \boldsymbol{v}_{<,1}^{(p)} \\ \boldsymbol{v}_{>,2}^{(p)T} \otimes \boldsymbol{b}_2^{(p)T} \otimes \boldsymbol{v}_{<,2}^{(p)} \\ \vdots \\ \boldsymbol{v}_{>,N}^{(p)T} \otimes \boldsymbol{b}_N^{(p)T} \otimes \boldsymbol{v}_{<,N}^{(p)} \end{bmatrix}, \quad \boldsymbol{g}^{(p)} = \boldsymbol{vec}\left(\mathcal{G}_{\mathcal{W}}^{(p)}\right). \tag{4-11}$$

The optimum is found by solving the normal equation:

$$\left(\boldsymbol{A}^{(p)T} \boldsymbol{A}^{(p)}\right) \boldsymbol{g}^{(p)} = \boldsymbol{A}^{(p)T} \boldsymbol{y}. \tag{4-12}$$

Reshaping $\boldsymbol{g}^{(p)}$ back into a third-order tensor results in the updated core $\mathcal{G}_{\mathcal{W}}^{(p)}$. The ALS algorithm sweeps back and forth, iterating from the first to the last core and back, until

convergence. At each iteration, Eq. (4-12) is solved for $\boldsymbol{g}^{(p)}$. Numerical stability is ensured by keeping the tensor train in site-p-mixed-canonical form through an additional orthogonalization step. To illustrate, consider again the TNBS in Figure 4-1d. Assume that we are iterating from left to right and the tensor train is in site-2-mixed-canonical form. After solving $\boldsymbol{g}^{(2)}$ it is reshaped into a matrix $\boldsymbol{G}^{(2)} \in \mathbb{R}^{r_{p-1}k_p \times r_p}$, which is then decomposed through a QR decomposition. Finally, $Q$ is reshaped back into a third-order left-orthogonal tensor $\mathcal{G}_{\mathcal{W}}^{(2)}$ and $R$ is contracted with the next core. The tensor train is now in site-3-mixed-canonical form, and the next iteration starts. More details about the orthogonalization step is given in [71]. The optimization with ALS converges monotonously, so a possible stopping criterion is:

$$\left\| J_h^{(1)} - J_{h+1}^{(1)} \right\|_2 \leq \epsilon, \tag{4-13}$$

where $J_h^{(1)}$ is the cost of the objective function in Eq. (4-10) during the first core update of the $h$-th sweep.

### 4-2-2 Regularization

The low-rank approximation of the weight tensor does not only decrease the computational burden, but additionally serves as a regularization mechanism. The regularization from low-rank constraints is however not sufficient for high-dimensional B-splines. Like noted before, the available estimation data for high-dimensional functions becomes extremely sparse and scattered. We therefore wish to extend the P-spline regularization in Eq. (4-17) to the TNBS format. Some work on penalties in the tensor train format is done in [50] and [57].

Without loss of generality, Figure 4-3 visualizes in graphical notation the necessary steps for including P-spline regularization when fitting a three-dimensional TNBS surface. Recall that in the multivariate case, penalizing the differences in adjacent weights along the $j$-th dimension is done by contracting the second index of the difference matrix $\boldsymbol{D}_\alpha$ with the $j$-th index of the weight tensor $\mathcal{W}$, then taking the norm of the result:

$$\begin{aligned} R_j(\mathcal{W}) &= \|\mathcal{W} \times_j^2 \boldsymbol{D}_\alpha\|_2^2 \\ &= \left\langle (\mathcal{W} \times_j^2 \boldsymbol{D}_\alpha),\ (\mathcal{W} \times_j^2 \boldsymbol{D}_\alpha) \right\rangle. \end{aligned} \tag{4-14}$$

This is illustrated in 4-3a, where the penalty is applied along the first dimension, e.g. $j = 1$. Decomposing $\mathcal{W}$ into a tensor train results in the network depicted in Figure 4-3b. To write this penalty again as a linear function of the $p$-th core, we contract everything in the network except these cores (Figure 4-3c). In this example, $\boldsymbol{C}_{>,1}^{(2)}$ and $\boldsymbol{C}_{-,1}^{(2)}$ are simply identity matrices.

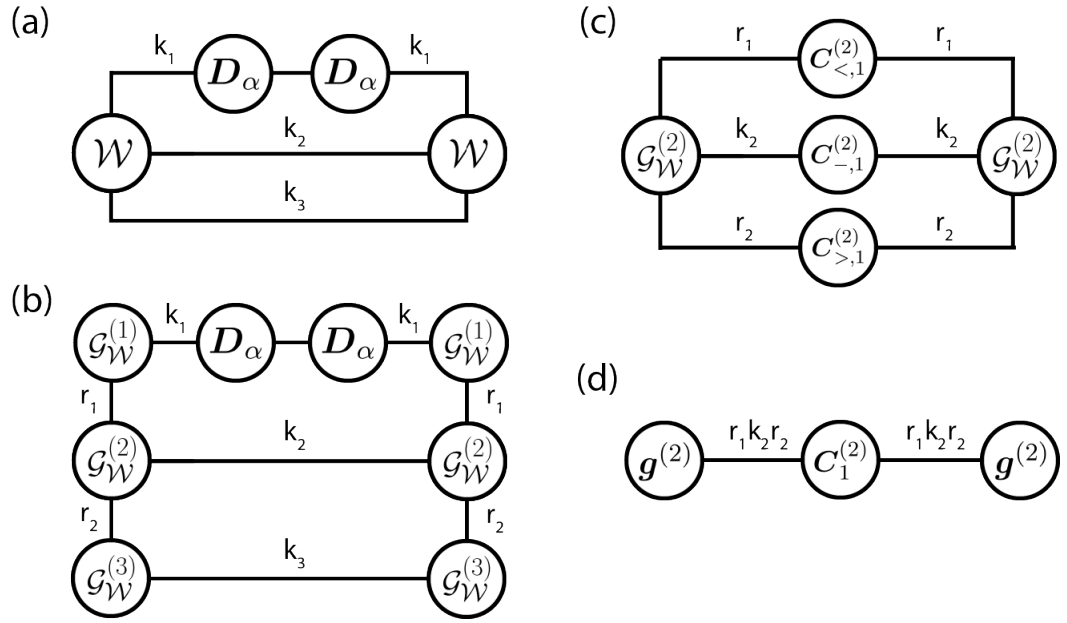



**Figure 4-3:** Derivation of the Tensor Network P-spline penalty.

Then, using Eq. (2-10), the penalty function can be rewritten in the form of Figure 4-3d:

$$R_j\left(\mathcal{G}_{\mathcal{W}}^{(p)}\right) = \boldsymbol{g}^{(p)T}\boldsymbol{C}_j^{(p)}\boldsymbol{g}^{(p)}, \tag{4-15}$$

where

$$\boldsymbol{C}_j^{(p)} = \left(\boldsymbol{C}_{>,j}^{(p)} \otimes \boldsymbol{C}_{-,j}^{(p)} \otimes \boldsymbol{C}_{<,j}^{(p)}\right). \tag{4-16}$$

The matrix $\boldsymbol{C}_j^{(p)}$ in Eq. (4-16) is constructed for every dimension $j$. Due to the site-p-mixed-canonical form of the tensor train, the contraction of two out of the three matrices $\boldsymbol{C}_{>,j}^{(p)}$, $\boldsymbol{C}_{-,j}^{(p)}$ and $\boldsymbol{C}_{<,j}^{(p)}$ result in identity matrices. This knowledge can be utilized for efficient implementation. Adding the penalties to the TNBS cost function results in the following regularized optimization problem:

$$\begin{aligned} \min_{\mathcal{W}} \quad & \|\boldsymbol{y} - \boldsymbol{s}\|_2^2 + \sum_{j=1}^{d} \lambda_j \, \|\mathcal{W} \times_j^2 \boldsymbol{D}_\alpha\|_2^2 \\ \text{s.t.} \quad & \text{TT-rank}(\mathcal{W}) = (r_1, r_2, \ldots, r_{d-1}). \end{aligned} \tag{4-17}$$

This problem can again be solved iteratively using the ALS method. The subproblem for updating the $p$-th core becomes:

$$\min_{\boldsymbol{g}^{(p)}} \quad \|\boldsymbol{y} - \boldsymbol{A}^{(p)}\boldsymbol{g}^{(p)}\|_2^2 + \sum_{j=1}^{d} \lambda_j \, \boldsymbol{g}^{(p)T}\boldsymbol{C}_j^{(p)}\boldsymbol{g}^{(p)}. \tag{4-18}$$

The normal equation is then:

$$\left(\boldsymbol{A}^{(p)T}\boldsymbol{A}^{(p)} + \sum_{j=1}^{d} \lambda_j \, \boldsymbol{C}_j^{(p)}\right) \boldsymbol{g}^{(p)} = \boldsymbol{A}^{(p)T}\boldsymbol{y}. \tag{4-19}$$

## 4-3 Complexity

Table 4-1 summarizes relevant computational complexities concerning function approximation using TNBS and ALS. While the complexities scale only linearly in the dimensions, it is important to realize that high TT-ranks easily degrade the performance of the optimization of the cores. Some functions might require high TT-ranks to be represented accurately. For example, functions that have highly complex surfaces will require large number of B-splines per dimension $k$. In turn, due to Eq. (2-12) it is likely that high $k$ results in high TT-rank requirements. Another observation from our experiments is that high TT-ranks increase the number of sweeps ALS requires until convergence. There is, therefore, a tradeoff between accuracy and speed when choosing the number of B-splines $k$. The construction of the B-spline vector $\boldsymbol{b}^{(p)}$ is very efficient and only has to be done once for every data sample. The number of data samples $N$ appears linearly in the complexities but may become a bottleneck for large datasets. A modification for this scenario is to only use a small random batch of the data when updating $\boldsymbol{g}^{(p)}$. This can speed up estimation time without significant loss of accuracy.

**Table 4-1:** Computational complexities of significant operations

| Operation | Complexity |
|---|---|
| Construct $\left\{\boldsymbol{b}_n^{(p)}\right\}_{i=1}^{N}$ | $O\left(Nn^2\right)$ |
| Construct $\left\{\boldsymbol{C}_j^{(p)}\right\}_{j=1}^{d}$ | $O\left((d+(m-\rho)^4)r^4\right)$ |
| Construct $\boldsymbol{A}^{(p)}$ | $O\left(N(m-\rho)r^2\right)$ |
| Solve $\boldsymbol{g}^{(p)}$ | $O\left(N(m-\rho)^2r^4+(m-\rho)^3r^6\right)$ |
| Evaluate $f$ | $O\left((\rho^2+(m-\rho)r^2)d\right)$ |

## 4-4 Experiments

In this section, we validate our work though several numerical experiments on synthetic datasets. The proposed regularized optimization procedure is implemented in MATLAB and executed on a personal computer with a 4.2 GHz Intel Core i5-7600K processor and 16 GB of random access memory (RAM). An open-source MATLAB implementation can be found at https://github.com/Ridvanz/Tensor-Network-B-splines.

