An approach to discrete operator learning based on sparse high-dimensional approximation

Daniel Potts[*], Fabian Taubert[†]

May 21, 2025

We present a dimension-incremental method for function approximation in bounded orthonormal product bases to learn the solutions of various differential equations. Therefore, we decompose the source function of the differential equation into parameters like Fourier or Spline coefficients and treat the solution of the differential equation as a high-dimensional function w.r.t. the spatial variables, these parameters and also further possible parameters from the differential equation itself. Finally, we learn this function in the sense of sparse approximation in a suitable function space by detecting coefficients of the basis expansion with the largest absolute values. Investigating the corresponding indices of the basis coefficients yields further insights on the structure of the solution as well as its dependency on the parameters and their interactions and allows for a reasonable generalization to even higher dimensions and therefore better resolutions of the decomposed source function.

Keywords and phrases : sparse approximation, nonlinear approximation, high-dimensional approximation, dimension-incremental algorithm, partial differential equations, operator learning

2020 AMS Mathematics Subject Classification : 35C09, 35C11, 41A50, 42B05, 65D15, 65D30, 65D32, 65D40, 65T40,

1 Introduction

In mathematical analysis, partial differential equations (PDEs) stand as formidable tools when it comes to modeling diverse phenomena across various scientific disciplines from fluid dynamics to quantum mechanics. Unfortunately, solving PDEs analytically as well as numerically can be quite difficult and thus became a challenging task. The numerical solution of PDEs is investigated thoroughly already since the mid-20th century until today by various well-known methods like finite difference methods [29], spectral methods [36, 6] or finite element methods

[1]Chemnitz University of Technology, Faculty of Mathematics, 09107 Chemnitz, Germany
potts@mathematik.tu-chemnitz.de
[2]Chemnitz University of Technology, Faculty of Mathematics, 09107 Chemnitz, Germany
fabian.taubert@mathematik.tu-chemnitz.de

(FEMs) [40]. On the other hand, with the age of artificial intelligence (AI), several new methods using machine learning techniques are currently arising and investigated for the solution of PDEs, including physics-informed neural networks (PINNs) [34, 25, 11, 33], convolutional neural networks [13], deep operator networks [12, 8, 31] multilevel Picard approximations [37, 18] and neural operators [30, 19, 26, 32, 28], among numerous others.

It is not necessary to emphasize that both the classic and the AI methods have various advantages and disadvantages, which are also getting studied frequently in the last years [3, 15]. Especially for high-dimensional PDEs and for many-query settings, where multiple solutions of the PDE w.r.t. varying parameters, initial or boundary conditions are needed, machine learning algorithms have proven to outperform classical methods significantly. One of the main reasons for this is the great performance of neural networks in the framework of operator learning on PDEs, i.e., learning the underlying solution operator of a PDE which maps initial and/or boundary conditions as well as other parameters to the PDE solution. For a comprehensive overview of operator learning theory, algorithms, and applications, we refer the reader to the recent surveys [5, 27].

However, for operator learning of simpler differential equations, classical methods can still compete with machine learning. As an example, consider the one-dimensional differential equation

$$\mathcal{L}u = f \tag{1.1}$$

with the differential operator $\mathcal{L} = \frac{d}{dx}$ and some initial condition $u(0) = 0$. Assume the right-hand side function f to be given (or at least be well approximated) by a partial Fourier sum

$$f(x) = \sum_{\ell=0}^{N-1} f_\ell \mathrm{e}^{2\pi \mathrm{i} \ell x}.$$

If we denote the vector of Fourier coefficients $\boldsymbol{f} = (f_0, \dots, f_{N-1}) \in \mathbb{C}^N$, we now aim for a solution u of the form

$$u(x, \boldsymbol{f}) = \sum_{\boldsymbol{k} \in I} u_{\boldsymbol{k}} T_{\boldsymbol{k}}(x, \boldsymbol{f}) \tag{1.2}$$

with

$$T_{\boldsymbol{k}}(\boldsymbol{z}) \coloneqq \prod_{j=1}^{d} T_{k_j}(z_j) \quad \text{with} \quad T_{k_j}(z_j) = \begin{cases} 1 & k_j = 0 \\ \sqrt{2}\cos(k_j \arccos(z_j)) & k_j \neq 0 \end{cases}$$

being multivariate Chebyshev polynomials of dimension $d = N + 1$, unknown coefficients $u_{\boldsymbol{k}} \in \mathbb{C}$ and an unknown, sparse index set $I \subset \mathbb{N}^{N+1}$. Note that the sparsity is a crucial requirement here, since a high-dimensional approximation of such form for non-sparse index sets I is almost always computationally unfeasible due to the curse of dimension. On the other hand, sparsity occurs naturally based on the way we discretize the function f as we show below. The behavior, that the solution is dominated by main effects and some low-order interactions, will reappear in our numerical experiments as a result of our general approach. A similar effect is known as the sparsity-of-effects principle in other fields of analysis, cf. [38, 16].

Due to the simple structure of $\mathcal{L}$ and the initial condition, we now know that

$$u(x, \boldsymbol{f}) = f_0 x + \sum_{\ell=1}^{N-1} \frac{f_\ell}{2\pi \mathrm{i} \ell} \mathrm{e}^{2\pi \mathrm{i} \ell x},$$

which can be rewritten using the univariate Chebyshev polynomials $T_0(z) = 1$ and $T_1(z) = \sqrt{2}z$ as

$$u(x, \boldsymbol{f}) = \frac{1}{2} T_1(x) T_1(f_0) \prod_{j=1}^{N-1} T_0(f_j) + \sum_{\ell=1}^{N-1} \mathrm{e}^{2\pi \mathrm{i} \ell x} \frac{T_1(f_\ell)}{2\sqrt{2}\pi \mathrm{i} \ell} \prod_{\substack{j=0 \\ j \neq \ell}}^{N-1} T_0(f_j),$$

since all the f_ℓ appear only linearly and decoupled. From this formula, we can directly read the structure of the index set I as

$$I = \left\{ \begin{bmatrix} 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} 2 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, \begin{bmatrix} 3 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots \right\}$$

with the first row corresponding to the spatial dimension x and the remaining rows corresponding to the coefficient dimensions $f_0, \dots, f_9$ (in this order).

Therefore, we in fact know the index set I and can compute the corresponding basis coefficients $u_{\boldsymbol{k}}$ simply by using e.g. Chebyshev rank-1 lattices [21]. So if the structure of a suitable index set I in (1.2) can be computed exactly, the hardest part of the approximation problem is already done and allows us to use high-dimensional cubature methods like Monte Carlo (MC) or Quasi-Monte Carlo (QMC) methods to derive the coefficients $u_{\boldsymbol{k}}$. A similar approach in the Fourier setting for a more complicated differential equation was investigated in [14] and showed great results, even for very high dimensions. Further, the parametrization of the right-hand side function f as well as the representation of the solution u by its basis coefficients $u_{\boldsymbol{k}}$ is closely related to the concept of so-called encoder and decoder mappings as for example used in [17]. There the existence of reasonable neural and spectral operators wrapped by such encoder and decoder mappings as well as their error bounds were studied. The fundamental difference of our approach is the missing decoder, since we use the Fourier coefficients $\boldsymbol{f}$ as the encoding of the function f directly to compute the solution u via (1.2).

Unfortunately, a direct analysis of the structure of the a priori unknown index set I is often extremely difficult or simply impossible due to the complexity of the considered differential problem. Thus, we use a dimension-incremental algorithm presented in [23] instead, using point samples to detect a suitable index set I adaptively. More precisely, our general aim is to solve the high-dimensional approximation problem of approximating (1.2) by detecting a reasonable and sparse index set I containing the indices $\boldsymbol{k}$ corresponding to the largest (in absolute value) coefficients $u_{\boldsymbol{k}}$. Thus, we are using samples $u(x^{(j)}, \boldsymbol{f}^{(j)})$ of the solution u, which we obtain by solving the differential equation (1.1) for the fixed parameters $\boldsymbol{f}^{(j)}$. The adaptive detection of a good index set I is made possible by the usage of adaptively chosen sampling points $(x^{(j)}, \boldsymbol{f}^{(j)})$ and hence training data in the algorithm, which is one of the biggest differences of this approach to several deep learning techniques for operator learning associated with PDEs, where random training data is assumed.

The dimension-incremental algorithm in [23], based on the method from [24], works in arbitrary bounded orthonormal product bases (BOPB) and is capable of computing proper approximations with satisfying error bounds based on the underlying cubature method as shown for the Fourier case in [2]. To generate the necessary point samples of the solution u, we will utilize classical differential equation solvers like the FEM. Once we detected the index set I, we can analyze its structure to gain insight on the interactions between the different parameters and variables and their influence on the solution u. Further, we can generalize the structure of the index set I to proceed to even higher-dimensional or more detailed versions of the differential problem with a reasonable a priori guess, which coefficients $u_{\boldsymbol{k}}$ will be important therein.

This leads to an efficient and interpretable approximation of the solution u, expressed similarly to (1.2), allowing direct evaluations of u for any right-hand side f that is either of the same form or can be well approximated by it—without repeatedly solving the PDE. This operator-based perspective on PDE solvers offers an appealing alternative to classical model-based approaches, in which the structure of the differential operator is explicitly exploited to construct efficient solution algorithms. Our method can be seen as a synthesis of model-based and data-driven approaches: although we rely on sample-based recovery, the use of a structured index set and the decomposition into interpretable basis functions connects our framework closely to analytical PDE solvers, where solution structures are derived symbolically or semi-analytically. As such, it provides both interpretability and generalizability, thereby addressing one of the major limitations of many purely machine learning-based methods. In contrast to neural operator approaches or physics-informed neural networks (PINNs), which often operate as black-box models with limited insight into the underlying solution structures, our method maintains a clear connection to the analytical properties of the PDE and offers explicit control over the approximation process through the choice of basis and index set.

The remainder of this paper is organized as follows: In Section 2, we properly introduce our notations and the theoretical framework for the application of the dimension-incremental algorithm from [23], which is also explained briefly. Section 3 then investigates several test examples to present the application of our method. These examples include the Poisson and heat equation, where we have access to an analytical solution for comparison of the results as well more advanced differential problems like parametric diffusion equations with random coefficients and the non-linear Burgers' equation. Finally, we give a brief conclusion in Section 4.

The Python code is available at `https://github.com/fabiantaubert/nabopb`. It contains the dimension-incremental algorithm, the necessary rank-1 lattice cubature methods and the differential equation applications.

2 Theory

2.1 Setting

We investigate differential equations of the general form

$$\mathcal{L}u = f \tag{2.1}$$

with a differential operator $\mathcal{L} : \mathcal{U} \to \mathcal{F}$, a right-hand side $f \in \mathcal{F}$ and the corresponding solution $u \in \mathcal{U}$. Here, $\mathcal{U}$ and $\mathcal{F}$ are suitable function spaces defined on the Lipschitz smooth

d-dimensional spatial domain $\Omega \subset \mathbb{R}^d$. A typical example for the spaces $\mathcal{U}$ and $\mathcal{F}$, which will also appear in our numerical experiments in Section 3, are the Sobolev space $H^1(\Omega)$ and its dual $H^{-1}(\Omega)$. Furthermore, together with boundary conditions like Dirichlet or Neumann conditions, the solution u can be defined even on the closure $\overline{\Omega}$.

To ensure that the differential problem (2.1) is well-posed, we require certain conditions for the existence and uniqueness of the solution. A problem is said to be well-posed if there is a solution, the solution is unique, and the solution depends continuously on the data. These conditions are typically guaranteed through appropriate assumptions on the operator $\mathcal{L}$, the function spaces $\mathcal{U}$ and $\mathcal{F}$, and the boundary conditions.

For example, in the case where $\mathcal{U} = H^1(\Omega)$ and $\mathcal{F} = H^{-1}(\Omega)$, the Lax-Milgram theorem can be applied for elliptic problems to guarantee the existence and uniqueness of the solution under suitable conditions on the bilinear form associated with $\mathcal{L}$. In other cases, the Fredholm alternative or other functional analysis results may be used to establish these properties. Additionally, the regularity of the solution often depends on the smoothness of the data and the boundary conditions, which in turn influences the choice of suitable function spaces for approximation.

2.2 Sparse Approximation

Assuming that (2.1) has a unique solution for each right-hand side function $f \in \mathcal{F}$, our goal is to learn the solution mapping $\mathcal{G} : \mathcal{F} \to \mathcal{U}$, ending up with an apporoximation $\tilde{\mathcal{G}}$ of $\mathcal{G}$.

In order to do so we start by approximating the function f by

$$f(\boldsymbol{x}) \approx \sum_{j=1}^{n} a_j A_j(\boldsymbol{x}) \qquad\qquad \boldsymbol{x} \in \Omega, \tag{2.2}$$

with some fixed functions $A_j, j = 1, \dots, n$, and coefficients $\boldsymbol{a} = (a_1, \dots, a_n) \in \mathbb{C}^n$. Favorable approximative parametrizations of the function f by the coefficients $\boldsymbol{a}$ should be accurate and efficient, i.e., the possible error should be reasonably bounded and there exist fast methods to compute the coefficients $\boldsymbol{a}$ for a given function f and vice versa.

Further, we assume the solution $u : \mathcal{D} \to \mathbb{C}$ to have a basis expansion of the form

$$u(\boldsymbol{x}, \boldsymbol{a}) \coloneqq \sum_{\boldsymbol{k} \in \mathbb{N}^{d+n}} c_{\boldsymbol{k}} \Phi_{\boldsymbol{k}}(\boldsymbol{x}, \boldsymbol{a}) \qquad\qquad (\boldsymbol{x}, \boldsymbol{a}) \in \mathcal{D}, \tag{2.3}$$

with $\{\Phi_{\boldsymbol{k}} : \boldsymbol{k} \in \mathbb{N}^{d+n}\}$ a bounded orthonormal product basis (BOPB) in the separable Hilbert space $\mathcal{H} = L_2(\mathcal{D}, \mu)$ on the Cartesian product type domain $\mathcal{D}$ and basis coefficients $c_{\boldsymbol{k}} \in \mathbb{C}, \boldsymbol{k} \in \mathbb{N}^{d+n}$. See [23, Sec. 1.1] for more details on the notion of a BOPB and the corresponding domains and spaces. Further, see Remark 2.1 for the relation between the domains $\mathcal{D}$ and $\overline{\Omega} \times C^n$, which would be the canonical domain for u since $\boldsymbol{x} \in \overline{\Omega}$ and $\boldsymbol{a} \in \mathbb{C}^n$.