### 4-4-1 Two-dimensional Function Approximation

First, we approximate a highly nonlinear two-dimensional function from relatively sparse data. This experiment allows us to visually verify the correctness of our algorithm. The function we will approximate is a scaled and shifted version of the Rastrigin function:

$$f(x_1,x_2)=\frac{1}{5}+\sum_{i=1}^{2}\left((10x_i-0.5)^2-\frac{1}{10}\cos(2\pi(10x_i-0.5))\right) \tag{4-20}$$

This function will be approximated over the domain $x_1, x_2 \in [0, 1]$, which is shown in Figure 4-4. We randomly sample 800 points from this surface, of which 400 are used to reconstruct the

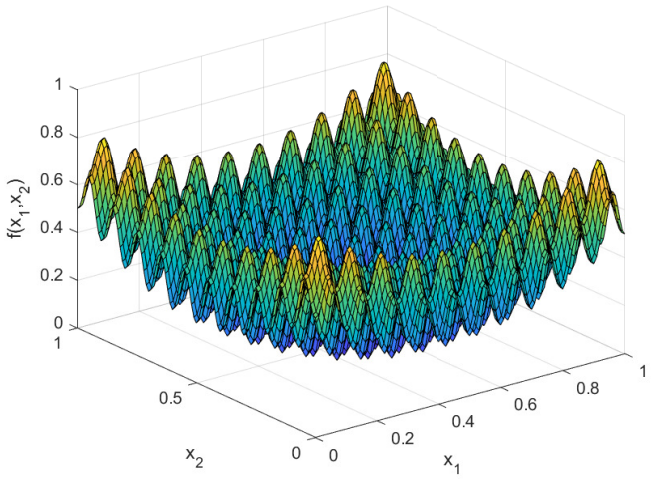



**Figure 4-4:** A scaled and shifted version of the Rastrigin function.

original function using the regularized TNBS model and the other 400 for testing. The degree of the B-splines is chosen as $\rho = 3$ uniformly and the number of knots per dimension $m = 103$. This gives us $k = 100$ B-splines along each dimension and $10^4$ multivariate B-splines in total. We place a penalty only on the second difference, i.e. $\alpha = 2$. The TT-ranks are chosen uniformly as $r = 3$ and the TT-cores are initialized randomly with uniformly distributed values from the interval $[0, 1]$. Subsequently, each core is normalized to unit norm. We first tune the regularization parameters manually to get a good approximation of the original function. The TT-cores are optimized for 50 sweeps in 0.2 seconds on a standard desktop computer. The total cost converges monotonously as seen in Figure 4-5. The experiment is

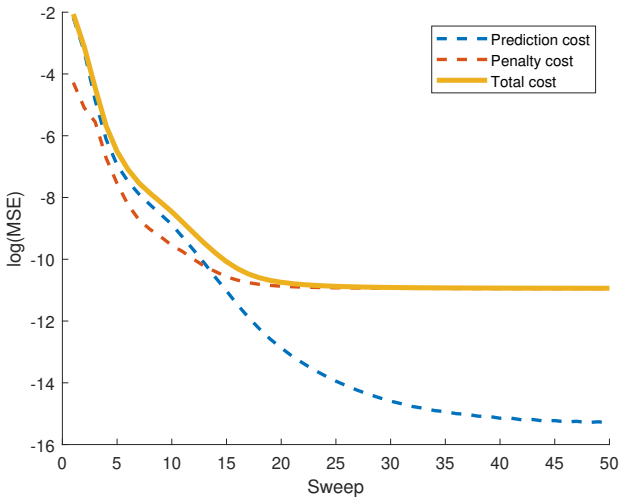



**Figure 4-5:** Costs during the optimization with ALS.

then repeated for different combinations of $[\lambda_1, \lambda_2]$. We increase the regularization parameters to large values to make sure the P-spline regularization induces smoothness. The results are shown in Figure 4-6. The TNBS approximates the surface closely for $\lambda_1 = \lambda_2 = 0.01$. As expected, increasing $\lambda_j$ for one dimension smooths the surface along that dimension. We can conclude that the algorithm is implemented correctly.

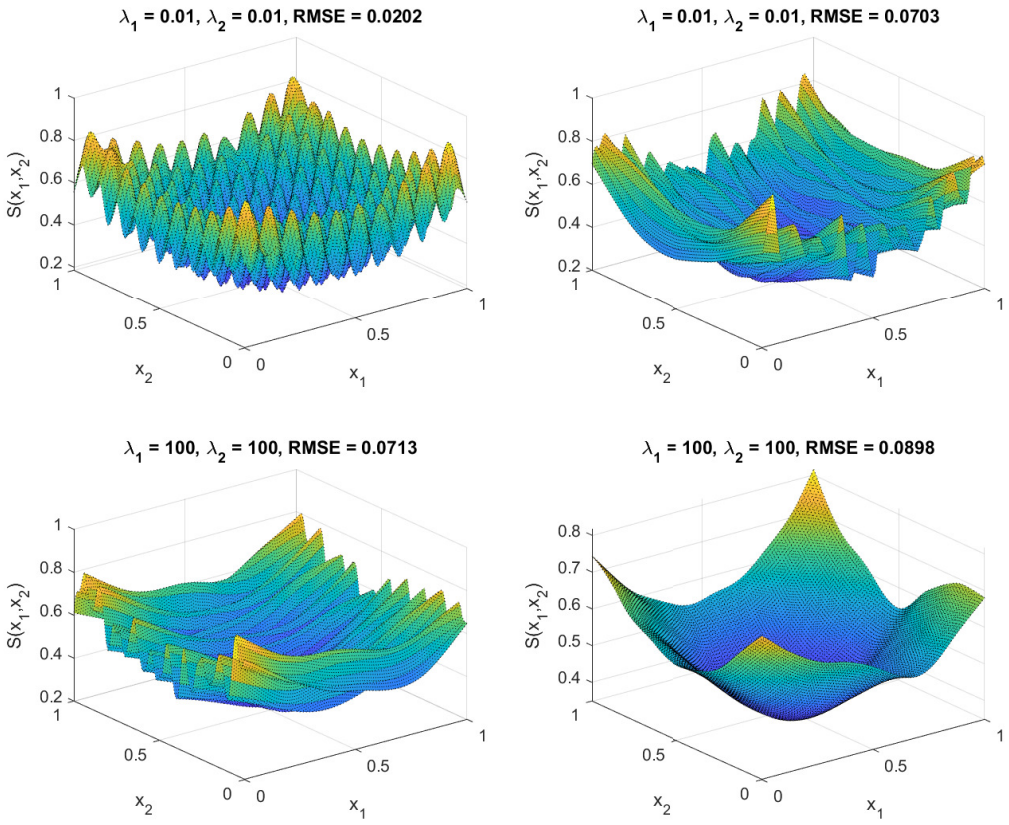



**Figure 4-6:** Results of the experiment for different values of $\lambda_1$ and $\lambda_2$.

### 4-4-2 High-dimensional Function Approximation

The next test measures how the performance of our algorithm scales to higher-dimensional functions. The function we will use is a sum of phase shifted sinusoids:

$$f(x_1, x_2, \ldots, x_d) = \sqrt{\frac{2}{d}} \sum_{p=1}^{d} \sin(2\pi x_p - p). \tag{4-21}$$

This function is designed to have zero mean and unit variance over the domain $x_1, x_2, \ldots, x_d \in [0, 1]$ for any number of inputs $d$. This allows easier comparison of performance using the RMSE metric. We will approximate Eq. (4-21) for an increasing number of dimensions. The regularization parameter $\lambda$ will be chosen equal for each dimension, e.g. $\lambda_j = \lambda \, \forall j$, and the penalty is placed on the second differences, e.g. $\alpha = 2$. The value for $\lambda$ is determined heuristically. We first tune for the univariate case, which results in $\lambda = 10^{-4}$. We find that we achieve reasonable performance when scaling $\lambda$ as follows:

$$\lambda = \frac{10^{-}4}{(k - \alpha)^d}. \tag{4-22}$$

The reasoning behind this heuristic is that the number of elements of the penalty term in Eq. (4-14) increases exponentially with the number of dimensions, while the least square cost stays approximately constant. The factor $(k-\alpha)^{-d}$ balances the tradeoff between smoothness and least square fit in higher dimensions. The number of B-splines $k$ indirectly influences the required TT-ranks and the number of sweeps for convergence, so we choose a low value, $k = 5$. The degrees of the B-splines are chosen $\rho = 3$. First, we want to determine a good value for the uniform TT-ranks $r$. We sample an estimation and test dataset from the function, both of size $N = 1000$. For different values of $r$ and number of inputs $d$ we fit a TNBS to the data samples. We use Eq. (4-13) as a stopping criterion and additionally specify a maximum number of sweeps of 200. The results are plotted in Figure 4-7.

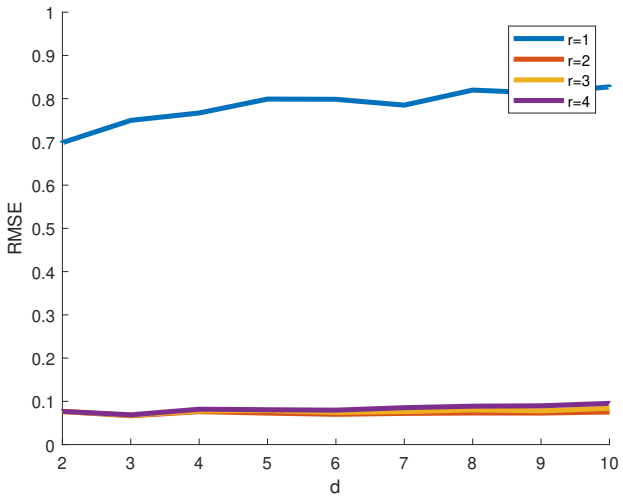



**Figure 4-7:** Results of the experiment for different values of $r$ and $d$.

To put these scores into perspective, note that for an unbiased estimator, the RMSE is equal to the square of the variance of the function. The variance of our uniformly sampled data points is 1 and their mean is 0. Therefore, an RMSE of 1 can be achieved with a function that always outputs zero. A score smaller than 1 means the approximation was able to find relevant patterns in the data. For RMSE values smaller than 0.1, the approximation is practically identical to the original function. A TNBS with TT-ranks of 1 appears to perform poorly for any $d$. Further inspection reveals that a TT-ranks of 1 constrains the B-spline weights to positive values. The TNBS model only approximates the positive half of the function as seen in Figure 4-8.

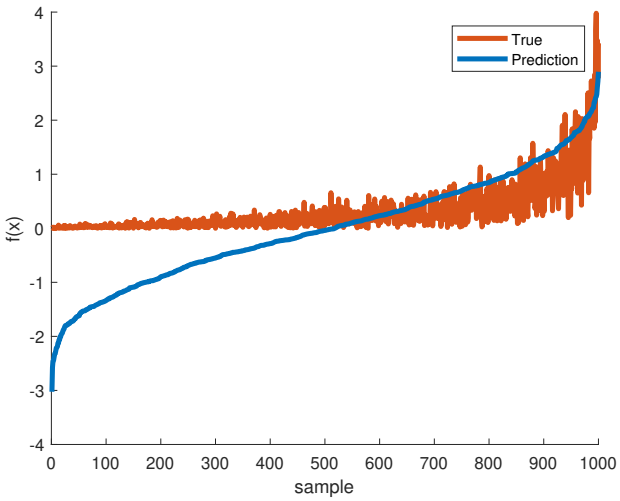



**Figure 4-8:** Test samples and the predicted values sorted ascending.

For $r = 2$, we achieve good approximations for the first ten values of $d$. Higher TT-ranks result in similar or worse performance, but do significantly increase optimization time. We therefore settle with $r = 2$ and further test the performance on higher dimensions with different number of samples $N$. Each optimization is executed 5 times. The averaged results are plotted in Figure 4-9. The left figure shows the RMSE scores while the right figure gives the time in seconds to train the models. From both graphs it is quite clear that the TNBS model scales well with the number of dimensions. Even for up to 20 inputs, the model is able to accurately approximate the function from relatively sparse data. Eventually, the performance degrades with the number of dimensions, but this simply indicates that more data is required. Even though the optimization time increases with the dimensions and the size of the data, it is still in the order of seconds. This is sufficiently fast for many function approximation applications.

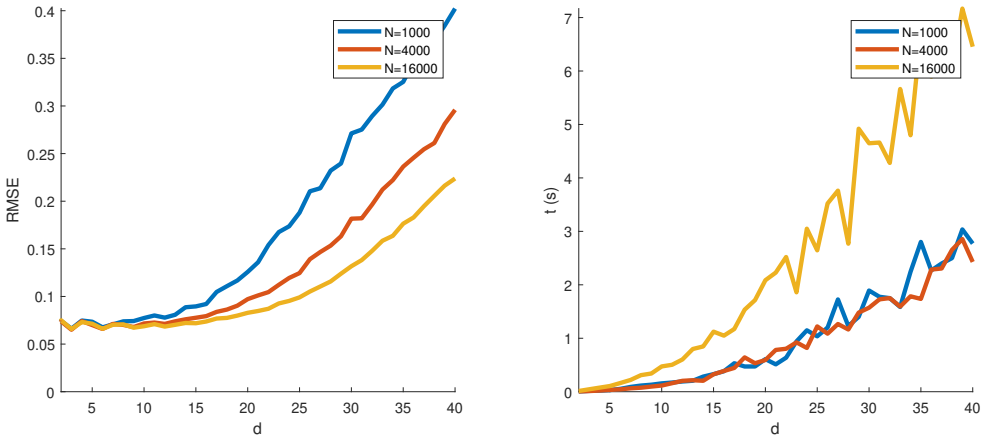



**Figure 4-9:** Results of the experiment for different values of $N$ and $d$. The RMSE scores are shown on the left and the optimization time in seconds is shown on the right.

# Chapter 5

# Paper

This chapter contains a preprint of a paper submitted to Automatica on March 17, 2020. Automatica is a leading archival publication in the field of systems and control. One particularly well-suited application of Tensor Network B-splines is black-box nonlinear system identification. The Nonlinear Autoregressive eXogenous (NARX) model [87] is able to represent a wide range of nonlinear systems and is useful when knowledge about the model structure of the system is limited. When the system under study has a large number of inputs and lags, the NARX approach requires the approximation of a high-dimensional function. The paper introduces the Tensor Network B-splines model and applies it to nonlinear system identification on a benchmark dataset using a NARX approach. A small remark: In this thesis, the univariate B-splines were indexed from 0 to $k-1$. In the paper, we have chosen to index them from 1 to $k$.

# Nonlinear system identification with regularized Tensor Network B-splines

Ridvan Karagoz [a], Kim Batselier [a],

[a] *DCSC, Delft University of Technology*

**Abstract**

This article introduces the Tensor Network B-spline model for the regularized identification of nonlinear systems using a nonlinear autoregressive exogenous (NARX) approach. Tensor network theory is used to alleviate the curse of dimensionality of multivariate B-splines by representing the high-dimensional weight tensor as a low-rank approximation. An iterative algorithm based on the alternating linear scheme is developed to directly estimate the low-rank tensor network approximation, removing the need to ever explicitly construct the exponentially large weight tensor. This reduces the computational and storage complexity significantly, allowing the identification of NARX systems with a large number of inputs and lags. The proposed algorithm is numerically stable, robust to noise, guaranteed to monotonically converge, and allows the straightforward incorporation of regularization. The TNBS-NARX model is validated through the identification of the cascaded watertank benchmark nonlinear system, on which it achieves state-of-the-art performance while identifying a 16-dimensional B-spline surface in 4 seconds on a standard desktop computer. An open-source MATLAB implementation is available on GitHub.

*Key words:* nonlinear system identification; NARX; B-splines; tensor network; curse of dimensionality.

## 1 Introduction

B-splines are basis functions for the spline function space [1], making them an attractive choice for approximating smooth continuous functions. For this reason, B-splines have had numerous applications in system identification [2,3,4,5,6,7,8] and control [9,10,11,12]. The generalization of B-splines to multiple dimensions is done through tensor products of their univariate basis functions. The number of basis functions and weights that define a multivariate B-spline surface, therefore, increase exponentially with the number of dimensions, i.e. B-splines suffer from the curse of dimensionality. Previous attempts to avoid this limitation include strategies such as dimensionality reduction, ANOVA decompositions and hierarchical structures [13]. The most effective method, i.e. hierarchical B-splines, relies on sparse grids [14] and reduces the storage complexity from $\mathcal{O}(k^d)$ to $\mathcal{O}(k\log(k)^{d-1})$ [15]. This is still exponential in the number of dimensions $d$. A recently emerging way to alleviate the curse of dimensionality is through the concept of tensor networks. Originally developed in the context of quantum physics, tensor networks efficiently represent high-dimensional tensors as a set of sparsely interconnected low-order tensors [16]. Combined with tensor algebra, tensor network structures can greatly decrease the computational complexity of many applications [17,18,19]. Due to their multilinear nature, multivariate B-splines easily admit a tensor network representation, which we call the Tensor Network B-splines (TNBS) model. Algorithms for optimization in the tensor network format make it possible to fit multivariate B-spline surfaces onto high-dimensional data by directly finding a low-rank tensor network approximation of the weight tensor, thereby overcoming the curse of dimensionality. This broadens the applicability of multivariate B-splines to high-dimensional problems that often occur in system identification and control. One particularly well-suited application of Tensor Network B-splines is black-box nonlinear system identification. The Nonlinear Autoregressive eXogenous (NARX) model [20] is able to represent a wide range of nonlinear systems and is useful when knowledge about the model structure of the system is limited. For the single-input–single-output (SISO) case, the discrete-time NARX model is expressed by the following nonlinear difference equation:

$$y_n = f(y_{n-1}, y_{n-2}, \ldots, u_n, u_{n-1}, u_{n-2}, \ldots) + \varepsilon_n. \quad (1)$$

The function $f$ is an unknown nonlinear mapping and $u_n$ and $y_n$ are the input and output samples at time

*Email addresses:* r.karagoz@hotmail.com (Ridvan Karagoz), K.Batselier@tudelft.nl (Kim Batselier).

step $n$. The error $\varepsilon_t$ is assumed to be Gaussian white noise. The most common models used in approximating $f$ are polynomials or neural networks [21]. The applicability of polynomial NARX is, however, often limited to weakly nonlinear systems due to computational complexity. Neural networks, on the other hand, require a lot of data to generalize well and can be time consuming to train. Under the reasonable assumption that $f$ is sufficiently smooth, the Tensor Network B-splines model is a suitable candidate to approximate the function from observed input and output data. The contributions of this paper are:

- Introduce the Tensor Network B-splines model.
- Present a regularized TNBS-NARX system identification algorithm.

The paper is structured as follows. Section 2 introduces relevant tensor and B-spline theory. Section 3 presents the TNBS model, the regularization technique and the NARX identification algorithm. Section 4 validates the TNBS-NARX approach through numerical experiments on a synthetic and a benchmark dataset. Section 5 concludes this paper and lists some recommendations.

## 2 Preliminaries

This section provides the basic terminology and definitions for tensors and tensor decompositions, followed by an introduction to B-splines. Most of the introduced tensor network definitions are based on [22,23,24,25]. A comprehensive treatment of B-splines is given in the book by de Boor [26].

### *2.1 Tensor basics*

A tensor is a multidimensional array of real numerical values, e.g. $\mathcal{A} \in \mathbb{R}^{k_1 \times k_2 \times \cdots \times k_d}$. Tensors can thus be considered generalizations of vectors and matrices. The order $d$ of the tensor is the number of dimensions of the array. Unless stated otherwise, subscript indices indicate a single element of a tensor, e.g. $a = \mathcal{A}_{i_1,i_2,\ldots,i_d}$. The size of each dimension is indicated by $k_p, p \in \{1, 2, \ldots, d\}$, such that $i_p \in \{1, 2, \ldots, k_p\}$. In this paper, scalars are denoted by lowercase letters ($a$), vectors are denoted by bold lowercase letters ($\boldsymbol{a}$), matrices are denoted by bold uppercase letters ($\boldsymbol{A}$) and higher-order tensors are denoted by calligraphic letters ($\mathcal{A}$). A convenient way of expressing tensors and their operations is using the graphical notation introduced by Roger Penrose in 1972 [23]. Figure 1 shows the representation of a scalar, vector, matrix and third-order tensor using this notation. Every node represents a tensor, the edges represent the indices and the number of edges, therefore, corresponds to its order. The vectorization of a tensor $\mathcal{A} \in \mathbb{R}^{k_1 \times k_2 \times \cdots \times k_d}$ is the reordering of its elements into a column vector, denoted by $\boldsymbol{vec}(\mathcal{A}) = \boldsymbol{a} \in \mathbb{R}^{k_1 k_2 \cdots k_d}$. The elements of $\boldsymbol{a}$ are denoted as:

$$\boldsymbol{a}_{i_1+(i_2-1)k_1+\ldots+(i_d-1)k_1k_2\ldots k_{d-1}} = \mathcal{A}_{i_1,i_2,\ldots,i_d}.$$

A tensor $\mathcal{T} \in \mathbb{R}^{k_1 \times k_2 \times \cdots \times k_d}$ is of rank one if it can be decomposed into the outer product of $d$ vectors $\boldsymbol{b}^{(p)} \in \mathbb{R}^{k_p}$, e.g:

$$\mathcal{T} = \boldsymbol{b}^{(1)} \circ \boldsymbol{b}^{(2)} \circ \cdots \circ \boldsymbol{b}^{(d)},$$

where $\circ$ denotes the outer product operation. The most essential operation in tensor algebra is contraction, which is the summing of elements over equal-sized indices. Given the tensors $\mathcal{A} \in \mathbb{R}^{k_1 \times k_2 \times k_3}$ and $\mathcal{B} \in \mathbb{R}^{k_3 \times k_4 \times k_5}$, contracting the index $i_3$ results in a tensor $\mathcal{A} \times_3^1 \mathcal{B} = \mathcal{C} \in \mathbb{R}^{k_1 \times k_2 \times k_4 \times k_5}$ whose elements are given by:

$$\mathcal{C}_{i_1,i_2,i_4,i_5} = \sum_{i_3} \mathcal{A}_{i_1,i_2,i_3}\, \mathcal{B}_{i_3,i_4,i_5}. \tag{2}$$

Contraction is indicated by the left-associative $\binom{m}{n}$-mode product operator [16], where $n$ and $m$ indicate the position of the indices of the first and second tensor respectively. In the graphical notation, contraction is indicated by connecting corresponding edges, as illustrated for (2) in Figure 2. An important equation [18] that relates contraction of a $d$-dimensional tensor with $d$ matrices to a linear operation is the following:

$$\begin{aligned} &\boldsymbol{vec}\left(\mathcal{A} \times_1^2 \boldsymbol{C}^{(1)} \times_2^2 \cdots \times_d^2 \boldsymbol{C}^{(d)}\right) \\ &= \left(\boldsymbol{C}^{(d)} \otimes \cdots \otimes \boldsymbol{C}^{(1)}\right) \boldsymbol{vec}(\mathcal{A}), \end{aligned} \tag{3}$$

where $\otimes$ denotes the Kronecker product. The outer product operation is a special case of contraction where the contracted indices have singleton dimensions. The outer product is depicted in the graphical notation by a dashed line connecting two nodes. The inner product between two equal-sized tensors is the sum of their entry-wise products, equivalent to contraction of the tensors over

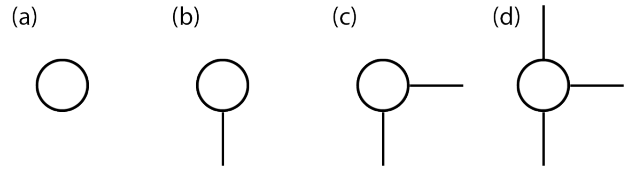



Fig. 1. Graphical notation of a (a) scalar, (b) vector, (c) matrix and (d) third-order tensor.

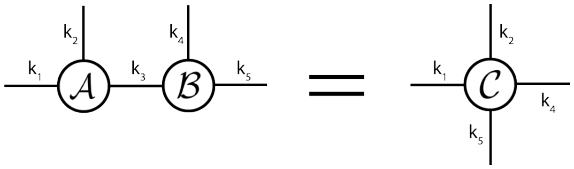



Fig. 2. Tensor contraction in graphical notation.

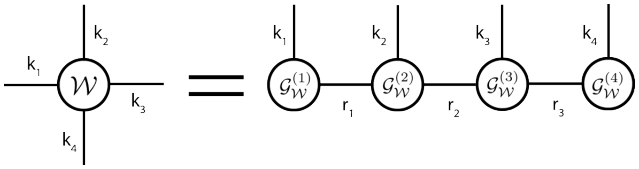



Fig. 3. Graphical notation of the tensor train decomposition for a fourth-order tensor.

all pairs of indices. Given two tensors $\mathcal{A} \in \mathbb{R}^{k_1 \times k_2 \times k_3}$ and $\mathcal{B} \in \mathbb{R}^{k_1 \times k_2 \times k_3}$, their inner product is given by:

$$\langle \mathcal{A}, \mathcal{B} \rangle = \sum_{i_1,i_2,i_3} \mathcal{A}_{i_1,i_2,i_3} \mathcal{B}_{i_1,i_2,i_3} = \boldsymbol{vec}(\mathcal{A})^T \boldsymbol{vec}(\mathcal{B}).$$

The Frobenius norm of a tensor is defined as the square root of the sum of squares of its entries:

$$\|\mathcal{A}\|_2 = \sqrt{\langle \mathcal{A}, \mathcal{A} \rangle}.$$

### 2.2 *Tensor trains*

The tensor train (TT) decomposition is a widely used tensor network format, popular for its low parametric format and the numerical stability of related optimization algorithms [25]. A tensor train expresses a tensor $\mathcal{W} \in \mathbb{R}^{k_1 \times k_2 \times \cdots \times k_d}$ of order $d$ in terms of third-order tensors $\mathcal{G}^{(p)}_{\mathcal{W}} \in \mathbb{R}^{r_{p-1} \times k_p \times r_p}$, also known as the TT-cores. Figure 3 shows the TT-decomposition of a four-dimensional tensor in graphical notation. The dimensions of the contracted indices, $r_p$, are called TT-ranks. The first and last TT-ranks, $r_0$ and $r_d$, are by definition equal to one. Keeping in mind that the $\binom{m}{n}$-mode product operator is left-associative, the tensor train in Figure 3 can be expressed as:

$$\mathcal{W} = \mathcal{G}^{(1)}_{\mathcal{W}} \times^1_2 \mathcal{G}^{(2)}_{\mathcal{W}} \times^1_3 \mathcal{G}^{(3)}_{\mathcal{W}} \times^1_4 \mathcal{G}^{(4)}_{\mathcal{W}}. \tag{4}$$

There exists a set of TT-ranks $r_p = R_p$ for which the decomposition is exact. When $r_p < R_p$, the tensor train represents an approximation of the original tensor. The lower the TT-ranks, the less accurate the decomposition, but the better the compression. When all $r_p$ and dimensions $k_p$ are equal, the storage complexity of the tensor train representation is $\mathcal{O}(kdr^2)$. A TT-decomposition with low TT-ranks can thus significantly reduce the memory footprint of high-dimensional data. For a prescribed set of TT-ranks or a prescribed accuracy, the TT-decomposition of a tensor can be computed with the TT-SVD [25] or the TT-Cross [27] algorithm. An important notion for TT-cores is orthogonality. A TT-core $\mathcal{G}^{(p)}_{\mathcal{W}}$ is left-orthogonal if it can be reshaped into a matrix $\boldsymbol{G}^{(p)} \in \mathbb{R}^{r_{p-1}k_p \times r_p}$ for which:

$$\boldsymbol{G}^{(p)T} \boldsymbol{G}^{(p)} = \boldsymbol{I}.$$

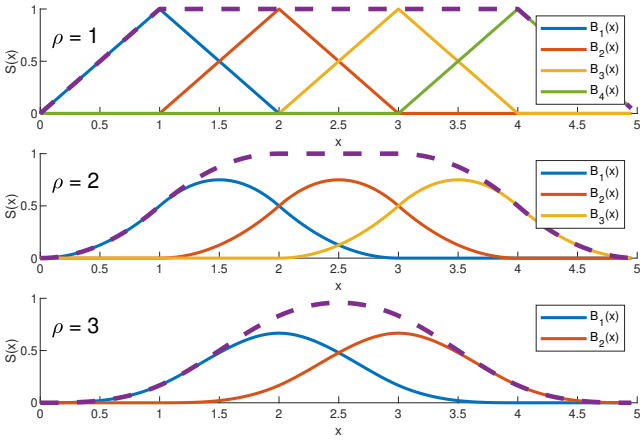



Fig. 4. Cardinal B-splines of degrees 1 to 3. The dashed purple lines represent the sum of the B-splines.