Having (2.3), we aim to approximate u by a truncation

$$S_I u(\boldsymbol{x}, \boldsymbol{a}) \coloneqq \sum_{\boldsymbol{k} \in I} c_{\boldsymbol{k}} \Phi_{\boldsymbol{k}}(\boldsymbol{x}, \boldsymbol{a}), \tag{2.4}$$

with some a priori unknown index set $I \subset \mathbb{N}^{d+n}, |I| < \infty$, followed by an approximation

$$S_I^{\mathcal{A}} u(\boldsymbol{x}, \boldsymbol{a}) \coloneqq \sum_{\boldsymbol{k} \in I} \hat{u}_{\boldsymbol{k}} \Phi_{\boldsymbol{k}}(\boldsymbol{x}, \boldsymbol{a}), \tag{2.5}$$

where $\hat{u}_{\boldsymbol{k}} \in \mathbb{C}, \boldsymbol{k} \in I$, are approximations of the true coefficients $c_{\boldsymbol{k}}$. Note that the detection of a "good" index set I in general leads to a non-linear approximation problem. With (2.5) we then have an approximation of the solution mapping: For every right-hand side f we determine the coefficients $\boldsymbol{a}$ and plug them into $S_I^{\mathcal{A}} u$, which yields us an explicit representation of an approximation of u. So in fact, combining the discretization mapping $f \mapsto \boldsymbol{a}$ with the approximation mapping $\boldsymbol{a} \mapsto S_I^{\mathcal{A}} u(\cdot, \boldsymbol{a})$ yields us an approximation $\tilde{G}$ of the solution map $G : \mathcal{F} \to \mathcal{U}$. Furthermore, the structure of the index set I as well as the size of the corresponding approximated coefficients $\hat{u}_{\boldsymbol{k}}$ may reveal interesting insights on the structure of the solution u and its dependence on the coefficients $\boldsymbol{a}$ and, equivalently, the dependence on the right-hand side function f.

Estimating the error of the approximation in the L_∞ norm and using the boundedness of our basis functions $\Phi_{\boldsymbol{k}}$, we get:

$$\begin{aligned}
\left\| u - S_I^{\mathcal{A}} u \right\|_\infty &\leq \left\| u - S_I u \right\|_\infty + \left\| S_I u - S_I^{\mathcal{A}} u \right\|_\infty \\
&= \left\| \sum_{\boldsymbol{k} \notin I} c_{\boldsymbol{k}} \Phi_{\boldsymbol{k}} \right\|_\infty + \left\| \sum_{\boldsymbol{k} \in I} (c_{\boldsymbol{k}} - \hat{u}_{\boldsymbol{k}}) \Phi_{\boldsymbol{k}} \right\|_\infty \\
&\leq \sum_{\boldsymbol{k} \notin I} |c_{\boldsymbol{k}}| \left\| \Phi_{\boldsymbol{k}} \right\|_\infty + \sum_{\boldsymbol{k} \in I} |c_{\boldsymbol{k}} - \hat{u}_{\boldsymbol{k}}| \left\| \Phi_{\boldsymbol{k}} \right\|_\infty \\
&\leq B \left(\sum_{\boldsymbol{k} \notin I} |c_{\boldsymbol{k}}| + \sum_{\boldsymbol{k} \in I} |c_{\boldsymbol{k}} - \hat{u}_{\boldsymbol{k}}| \right)
\end{aligned}$$

Note that the boundedness constant B depends on the BOPB and therefore on the space $\mathcal{H}$ of our approximation. As an example, consider the well-known Fourier system using trigonometric polynomials $\exp(2\pi \mathrm{i} \langle \boldsymbol{k}, \cdot \rangle)$ as basis functions, which comes with the boundedness constant $B = 1$ independent of the dimenson d. For the Chebyshev basis already introduced in Section 1, the boundedness constant is $B = 2^{d/2}$. If the approximation problem considered has a limited number of interacting dimensions $d_s \leq d$, the effective boundedness constant can be replaced with $B = 2^{d_s/2}$ instead.

The terms inside the brackets, which we will refer to as the truncation error $\sum_{\boldsymbol{k} \notin I} |c_{\boldsymbol{k}}|$ and the coefficient approximation error $\sum_{\boldsymbol{k} \in I} |c_{\boldsymbol{k}} - \hat{u}_{\boldsymbol{k}}|$, are mainly influenced by the index set I and the approximated coefficients $\hat{u}_{\boldsymbol{k}}$. Hence, we need not only to compute good approximations of the coefficients $c_{\boldsymbol{k}}$ but also to detect a "good" index set I, hopefully containing the largest (in terms of absolute value) coefficients $c_{\boldsymbol{k}}$, to make (2.5) a reasonable approximation of the solution u.

2.3 The dimension-incremental method

In the present work, we will use the nonlinear approximation method for high-dimensional function approximation proposed in [23] in order to receive the desired approximation (2.5). First, we summarize some main aspects of the dimension-incremental algorithm here and refer to [23, Sec. 2] for more detailed explanations and proper definitions of the used notations. A simplified version of the algorithm is also given in Algorithm 1.

Suppose for now for simplicity that we are interested in the approximation of a d-dimensional target function $g : \mathcal{D} \to \mathbb{C}$ of the form $g = \sum_{\boldsymbol{k} \in I} \hat{g}_{\boldsymbol{k}} \Phi_{\boldsymbol{k}}$ with the unknown index set $I \subset \mathbb{N}^d$. Later, the target function g will be the solution u in $d + n$ dimensions. Motivated by the

estimate above, we aim for an s-sparse index set I, i.e., we have $|I| = s$, corresponding to basis coefficients $c_{\boldsymbol{k}}$ with large absolute values.

Roughly spoken, the algorithm uses samples of g to detect reasonable indices k_j of $\boldsymbol{k} = (k_j)_{j=1}^d, \boldsymbol{k} \in I$, in each dimension $j = 1, \dots, d$ and reasonable combinations thereof. In order to do so, only a search space $\Gamma \supset I$ is needed in advance. Commonly, we choose search spaces like the full (non-negative) grid $\Gamma = [0, N]^d$ with some parameter $N \in \mathbb{N}$, which we will call extension from now on. If there is additional initial knowledge on the structure of the desired index set I, the choice of Γ can be improved.

The algorithm starts by investigating the one-dimensional projections $\mathcal{P}_{\{t\}}(\Gamma) \coloneqq \{k \in \mathbb{N} \,|\, \exists \boldsymbol{k} \in \Gamma : \boldsymbol{k}_t = k\}$ for all $t = 1, \dots, d$ by constructing a suitable cubature rule for integrals of the form

$$\hat{g}_{\{t\},k_t}(\tilde{\boldsymbol{x}}) \coloneqq \int g(\xi, \tilde{\boldsymbol{x}})_{\{t\}} \overline{\Phi_{\{t\},k_t}(\xi)} \mathrm{d}\xi \tag{2.6}$$

with $\Phi_{\{t\},k_t}$ the one-dimensional basis function of the t-th dimension of our BOPB. The notation $g(\xi, \tilde{\boldsymbol{x}})_{\{t\}}$ refers to sampling values of g using ξ in the t-th dimension and $\tilde{\boldsymbol{x}}$ for the remaining dimensions. The algorithm then computes these so-called projected coefficients $\hat{g}_{\{t\},k_t}$ using this cubature rule and samples of the target function g for a particular random anchor $\tilde{\boldsymbol{x}}$. The absolute value of these projected coefficients $\hat{g}_{\{t\},k_t}$ can be seen as an indicator whether or not k_t is important, i.e., if k_t should appear in the t-th component of any index $\boldsymbol{k} \in I$. Hence, the algorithm takes the sparsity s largest projected coefficients fulfilling $|\hat{g}_{\{t\},k_t}| \geq \delta_+$ for some initially chosen detection threshold δ_+ and adds the corresponding indices k_t to a temporary index set $I_{\{t\}}$. Since the computation of the projected coefficients $\hat{g}_{\{t\},k_t}$ involves randomness due to the randomly drawn anchor $\tilde{\boldsymbol{x}}$, this computation is repeated r times with r being the number of detection iterations with different anchors $\tilde{\boldsymbol{x}}^{(j)}, j = 1, \dots, r$.

The temporary index sets $I_{\{t\}}$ with the reasonable indices for each dimension $t = 1, \dots, d$ are now combined to proceed in a dimension-incremental way. Starting with $t = 2$ a new candidate set $K \coloneqq (I_{\{1,\dots,t-1\}} \times I_{\{t\}}) \cap \mathcal{P}_{\{1,\dots,t\}}(\Gamma)$ is formed, containing now higher-dimensional indices $\boldsymbol{k} \in \mathbb{N}^{|\{1,\dots,t\}|}$. Again, a suitable t-dimensional cubature method, e.g., multiple rank-1 lattices as in [20], is constructed and evaluated using samples of the target function g to compute the projected coefficients

$$\hat{g}_{\{1,\dots,t\},\boldsymbol{k}}(\tilde{\boldsymbol{x}}) \coloneqq \int g(\boldsymbol{\xi}, \tilde{\boldsymbol{x}}) \overline{\Phi_{\{1,\dots,t\},\boldsymbol{k}}(\boldsymbol{\xi})} \mathrm{d}\boldsymbol{\xi}$$

for the indices $\boldsymbol{k} \in K$, which is the natural generalization of (2.6) to multiple dimensions $\{1, \dots, t\}$. As before, the algorithm collects those indicies $\boldsymbol{k} \in \mathbb{N}^t$ with the sparsity s largest (in absolute value) projected coefficients $\hat{g}_{\{1,\dots,t\},\boldsymbol{k}}$ in the temporary index set $I_{\{1,\dots,t\}}$. These indices are now the tuples, which can still appear in the first t components of the indices in the final index set I. If $t < d$ holds, this process is again influenced by the randomly chosen anchor $\tilde{\boldsymbol{x}}$ and therefore repeated r times. This process is then repeated with $t + 1$ instead of t until $t = d$, where the projected coefficients $\hat{g}_{\{1,\dots,d\},\boldsymbol{k}}$ do not longer depend on a random anchor $\tilde{\boldsymbol{x}}$ at all. We finally set $I = I_{\{1,\dots,d\}}$ and $\hat{g}_{\boldsymbol{k}} \coloneqq \hat{g}_{\{1,\dots,d\},\boldsymbol{k}}$ for all $\boldsymbol{k} \in I_{\{1,\dots,d\}}$. The final output of the algorithm is the desired index set I as well as approximations $\hat{g}_{\boldsymbol{k}}$ of the true basis coefficients for each $\boldsymbol{k} \in I$.

A crucial requirement for this algorithm is the possibility to access the sampling values $g(\boldsymbol{x})$ for arbitrary $\boldsymbol{x}$ during the algorithm, e.g. by a black-box function handle of g. This is since the

Algorithm 1 Dimension-incremental Algorithm (Simplified)

Input:	$\Gamma \subset \mathbb{N}^d$	search space
	g	target function g as black box (function handle)
	$s \in \mathbb{N}$	sparsity parameter
	$\delta_+ > 0$	detection threshold
	$r \in \mathbb{N}$	number of detection iterations

(Step 1) [Single component identification]
for $t := 1, \dots, d$ **do**
Set $I_{\{t\}} := \emptyset$.
Compute a suitable cubature method for $\mathcal{P}_{\{t\}}(\Gamma)$.
for $i := 1, \dots, r$ **do**
Draw a random anchor $\tilde{\boldsymbol{x}}$.
Sample g at the necessary sampling points (the cubature nodes combined with $\tilde{\boldsymbol{x}}$).
Compute the projected coefficients $\hat{g}_{\{t\},k_t}(\tilde{\boldsymbol{x}})$ for $k_t \in \mathcal{P}_{\{t\}}(\Gamma)$.
Add the (up to) s indices k_t with the largest proj. coef. $|\hat{g}_{\{t\},k_t}(\tilde{\boldsymbol{x}})| \geq \delta_+$ to $I_{\{t\}}$.
end for i
end for t
(Step 2) [Coupled component identification]
for $t := 2, \dots, d$ **do**
If $t < d$, set $\tilde{r} := r$ and otherwise $\tilde{r} := 1$.
Set $I_{\{1,\dots,t\}} := \emptyset$.
Construct the index set $K := (I_{\{1,\dots,t-1\}} \times I_{\{t\}}) \cap \mathcal{P}_{\{1,\dots,t\}}(\Gamma)$.
Compute a suitable cubature method for K.
for $i := 1, \dots, \tilde{r}$ **do**
Draw a random anchor $\tilde{\boldsymbol{x}}$.
Sample g at the necessary sampling points (the cubature nodes combined with $\tilde{\boldsymbol{x}}$ if $t < d$).
Compute the projected coefficients $\hat{g}_{\{1,\dots,t\},\boldsymbol{k}}(\tilde{\boldsymbol{x}})$ for $\boldsymbol{k} \in K$.
Add the (up to) s indices $\boldsymbol{k}$ with the largest proj. coef. $|\hat{g}_{\{1,\dots,t\},\boldsymbol{k}}(\tilde{\boldsymbol{x}})| \geq \delta_+$ to $I_{\{1,\dots,t\}}$.
end for i
end for t
(Step 3)
Set $I := I_{\{1,\dots,d\}}$ and $\hat{g}_{\boldsymbol{k}} := \hat{g}_{\{1,\dots,d\},\boldsymbol{k}}$ for all $\boldsymbol{k} \in I_{\{1,\dots,d\}}$.

Output:	$I \subset \Gamma \subset \mathbb{N}^d$	detected index set
	$(\hat{g}_{\boldsymbol{k}})_{\boldsymbol{k} \in I} \in \mathbb{C}^{\lvert I \rvert}$	approximated coefficients with $\lvert \hat{g}_{\boldsymbol{k}} \rvert \geq \delta_+$

necessary sampling points $\boldsymbol{x}$, for which the corresponding sampling values $g(\boldsymbol{x})$ are needed, are not known a priori, but are computed adaptively during the algorithm based on the current candidate sets K and the constructed cubature methods, combined with the random anchors $\tilde{\boldsymbol{x}}$. We again encourage the reader to see [23] for a more detailed and rigorous explanation of this concept, the theorem on the theoretical detection guarantee and simple numerical examples as well as several comments and discussions on the capabilities and restrictions of this algorithm. Further, similar approaches as in [39, Sec. 3.1.2] for the selection of the largest projected coefficients $\hat{g}_{\{1,\dots,t\},\boldsymbol{k}}(\tilde{\boldsymbol{x}})$ of the candidate sets K in each step might speed up these detections and therefore the whole algorithm, if additional information on the behavior of these projected coefficients is known.