Likewise, $\mathcal{G}^{(p)}_{\mathcal{W}}$ is is right-orthogonal if it can be reshaped into a matrix $\boldsymbol{G}^{(p)} \in \mathbb{R}^{r_{p-1} \times k_p r_p}$ for which:

$$\boldsymbol{G}^{(p)} \boldsymbol{G}^{(p)T} = \boldsymbol{I}.$$

A tensor train is in site-k-mixed-canonical form [28] when for for its TT-cores the following applies:

$$\mathcal{G}^{(p)}_{\mathcal{W}} = \begin{cases} \text{left-orthogonal}, & 1 \leq p \leq k-1 \\ \text{right-orthogonal}, & k+1 \leq p \leq d. \end{cases} \tag{5}$$

For a site-k-mixed-canonical tensor train holds that its norm is contained in the $k$-th TT-core, i.e.:

$$\|\mathcal{W}\|_2 = \|\mathcal{G}^{(k)}_{\mathcal{W}}\|_2.$$

### 2.3 *B-splines*

A univariate spline $S$ is a piecewise polynomial function that maps values from an interval $[a, b]$ to the set of real numbers, e.g. $S : [a, b] \in \mathbb{R} \to \mathbb{R}$. Any spline of degree $\rho$ can be expressed as a unique linear combination of B-splines of the same degree:

$$S(x) = \sum_{i=1}^{k} B_i(x) w_i = \boldsymbol{b}^T \boldsymbol{w} \tag{6}$$

$$= \begin{bmatrix} B_{1,\rho}(x) & B_{2,\rho}(x) & \cdots & B_{k,\rho}(x) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix}. \tag{7}$$

The B-spline basis functions $B_{i,\rho}(x)$ are defined by the knot sequence and degree $\rho$, and they are

contained in the basis vector $\boldsymbol{b}$. A knot sequence $\boldsymbol{t} = \{t_0, t_1, \ldots, t_{m-1}, t_m\}$ is defined as a non-decreasing and finite sequence of real numbers that define the partitioning of the domain [a,b], i.e. $a = t_0 \leq t_1 \leq \cdots \leq t_{m-1} \leq t_m = b$, such that $S(x)$ is a polynomial on any interval $[t_i, t_{i+1}]$. The number of B-spline basis functions $k$ relates to the degree $\rho$ and number of knots $m+1$ by $k = m - \rho$. B-spline basis functions of arbitrary degree $\rho$ can be recursively constructed by means of the Cox-de Boor formula [26]:

$$B_{i,0}(x) = \begin{cases} 1 \text{ if } & t_{i-1} \leq x < t_i \\ 0 & \text{otherwise} \end{cases},$$
$$B_{i,\rho+1}(x) = \frac{x - t_{i-1}}{t_{i+\rho-1} - t_{i-1}} B_{i,\rho}(x) + \frac{t_{i+\rho} - x}{t_{i+\rho} - t_i} B_{i+1,\rho}(x). \tag{8}$$

If the knots are equidistantly distributed over the domain of the spline, the spline is called uniform. If the uniform knot sequence is also a subset of $\mathbb{Z}$, i.e. a sequence of integers, the spline is referred to as a cardinal spline [29]. In this article, all knot sequences will be considered uniform, as they allow for efficient evaluation of $\boldsymbol{b}$ using a matrix expression [30] instead of (8). Figure 4 illustrates B-splines of degree 1 to 3 on the cardinal knot sequence $\boldsymbol{t} = \{0, 1, 2, 3, 4, 5\}$. The dashed purple lines represent the sum of the basis functions. The shape of a B-spline curve $S(x)$ is only fully adjustable within its natural domain $\mathcal{D}_n = [t_\rho, t_{m-\rho}]$, because the sum of the B-spline basis functions at any point within this domain equals one. It is desirable to have full control over the shape of the B-spline curve over the whole range of data samples. The knot sequences in this article will be chosen such that $\mathcal{D}_n$ coincides with the unit interval $[0, 1]$.

### *2.4 Multivariate B-splines*

B-splines generalize to multiple input dimensions through tensor products of univariate basis functions. One can construct a $d$-dimensional spline $S$ as a linear combination of multivariate B-splines:

$$\begin{aligned} &S(x_1, x_2, \ldots, x_d) \\ &= \sum_{i_1=1}^{k_1} \sum_{i_2=1}^{k_2} \cdots \sum_{i_d=1}^{k_d} B_{i_1}(x_1) B_{i_2}(x_2) \cdots B_{i_d}(x_d) \mathcal{W}_{i_1 i_2 \cdots i_d} \\ &= \langle \mathcal{B}, \mathcal{W} \rangle . \end{aligned} \tag{9}$$

For notational convenience, we omitted the degrees $\rho$. The B-spline tensor $\mathcal{B}$ contains the multivariate basis functions and is defined as:

$$\mathcal{B} = \boldsymbol{b}^{(1)} \circ \boldsymbol{b}^{(2)} \circ \cdots \circ \boldsymbol{b}^{(d)},$$

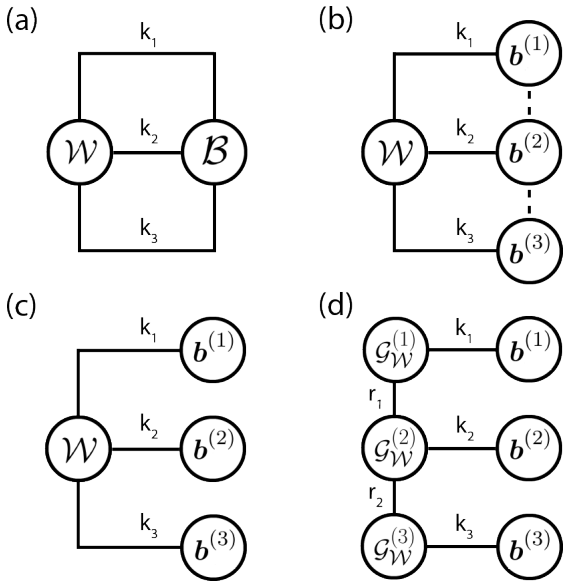



Fig. 5. Derivation of the Tensor Network B-splines model of order 3.

where $\boldsymbol{b}^{(p)}$ is the univariate basis vector of the $p$-th input variable, i.e.

$$\boldsymbol{b}^{(p)} = \Big[ B_{1,\rho}(x_p) \; B_{2,\rho}(x_p) \; \cdots \; B_{k_p,\rho}(x_p) \Big]^T. \tag{10}$$

We will assume equal knots and degree for each dimension, hence $k_p = k, \; \forall p$. The representation of B-spline surfaces in (9) is severely limited by the exponential increase in the number of basis functions and weights, $\mathcal{O}((k)^d)$.

## 3 Tensor Network B-splines

For our purposes, the input variables $x_p$ are the lagged inputs and outputs of (1). For a large number of lags or inputs, it can therefore quickly become computationally infeasible to store or operate on the tensors $\mathcal{B}$ and $\mathcal{W}$. Using tensor network theory, the multivariate B-spline surface can be represented in a low-parametric format. In this section, we derive the TNBS model and use it to approximate the function $f$ in (1) from observed input and output data.

### *3.1 Model structure*

We illustrate the model structure using a three-dimensional Tensor Network B-spline surface as an example, which is derived as follows:

$$
\begin{aligned}
&S(x_1, x_2, x_3) \\
&\quad = \langle \mathcal{B}, \mathcal{W} \rangle && (11)\\
&\quad = \mathcal{W} \times_1^1 \boldsymbol{b}^{(1)} \times_2^1 \boldsymbol{b}^{(2)} \times_3^1 \boldsymbol{b}^{(3)} && (12)\\
&\quad = (\mathcal{G}_{\mathcal{W}}^{(1)} \times_2^1 \boldsymbol{b}^{(1)})(\mathcal{G}_{\mathcal{W}}^{(1)} \times_2^1 \boldsymbol{b}^{(2)})(\mathcal{G}_{\mathcal{W}}^{(1)} \times_2^1 \boldsymbol{b}^{(3)}). && (13)
\end{aligned}
$$

Figure 5 will be used as a visual reference to walk through these equations. Given the weight tensor $\mathcal{W} \in \mathbb{R}^{k_1 \times k_2 \times k_3}$ and B-spline tensor $\mathcal{B} \in \mathbb{R}^{k_1 \times k_2 \times k_3}$ in (11), their inner product is equal to the contraction over all pairs of indexes, as seen in Figure 5a. As $\mathcal{B}$ is a rank one tensor, it can be decomposed into the outer product of three B-spline vectors $\boldsymbol{b}^{(p)}$ (Figure 5b). The outer product operation is a special case of contraction where the contracted indexes have singleton dimensions. Singleton contractions that close a loop in a tensor network are redundant, and hence omitted in Figure 5c. Now $S(x_1, x_2, x_3)$ in (12) is simply the contraction of $\mathcal{W}$ with the B-spline basis vectors. Finally, $\mathcal{W}$ is decomposed into a tensor train in Figure 5d. A point $(x_1, x_2, x_3)$ on the TNBS surface in (13) is evaluated by constructing the B-spline vectors $\boldsymbol{b}^{(p)}$, contracting them with the corresponding tensor train cores and finally multiplying the sequence of resulting matrices. Due to the constraints, $r_0 = r_d = 1$, this results in a scalar output. Extending the TNBS model to $l$ outputs can be realized by removing one of these constraints, e.g. $r_0 = l$. In general, a $d$-dimensional TNBS surface is represented by:

$$
S(x_1, x_2, \ldots, x_d) = \prod_{p=1}^{d} (\mathcal{G}_{\mathcal{W}}^{(p)} \times_2^1 \boldsymbol{b}^{(p)}). \tag{14}
$$

### *3.2 Identification algorithm*

We illustrate, without loss of generality, the proposed identification algorithm by means of the following example. Suppose we have the following NARX system model:

$$
y_n = f(u_n, y_{n-1}, u_{n-1}) + \varepsilon_n. \tag{15}
$$

We want to identify this model from a set of observed input and output data $\{(y_n, u_n)\}_{n=1}^{N}$. We approximate the function $f$ with the three-dimensional TNBS from Figure 5d, by minimizing the least-squared cost function:

$$
\begin{aligned}
&\min_{\mathcal{W}} \ \|\boldsymbol{y} - \boldsymbol{s}\|_2^2 && (16)\\
&\text{s.t. TT-rank}(\mathcal{W}) = (r_1, r_2),
\end{aligned}
$$

where

$$
\boldsymbol{y} = \begin{bmatrix} y_2 \\ y_3 \\ \vdots \\ y_N \end{bmatrix}, \qquad \boldsymbol{s} = \begin{bmatrix} f(u_2, y_1, u_1) \\ f(u_3, y_2, u_2) \\ \vdots \\ f(u_N, y_{N-1}, u_{N-1}) \end{bmatrix}.
$$

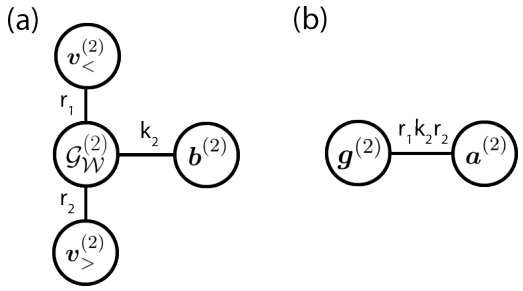



Fig. 6. The tensor network written as a vector inner product.

To solve (16) directly for the TT-cores, we use the alternating linear scheme (ALS) [31]. The TT-ranks are chosen beforehand and the TT-cores are initialized randomly. ALS then iteratively optimizes one tensor core at a time while holding the others fixed. Optimizing one core is equal to solving a small linear subsystem. Suppose we wish to update the second core from Figure 5d. The idea is to contract everything in the network up until the nodes adjacent to $\mathcal{G}_{\mathcal{W}}^{(2)}$ (Figure 6a), whereupon (3) is used to rewrite the network as an inner product of two vectors (Figure 6b):

$$
\begin{aligned}
y_n &= \mathcal{G}_{\mathcal{W}}^{(2)} \times_1^1 \boldsymbol{v}_<^{(2)} \times_2^1 \boldsymbol{b}^{(2)} \times_3^2 \boldsymbol{v}_>^{(2)} \\
&= \mathcal{G}_{\mathcal{W}}^{(2)} \times_1^2 \boldsymbol{v}_<^{(2)T} \times_2^2 \boldsymbol{b}^{(2)T} \times_3^2 \boldsymbol{v}_>^{(2)} \\
&= \left(\boldsymbol{v}_>^{(2)T} \otimes \boldsymbol{b}^{(2)T} \otimes \boldsymbol{v}_<^{(2)}\right) \boldsymbol{vec}\left(\mathcal{G}_{\mathcal{W}}^{(2)}\right) \\
&= \boldsymbol{a}^{(2)T} \boldsymbol{g}^{(2)}.
\end{aligned} \tag{17}
$$

More generally, rewriting (14) for the $n$-th data sample as a linear function of the elements of the $p$-th core gives:

$$
y_n = \left(\boldsymbol{v}_{>,n}^{(p)T} \otimes \boldsymbol{b}_n^{(p)T} \otimes \boldsymbol{v}_{<,n}^{(p)}\right) \boldsymbol{vec}\left(\mathcal{G}_{\mathcal{W}}^{(p)}\right), \tag{18}
$$

where

$$
\begin{aligned}
\boldsymbol{v}_{<,n}^{(p)} &= \prod_{j=1}^{p-1} (\mathcal{G}_{\mathcal{W}}^{(j)} \times_2^1 \boldsymbol{b}_n^{(j)}) \in \mathbb{R}^{1 \times r_{p-1}} \\
\boldsymbol{v}_{>,n}^{(p)} &= \prod_{j=p+1}^{d} (\mathcal{G}_{\mathcal{W}}^{(j)} \times_2^1 \boldsymbol{b}_n^{(j)}) \in \mathbb{R}^{r_p}
\end{aligned}
$$

for $2 \leq p \leq d-1$, and $\boldsymbol{v}_{<,n}^{(1)} = \boldsymbol{v}_{>,n}^{(d)} = 1$. Computing (18) for all $N$ data samples results in a system of linear equations. The subproblem for updating the $p$-th core thus becomes:

$$
\min_{\boldsymbol{g}^{(p)}} \ \left\| \boldsymbol{y} - \boldsymbol{A}^{(p)} \boldsymbol{g}^{(p)} \right\|_2^2, \tag{19}
$$

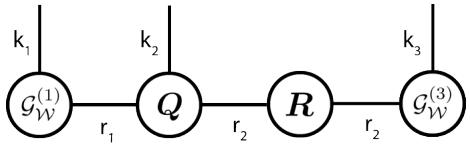



Fig. 7. QR decomposition of the second core during a left to right sweep.

where

$$\boldsymbol{A}^{(p)} = \begin{bmatrix} \boldsymbol{v}_{>,1}^{(p)T} \otimes \boldsymbol{b}_1^{(p)T} \otimes \boldsymbol{v}_{<,1}^{(p)} \\ \boldsymbol{v}_{>,2}^{(p)T} \otimes \boldsymbol{b}_2^{(p)T} \otimes \boldsymbol{v}_{<,2}^{(p)} \\ \vdots \\ \boldsymbol{v}_{>,N}^{(p)T} \otimes \boldsymbol{b}_N^{(p)T} \otimes \boldsymbol{v}_{<,N}^{(p)} \end{bmatrix}, \quad \boldsymbol{g}^{(p)} = \boldsymbol{vec}\left(\mathcal{G}_{\mathcal{W}}^{(p)}\right). \tag{20}$$

The optimum is found by solving the normal equation:

$$\left(\boldsymbol{A}^{(p)T}\boldsymbol{A}^{(p)}\right)\boldsymbol{g}^{(p)} = \boldsymbol{A}^{(p)T}\boldsymbol{y}. \tag{21}$$

Reshaping $\boldsymbol{g}^{(p)}$ back into a third-order tensor results in the updated core $\mathcal{G}_{\mathcal{W}}^{(p)}$. The ALS algorithm sweeps back and forth, iterating from the first to the last core and back, until convergence. At each iteration, (21) is solved for $\boldsymbol{g}^{(p)}$. Numerical stability is ensured by keeping the tensor train in site-p-mixed-canonical form through an additional orthogonalization step. To illustrate, consider again the TNBS in Figure 5d. Assume that we are iterating from left to right and the tensor train is in site-2-mixed-canonical form. After solving $\boldsymbol{g}^{(p)}$ it is reshaped into a matrix $\boldsymbol{G}^{(p)} \in \mathbb{R}^{r_{p-1}k_p \times r_p}$, which is then decomposed through a QR decomposition. The tensor network is now in the form of Figure 7. Finally, $Q$ is reshaped back into a third-order left-orthogonal tensor $\boldsymbol{G}^{(2)}$ and $R$ is contracted with the next core. The tensor train is now in site-3-mixed-canonical form, and the next iteration starts. More details about the orthogonalization step are given in [31]. The optimization with ALS converges monotonously, so a possible stopping criterion is:

$$\left\| J_h^{(1)} - J_{h+1}^{(1)} \right\|_2 \leq \epsilon, \tag{22}$$

where $J_h^{(1)}$ is the cost of the objective function in (19) during the first core update of the $h$-th sweep. A modified version of ALS method, MALS [31], updates two cores simultaneously and is computationally more expensive, but is able to adaptively determine the optimal TT-ranks for a specified accuracy. Another adaptive method is the tensor network Kalman filter [17], which can be used for online optimization of the cores.

### *3.3 Regularization*

In addition to decreasing computational burden, the TT-rank constraints serve as a regularization mechanism. This regularization is however insufficient for high-dimensional B-splines, as the volume of the domain of the TNBS increases exponentially. The available estimation data becomes sparse and scattered, which can lead to an ill-posed optimization problem. B-spline curves inherently possess the ability to regularize by adjustment of their degree or knot placement. The choice of knots has been a subject of much research [32], but due to lack of an attractive all-purpose scheme, we opt for a non-parametric approach known as P-splines [33]. P-splines induce smoothness by combining uniform B-splines with a discrete penalty placed on the $\alpha$-th difference between adjacent weights. For univariate splines, the following penalty function is added to the cost function:

$$R(\boldsymbol{w}) = \|\boldsymbol{D}_\alpha \boldsymbol{w}\|_2^2. \tag{23}$$

The matrix $\boldsymbol{D}_\alpha \in \mathcal{R}^{(k+1-\alpha)\times(k+1)}$ is the $\alpha$-th order difference matrix such that $\boldsymbol{D}_\alpha \boldsymbol{w} = \Delta^\alpha \boldsymbol{w}$ results in a vector of $\alpha$-th order differences of $\boldsymbol{w}$. This matrix can be constructed by using the difference operator $\alpha$ times consecutively on the identity matrix. For $\alpha = 0$ this is equal to Tikhonov regularization and for $\alpha = 1$ we get Total Variation regularization. For example, given are a weight vector and the first-order difference matrix:

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}, \qquad \boldsymbol{D}_1 = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \end{bmatrix}.$$

The penalty term then equals:

$$\|\boldsymbol{D}_1\boldsymbol{w}\|_2^2 = (\boldsymbol{D}_1\boldsymbol{w})^T(\boldsymbol{D}_1\boldsymbol{w})$$

$$= \begin{bmatrix}(w_1 - w_2) & (w_2 - w_3)\end{bmatrix} \begin{bmatrix}(w_1 - w_2) \\ (w_2 - w_3)\end{bmatrix}$$

$$= (w_1 - w_2)^2 + (w_2 - w_3)^2.$$

We wish to extend the penalty in (23) to the TNBS format. Without loss of generality, Figure 8 visualizes the necessary steps in graphical notation for a three-dimensional B-spline surface. In the multivariate case, the differences in adjacent weights in the weight tensor $\mathcal{W}$ have to be penalized along each dimension individually. This is done by contracting the second index of the difference matrix $\boldsymbol{D}_\alpha$ with the dimension of the weight tensor $\mathcal{W}$ along which the penalty is applied, then taking the norm of the result. For a B-spline curve with $d$ inputs, the penalty on the $\alpha$-th order differences along the $j$-th dimension is given by:

$$\begin{aligned} R(\mathcal{W}) &= \|\mathcal{W} \times_j^2 \boldsymbol{D}_\alpha\|_2^2 \\ &= \left\langle (\mathcal{W} \times_j^2 \boldsymbol{D}_\alpha),\ (\mathcal{W} \times_j^2 \boldsymbol{D}_\alpha) \right\rangle. \end{aligned} \tag{24}$$

This is illustrated in 8a, where the penalty is applied along the first dimension, e.g. $j = 1$. Decomposing $\mathcal{W}$

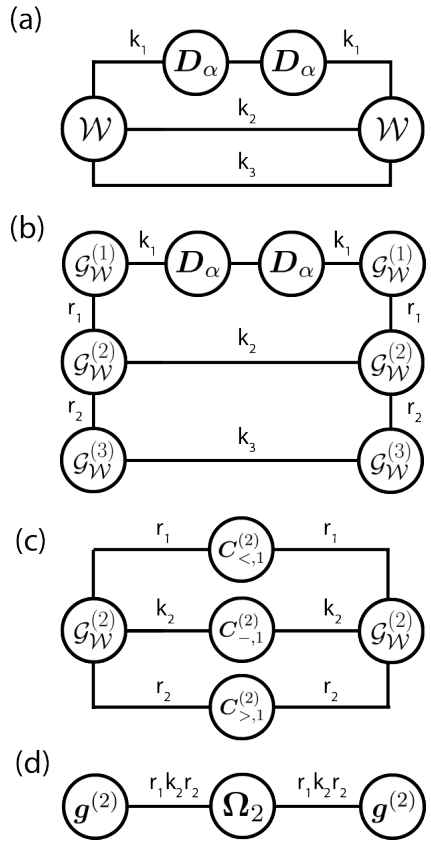



Fig. 8. Derivation of the Tensor Network P-spline penalty.

into a tensor train results in the network depicted in 8b. To write this penalty again as a linear function of the $p$-th core, we contract everything in the network except these cores (Figure 8c). In this example, $\boldsymbol{C}^{(2)}_{>,1}$ and $\boldsymbol{C}^{(2)}_{-,1}$ are simply identity matrices. Then, using (3), the penalty function can be rewritten in the form of Figure 8d:

$$R\left(\mathcal{G}^{(p)}_{\mathcal{W}}\right) = \boldsymbol{g}^{(p)T}\boldsymbol{\Omega}^{(p)}_j\boldsymbol{g}^{(p)}, \tag{25}$$

where

$$\boldsymbol{\Omega}^{(p)}_j = \left(\boldsymbol{C}^{(p)}_{>,j} \otimes \boldsymbol{C}^{(p)}_{-,j} \otimes \boldsymbol{C}^{(p)}_{<,j}\right). \tag{26}$$

The matrix $\boldsymbol{\Omega}^{(p)}_j$ in (26) is constructed for every dimension $j$. Due to the site-p-mixed-canonical form of the tensor train, the contraction of two out of the three matrices $\boldsymbol{C}^{(p)}_{>,j}$, $\boldsymbol{C}^{(p)}_{-,j}$ and $\boldsymbol{C}^{(p)}_{<,j}$ result in identity matrices. This knowledge can be utilized for efficient implementation. Adding the penalties to the cost function results in the following regularized optimization problem:

$$\begin{aligned}&\min_{\mathcal{W}} \ \|\boldsymbol{y}-\boldsymbol{s}\|_2^2 + \sum_{j=1}^{d}\lambda_j \|\mathcal{W} \times_j^2 \boldsymbol{D}_\alpha\|_2^2 \\ &\text{s.t. TT-rank}(\mathcal{W}) = (r_1, r_2, \ldots, r_{d-1}).\end{aligned} \tag{27}$$

The smoothing parameter $\lambda_j \geq 0$ controls the penalization of the roughness along dimension $j$. The subproblem for updating the $p$-th core becomes:

$$\min_{\boldsymbol{g}^{(p)}} \ \|\boldsymbol{y}-\boldsymbol{A}^{(p)}\boldsymbol{g}^{(p)}\|_2^2 + \sum_{j=1}^{d}\lambda_j\, \boldsymbol{g}^{(p)T}\boldsymbol{\Omega}^{(p)}_j\boldsymbol{g}^{(p)}. \tag{28}$$

The normal equation is then:

$$\left(\boldsymbol{A}^{(p)T}\boldsymbol{A}^{(p)} + \sum_{j=1}^{d}\lambda_j\, \boldsymbol{\Omega}^{(p)}_j\right)\boldsymbol{g}^{(p)} = \boldsymbol{A}^{(p)T}\boldsymbol{y}. \tag{29}$$

The whole procedure of identifying a TNBS model from measured data is summarized as pseudo-code in Algorithm 1.