2.4 Black-box sampling the differential equation

For our application of Algorithm 1, we have u as the target function and hence sampling points of the form $(\boldsymbol{x}, \boldsymbol{a})$. Each "sample" $u(\boldsymbol{x}, \boldsymbol{a})$ is then the value of the solution u of (2.1) for a given parameter $\boldsymbol{a}$, evaluated at the spatial point $\boldsymbol{x}$. In order do receive such samples, we utilize numerical solvers, e.g. the finite element method. With the given parameter $\boldsymbol{a}$, we can compute the (fixed) right-hand side function f, solve the differential equation for this particular f numerically and evaluate the approximated solution at $\boldsymbol{x}$.

Since the algorithm only requests the final sampling value $u(\boldsymbol{x}, \boldsymbol{a})$, the choice of the particular method or numerical solver for the differential equation is completely free, as long as the approximated sampling value we compute and return to the algorithm is a reasonable approximation of the true value $u(\boldsymbol{x}, \boldsymbol{a})$. This non-intrusive behavior of our algorithm is the reason for its generality. The properties of the differential equation (2.1) as well as possible difficulties therein are mostly dealt with by the numerical solver. As long as there is any possibility to obtain reasonable estimates of $u(\boldsymbol{x}, \boldsymbol{a})$ for a given $(\boldsymbol{x}, \boldsymbol{a})$, we can plug this method into our black-box sampling step and are done. Note that while the accuracy and efficiency of the numerical solver used obviously directly affect the accuracy and efficiency of our dimension-incremental method, we will not investigate the properties of these solvers in more detail in this work.

Remark 2.1. *Generally, the product type domain $\mathcal{D}$, where we can apply Algorithm 1, will not coincide with the domain $\overline{\Omega} \times \mathbb{C}^n$ of our solution u. Hence, we need to transform and or restrict this domain carefully, such that the sampling points given by our algorithm are suitable for the differential operator.*

Since $\overline{\Omega}$ will be some compact domain for many applications, it is often enough to apply a simple transformation $\mathcal{T}$ for the spatial variable $\boldsymbol{x}$, e.g., the continuous and bijective linear transformation $\mathcal{T}\boldsymbol{x} = m_1\boldsymbol{x} + m_2\mathbf{1}$ with two constants $m_1, m_2 \in \mathbb{R}$, mapping $\boldsymbol{x}$ to the desired domain $\overline{\Omega}$.

On the other hand, the parameters $\boldsymbol{a} \in \mathbb{C}^n$ are more difficult to handle. In this case, we will often have to restrict the domain of $\boldsymbol{a}$ to e.g. some compact interval again, before thinking about possible transformations as we did for the spatial part. This is obviously a loss of generality and the restriction needs to be performed carefully, such that most reasonable source functions f can still be approximated well enough using the restricted $\boldsymbol{a}$.

For examples of such transformations and restrictions, we refer to the particular examples in the following section.

Remark 2.2. *While we stick to the mentioned setting* (2.1) *for the theoretical part of this paper to preserve clarity in the notations, our numerical experiments in Section 3 also include some further variations, which we will only briefly mention in this remark.*

First, we can also consider parametrized differential operators $\mathcal{L}_{\boldsymbol{\theta}}$ with some parameter $\boldsymbol{\theta} \in \mathbb{R}^{n_\theta}, n_\theta \in \mathbb{N}$, and the corresponding solution mapping $\mathcal{G} : \mathcal{F} \times \mathbb{R}^n \to \mathcal{U}$. Correspondingly, (2.3) *then becomes*

$$u(\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{\theta}) \coloneqq \sum_{\boldsymbol{k} \in \mathbb{N}^{d+n+n_\theta}} c_{\boldsymbol{k}} \Phi_{\boldsymbol{k}}(\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{\theta}) \tag{2.7}$$

with another separable Hilbert space $\mathcal{H}$ and corresponding BOPB $\{\Phi_{\boldsymbol{k}} : \boldsymbol{k} \in \mathbb{N}^{d+n+n_\theta}\}$. The truncated and approximated version (2.5) *is then modified in the same way, now with an unknown index set $I \subset \mathbb{N}^{d+n+n_\theta}$. An example of this variation can be found in Section 3.4.*

Similarly, we can consider time-dependent differential operators $\mathcal{L}$ with respect to some time variable $\tau \in [0,T]$ and their corresponding solution mapping $\mathcal{G} : \mathcal{F} \times [0,T] \to \mathcal{U}$. As before, we end up with the representation

$$u(\boldsymbol{x}, \tau, \boldsymbol{a}) \coloneqq \sum_{\boldsymbol{k} \in \mathbb{N}^{d+1+n_\theta}} c_{\boldsymbol{k}} \Phi_{\boldsymbol{k}}(\boldsymbol{x}, \tau, \boldsymbol{a})$$

and proceed similarly as above, now with the $d+1+n$-dimensional separable Hilbert space $\mathcal{H}$ and an unknown index set $\mathrm{I} \subset \mathbb{N}^{d+1+n}$.

In each case, we proceed to the approximation $S_I^{\mathcal{A}} u$ from (2.5). This time, the analysis of the index set I and the coefficients $\hat{u}_{\boldsymbol{k}}$ can give additional information about the dependence and interaction of the spatial variable $\boldsymbol{x}$ and the right-hand side f not only with each other, but also with the parameters $\boldsymbol{\theta}$ or the time variable t.

Obviously, a combination of these two variations, i.e. a parameter- and time-dependent differential equation, can be treated in an analogous way, cf. Section 3.5 with the one-dimensional heat equation.

3 Numerics

In this Section, we will test our approach on several test problems, such as multiple diffusion equations and Burgers' equation, and discuss the results. We show, that our approach leads to sparse index sets I that can be used directly for the high-dimensional approximation of the solution u or further generalized to even higher-dimensional problems. We will briefly investigate such a generalization for the first model example at the end of Section 3.1.1. For the other examples, we focus on the detection of a suitable index set I and omit the generalization to higher dimensions, since the first part is the main goal of Algorithm 1.

As mentioned in Section 2, we need to choose a suitable BOPB for the solution u to achieve the basis expansion (2.3). In all of the following examples, we will work with the tensorized Chebyshev polynomials

$$T_{\boldsymbol{k}}(\boldsymbol{z}) \coloneqq \prod_{j=1}^{d+n} T_{k_j}(z_j) \quad \text{with} \quad T_{k_j}(z_j) = \begin{cases} 1 & k_j = 0 \\ \sqrt{2}\cos(k_j \arccos(z_j)) & k_j \neq 0 \end{cases}$$

on the domain $\mathcal{D} = [-1,1]^{d+n}$ as used in [23]. Hence, the approximation (2.5) inserting $\boldsymbol{z} \coloneqq (\boldsymbol{x}, \boldsymbol{a})$ with $\boldsymbol{x} \in \mathbb{R}^d$ and $\boldsymbol{a} \in \mathbb{C}^n$ becomes

$$S_I^{\mathcal{A}} u(\boldsymbol{x}, \boldsymbol{a}) = \sum_{\boldsymbol{k} \in I} \hat{u}_{\boldsymbol{k}} T_{\boldsymbol{k}}(\boldsymbol{x}, \boldsymbol{a}), \tag{3.1}$$

where $I \subset \mathbb{N}^{d+n}$ is the detected index set and $\hat{u}_{\boldsymbol{k}}$ are the approximations of the corresponding exact basis coefficients $c_{\boldsymbol{k}}$ from (2.3).

To investigate the accuracy of our method, we consider for a fixed coefficient $\boldsymbol{a} \in \mathbb{C}^n$ the relative ℓ_2-error

$$\mathrm{err}(\boldsymbol{a}) \coloneqq \frac{\left\| S_I^{\mathcal{A}} u(\boldsymbol{x}, \boldsymbol{a}) - u(\boldsymbol{x}, \boldsymbol{a}) \right\|_{\ell_2}}{\left\| u(\boldsymbol{x}, \boldsymbol{a}) \right\|_{\ell_2}} = \frac{\left(\sum_{j=1}^{G} \left| S_I^{\mathcal{A}} u(\boldsymbol{x}^{(j)}, \boldsymbol{a}) - u(\boldsymbol{x}^{(j)}, \boldsymbol{a}) \right|^2 \right)^{\frac{1}{2}}}{\left(\sum_{j=1}^{G} \left| u(\boldsymbol{x}^{(j)}, \boldsymbol{a}) \right|^2 \right)^{\frac{1}{2}}}, \tag{3.2}$$

where $\boldsymbol{x}^{(j)}$ for $j = 1, \dots, G$ are equidistant grid points in the spatial domain Ω. We then proceed by computing this error for numerous, randomly drawn coefficients $\boldsymbol{a}$ and investigating the corresponding range as well as the first quartile, the median and the second quartile of this statistical test (dividing the results into four equal parts).

All tests are performed in Python$^{\mathrm{TM}}$ and can be found together with the algorithm in [35]. The overall runtime depends significantly on the specific problem, the desired accuracy, the chosen parameters, and the available computational resources. Therefore, the runtime for each example is briefly discussed in the corresponding sections. To accelerate the sampling process, we used parallelization with up to 180 separate workers, but not GPU computing. If not stated otherwise, the dimension-incremental algorithm uses the following parameters and settings:

- the cubature method: Chebyshev multiple rank-1 lattices as described in [23, Sec. 4.2]
- the search space Γ: (non-negative) full grid $[0, N]^{d+n}$ in $d+n$ dimensions with extension N and no superposition assumption
- the detection threshold $\delta_+ = 10^{-12}$
- the number of detection iterations $r = 5$.

The sparsity s will be given for each test separately. See [23] for more detailed information on these parameters and settings and how they affect the behavior of the algorithm.

3.1 The one-dimensional Poisson equation

The following example considers a rather simple differential equation in order to demonstrate the application of our proposed method for the first time.

Given a source function $f : (0,1) \to \mathbb{C}$, the one-dimensional Poisson equation with homogeneous Dirichlet boundary conditions reads as

$$\begin{aligned} -\frac{d^2}{dx^2} u(x) &= f(x), \quad x \in (0,1), \\ u(0) = u(1) &= 0. \end{aligned} \tag{3.3}$$

For this differential operator $\mathcal{L} = -\frac{d^2}{dx^2}$ and these particular boundary conditions, we go with the usual choice of function spaces $\mathcal{U} = H_0^1((0,1))$ and $\mathcal{F} = H^{-1}((0,1))$, the dual of $H_0^1((0,1))$. As described in Section 2, our first step is to find a suitable parametrization (2.2) of the function f. For this first example, we will consider two different approaches here:

- a parametrization of the source function f by its first Fourier coefficients,
- a parametrization of the source function f by a B-Spline approximation.

In all of these cases we restrict ourselves to a discretization of f using only n parameters. Together with the spatial dimension $d = 1$ this results in a $n+1$-dimensional approximation problem. While choosing a larger n should lead to more accurate approximations of f and thus to an overall better quality of the approximation $S_I^{\mathcal{A}} u$ for general f, the additional dimensions will also result in higher sampling and computational complexities. Therefore, we have to choose reasonable limits for n in the upcoming examples.

3.1.1 Fourier series parametrization

We consider a parametrization of the source function f by its first $n \in 2\mathbb{N}+1$ Fourier coefficients $\boldsymbol{a} = (a_{-\frac{n-1}{2}}, \dots, a_{\frac{n-1}{2}}) \in \mathbb{C}^n$, i.e.,

$$f(x) \approx \sum_{\ell=-\frac{n-1}{2}}^{\frac{n-1}{2}} a_\ell \mathrm{e}^{2\pi \mathrm{i} \ell x}. \tag{3.4}$$

These Fourier coefficients $\boldsymbol{a}$ can be computed efficiently for reasonable functions f using the well known FFT. Note that this approximation of f will always be 1-periodic, forcing the implicit assumption that the function f is either a 1-periodic function itself or can be well approximated by such a function up to some extend.

Using this truncated Fourier series as the right-hand side of the differential equation (3.3) the solution u of the one-dimensional Poisson equation is then given analytically by

$$u(x, \boldsymbol{a}) = \frac{a_0}{2} x(1-x) + \sum_{\substack{\ell=-\frac{n-1}{2} \\ \ell \neq 0}}^{\frac{n-1}{2}} \frac{a_\ell}{4\pi^2 \ell^2} (\mathrm{e}^{2\pi \mathrm{i} \ell x} - 1). \tag{3.5}$$

This formula can be used directly as the black-box sampling strategy necessary for our algorithm, see Section 2.4, to generate the necessary sampling values $u(x^*, \boldsymbol{a}^*)$ for any sampling point $(x^*, \boldsymbol{a}^*)$. To demonstrate the general application of Algorithm 1, we use this direct method to be able to neglect errors made when solving the differential equation numerically and focus on the approximation of the solution u directly.

As mentioned in Remark 2.1, we need to pay attention since the original domain $[0,1] \times \mathbb{C}^n$ doesn't match our function approximation domain $\mathcal{D} = [-1,1]^{n+1}$. For the spatial part, we apply the transformation $\mathcal{T} x = \frac{1}{2}(x+1)$ to perform the shift between $[-1,1]$ and $[0,1]$. For simplicity, we directly assume the restriction $\boldsymbol{a} \in [-1,1]^n$ for the Fourier coefficients $\boldsymbol{a}$ such that we can omit further transformations. Note that this implies that we are only interested in right-hand side functions f, which can be well approximated by using such Fourier coefficients $\boldsymbol{a}$ during this artificial example. Overall, the final function, which we are going to approximate here, is now

$$\tilde{u}(\tilde{x}, \boldsymbol{a}) \coloneqq u(\mathcal{T}\tilde{x}, \boldsymbol{a}) = u\left(\frac{1}{2}(\tilde{x}+1), \boldsymbol{a}\right) \qquad \tilde{x} \in [-1,1], \boldsymbol{a} \in [-1,1]^n.$$

Using the explicit formula (3.5), we get

$$\tilde{u}(\tilde{x}, \boldsymbol{a}) = \frac{a_0}{8}(1-\tilde{x}^2) + \sum_{\substack{\ell=-\frac{n-1}{2} \\ \ell \neq 0}}^{\frac{n-1}{2}} \frac{a_\ell}{4\pi^2 \ell^2} ((-1)^\ell \mathrm{e}^{\pi \mathrm{i} \ell \tilde{x}} - 1). \tag{3.6}$$

Remark 3.1. *Note that the particular choice of the domain $\mathcal{D}$ and the corresponding basis of $\mathcal{H}$ are not unique. Since we are only restricting the Fourier coefficients $\boldsymbol{a}$ and not transforming them, the solution u is obviously not periodic with respect to these variables, so our decision to use the tensorized Chebyshev polynomials for the approximation is reasonable here. However, we could have applied various transformations $\mathcal{T}$ to $\boldsymbol{a}$, including those that force a periodic*

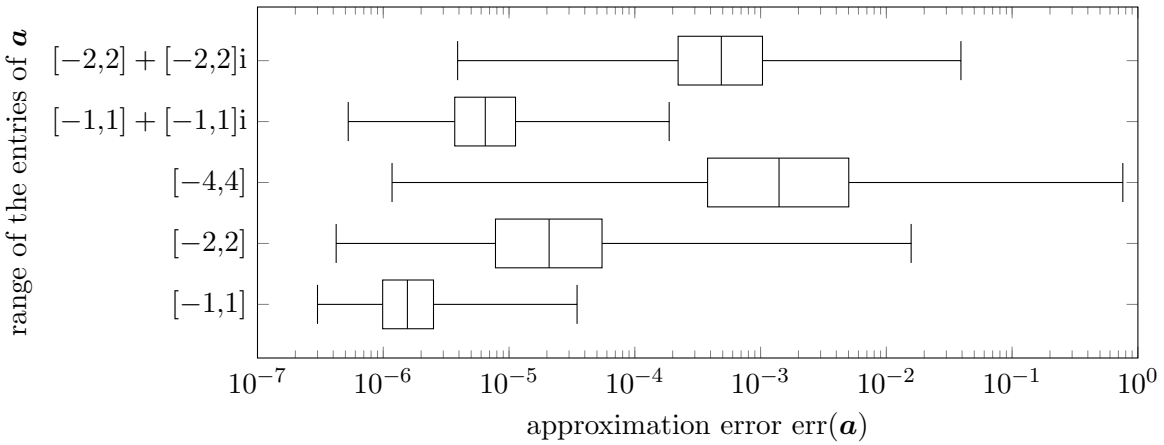


Figure 3.1: The relative approximation error $\mathrm{err}(\boldsymbol{a})$ for 10000 randomly drawn $\boldsymbol{a}$ when using the Fourier series parametrization. The box-and-whisker plots show the median, the first and the second quartile as well as the maximal and minimal error observed. The five plots indicate different choices for the range of the Fourier coefficient $\boldsymbol{a}$, including two cases with complex-valued Fourier coefficients. The range $[-1, 1]$ coincides with the training data used.

dependence of u on $\boldsymbol{a}$ such as the tent-transform, cf. [7]. Then, together with the periodicity in $x \in [0, 1]$ due to the boundary conditions $u(0) = u(1) = 0$, the solution u would be periodic (but not smooth) in all $n + 1$ dimensions. In such a scenario, we could use the high-dimensional torus domain $\mathcal{D} = \mathbb{T}^{n+1}$ as well as a Fourier basis for the approximation space $\mathcal{H}$.

We use $n = 9$ as the amount of Fourier coefficients a_ℓ for our tests, which results in the overall dimension $d + n = 10$. Further, we choose the sparsity $s = 1000$ and the extension $N = 64$ of the search space Γ. Since we do not use a numerical solver but the exact solution, each solution sample can be generated in about 10^{-4} seconds, resulting in an overall runtime of our algorithm of about 2 minutes.

The accuracy of our approximation is shown in Figure 3.1. Therein, we used 10000 randomly drawn Fourier coefficients $\boldsymbol{a}$ and computed the relative ℓ_2-error $\mathrm{err}(\boldsymbol{a})$ using $G = 1000$ equidistant grid points in the spatial domain. We use box-and-whisker plots to illustrate the statistical distribution here, where the central line inside the box indicates the median. On each side of the median, the box contains 25% of the data. Outside the box, the whiskers indicate the maximal and minimal error observed. Since we did not specify outliers in our data, the box-and-whisker plot truly covers the full range of observed errors.

The first plot with the range $[-1, 1]$ is the true approximation error, since it used the same range for the entries of $\boldsymbol{a}$ as we used during our approximation. Although computed coefficients $\hat{u}_{\boldsymbol{k}}$ from (3.1) smaller than 10^{-7} are not necessarily true basis coefficients but mainly artifacts because of numerical errors, the overall approximation accuracy is still satisfactory. The other plots in Figure 3.1 show results with larger or even complex domains for the test Fourier coefficients $\boldsymbol{a}$. For this transfer learning scenario it shows, that our approximation is also applicable for slightly larger domains of $\boldsymbol{a}$, and therefore more source functions f, than the restricted ones from the training setting. However, the further the test cases are from the training setting, the larger our relative approximation error $\mathrm{err}(\boldsymbol{a})$ grows. Thus, we strongly recommend using another restriction $\boldsymbol{a} \in [-\alpha, \alpha]^n$, $\alpha > 1$, and transforming them similarly

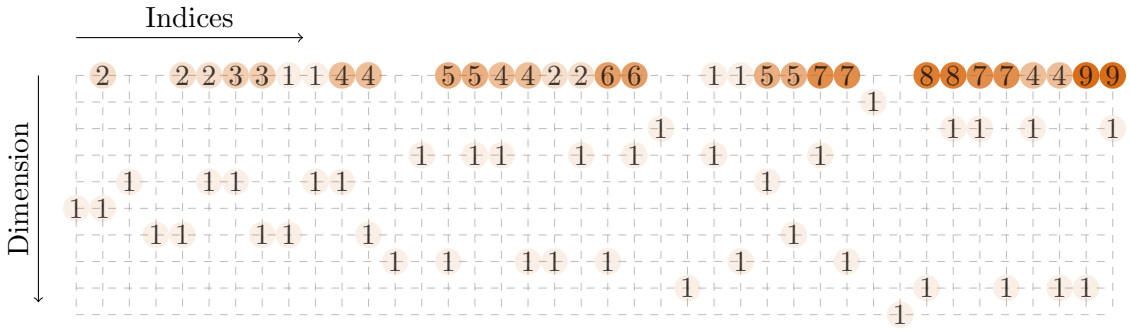


Figure 3.2: An abstract visualization of the first 40 indices $\boldsymbol{k}$ detected when using the Fourier series parametrization. The leftmost column contains the index $\boldsymbol{k}$ corresponding to the largest (in absolute value) basis coefficient $\hat{u}_{\boldsymbol{k}}$, the second column the index for the second largest and so on. The rows identify the 10 dimensions corresponding to the variables x and $a_{-4}, \dots, a_4$ from top to bottom in this order. Zeros are neglected to preserve clarity.

to the spatial variable x already in the training process, if desired functions f require such Fourier coefficients $\boldsymbol{a}$.

The detected indices show a clear structure as can be seen for the first indices in Figure 3.2. For all dimensions corresponding to an entry of the Fourier coefficient vector $\boldsymbol{a}$, there exists no other entry than 0 or 1. This effect is not caused by the particular sparsity s we have chosen, since the algorithm already neglects every other possible entry (so the numbers from 2 to 64 for our choice of N) in the single component identification step, cf. Step 1 in Algorithm 1, such that they can never appear at all in these dimensions. This result is exactly what we expected knowing the explicit formula (3.6), since therein all the Fourier coefficients $a_{-4}, \dots, a_4$ appear only linearly. Additionally, the only indices where the entry corresponding to a_0 is non-zero are the first and second one, which can be also seen in Figure 3.2 as the first and second column. Again, this matches our expectations, since a_0 only appears in the first term of (3.6), which can be rewritten in terms of the Chebyshev polynomials as

$$\begin{aligned}\frac{a_0}{8}(1-\tilde{x}^2) &= \frac{1}{8\sqrt{2}} T_1(a_0)\left(\frac{1}{2}T_0(x) - \frac{1}{2\sqrt{2}}T_2(x)\right) \prod_{\substack{\ell=-\frac{n-1}{2} \\ \ell\neq 0}}^{\frac{n-1}{2}} T_0(a_\ell) \\ &= \frac{1}{16\sqrt{2}} T_{\boldsymbol{k}^{(1)}}(x, \boldsymbol{a}) - \frac{1}{32} T_{\boldsymbol{k}^{(2)}}(x, \boldsymbol{a}),\end{aligned}$$

with

$$\boldsymbol{k}^{(1)} = [\underbrace{0}_{T_0(x)}, \underbrace{0,0,0,0}_{T_0(a_\ell)}, \underbrace{1}_{T_1(a_0)}, \underbrace{0,0,0,0}_{T_0(a_\ell)}]^T$$

and

$$\boldsymbol{k}^{(2)} = [\underbrace{2}_{T_2(x)}, \underbrace{0,0,0,0}_{T_0(a_\ell)}, \underbrace{1}_{T_1(a_0)}, \underbrace{0,0,0,0}_{T_0(a_\ell)}]^T$$

being the indices mentioned above. Finally, we would expect no couplings between the different Fourier coefficients $a_{-4}, \dots, a_4$ since they never appear together in the parts of the sum in the right part of (3.6). While this behavior can be observed for the first detected indices in Figure 3.2, this does not hold for all of our detected indices. At some point, the value of the remaining true Chebyshev coefficients become so small, that the algorithm can not distinguish their corresponding indices from false ones like $[2, 1, 0, 1, 1, 1, 0, 1, 0, 1]^T$, which seem to produce similar coefficient values due to small numerical errors. However, since the size of the coefficients where this effect happens is already very small, i.e. about 10^{-8}, this does not harm the overall approximation. Obviously, this minor problem is simply caused by the large sparsity $s = 1000$ and could also be prevented up to some extend by using a search space Γ that does not contain indices $\boldsymbol{k}$ with so many non-zero entries.

Example 3.2 (High-dimensional extension of the detected index set I)**.** *As stated already in Section 1, we can use the structure of the detected index set I to extend our approach to even higher dimensions. We demonstrate this approach here for the current differential equation with the Fourier series parametrization due to its simple structure and our explicit knowledge of the true solution* (3.6)*.*

We increase the number n of Fourier coefficients a_ℓ used to 99 *in order to achieve a better resolution of the right-hand side function f than before. Obviously, this leads us to the approximation of the now* 100*-dimensional function $u(x, \boldsymbol{a})$. However, analyzing the detected index set I from our* 10*-dimensional test above, we can construct a good index set I directly by generalizing the main structural features of I. In detail, we will construct our new index set I in the following way:*

- *The first dimension (corresponding to the spatial variable x) may contain any number from* 0 *to N_x.*
- *The entries of the dimensions* 2 *to* 100 *are either all zero or contain at most one non-zero entry. This non-zero entry, if existing, must be* 1*.*

This index set I then contains $(N_x + 1) \cdot 100$ indices of a similar structure as in Figure 3.2. Note that we did not include the fact, that there were only two indices $\boldsymbol{k}^{(1)}$ and $\boldsymbol{k}^{(2)}$ with a non-zero entry in the dimension corresponding to a_0.

We perform our test using $N_x = 999$, so using an index set I containing 10^5 indices $\boldsymbol{k}$. We compute the corresponding basis cofficients $u_{\boldsymbol{k}}$ using the same Chebyshev multiple rank-1 lattice approach from [21] as before in Algorithm 1. Our full dimension-incremental algorithm in just 10 *dimensions already needed around* 400000 *samples for this simple example. Now we only need about* 300000 *samples to approximate all the basis coefficients $\hat{u}_{\boldsymbol{k}}, \boldsymbol{k} \in I$ in this 100-dimensional example. This would be an impossible goal when applying our full algorithm 1 directly to the* 100*-dimensional approximation problem instead. Especially for real applications, where the sampling values are not generated by an explicit formula but by a differential equation solver (like the FEM), the reduction of the amount of samples needed is an important tool, since the corresponding calls of the differential equation solver will be the dominating part of the computational complexity of the whole algorithm. The same problem appeared in [22] and was the main motivation for the method proposed there. The relative approximation errors range from 10^{-8} to at most 10^{-6}, which is a further improvement compared to the relative errors for the range $[-1, 1]$ in Figure 3.1. Note that all these tests were performed with right-hand side functions f of the form* (3.4)*, so as before there was no*

error in the discretization of this function. However, the higher resolution of this approach with $n = 99$ allows for a much better discretization error (of the function f) if we work with more general right-hand side functions f.

3.1.2 B-spline parametrization

We approximate the right-hand side f by a sum of n B-splines, i.e.

$$f(x) \approx \sum_{\ell=0}^{n-1} a_\ell B_\ell^{(m)}(x),$$

where $B_\ell^{(m)}$ are versions of the cardinal B-spline $B^{(m)}$ of order m. Originally, they are recursively defined via

$$B^{(1)}(x) \coloneqq \begin{cases} 1, & -\frac{1}{2} < x < \frac{1}{2} \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad B^{(m)}(x) \coloneqq \int_{x-\frac{1}{2}}^{x+\frac{1}{2}} B^{(m-1)}(y)\mathrm{d}y.$$

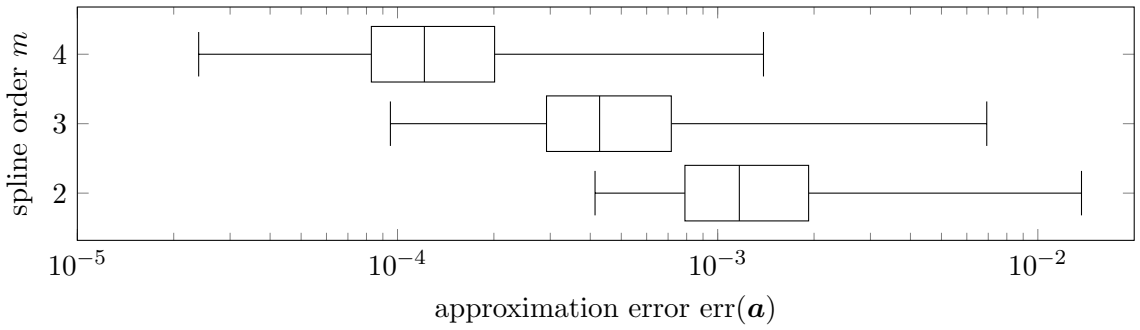
We use the additional index ℓ to indicate, that we scaled and shifted them w.r.t. the interval $[0, 1]$ and the desired amount of B-splines n, such that their peaks are equidistantly spaced along the interval and each spline overlaps $m - 1$ neighboring splines in each direction.