**Algorithm 1** TNBS-NARX identification

**Input:** Data $\{(y_n, u_n)\}_{n=1}^N$, TT-ranks $\{r_p\}_{p=1}^d$, number of knots $m$, degree $\rho$, regularization parameters $\{\lambda_j\}_{j=1}^d$

**Output:** TT-cores $\left\{\mathcal{G}^{(p)}_{\mathcal{W}}\right\}_{p=1}^d$

```
Initialize random TT-cores
Construct {{b_n^(p)}_{n=1}^N}_{p=1}^d from data
while stopping criteria not satisfied do
    for p = 1, 2, ..., d − 1 do
        Construct A^(p) (20) and {Ω_j^(p)}_{j=1}^d (26)
        g^(p) ← Solve (29)
        G_W^(p) ← Orthogonalize and reshape g^(p)
    end for
    for p = d, d − 1, ..., 2 do
        Repeat the above
    end for
end while
```

Table 1 summarizes relevant computational complexities concerning the TNBS-NARX method. While the complexities scale only linearly in the dimensions, it is important to realize that high TT-ranks easily degrade the performance of optimization of the cores. There is, therefore, a tradeoff between accuracy and speed. The number of data samples $N$ also appears linearly in the complexities but may become a bottleneck for large datasets. A modification for this scenario is to use a small random batch of the data when updating $\boldsymbol{g}^{(p)}$. This can speed up estimation time without significant loss of accuracy.

## 4 Experiments

In this section, we demonstrate the proposed system identification method. The algorithm is implemented in MATLAB and executed on a personal computer with a 4.2 GHz Intel Core i5-7600K processor and 16 GB of random access memory (RAM). An open-source MATLAB

Table 1
Computational complexities of significant operations

| Operation | Complexity |
|---|---|
| Construct $\left\{\boldsymbol{b}_n^{(p)}\right\}_{n=1}^{N}$ | $O\left(Nn^2\right)$ |
| Construct $\left\{\boldsymbol{\Omega}_j^{(p)}\right\}_{j=1}^{d}$ | $O\left((d+(m-\rho)^4)r^4\right)$ |
| Construct $\boldsymbol{A}^{(p)}$ | $O\left(N(m-\rho)r^2\right)$ |
| Solve $\boldsymbol{g}^{(p)}$ | $O\left(N(m-\rho)^2r^4+(m-\rho)^3r^6\right)$ |
| Evaluate $f$ | $O\left((\rho^2+(m-\rho)r^2)d\right)$ |

implementation can be found at `https://github.com/Ridvanz/Tensor-Network-B-splines`.

### *4.1 Synthetic dataset*

First, we validate the proposed methods through the identification of an artificial nonlinear dynamical system that is exactly representable in the TNBS-NARX format. The lagged inputs and outputs are chosen as $u_{n-\mu}$ and $y_{n-\mu}$ respectively, where $\mu \in (1,2,3,4)$, such that the system equation is of the following form:

$$y_n = f(y_{n-1}, y_{n-2}, y_{n-3}, y_{n-4}, u_{n-1}, u_{n-2}, u_{n-3}, u_{n-4}) \tag{30}$$

The nonlinear mapping $f$ is modeled as an 8-dimensional TNBS. We choose the degree of the B-splines $\rho = 2$ and the number of knots per dimension $m = 6$. A random weight tensor $\mathcal{W}$ of size $(m-\rho)^d = 4^8$ is generated of which the elements equal either $w_{min} = -4$ or $w_{max} = 5$ with equal probability. The generated tensor is decomposed using the TT-SVD algorithm, truncating the TT-ranks to a value of 5 uniformly. The resulting tensor train represents the true weights of our nonlinear system. For the input signal $\boldsymbol{u}$ we generate a random sequence of length 3000, with values uniformly distributed in the range $[0, 1]$. This sequence is smoothed with a Gaussian window of size 5 to dampen higher frequencies. We initialize the output signal $\boldsymbol{y}$ with 4 zeros and recursively evaluate the next output with (30), until we have a signal of length 3000. The first 200 samples of the input (blue) and output (red) signals are plotted in in Figure 4.1. The signals are split in an identification set of 2000 samples and a test set of 1000 samples.

We test the performance of our TNBS-NARX identification algorithm with different levels of Gaussian white noise on the estimation data. Noise is only added to the output signal. The variances for the white noise signals are chosen based on the desired signal to noise ratios SNR. The signal powers are determined after subtracting their means. For simplicity, we penalize the second difference ($\alpha = 2$) of the weights equally for each dimension, e.g. $\lambda_p = \lambda,\ \forall p$. The experiment is run using three different values for lambda. All other model parameters are set to the true values of the synthetic model. The TT-cores are estimated using Algorithm 1. For consistency, we simply choose a max number (16) of sweeps as

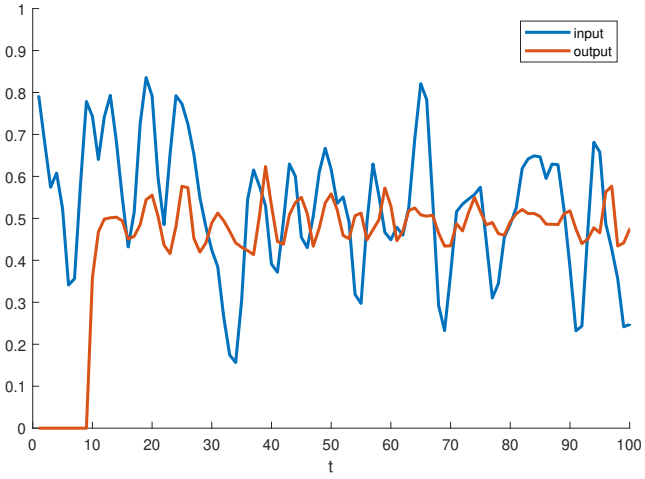



Fig. 9. Input and output signals of the synthetic dataset.

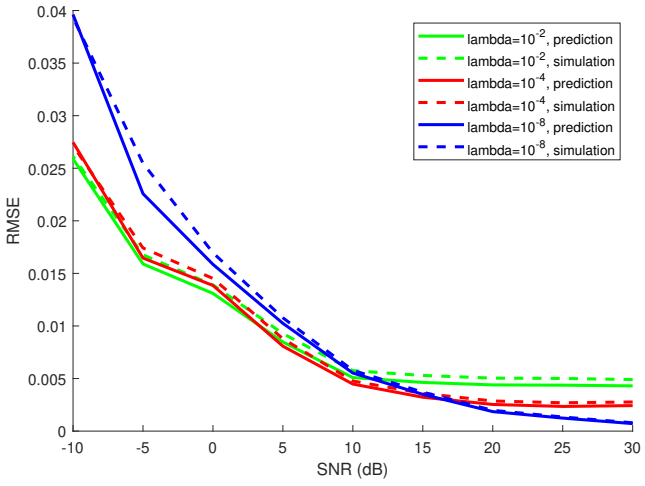



Fig. 10. Prediction and simulation performance on synthetic test set.

stopping criteria. The root mean squared error (RMSE) is used as the performance metric to evaluate the accuracy on the test set for both prediction and simulation.

$$e_{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

Figure 10 plots the RMSE of the different experiments as a function of the SNR in dB. The prediction errors are consistently lower than the simulation errors. The effect of the regularization is in line with expectations, i.e. for increasing SNR values, more regularization is needed to avoid overfitting to noise, so larger penalties give better performance. Overall, the TNBS is able to identify the system accurately, even for relatively noisy estimation data.

Table 2
Comparison of methods on Cascaded tanks benchmark

| Method | Prediction | Simulation |
|---|---|---|
| LTI [35] | 0.056 | 0.588 |
| Volterra FB [35] | 0.049 | 0.397 |
| Flexible SS [36] | - | 0.45 |
| NOMAD [37] | - | 0.376 |
| PWARX [38] | - | 0.350 |
| Sparse Bay. DNN [39] | 0.0472 | 0.344 |
| TNBS-NARX | 0.0461 | 0.3018 |

### *4.2 Cascaded tanks dataset*

The cascaded tanks system is a benchmark dataset for nonlinear system identification. A detailed description of the system and the data is given in [34]. The system consists of two tanks, a water reservoir and a pump. The water in the reservoir is pumped in the upper tank, from which it flows to the lower tank through a small opening and then back into the reservoir. The system input $u_n$ is the pump voltage and the system output $y_n$ is the water level of the lower tank. If too much water is pumped into the upper tank it overflows, causing a hard saturation nonlinearity in the system dynamics. The input signals are low-frequency multisine signals. Both the estimation and test set have a length of $N = 1024$ samples. The major challenges of this benchmark are the hard saturation nonlinearity and the relatively small size of the estimation set. The performance metric used is again RMSE.

The original data is first normalized to the interval [0,1]. Both input and output lags are chosen as $u_{n-\mu}$ and $y_{n-\mu}$ respectively, where $\mu \in \{1, 2, 3, 4, 8, 12, 16, 32\}$. The large lags are included to capture the relevant slow system dynamics. We choose the degree of the B-splines $\rho = 3$ and the number of knots $m = 7$. We penalize the first-order difference only, i.e. $\alpha = 1$, and set the TT-ranks to 8 uniformly. We choose $\lambda$ through 3-fold cross-validation on the estimation set. A total of 12 sweeps are performed in the optimization with Algorithm 1. After tuning lambda, the full identification set is used to identify the final model, which takes about 4 seconds. Using TNBS, the number of weights to represent the 16-dimensional B-spline surface is reduced from approximately $4.3 \times 10^9$ to 3648. The performance on prediction and simulation are listed and compared in table 2. To the best of our knowledge, the algorithm slightly outperforms the current state-of-the-art results on both prediction and simulation. Figure 11 shows the true and simulated output on the test set. It is apparent that the TNBS-NARX model was able to accurately capture the nonlinear system dynamics with relatively sparse estimation data.

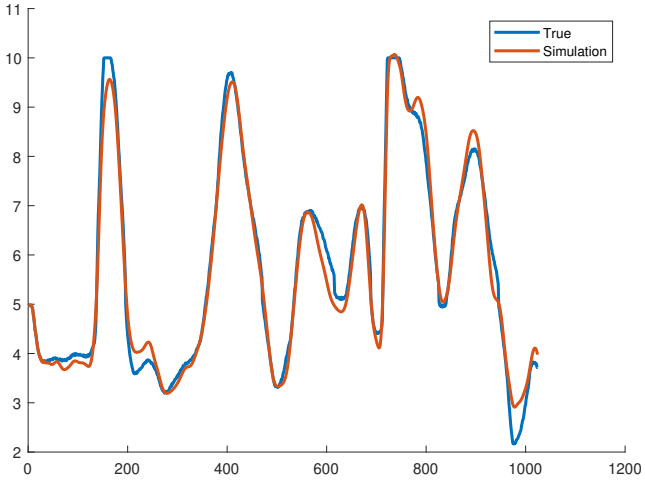



Fig. 11. Simulation on cascaded tanks dataset.

## 5 Conclusions

This article presents a new algorithm for nonlinear system identification using a NARX model of which the nonlinear mapping is approximated using the introduced Tensor Network B-splines. Tensor Network theory enables to work with B-spline surfaces directly in a high-dimensional feature space, allowing the identification of NARX systems with a large number of lags and inputs. The identification algorithm is guaranteed to monotonically converge and numerical stability is ensured through orthogonality of the TT-cores. The efficiency and accuracy of the algorithm is demonstrated through numerical experiments on SISO nonlinear systems. Extension of TNBS-NARX to multiple inputs is straightforward through the addition of input variables. Multiple outputs can be realized efficiently by adding an index to one of the TT-cores, as done in [18]. Future work includes the implementation of an online optimization scheme, as an alternative to ALS, and the development of control strategies for identified TNBS-NARX systems.

## References


[1] C. De Boor. Splines as Linear Combinations of B-splines. A Survey. Technical report, Wisconsin Univ Madison Mathematics Research Center, 1976.

[2] G. Lightbody, P. O'Reilly, G. W. Irwin, J. McCormick, et al. Neural modelling of chemical plant using MLP and B-spline networks. *Control Engineering Practice*, 5(11):1501–1515, 1997.