This time, we use a classical differential equation solver to acquire the sample values $u(x^*, \boldsymbol{a}^*)$ for any sampling point $(x^*, \boldsymbol{a}^*)$. In particular, we will apply the function `solve_bvp` from the submodule `scipy.integrate` here, which is capable of solving first order systems of ODEs with two-point boundary conditions. As mentioned in Remark 2.1, we need to transform the points $(x, \boldsymbol{a})$ such that they fit to the domain $\mathcal{D} = [-1, 1]^{n+1}$. We perform the same steps as in the previous example by transforming $\mathcal{T}x = \frac{1}{2}(x + 1)$ and restricting $\boldsymbol{a} \in [-1, 1]^n$ throughout this example. Hence, once again the final function, which we are going to approximate, is $\tilde{u}(\tilde{x}, \boldsymbol{a}) \coloneqq u(\mathcal{T}\tilde{x}, \boldsymbol{a}) = u(\frac{1}{2}(\tilde{x} + 1), \boldsymbol{a})$.

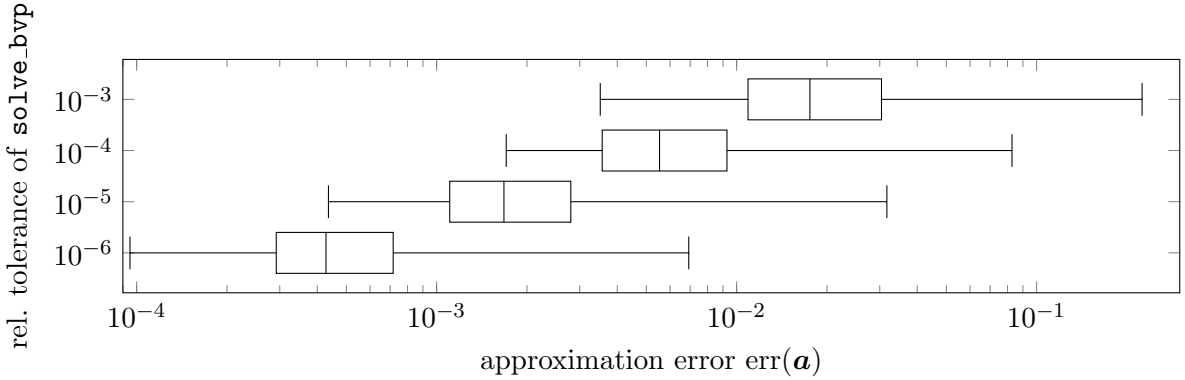
Additionally, we also use the same amount $n = 9$ of Spline coefficients a_ℓ, again resulting in 10-dimensional approximation problem. The sparsity $s = 1000$ and the extension $N = 64$ of the search space Γ also remain the same. With the differential equation solver needing roughly 0.1 seconds per call, we ended up with an overall runtime of our method of almost 6 hours.

The approximation error shown in Figure 3.3 is derived by evaluating our approximation as well as the solution given by the function `solve_bvp` on 1000 equidistant spatial points and considering the respective ℓ_2-error $\mathrm{err}(\boldsymbol{a})$. The reference solution for the error erstimation was obtained by using a relative tolerance of 10^{-9} in `solve_bvp`. As in the previous example, we use box-and-whisker plots to visualize the statistical distribution of the results, i.e., the range and the median of the observed errors as well as their quartiles.

Figure 3.3a shows the results for different choices of the spline order m when parametrizing the right-hand side function f. The piecewise linear splines ($m = 2$) result in rather unsatisfying errors, probably caused by the lack of smoothness. Higher-order splines as $m = 3$ and $m = 4$ provide better results, especially when investigating the range and the worst case of the possible errors $\mathrm{err}(\boldsymbol{a})$. Although the overall error size in Figure 3.3a might seem a bit large, it is matching the default relative tolerance 10^{-6} of the function `solve_bvp`, which we used for these tests. A lower accuracy of the underlying differential equation solver leads to a lower accuracy of our method, which can be observed in Figure 3.3b. Here, we fixed the spline order



(a) Varying the spline order m for fixed relative tolerance 10^{-6} (default).



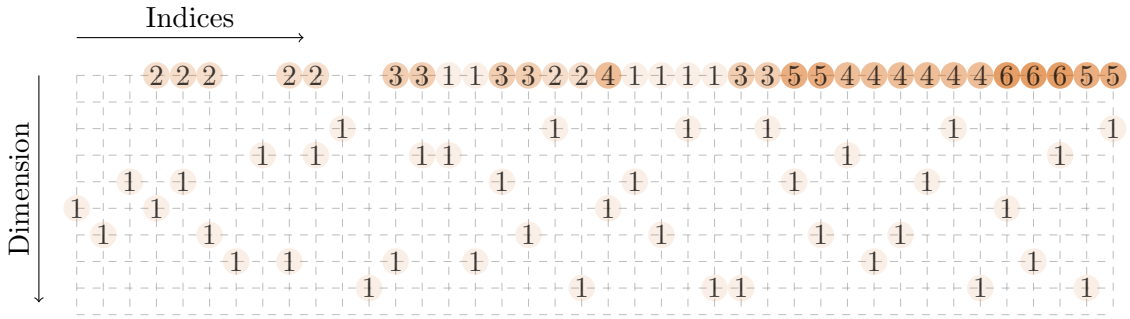
(b) Varying the relative tolerance for fixed spline order $m = 3$.

Figure 3.3: The relative approximation error err($\boldsymbol{a}$) for 10000 randomly drawn $\boldsymbol{a}$ for different choices of the spline order m and the relative tolerance of the solver function `solve_bvp`. The box-and-whisker plots show the median, the first and the second quartile as well as the maximal and minimal error observed.

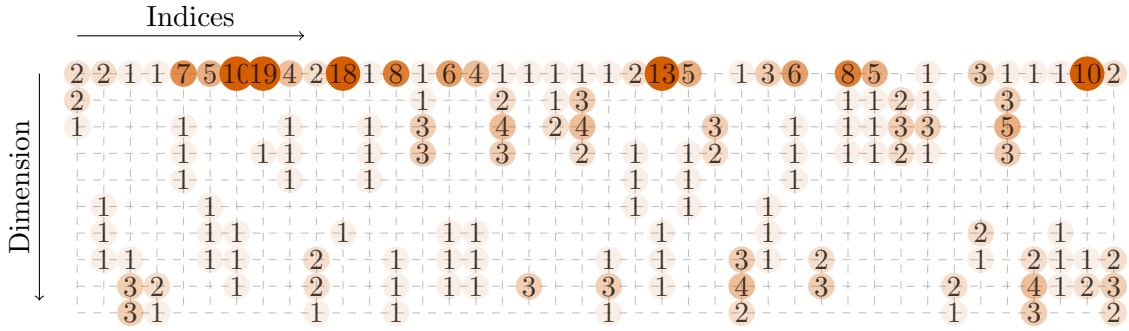
$m = 3$ and varied the relative tolerance of the function `solve_bvp`. As mentioned before, we will not go into further detail about the properties of the differential equation solvers used. However, we wanted to briefly mention the influence of the accuracy of the underlying solver at least for this first example.

Figure 3.4a shows two parts of the detected index set I for the spline order $m = 3$. As in Example 3.1.1, we notice a sparse structure of the first detected indices. This time, we do not have an explicit representation of the true solution u and only use approximations of the solution given by the differential equation solver as our samples. Hence, we are not capable of comparing these indices and the corresponding values to the true ones as before. However, the structure, which can be observed in Figure 3.4a, is still highly reasonable: It shows, that the algorithm is prioritizing two-dimensional couplings with small entries in the first dimension corresponding to the spatial variable x. For later indices as shown for example in Figure 3.4b, there appear some higher-dimensional couplings and even some values greater than 1 outside of the spatial dimension. The coupling B-spline coefficients a_ℓ are always adjacent. This is caused by the overlapping nature of the B-splines. Even for later indices (apart from numerical errors as described below) this behavior will continue.

On the other hand, each of the 1000 detected indices contains at least one non-zero entry in the dimensions 2 to 10, i.e., the corresponding Chebyshev series does not contain a single



(a) The detected indices from number 1 to number 40.



(b) The detected indices from number 201 to number 240.

Figure 3.4: Abstract visualizations of 40 detected indices $\boldsymbol{k}$ (from left to right) for Example 3.1.2. The indices $\boldsymbol{k}$ are sorted in descending order according to the size of the corresponding approximated coefficient $\hat{u}_{\boldsymbol{k}}$. The rows identify the 10 dimensions corresponding to the variables x and $a_{-4}, \ldots, a_4$ from top to bottom in this order. Zeros are neglected to preserve clarity.

term that depends only on x. Unfortunately, there are also some artifact indices again, which do not contain a single zero entry but unreasonably large numbers (≥ 10) in these dimensions. As before, these are mainly caused by numerical errors and could be reduced by choosing the search space Γ more restrictively.

Finally, all the computed coefficients are real-valued this time. While the domain of the coefficients $\boldsymbol{a}$ is the same as before, multiplying them with the cardinal B-splines instead of Fourier terms causes the source function f and therefore also the solution u to be real-valued for each possible coefficient $\boldsymbol{a}$.

3.2 A piece-wise continuous differential equation

As a second one-dimensional example, we consider the ordinary differential equation

$$\begin{aligned} -\frac{d}{dx}(a(x)\frac{d}{dx}u(x)) &= f(x), \quad x \in (-1,1), \\ u(-1) &= u(1) = 0, \end{aligned} \tag{3.7}$$

with the piece-wise constant coefficient function

$$a(x) = \begin{cases} \frac{1}{2}, & x \in (-1, 0), \\ 1, & x \in [0, 1). \end{cases}$$

This example was investigated in [33, Sec. 2.3] and threw up tremendous problems when using physics-informed neural networks (PINNs) since it has no classical but only a weak solution u for the given right-hand side function f

$$f(x) = \begin{cases} 0, & x \in (-1, 0), \\ -2, & x \in [0, 1). \end{cases} \tag{3.8}$$

Therefore, we are interested in solving (3.7) using our approach and comparing the result afterwards for this particular right-hand side function f. The exact solution for this scenario is also given in [33] and reads as

$$u(x) = \begin{cases} -\frac{2}{3}x - \frac{2}{3}, & x \in (-1, 0), \\ x^2 - \frac{1}{3}x - \frac{2}{3}, & x \in [0, 1). \end{cases} \tag{3.9}$$

For this ODE, we have the differential operator $\mathcal{L} = -\frac{d}{dx}a(x)\frac{d}{dx}$ and choose $\mathcal{U} = H_0^1((-1, 1))$ and $\mathcal{F} = H^{-1}((-1, 1))$ as the function spaces as well as the approximation domain $\mathcal{D} = [-1, 1]^{n+1}$ and the tensorized Chebyshev polynomials as the BOPB. In order to resolve (3.8) properly, we choose a discretization of f similar to Section 3.1.2 using B-splines of order $m = 1$, so characteristic functions on non-overlapping intervals. Precisely, we resolve the right-hand side f as

$$f(x) = \sum_{\ell=0}^{7} b_\ell \mathbb{1}_{[-1+\frac{\ell}{4}, -1+\frac{\ell+1}{4}]}(x), \tag{3.10}$$

such that the particular function f given in (3.8) is obtained exactly for the spline coefficients $\boldsymbol{b} = [0, 0, 0, 0, -2, -2, -2, -2]^T$. Then, the general exact solution reads as

$$u(x, \boldsymbol{b}) = \begin{cases} \sum_{\ell=0}^{7} -2b_\ell W_\ell(x) + 2C_1 x + C_2, & x \in (-1, 0), \\ \sum_{\ell=0}^{7} -b_\ell W_\ell(x) + C_1 x + C_2, & x \in [0, 1), \end{cases} \tag{3.11}$$

with

$$W_\ell(x) \coloneqq \begin{cases} 0, & x \in (-1, -1 + \frac{\ell}{4}), \\ \frac{1}{2}x^2 + (1 - \frac{\ell}{4})x + \frac{1}{2}(-1 + \frac{\ell}{4})^2, & x \in [-1 + \frac{\ell}{4}, -1 + \frac{\ell+1}{4}), \\ \frac{1}{4}x + \frac{7}{32} - \frac{\ell}{16}, & x \in [-1 + \frac{\ell+1}{4}, 1) \end{cases}$$

being the second anti-derivative of the characteristic function $\mathbb{1}_{[-1+\frac{\ell}{4}, -1+\frac{\ell+1}{4}]}(x)$. The boundary conditions from (3.7) yield $C_2 = 2C_1$ and $C_1 = \sum_{\ell=0}^{7} b_\ell \frac{15-2\ell}{96}$.

To obtain the necessary samples of the (weak) solution of (3.7), we utilize the popular open-source computing platform FEniCS and its Python interface. The underlying finite element method is very well capable of computing the weak solution of (3.7), making it a perfect tool for our black-box sampling step. We use a rather coarse mesh with only 100 nodes. Note

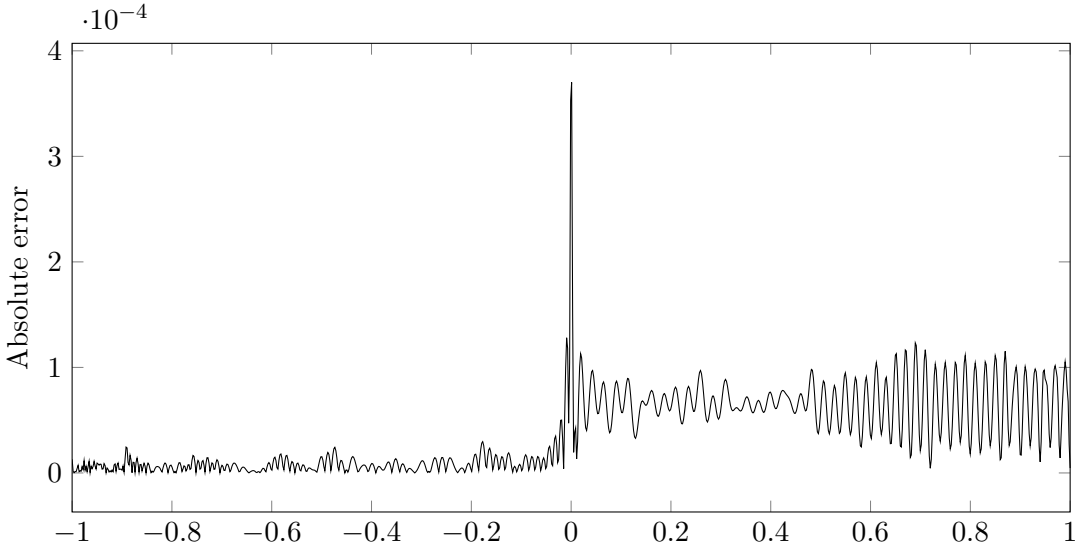


Figure 3.5: The (absolute) pointwise approximation error of our approximation when using the right-hand side (3.8) compared to the exact solution (3.9).

that the domain of the spatial variable x is already $[-1, 1]$ this time and needs no further transformation. On the other hand, we scale the spline coefficients b_ℓ with a factor of 2 (or $\frac{1}{2}$, respectively) to cover the range $[-2, 2]^8$, such that the particular right-hand side f given in (3.8) can be resolved exactly as described above.