[3] K. F. C. Yiu, S. Wang, K. L. Teo, and A. C. Tsoi. Nonlinear system modeling via knot-optimizing B-spline networks. *IEEE transactions on neural networks*, 12(5):1013–1022, 2001.

[4] Z. Lin, D. S. Reay, B. W. Williams, and X. He. Online modeling for switched reluctance motors using B-spline neural networks. *IEEE Transactions on Industrial electronics*, 54(6):3317–3322, 2007.

[5] L. dos Santos Coelho and M. W. Pessôa. Nonlinear identification using a B-spline neural network and chaotic immune approaches. *Mechanical Systems and Signal Processing*, 23(8):2418–2434, 2009.

[6] L. Mirea. Dynamic multivariate B-spline neural network design using orthogonal least squares algorithm for non-linear system identification. In *2014 18th International Conference on System Theory, Control and Computing (ICSTCC)*, pages 720–725. IEEE, 2014.

[7] M. Folgheraiter. A combined B-spline-neural-network and ARX model for online identification of nonlinear dynamic actuation systems. *Neurocomputing*, 175:433–442, 2016.

[8] P. Csurcsia, J. Schoukens, I. Kollár, and J. Lataire. Nonparametric time-domain identification of linear slowly time-variant systems using B-splines. *IEEE Transactions on Instrumentation and Measurement*, 64(1):252–262, 2014.

[9] S. Cong and R. Song. An improved B-spline fuzzy-neural network controller. In *Proceedings of the 3rd World Congress on Intelligent Control and Automation (Cat. No. 00EX393)*, volume 3, pages 1713–1717. IEEE, 2000.

[10] D. S. Reay. CMAC and B-spline neural networks applied to switched reluctance motor torque estimation and control. In *IECON'03. 29th Annual Conference of the IEEE Industrial Electronics Society (IEEE Cat. No. 03CH37468)*, volume 1, pages 323–328. IEEE, 2003.

[11] X. Hong and S. Chen. The system identification and control of Hammerstein system using non-uniform rational B-spline neural network and particle swarm optimization. *Neurocomputing*, 82:216–223, 2012.

[12] X. Zhang, Y. Zhao, K. Guo, G. Li, and N. Deng. An adaptive B-spline neural network and its application in terminal sliding mode control for a mobile satcom antenna inertially stabilized platform. *Sensors*, 17(5):978, 2017.

[13] M. Brown, K. M. Bossley, D. J. Mills, and C. J. Harris. High dimensional neurofuzzy systems: overcoming the curse of dimensionality. In *Proceedings of 1995 IEEE International Conference on Fuzzy Systems.*, volume 4, pages 2139–2146. IEEE, 1995.

[14] C. Zenger and W. Hackbusch. Sparse grids. 1991.

[15] J. Garcke et al. Sparse grid tutorial. *Mathematical Sciences Institute, Australian National University, Canberra Australia*, page 7, 2006.

[16] A. Cichocki, N. Lee, I. Oseledets, A. Phan, Q. Zhao, D. P. Mandic, et al. Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. *Foundations and Trends® in Machine Learning*, 9(4-5):249–429, 2016.

[17] K. Batselier, Z. Chen, and N. Wong. A Tensor Network Kalman filter with an application in recursive MIMO Volterra system identification. *Automatica*, 84:17–25, 2017.

[18] K. Batselier, Z. Chen, and N. Wong. Tensor Network alternating linear scheme for MIMO Volterra system identification. *Automatica*, 84:26–35, 2017.

[19] K. Batselier, C. Ko, and N. Wong. Tensor Network subspace identification of polynomial state space models. *Automatica*, 95:187–196, 2018.

[20] I. J. Leontaritis and S. A. Billings. Input-output parametric models for non-linear systems part I: deterministic non-linear systems. *International journal of control*, 41(2):303–328, 1985.

[21] S. A. Billings. *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons, 2013.

[22] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[23] R. Penrose. Applications of negative dimensional tensors. *Combinatorial mathematics and its applications*, 1:221–244, 1971.

[24] A. Cichocki. Era of big data processing: A new approach via tensor networks and tensor decompositions. *arXiv preprint arXiv:1403.2048*, 2014.

[25] I. V. Oseledets. Tensor-Train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[26] C. de Boor. *A practical guide to splines*, volume 27. springer-verlag New York, 1978.

[27] I. Oseledets and E. Tyrtyshnikov. TT-cross approximation for multidimensional arrays. *Linear Algebra and its Applications*, 432(1):70–88, 2010.

[28] Ulrich Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326(1):96–192, 2011.

[29] I. J. Schoenberg. *Cardinal spline interpolation*, volume 12. Siam, 1973.

[30] K. Qin. General matrix representations for B-splines. *The Visual Computer*, 16(3):177–186, 2000.

[31] S. Holtz, T. Rohwedder, and R. Schneider. The alternating linear scheme for tensor optimization in the tensor train format. *SIAM Journal on Scientific Computing*, 34(2):A683–A713, 2012.

[32] R. L. Eubank. *Nonparametric regression and spline smoothing*. CRC press, 1999.

[33] P. H. C. Eilers and B. D. Marx. Flexible smoothing with B-splines and penalties. *Statistical science*, pages 89–102, 1996.

[34] M. Schoukens, P. Mattson, T. Wigren, and J. P. Noël. Cascaded tanks benchmark combining soft and hard nonlinearities. In *Workshop on Nonlinear System Identification Benchmarks*, pages 20–23, 2016.

[35] J. Schoukens and F. G. Scheiwe. Modeling Nonlinear Systems Using a Volterra Feedback Model. `http://www.nonlinearbenchmark.org/FILES/SLIDES/2016-NSIB-Schoukensb.pdf`, Workshop on Nonlinear System Identification Benchmarks, 2016. Online; accessed 1 March 2020.

[36] A. Svensson and T. B. Schön. A flexible state-space model for learning nonlinear dynamical systems. *Automatica*, 80:189–199, 2017.

[37] M. Brunot, A. Janot, and F. Carrillo. Continuous-time nonlinear systems identification with output error method based on derivative-free optimisation. *IFAC-PapersOnLine*, 50(1):464–469, 2017.

[38] P. Mattsson, D. Zachariah, and P. Stoica. Identification of cascade water tanks using a PWARX model. *Mechanical systems and signal processing*, 106:40–48, 2018.

[39] H. Zhou, C. Ibrahim, and W. Pan. A Sparse Bayesian Deep Learning Approach for Identification of Cascaded Tanks Benchmark. *arXiv preprint arXiv:1911.06847*, 2019.

# Chapter 6

# Conclusion

## 6-1 Summary

B-spline curves are an attractive choice for applications involving the approximation of smooth continuous functions. Their basis functions have local support and can be constructed efficiently, especially when defined on uniform knot sequences. They can effectively fit to data using convex optimization techniques and can accommodate several regularization methods to help avoid overfitting. Multivariate B-splines inherit these properties, but the number of elements in both the weight tensor $\mathcal{W}$ and B-spline tensor $\mathcal{B}$ scale exponentially with the number of input dimensions $d$. Tensor Network theory has enabled us to work with B-spline surfaces directly in a high-dimensional feature space, allowing the approximation of high-dimensional functions with low computational effort. The introduced Tensor Network B-splines (TNBS) model has a low storage complexity, its optimization is guaranteed to monotonically converge and numerical stability is ensured through orthogonality of the TT-cores. The addition of P-spline regularization ensures smoothness and good generalization properties while fitting high-dimensional B-splines to sparse and scattered data. The efficiency and accuracy of the introduced approximation algorithm is demonstrated through numerical experiments. The research for this thesis resulted in a paper and an open-source implementation of the TNBS framework in MATLAB.

## 6-2 Limitations and Future Work

The presented Tensor Network B-splines framework has some limitations and potential room for improvement. This section lists some current limitations and possible ideas for future development in this area.

- A limitation to splines in general is that they are only defined on an interval $[a, b]$. This means that the range of the input data has to be known beforehand. For most applications, the range of the inputs are obvious, either by considering physical constraints or by looking at the distribution of the data. For highly non-uniform datasets, e.g. exponentially distributed data, one can apply invertable data transformations such as the logarithm and square root transformations.

- Large numbers of B-splines per dimension require higher TT-ranks for an accurate approximation of the weight tensor, which in turn requires more sweeps during optimization. This means there is a computational trade-off between the number of dimensions and the complexity of the TNBS surface. In other words, choosing large values for both $k$ and $d$ still leads to a computationally infeasible optimization problem with the current model.

- While we mention that B-splines are highly interpretable, interpreting the cores of TNBS is less straightforward. A possible solution is to convert the optimized TT-train to a CP-decomposition, as done in [88]. From the CP-representation of the B-spline weight tensor, it is possible to extract more information about the underlying B-spline surface.

- The current TNBS format assumes a single output. One method to approximate functions with multiple outputs is to fit a TNBS model for each individual output in parallel. However, larger overall compression can be achieved when extending the tensor train to multiple outputs and optimizing for all outputs directly. This can simply be done by adding an index to one of the cores. Examples of vector output extensions for tensor trains are given in [54] and [53].

- Our implementation of ALS includes batch sampling to cope with larger datasets. Another possibility for speeding up ALS is by only approximating the solution of $\boldsymbol{g}^{(p)}$ when updating a core. Our trials show that an exact solution to the linear subsystem in Eq. (4-3) is not necessary for ALS to converge, as long as the solution is sufficiently close to the optimum. Solving for $\boldsymbol{g}^{(p)}$ with iterative methods like preconditioned conjugate gradients (PCG) could win an order in overall optimization speed. The main challenge here is determining the accuracy or number of iterations of the PCG method.

- Some applications require the approximation of a function in real-time. Examples are the identification of time-varying systems, or the mapping of states/actions to values in reinforcement learning. Online optimization with TNBS can be achieved by replacing the ALS method by a TN Kalman filter [54]. This allows updating the tensor train efficiently using only the most recent data sample. However, the addition of P-spline regularization to this optimization scheme is not straightforward and needs further research.

- Approximating functions is an essential cornerstone in the field of machine learning. The TNBS model could be very suitable for supervised learning tasks such as regression and classification. Classification can be performed with TNBS by choosing an appropriate error function, such as the cross entropy or sigmoid function. The resulting subsystem when updating a core would still be convex, but no longer linear. Hence a convex optimization problem would need to be solved at each update of the core. The convergence properties of such an 'Alternating Convex Scheme' are to be investigated. Building a TNBS classifier, along with an appropriate tensor train optimization method, is an interesting future development.

# Bibliography


[1] Karl Weierstrass. "Über die analytische Darstellbarkeit sogenannter willkürlicher Functionen einer reellen Veränderlichen". In: *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin* 2 (1885), pp. 633–639.

[2] George M Phillips and Peter J Taylor. *Theory and applications of numerical analysis*. Elsevier, 1996.

[3] Volker Barthelmann, Erich Novak, and Klaus Ritter. "High dimensional polynomial interpolation on sparse grids". In: *Advances in Computational Mathematics* 12.4 (2000), pp. 273–288.

[4] Harvey J Motulsky and Lennart A Ransnas. "Fitting curves to data using nonlinear regression: a practical and nonmathematical review". In: *The FASEB journal* 1.5 (1987), pp. 365–374.

[5] David G Kleinbaum et al. *Applied regression analysis and other multivariable methods*. Vol. 601. Duxbury Press Belmont, CA, 1988.

[6] Cheng Lin Liu and Hiroshi Sako. "Class-specific feature polynomial classifier for pattern classification and its application to handwritten numeral recognition". In: *Pattern recognition* 39.4 (2006), pp. 669–681.

[7] Ninh Pham and Rasmus Pagh. "Fast and scalable polynomial kernels via explicit feature maps". In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2013, pp. 239–247.

[8] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. Vol. 2. 3. MIT press Cambridge, MA, 2006.

[9] Oliver Nelles. *Nonlinear system identification: from classical approaches to neural networks and fuzzy models*. Springer Science & Business Media, 2013.

[10] Stephen A Billings. *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons, 2013.

[11] Carl Runge. "Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten". In: *Zeitschrift für Mathematik und Physik* 46.224-243 (1901), p. 20.

[12] Gordon K Smyth. "Polynomial approximation". In: *Wiley StatsRef: Statistics Reference Online* (2014).

[13] Eliahu Ibrahim Jury. "Stability of multidimensional scalar and matrix polynomials". In: *Proceedings of the IEEE* 66.9 (1978), pp. 1018–1047.

[14] Richard E Bellman. *Adaptive control processes: a guided tour*. Vol. 2045. Princeton university press, 2015.

[15] Jalil Rashidinia. "Applications of Splines to Numerical Solution of Differential Equations". PhD thesis. Aligarh Muslim University, 1990.

[16] Carl De Boor. *Splines as Linear Combinations of B-Splines. A Survey*. Tech. rep. Wisconsin Univ Madison Mathematics Research Center, 1976.