The amount $n = 8$ of spline coefficients b_ℓ is already fixed such that we have the overall dimension $d = 9$ for this problem. We use the sparsity $s = 4000$ with the extension $N = 256$ for this test example. The dimension-incremental method needed about 75 minutes for this approach. FEniCS itself computes one solution sample for this problem in about 10^{-2} seconds.

Figure 3.5 illustrates the pointwise error of our approximation for the particular function f given in (3.8) when compared to the true solution (3.9). We note that the kink of the exact solution u at the point $x = 0$ leads to larger pointwise errors in this region, which is not surprising given our smooth basis functions and hence the smoothness of our approximation. Further, it appears that in the right half of the domain, the pointwise errors are no longer unbiased but oscilliate around some constant greater than zero.

The detected index set, partially illustrated in Figure 3.6, shows the usual structure from the previous example. The linear dependence of the spline coefficients $\boldsymbol{b}$ in the analytical solution (3.11) is detected perfectly, neglecting any entries larger than 1 in these dimensions already in Step 1 of Algorithm 1. This is highly remarkable when compared to our first example in Section 3.1.1, since we are using a numerical solver instead of an analytical solution this time. So while the samples of the solution u contain small numerical errors, our algorithm still successfully neglects the unnecessary values in Step 1. The first dimension, corresponding to the spatial variable x, contains again larger values, caused by the highly piecewise structure of the solution (3.11).

3.3 The multi-dimensional Poisson equation

While the previous examples considered ordinary differential equations, we now progress to partial differential equations with the two-dimensional version of (3.3). The general Poisson

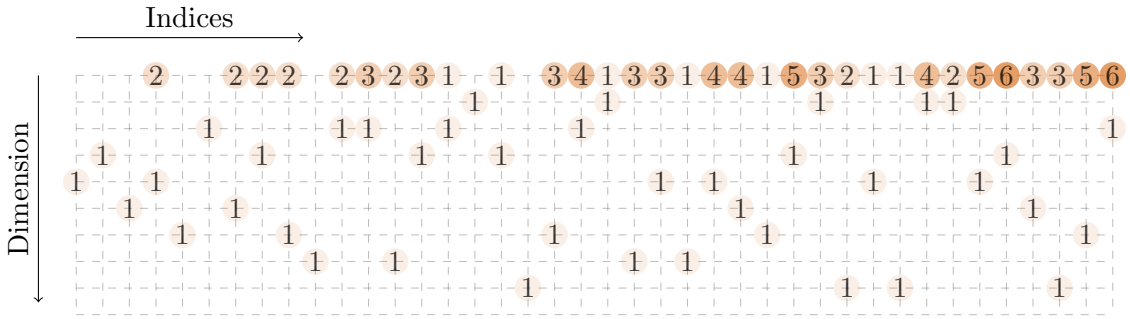


Figure 3.6: An abstract visualization of the first 40 indices $\boldsymbol{k}$ detected for the piece-wise continuous differential equation example. The indices $\boldsymbol{k}$ are ordered by the absolute values of their corresponding approximated coefficients $\hat{u}_{\boldsymbol{k}}$ in descending order from left to right. The rows identify the 9 dimensions corresponding to the spatial variable x and the $n = 8$ spline coefficients $\boldsymbol{b}$ used. Zeros are neglected to preserve clarity.

equation with homogeneous Dirichlet boundary conditions is given by

$$\begin{aligned} -\Delta u(\boldsymbol{x}) &= f(\boldsymbol{x}), \quad && \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) &= 0, && \boldsymbol{x} \in \delta\Omega, \end{aligned} \tag{3.12}$$

with the spatial domain $\Omega = (0,1)^d$. We restrict ourself to the two-dimensional version $d = 2$ in this work, while $d = 3$ is also a common setting in applications. With the differential operator $\mathcal{L} = -\Delta$ and the homogeneous Dirichlet boundary conditions, we use the common function spaces $\mathcal{U} = H_0^1(\Omega)$ and $\mathcal{F} = H^{-1}(\Omega)$, which is the direct generalization of the function spaces used in Section 3.1. Also, the approximation space $\mathcal{H} = L_2(\mathcal{D})$ is again equipped with the tensorized Chebyshev polynomials on the now $n+2$-dimensional domain $\mathcal{D} = [-1,1]^{n+2}$.

Motivated by the example from Section 3.1, we use a two-dimensional Fourier series to parametrize the right-hand side function f in this example. In detail, we parametrize f by

$$f(\boldsymbol{x}) \approx \sum_{\boldsymbol{\ell} \in J} a_{\boldsymbol{\ell}} \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{\ell} \boldsymbol{x}}$$

with the index set J, again containing a total of n indices. As in the one-dimensional case, this choice should result in a rather simple detected index set I when applying our algorithm.

As in Section 3.2, we use FEniCS to solve the PDE (3.12) with the finite element method for given $\boldsymbol{a}$ this time. For the finite element mesh, we used a uniform unit square mesh with 51 equidistant points in each direction. Since each square cell is split into two triangular elements, we thus end up with 5000 finite elements for our approximation. As in the ODE examples, we still need to transform the sampling points $(\boldsymbol{x}, \boldsymbol{a})$. Hence, we proceed similarly as in Section 3.1.1 by using the transformation $\mathcal{T}\boldsymbol{x} = \frac{1}{2}(\boldsymbol{x} + \boldsymbol{1})$ and simply restricting $\boldsymbol{a} \in [-1,1]^n$.

We set $J \coloneqq \{-1,0,1\}^2$ in order to have $n = 9$ Fourier coefficients. Combined with the spatial dimension $d = 2$, we end up with an 11-dimensional approximation problem this time. Further, we choose the sparsity $s = 1000$ and the extension $N = 64$ of the search space Γ. FEniCS needs up to 3 seconds for each single solution this time, leaving our dimension-incremental method with a total runtime of roughly 100 minutes.

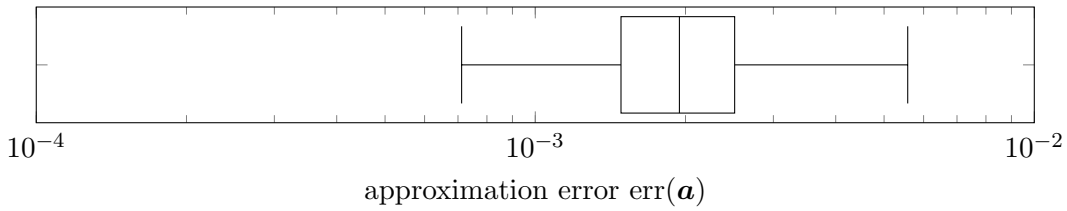


Figure 3.7: The relative approximation error err($\boldsymbol{a}$) for 10000 randomly drawn $\boldsymbol{a}$ for the two-dimensional Poisson equation example. The box-and-whisker plots show the median, the first and the second quartile as well as the maximal and minimal error observed.

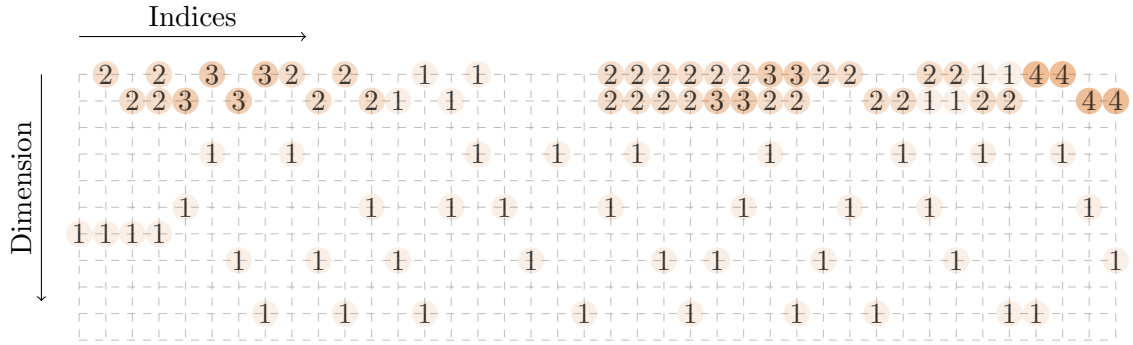


Figure 3.8: An abstract visualization of the first 40 indices $\boldsymbol{k}$ detected for the two-dimensional Poisson equation example. The indices $\boldsymbol{k}$ are ordered by the absolute values of their corresponding approximated coefficients $\hat{u}_{\boldsymbol{k}}$ in descending order from left to right. The rows identify the 11 dimensions corresponding to the two spatial variables $\boldsymbol{x} = (x_1, x_2)^T$ and the $n = 9$ Fourier coefficients $\boldsymbol{a}$ used. Zeros are neglected to preserve clarity.

Figure 3.7 illustrates the relative approximation error err($\boldsymbol{a}$) as before in the one-dimensional case using 10000 randomly drawn coefficients $\boldsymbol{a}$. Note that this error is computed by comparing our approximation to the solution the FEM solver produces for the given $\boldsymbol{a}$ on the nodes of the FE mesh using the same parameters as we did during the execution of our algorithm. We observe errors of sizes around 10^{-3}, which are obviously larger than before in Section 3.1.1. However, this effect is primarily caused by the fact, that we are no longer using a direct representation of the analytic solution as for the one-dimensional example to generate our sampling points. The error sizes are still reasonable and can compete with the used PDE solver, taking into account the sparsity s and extension N used here.

The structure of the detected index set I, where the first part is shown in Figure 3.8, is pretty similar to the one seen in Section 3.1.1. Even though we are not using the exact solution for our training samples anymore, our algorithm is still able to identify that for the dimensions corresponding to the Fourier coefficients $\boldsymbol{a}$ the only necessary entries are 0 and 1. The entries of the first two dimensions, corresponding to the spatial dimensions $\boldsymbol{x}$, also contain larger numbers, but are growing significantly slower than in the one-dimensional example. This is due to the fact, that in this two-dimensional case also all possible combinations of the entries in these two dimensions have to be exploited. Overall, the discovered structure resembles

the one from our first example quite nicely and seems like the kind of structure for such an index set we would expect as the canonical generalization to multi-dimensional examples even without examining an analytical solution.

3.4 A diffusion equation with an affine random coefficient

The differential equation (3.7) is also a one-dimensional diffusion equation, for which the coefficient a could be also called diffusion coefficient. Now, we investigate a two-dimensional diffusion equation on $\Omega = [0,1]^2$ with a randomized diffusion coefficient a, which is not only a PDE instead of an ODE but also a parametrized differential equation, i.e.

$$\begin{aligned} -\nabla \cdot (a(\boldsymbol{x},\boldsymbol{y})\nabla u(\boldsymbol{x},\boldsymbol{y})) &= f(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega,\, \boldsymbol{y} \in \Omega_{\boldsymbol{y}}, \\ u(\boldsymbol{x},\boldsymbol{y}) &= 0, \qquad \boldsymbol{x} \in \partial\Omega,\, \boldsymbol{y} \in \Omega_{\boldsymbol{y}}. \end{aligned} \tag{3.13}$$

Here, the differential operator ∇ is always used w.r.t. the spatial variable $\boldsymbol{x}$. While there exist multiple kinds of randomized diffusion coefficients a, as can be seen for example in [22], we will only work with an affine random coefficient a here. In more detail, we consider the particular example from [9, Sec. 11], where we have for $n_{\boldsymbol{y}} = 20$ the affine coefficient

$$a(\boldsymbol{x},\boldsymbol{y}) \coloneqq 1 + \sum_{j=1}^{n_{\boldsymbol{y}}} y_j \psi_j(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega,\, \boldsymbol{y} \in [-1,1]^{20}$$

with the random variables $\boldsymbol{y} \sim \mathcal{U}([-1,1]^{n_{\boldsymbol{y}}})$ and

$$\psi_j(\boldsymbol{x}) \coloneqq cj^{-\mu} \cos(2\pi m_1(j)\, x_1)\, \cos(2\pi m_2(j)\, x_2), \qquad \boldsymbol{x} \in \Omega,\, j \geq 1.$$

Here, $c > 0$ is a constant and $\mu > 1$ the decay rate. In our numerical example below, we use the values $c = 0.9/\zeta(2)$ and $\mu = 2$ already used in [9]. Further, $m_1(j)$ and $m_2(j)$ are defined as

$$m_1(j) \coloneqq j - \frac{k(j)(k(j)+1)}{2} \quad \text{and} \quad m_2(j) \coloneqq k(j) - m_1(j)$$

with $k(j) \coloneqq \lfloor -1/2 + \sqrt{1/4 + 2j} \rfloor$. For some explicit values of $m_1(j), m_2(j)$ and $k(j)$ as well as more details on this differential problem, see [9]. As before, we consider the common function spaces $\mathcal{U} = H_0^1(\Omega)$ and $\mathcal{F} = H^{-1}(\Omega)$ for this differential operator.