[17] Michael Unser, Akram Aldroubi, and Murray Eden. "B-spline signal processing. I. Theory". In: *IEEE transactions on signal processing* 41.2 (1993), pp. 821–833.

[18] Martin Brown and Christopher John Harris. *Neurofuzzy adaptive modelling and control*. Prentice Hall, 1994.

[19] Stephen H Lane, David A Handelman, and Jack J Gelfand. "Theory and development of higher-order CMAC neural networks". In: *IEEE Control Systems Magazine* 12.2 (1992), pp. 23–30.

[20] Matthias Eck and Hugues Hoppe. *Automatic reconstruction of B-spline surfaces of arbitrary topological type*. Technische Hochschule Darmstadt. Fachbereich Mathematik, 1996.

[21] Richard F Riesenfeld. "Applications of B-spline approximation to geometric problems of computer-aided design". PhD thesis. Syracuse University Syracuse, NY, 1973.

[22] Dmytro Shulga et al. "Tensor B-Spline Numerical Methods for PDEs: a High-Performance Alternative to FEM". In: *arXiv preprint arXiv:1904.03057* (2019).

[23] Klaus Hollig. *Finite element methods with B-splines*. Vol. 26. Siam, 2003.

[24] Philippe Thévenaz, Thierry Blu, and Michael Unser. "Interpolation revisited [medical images application]". In: *IEEE Transactions on medical imaging* 19.7 (2000), pp. 739–758.

[25] Henri Bachau et al. "Applications of B-splines in atomic and molecular physics". In: *Reports on Progress in Physics* 64.12 (2001), p. 1815.

[26] António E Ruano et al. "Supervised training algorithms for B-spline neural networks and fuzzy systems". In: *Proceedings Joint 9th IFSA World Congress and 20th NAFIPS International Conference (Cat. No. 01TH8569)*. IEEE. 2001, pp. 2830–2835.

[27] Zhengyu Lin et al. "Online modeling for switched reluctance motors using B-spline neural networks". In: *IEEE Transactions on Industrial electronics* 54.6 (2007), pp. 3317–3322.

[28] Letiţia Mirea. "Dynamic multivariate B-spline neural network design using orthogonal least squares algorithm for non-linear system identification". In: *2014 18th International Conference on System Theory, Control and Computing (ICSTCC)*. IEEE. 2014, pp. 720–725.

[29] G Lightbody et al. "Neural modelling of chemical plant using MLP and B-spline networks". In: *Control Engineering Practice* 5.11 (1997), pp. 1501–1515.

[30] Ka Fai Cedric Yiu et al. “Nonlinear system modeling via knot-optimizing B-spline networks”. In: *IEEE transactions on neural networks* 12.5 (2001), pp. 1013–1022.

[31] Michele Folgheraiter. “A combined B-spline-neural-network and ARX model for online identification of nonlinear dynamic actuation systems”. In: *Neurocomputing* 175 (2016), pp. 433–442.

[32] Donald S Reay. “CMAC and B-spline neural networks applied to switched reluctance motor torque estimation and control”. In: *IECON'03. 29th Annual Conference of the IEEE Industrial Electronics Society (IEEE Cat. No. 03CH37468)*. Vol. 1. IEEE. 2003, pp. 323–328.

[33] Xia Hong and Sheng Chen. “The system identification and control of Hammerstein system using non-uniform rational B-spline neural network and particle swarm optimization”. In: *Neurocomputing* 82 (2012), pp. 216–223.

[34] Xiaolei Zhang et al. “An adaptive B-spline neural network and its application in terminal sliding mode control for a mobile satcom antenna inertially stabilized platform”. In: *Sensors* 17.5 (2017), p. 978.

[35] Shuang Cong and Ruixiang Song. “An improved B-spline fuzzy-neural network controller”. In: *Proceedings of the 3rd World Congress on Intelligent Control and Automation (Cat. No. 00EX393)*. Vol. 3. IEEE. 2000, pp. 1713–1717.

[36] Christophe Abraham et al. “Unsupervised curve clustering using B-splines”. In: *Scandinavian journal of statistics* 30.3 (2003), pp. 581–595.

[37] Steve R Gunn et al. “Support vector machines for classification and regression”. In: *ISIS technical report* 14.1 (1998), pp. 5–16.

[38] Li Wang, Lijian Yang, et al. “Spline-backfitted kernel smoothing of nonlinear additive autoregression model”. In: *The Annals of Statistics* 35.6 (2007), pp. 2474–2503.

[39] Grace Wahba. *Spline models for observational data*. Vol. 59. Siam, 1990.

[40] M Brown et al. “High dimensional neurofuzzy systems: overcoming the curse of dimensionality”. In: *Proceedings of 1995 IEEE International Conference on Fuzzy Systems.* Vol. 4. IEEE. 1995, pp. 2139–2146.

[41] Tarek Sayed, Arash Tavakolie, and Abdolmehdi Razavi. “Comparison of adaptive network based fuzzy inference systems and B-spline neuro-fuzzy mode choice models”. In: *Journal of computing in civil engineering* 17.2 (2003), pp. 123–130.

[42] Christoph Zenger and W Hackbusch. “Sparse grids”. In: (1991).

[43] Jochen Garcke et al. “Sparse grid tutorial”. In: *Mathematical Sciences Institute, Australian National University, Canberra Australia* (2006), p. 7.

[44] Tamara G Kolda and Brett W Bader. “Tensor decompositions and applications”. In: *SIAM review* 51.3 (2009), pp. 455–500.

[45] Rose Yu et al. “Long-term forecasting using tensor-train rnns”. In: *arXiv preprint arXiv:1711.00073* (2017).

[46] Yinchong Yang, Denis Krompass, and Volker Tresp. “Tensor-train recurrent neural networks for video classification”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 3891–3900.

[47] Timur Garipov et al. “Ultimate tensorization: compressing convolutional and fc layers alike”. In: *arXiv preprint arXiv:1611.03214* (2016).

[48] Ching-Yun Ko et al. “Deep Compression of Sum-Product Networks on Tensor Networks”. In: *arXiv preprint arXiv:1811.03963* (2018).

[49] Namgil Lee and Andrzej Cichocki. “Very large-scale singular value decomposition using tensor train networks”. In: *arXiv preprint arXiv:1410.6895* (2014).

[50] Zhongming Chen et al. “Parallelized tensor train learning of polynomial classifiers”. In: *IEEE transactions on neural networks and learning systems* 29.10 (2017), pp. 4621–4632.

[51] Cong Chen et al. “A Support Tensor Train Machine”. In: (2018).

[52] Alexander Novikov, Mikhail Trofimov, and Ivan Oseledets. “Exponential machines”. In: *arXiv preprint arXiv:1605.03795* (2016).

[53] Edwin Stoudenmire and David J Schwab. “Supervised learning with tensor networks”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 4799–4807.

[54] Kim Batselier, Zhongming Chen, and Ngai Wong. “A Tensor Network Kalman filter with an application in recursive MIMO Volterra system identification”. In: *Automatica* 84 (2017), pp. 17–25.

[55] Kim Batselier, Zhongming Chen, and Ngai Wong. “Tensor Network alternating linear scheme for MIMO Volterra system identification”. In: *Automatica* 84 (2017), pp. 26–35.

[56] Kim Batselier, Ching-Yun Ko, and Ngai Wong. “Tensor network subspace identification of polynomial state space models”. In: *Automatica* 95 (2018), pp. 187–196.

[57] Ching Yun Ko et al. “Fast and accurate tensor completion with tensor trains: a system identification approach”. In: *CoRR abs/1804.06128* (2018).

[58] Johann A Bengua et al. “Efficient tensor completion for color image and video recovery: Low-rank tensor train”. In: *IEEE Transactions on Image Processing* 26.5 (2017), pp. 2466–2479.

[59] Ho N Phien et al. “Efficient tensor completion: Low-rank tensor train”. In: *arXiv preprint arXiv:1601.01083* (2016).

[60] Dmitry Savostyanov and Ivan Oseledets. “Fast adaptive interpolation of multi-dimensional arrays in tensor train format”. In: *The 2011 International Workshop on Multidimensional (nD) Systems*. IEEE. 2011, pp. 1–8.

[61] Pavel Izmailov, Alexander Novikov, and Dmitry Kropotov. “Scalable gaussian processes with billions of inducing inputs via tensor train decomposition”. In: *arXiv preprint arXiv:1710.07324* (2017).

[62] Irina Georgieva and Clemens Hofreither. “An algorithm for low-rank approximation of bivariate functions using splines”. In: *Journal of Computational and Applied Mathematics* 310 (2017), pp. 80–91.

[63] Roger Penrose. “Applications of negative dimensional tensors”. In: *Combinatorial mathematics and its applications* 1 (1971), pp. 221–244.

[64] Andrzej Cichocki. “Era of big data processing: A new approach via tensor networks and tensor decompositions”. In: *arXiv preprint arXiv:1403.2048* (2014).

[65] Ivan V Oseledets. "Tensor-train decomposition". In: *SIAM Journal on Scientific Computing* 33.5 (2011), pp. 2295–2317.

[66] A. Cichocki et al. "Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions". In: *Foundations and Trends® in Machine Learning* 9.4-5 (2016), pp. 249–429.

[67] Lars Grasedyck, Daniel Kressner, and Christine Tobler. "A literature survey of low-rank tensor approximation techniques". In: *GAMM-Mitteilungen* 36.1 (2013), pp. 53–78.

[68] Pierre Comon, Xavier Luciani, and André LF De Almeida. "Tensor decompositions, alternating least squares and other tales". In: *Journal of Chemometrics: A Journal of the Chemometrics Society* 23.7-8 (2009), pp. 393–405.

[69] Ivan Oseledets and Eugene Tyrtyshnikov. "TT-cross approximation for multidimensional arrays". In: *Linear Algebra and its Applications* 432.1 (2010), pp. 70–88.

[70] Ulrich Schollwöck. "The density-matrix renormalization group in the age of matrix product states". In: *Annals of Physics* 326.1 (2011), pp. 96–192.

[71] Sebastian Holtz, Thorsten Rohwedder, and Reinhold Schneider. "The alternating linear scheme for tensor optimization in the tensor train format". In: *SIAM Journal on Scientific Computing* 34.2 (2012), A683–A713.

[72] Sebastian Holtz, Thorsten Rohwedder, and Reinhold Schneider. "On manifolds of tensors of fixed TT-rank". In: *Numerische Mathematik* 120.4 (2012), pp. 701–731.

[73] Silvere Bonnabel. "Stochastic gradient descent on Riemannian manifolds". In: *IEEE Transactions on Automatic Control* 58.9 (2013), pp. 2217–2229.

[74] Carl de Boor. *A Practical Guide to Spline*. Vol. Volume 27. Jan. 1978. DOI: 10.2307/2006241.

[75] Isaac Jacob Schoenberg. "Contributions to the problem of approximation of equidistant data by analytic functions". In: *IJ Schoenberg Selected Papers*. Springer, 1988, pp. 3–57.

[76] Isaac J Schoenberg. *Cardinal spline interpolation*. Vol. 12. Siam, 1973.

[77] Michael Unser, Akram Aldroubi, and Murray Eden. "On the asymptotic convergence of B-spline wavelets to Gabor functions". In: *IEEE transactions on information theory* 38.2 (1992), pp. 864–872.

[78] Julian Valentin. *Flavors and types of B-splines*. https://bsplines.org/flavors-and-types-of-b-splines. 2019.

[79] Elaine Cohen and Richard F Riesenfeld. "General matrix representations for Bézier and B-spline curves". In: *Computers in Industry* 3.1-2 (1982), pp. 9–15.

[80] Kaihuai Qin. "General matrix representations for B-splines". In: *The Visual Computer* 16.3 (2000), pp. 177–186.

[81] Paul Dierckx. *Curve and surface fitting with splines*. Oxford University Press, 1995.

[82] Paul HC Eilers and Brian D Marx. "Flexible smoothing with B-splines and penalties". In: *Statistical science* (1996), pp. 89–102.

[83] Randall L Eubank. *Nonparametric regression and spline smoothing*. CRC press, 1999.

[84] Christian H Reinsch. "Smoothing by spline functions". In: *Numerische mathematik* 10.3 (1967), pp. 177–183.

[85] Gareth James et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[86] Leonid I Rudin, Stanley Osher, and Emad Fatemi. "Nonlinear total variation based noise removal algorithms". In: *Physica D: nonlinear phenomena* 60.1-4 (1992), pp. 259–268.

[87] I. J. Leontaritis and S. A. Billings. "Input-output parametric models for non-linear systems part I: deterministic non-linear systems". In: *International journal of control* 41.2 (1985), pp. 303–328.

[88] Anh-Huy Phan et al. "Tensor networks for latent variable analysis: Higher order canonical polyadic decomposition". In: *IEEE transactions on neural networks and learning systems* (2019).