Remark 3.3. *We already considered the numerical solution of this problem in [22, Sec. 4.3] using a slightly different approach. Therein, we discretize the spatial domain $\Omega = [0,1]^2$ and compute approximations like (2.7) with $\boldsymbol{\theta} \coloneqq \boldsymbol{y}$ in the Fourier setting for every fixed node $\boldsymbol{x}_g, g = 1, \ldots, G$. The key ingredient there is, that the a priori unknown index set I is chosen similarly for each of the G approximations $S_I^{\mathcal{A}} u(\boldsymbol{x}_g, \cdot)$, which allows us to compute all these approximations using only a single call of a modification of the sparse approximation algorithm with slightly more samples and computation time needed. For a given random coefficient $\boldsymbol{y}^*$, we then compute the values of $S_I^{\mathcal{A}} u(\boldsymbol{x}_g, \boldsymbol{y}^*)$ at all the nodes $\boldsymbol{x}_g$ and interpolated between them to receive a solution on the complete domain Ω.*

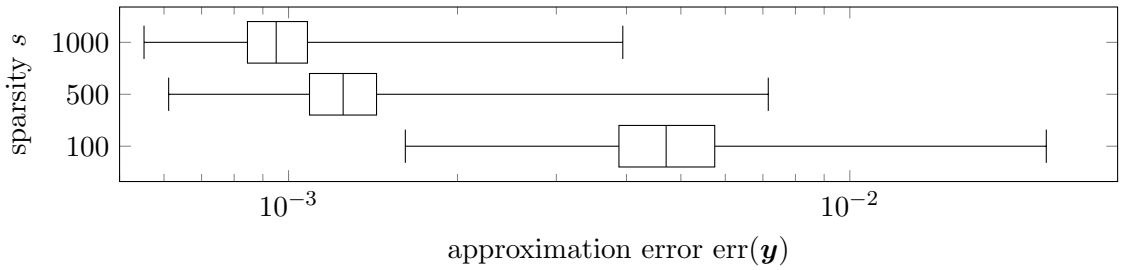


Figure 3.9: The relative approximation error err($\boldsymbol{y}$) for 10000 randomly drawn variables $\boldsymbol{y}$ for the diffusion equation example for different sparsities s. The box-and-whisker plots show the median, the first and the second quartile as well as the maximal and minimal error observed.

In contrast to all other examples considered in this work, we decided to use the fixed right-hand side $f \equiv 1$ without parametrization, since we are mainly interested in a comparison with the results from [22]. Hence, we neglect the space $\mathcal{F}$ for this example and proceed with the solution operator $\mathcal{G} : \Omega_{\boldsymbol{y}} \to \mathcal{U}$ this time. The approximation space is still $\mathcal{H} = L_2(\mathcal{D})$, again using the tensorized Chebyshev polynomials and $\mathcal{D} = [-1, 1]^{n_{\boldsymbol{y}}+2}$. The random variables $\boldsymbol{y}$ already match that domain, so we only transform $\boldsymbol{x}$ as usual using the transformation $\mathcal{T}\boldsymbol{x} = \frac{1}{2}(\boldsymbol{x} + \mathbf{1})$. Note that one could still use a similar approach as in the previous examples to easily parametrize the right-hand side f as well in this example. As in Section 3.3, we utilize FEniCS to solve the differential equation using 5000 finite elements.

Since there is no parametrization of the right-hand side f, but 2 spatial dimension as well as $n_{\boldsymbol{y}} = 20$ dimensions for the random variable $\boldsymbol{y}$, we still end up with a total of 22 dimensions for our approximation problem. Unfortunately, the sampling complexity as well as the computational complexity of our approach using the Chebyshev basis include an exponential factor on the maximal number of non-zero entries of the indices $\boldsymbol{k}$ appearing in the candidate sets K during step 2 of Algorithm 1. In order to prevent cases, where numerical errors cause candidates $\boldsymbol{k}$ with (almost) non zeros in any dimensions, we impose a superposition dimension $d_s = 7$ on our 22-dimensional search space Γ with extension $N = 64$ for this example. Unfortunately, the more difficult structure of the differential equation lead up to a FEniCS computation time of up to 14 seconds per call, resulting in a total runtime of about 60 hours (for sparsity $s = 1000$).

Figure 3.9 illustrates the relative approximation error err($\boldsymbol{y}$) for different sparsities s, this time with respect to the random variable $\boldsymbol{y}$. As in the previous example, the error is computed using the solution of the FEM with the same settings as comparison values. The error is again of reasonable size, even though this differential problem is significantly more difficult than the previous two-dimensional example in Section 3.3. Further, the largest nodal error as considered in [22, Sec. 4, Fig. 6], so the largest error at any of the nodes of the FE mesh when evaluating the approximation for 10000 randomly drawn $\boldsymbol{y}$ and considering the respective ℓ_2 norm, is just slightly larger than for the uniform sparse FFT from [22]. This small increase is probably caused by the fact, that we are no longer focusing on particular nodes and basis expansions of the solution u at these nodes, but a full basis expansion of u also considering the spatial variable.

The structure of the 22-dimensional index set is pretty similar to the previous examples and not illustrated here due to the high amount of dimensions. Surprisingly, the range of the

entries in the dimensions corresponding to the random variables $\boldsymbol{y}$ is rather restricted. Already in the one-dimensional detections (Step 1 in Algorithm 1), the algorithm does not detect a full range of 65 possible entries (from 0 to $N = 64$), but less than 20 possible entries for y_1, decaying rapidly down to only 4 possible entries (so 0, 1, 2 and 3) for the later dimensions like y_{15}. While we already saw the extreme version of this behavior for the Poisson equation using the Fourier series parametrization, where the only possible entries were 0 and 1, we did not observe anything similar for the other examples like in Section 3.1.2.

3.5 Heat equation

Our next example is the heat equation in one dimension with homogeneous boundary conditions on the domain $\Omega = (0, L)$, i.e.

$$\begin{aligned} \partial_\tau u - \alpha^2 \partial_{xx} u = 0, \qquad & x \in (0, L),\ \tau \in (0, T) \\ u(x, 0) = f(x) \qquad & x \in (0, L) \\ u(0, \tau) = u(L, \tau) = 0 \qquad & \tau \in (0, T). \end{aligned} \tag{3.14}$$

In this time-dependent differential equation, the source term on the right-hand side is zero and therefore does not require parametrization. Instead, our focus lies on the initial condition $u(x, 0) = f(x)$, which characterizes the system's state at the initial time $\tau = 0$. Consequently, we aim to parametrize the function f.

We are interested in the well-known solution of the heat equation

$$u(x, \tau) = \sum_{\ell=1}^{\infty} a_\ell \sin\left(\frac{\ell \pi x}{L}\right) \exp\left(\frac{-\ell^2 \pi^2 \alpha^2 \tau}{L^2}\right) \qquad x \in [0, L], \tau \in [0, T], \tag{3.15}$$

which can be derived exactly for the initial condition

$$u(x, 0) = f(x) = \sum_{\ell=1}^{\infty} a_\ell \sin \frac{\ell \pi x}{L} \qquad x \in [0, L] \tag{3.16}$$

with $a_\ell \in \mathbb{C}, \ell \in \mathbb{N}$, using Fourier's approach.

While this solution holds for arbitrary $\tau \geq 0$, we set the final time $T = 1$. Further, we set the length $L = 1$ and the diffusivity constant $\alpha = 0.25$. Due to the time-dependence of the differential equation (3.14), the function space $\mathcal{U}$ is a little more complicated than in the previous examples. We need to ensure spatial regularity, meaning that u should be square-integrable in time and satisfy $u(\tau) \in H_0^1(\Omega)$ for almost every time τ, since the term $\partial_{xx} u$ and homogeneous boundary conditions are present. Additionally, we require a notion of time regularity: the weak time derivative of u should exist and take values in the dual space $H^{-1}(\Omega)$. A formal and compact way to express this space uses so-called Bochner spaces (see, e.g., [10, Sec. 5.9.2] or [1]), which we omit here for simplicity. Accordingly, we have $\mathcal{F} = L_2(\Omega)$ to ensure that the initial state $u(x, 0)$ is square-integrable over the spatial domain Ω. Due to the time dependence, the solution operator we are analyzing this time is of the form $\mathcal{G} : \mathcal{F} \times [0, T] \to \mathcal{U}$, cf. Remark 2.2.

We parametrize the function f by truncating the sum (3.16) to n terms. Similar to Section 3.1.1, we restrict the coefficients $a_\ell \in [-1, 1]$ for all $\ell = 1, \ldots, n$ and transform both the spatial and time variable by $\mathcal{T} x = \frac{1}{2}(x + 1)$ and $\mathcal{T} \tau = \frac{1}{2}(\tau + 1)$. The differential equation is solved using SciPy's function `solve_ivp`. In particular, we use the Radau IIA implicit Runge-Kutta

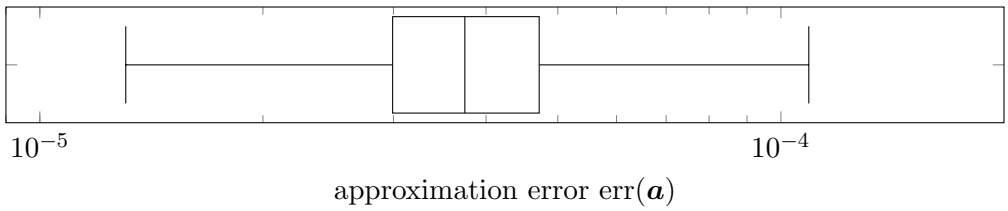


Figure 3.10: The relative approximation error err($\boldsymbol{a}$) for 10000 randomly drawn $\boldsymbol{a}$ for the heat equation example. The box-and-whisker plots show the median, the first and the second quartile as well as the maximal and minimal error observed.

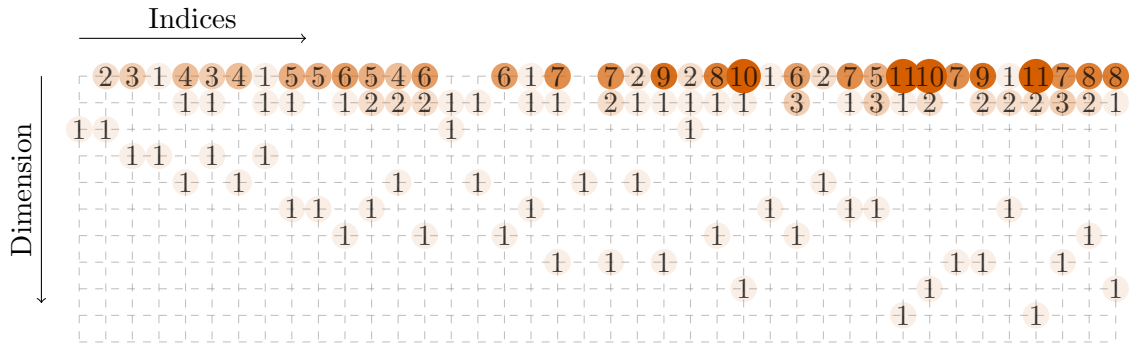


Figure 3.11: An abstract visualization of the first 40 indices $\boldsymbol{k}$ detected for the heat equation example. The indices $\boldsymbol{k}$ are ordered by the absolute values of their corresponding approximated coefficients $\hat{u}_{\boldsymbol{k}}$ in descending order from left to right. The rows identify the 11 dimensions corresponding to the spatial variable x, the time variable τ and the $n = 9$ coefficients $\boldsymbol{a}$ used. Zeros are neglected to preserve clarity.

method of order five, suitable for stiff differential equations with a desired relative tolerance of 10^{-8} and absolute tolerance of 10^{-10}. We choose the number of coefficients $n = 9$ to end up with an 11-dimensional approximation problem, the sparsity $s = 1000$ and the extension $N = 64$ for our algorithm. Note that using `solve_ivp` is a very costly approach for the solution of the heat equation, needing roughly 20 seconds for a single sample of the solution. Therefore, the execution of our full algorithm took around 32 hours.

Figure 3.10 shows the relative approximation error err($\boldsymbol{a}$) computed for 10000 randomly drawn coefficients $\boldsymbol{a}$. The error is computed similarly to (3.2) for 100 equidistant nodes in space and time each and using the exact solution (3.15) as comparison. The average error size is less than 10^{-4} for this example. Note, that the function `solve_ivp` itself reaches an accuracy of around 10^{-6} with the given parameters, which can be seen as the noise on the data samples we are giving to our dimension-incremental method here.

The detected index set I shows several interesting features this time. As can be seen for the first 40 indices in Figure 3.11, the dimensions corresponding to the coefficients a_ℓ contain exactly one non-zero entry for each index, which happens to be 1. This stays true for all the detected indices up to some artifacts again. As in previous examples, this is due to the fact, that the coefficients a_ℓ appear only linearly and separated from each other in the sum in (3.15). Our algorithm once again captures this behavior even for this more complicated

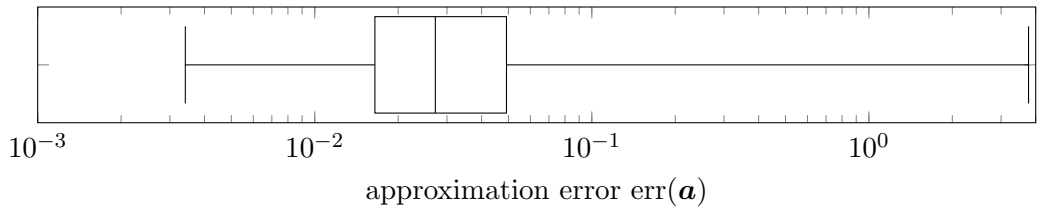


Figure 3.12: The relative approximation error err($\boldsymbol{a}$) for 10000 randomly drawn $\boldsymbol{a}$ for the Burgers' equation example. The box-and-whisker plots show the median, the first and the second quartile as well as the maximal and minimal error observed.

example. The size of the entries in the first dimension, so the dimension corresponding to the spatial variable x seems to grow rapidly and much faster than in the second dimension, which corresponds to the time variable τ. However, the largest entries in the first dimension is 24 while the largest one in the second dimension is 22. For increased sparsity s, the size of the entries in the first dimension stays the same while larger entries in the second dimension are added. This again confirms, that the time-dependent exponential term in (3.15) is the main difficulty in this approximation problem.

3.6 Burgers' equation

Our final example will be the non-linear Burgers' equation in one dimension with homogeneous boundary conditions, i.e.

$$\begin{aligned} \partial_\tau u + u\partial_x u = \nu \partial_{xx} u, \quad & x \in (0,L),\, \tau \in (0,T) \\ u(x,0) = f(x) \qquad & x \in (0,L) \\ u(0,\tau) = u(L,\tau) = 0 \qquad & \tau \in (0,T). \end{aligned} \tag{3.17}$$

We will use the viscosity $\nu = 0.05$ here. As in the previous example, we set the final time $T = 1$ and the length $L = 1$, i.e. $\Omega = (0,1)$. However, we are interested in the solution u only at the final time $T = 1$ this time, i.e. $u(x,1)$. This again complicates the notion of our solution space $\mathcal{U}$ a little, which is why we omit it here. The idea however is similar to the one described for the heat equation, adjusted accordingly for the neglection of the time variable τ.

We use the approximation via the sine expansion

$$f(x) \approx \sum_{\ell=1}^{n} a_\ell \sin(\ell \pi x) \qquad x \in [0,1] \tag{3.18}$$

for the initial condition f as our parametrization. This approach is similar to the one in Section 3.5. Analogously, we restrict $\boldsymbol{a} \in [-1,1]^n$, transform $\mathcal{T}x = \frac{1}{2}(x+1)$ and set the number of coefficients $n = 9$, such that we end up with a 10-dimensional approximation problem. Further, we use the sparsity $s = 500$ and the extension $N = 32$ here. We use the function `solve_ivp` with the Radau method again, this time with the relative tolerance 10^{-6} and absolute tolerance 10^{-8}. The solver itself then needs about 6 seconds for each solution. However, the more difficult structure of the index set we are detecting due to the non-linearity of our problem leaves our full method with a runtime of roughly 114 hours.

In Figure 3.12 we directly observe how the relative approximation error err($\boldsymbol{a}$) is way larger than for the heat equation in Section 3.5. The non-linearity of Burgers' equation increases the

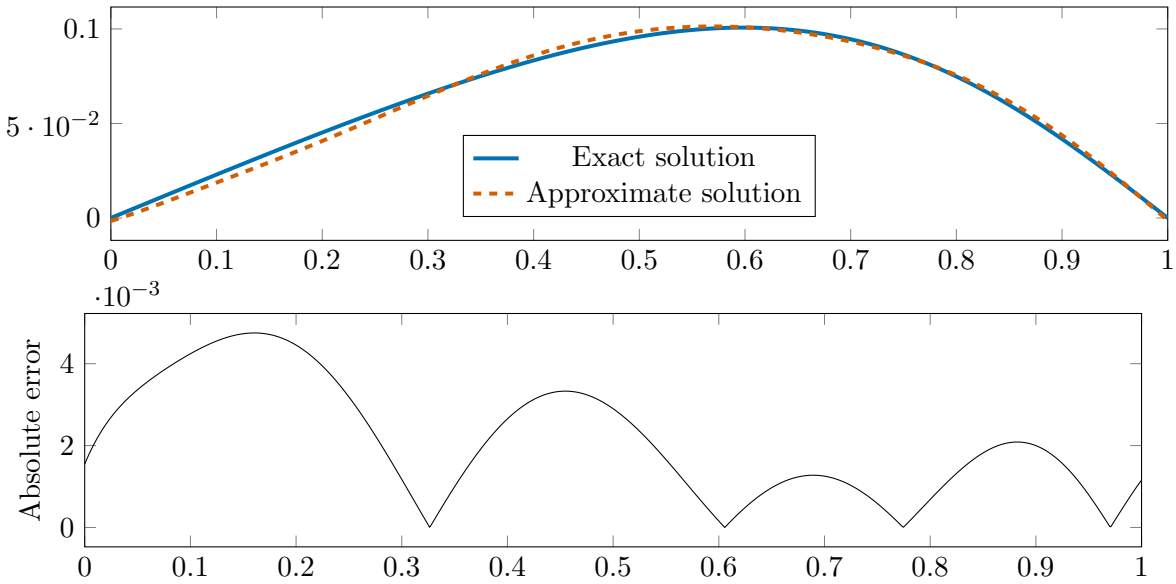


Figure 3.13: Comparison between the exact solution (3.20) and the approximate solution at $\tau = 1$ and the corresponding (absolute) pointwise approximation error.

difficulty of this approximation problem significantly, even when neglecting the time variable τ here. However, the error is in the order of magnitude of 10^{-2} most of the times, which is still reasonable given this difficulty and the smaller sparsity $s = 500$ used here. Note that approximation errors $\mathrm{err}(\boldsymbol{a})$ exceeding 10^{-1} occur only rarely and would be classified as outliers under the common convention of using 1.5 times the interquartile range.

Next, we are interested in an explicit solution to Burgers' equation. If we use the initial condition

$$f(x) = 2\pi\nu \frac{\sin(\pi x)}{\alpha + \cos(\pi x)} \qquad x \in [0, L], \tag{3.19}$$

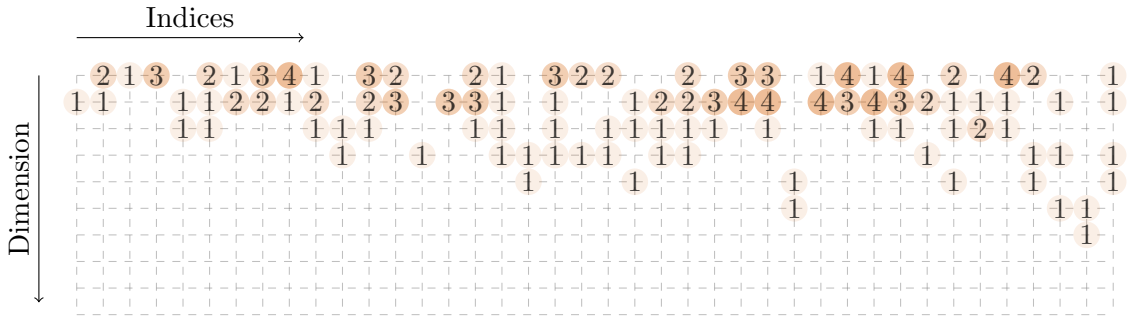
the solution u is given as

$$u(x, \tau) = 2\pi\nu \frac{\sin(\pi x)\exp(-\pi^2\nu\tau)}{\alpha + \cos(\pi x)\exp(-\pi^2\nu\tau)} \qquad x \in [0, L], \tau \in [0, T]. \tag{3.20}$$

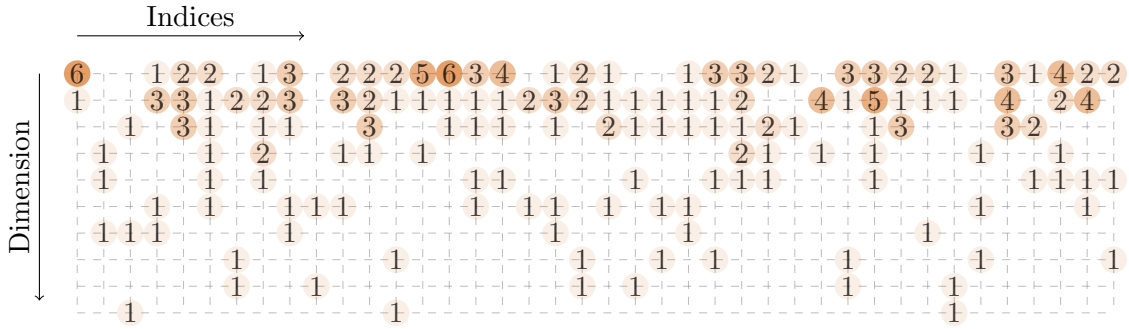
Further details and other explicit solutions to Burgers' equation can be found in [4]. We set $\alpha = 2$ here and approximate (3.19) by an $n = 9$-term sine expansion of the form (3.18), which yields a relative error of less than 10^{-5}. Note that the corresponding coefficients a_ℓ have absolute values of less than 0.2, matching our inital restriction $a_\ell \in [-1, 1]$.

Although we are starting with just an approximation of the true initial condition (3.19), we end up with a reasonable solution as shown in Figure 3.13. We notice the smooth oscilliations of our approximation around the true solution, which could of course be further reduced by increasing the sparsity s and the extension N.

Finally, Figure 3.14 shows some of the detected indices for this example. We observe similar behavior in the first dimension corresponding to the spatial dimension x as in previous examples. However, for the remaining dimensions corresponding to the coefficients a_ℓ, two main differences can be seen: First, the entries in these dimension are not only 0 and 1, i.e.,



(a) The detected indices from number 1 to number 40.



(b) The detected indices from number 201 to number 240.

Figure 3.14: Abstract visualizations of 40 detected indices $\boldsymbol{k}$ (from left to right) for Example 3.6. The indices $\boldsymbol{k}$ are sorted in descending order according to the size of the corresponding approximated coefficient $\hat{u}_{\boldsymbol{k}}$. The rows identify the 10 dimensions corresponding to the variables x and $a_1, \ldots, a_9$ from top to bottom in this order. Zeros are neglected to preserve clarity.

the solution does not depend only linearly on the coefficients a_ℓ. Second, there are several interactions between these dimensions (and the spatial dimension x) noticeable. While we saw both effects already in the example in Section 3.1.2, they are much more prominent here, i.e. occuring way earlier (already in the first 10 indices) and stronger (up to six non-zero entries).

4 Conclusion

We presented an adaptive approach that uses the dimension-incremental algorithm from [23] in combination with classical differential equation solvers like the FEM in order to approximate solution operators of differential equations. We transformed the problem of operator learning for differential equations by parametrizing e.g. the source function f, which led us to a high-dimensional approximation problem for a function with an unknown structure. Algorithm 1 detects a reasonable index set I by using samples of the solution u computed by the differential equation solver mentioned above. This index set I not only allows a good approximation of the respective solution u for any source function f suitable to this parametrization, but also gives us important information about the structure of the solution and its dependence on the spatial variable $\boldsymbol{x}$, the discretization parameters of the source function f and possible

other variables and parameters such as the time t. Such information can then be used to manually generalize the index set I to even higher dimensions that arise when refining the resolution of the source function f.

We have studied the behavior of our proposed methods on several examples. These numerical tests yielded reasonable approximations to the solutions of the PDEs. Especially for the easier examples, the structure of the obtained index sets I matched our general expectations and (if available) the structure of the underlying analytical solution. Our brief test of generalization of the index set I to even higher dimensions for the one-dimensional Poisson equation also showed promising results. The more advanced examples demonstrated the applicability of our approach to more difficult settings and problems. Our numerical tests are available at `https://github.com/fabiantaubert/nabopb` together with the dimension-incremental algorithm itself.

Overall, the presented algorithm performed satisfactorily and provided useful details about the structure of the solutions to the differential equations. Thus, while the field of operator learning is strongly dominated by machine learning algorithms such as PINNs, more classical approaches such as our proposed method can open new perspectives, especially to overcome still existing drawbacks of neural networks like the lack of interpretability.

References

[1] F. Bartel and D. Dũng. Sampling recovery in Bochner spaces and applications to parametric PDEs with log-normal random inputs. *arXiv:2409.05050*, 2024.

[2] F. Bartel and F. Taubert. Nonlinear approximation with subsampled rank-1 lattices. *Fourteenth International Conference on Sampling Theory and Applications*, 2023.

[3] J. Blechschmidt and O. G. Ernst. Three ways to solve partial differential equations with neural networks — a review. *GAMM-Mitteilungen*, 44(2):e202100006, 2021.

[4] M. P. Bonkile, A. Awasthi, C. Lakshmi, V. Mukundan, and V. S. Aswin. A systematic literature review of Burgers' equation with recent advances. *Pramana - Journal of Physics*, 90(6):69, 2018.

[5] N. Boullé and A. Townsend. Chapter 3 - a mathematical guide to operator learning. In S. Mishra and A. Townsend, editors, *Numerical Analysis Meets Machine Learning*, volume 25 of *Handbook of Numerical Analysis*, pages 83–125. Elsevier, 2024.

[6] J. E. S. Cardona and M. Hecht. Learning partial differential equations by spectral approximates of general Sobolev spaces. *arXiv:2301.04887*, 2023.

[7] R. Cools, F. Y. Kuo, D. Nuyens, and G. Suryanarayana. Tent-transformed lattice rules for integration and approximation of multivariate non-periodic functions. *J. Complexity*, 36:166–181, 2016.

[8] N. Demo, M. Tezzele, and G. Rozza. A DeepONet multi-fidelity approach for residual learning in reduced order modeling. *Adv. Model. and Simul. in Eng. Sci.*, 10(12), 2023.

[9] M. Eigel, C. J. Gittelson, C. Schwab, and E. Zander. Adaptive stochastic Galerkin FEM. *Comput. Methods Appl. Mech. Engrg.*, 270:247–269, 2014.

[10] L. C. Evans. *Partial Differential Equations*, volume 19 of *Graduate Studies in Mathematics*. American Mathematical Society, 2nd edition, 2010.

[11] X. Feng, Y. Qian, and W. Shen. MC-Nonlocal-PINNs: Handling nonlocal operators in PINNs via Monte Carlo sampling. *Numer. Math. Theor. Meth. Appl.*, 16(3):769–791, 2023.

[12] S. Goswami, M. Yin, Y. Yu, and G. E. Karniadakis. A physics-informed variational DeepONet for predicting crack path in quasi-brittle materials. *Comput. Methods Appl. Mech. Engrg.*, 391:114587, 2022.

[13] V. Grimm, A. Heinlein, and A. Klawonn. A short note on solving partial differential equations using convolutional neural networks. In *Domain Decomposition Methods in Science and Engineering XXVII*, pages 3–14, Cham, 2024. Springer Nature Switzerland.

[14] C. Gross and M. Iwen. Sparse spectral methods for solving high-dimensional and multiscale elliptic PDEs. *Found. Comput. Math.*, 2024.

[15] T. G. Grossmann, U. J. Komorowska, J. Latz, and C.-B. Schönlieb. Can physics-informed neural networks beat the finite element method? *IMA Journal of Applied Mathematics*, 89(1):143–174, 05 2024.

[16] A. Hashemi, H. Schaeffer, R. Shi, U. Topcu, G. Tran, and R. Ward. Generalization bounds for sparse random feature expansions. *Applied and Computational Harmonic Analysis*, 62:310–330, 2023.

[17] L. Herrmann, C. Schwab, and J. Zech. Neural and spectral operator surrogates: unified construction and expression rate bounds. *Adv. Comput. Math. (accepted)*, 2024.

[18] M. Hutzenthaler and T. A. Nguyen. Multilevel Picard approximations of high-dimensional semilinear partial differential equations with locally monotone coefficient functions. *Appl. Numer. Math.*, 181:151–175, 2022.

[19] P. Jin, S. Meng, and L. Lu. MIONet: Learning multiple-input operators via tensor product. *SIAM J. Sci. Comput.*, 44(6):A3490–A3514, 2022.

[20] L. Kämmerer. Multiple rank-1 lattices as sampling schemes for multivariate trigonometric polynomials. *J. Fourier Anal. Appl.*, 24:17–44, 2018.

[21] L. Kämmerer. An efficient spatial discretization of spans of multivariate Chebyshev polynomials. *Appl. Comput. Harmon. Anal.*, 77:101761, 2025.

[22] L. Kämmerer, D. Potts, and F. Taubert. The uniform sparse FFT with application to PDEs with random coefficients. *Sampl. Theory Signal Proces. Data Anal.*, 20(19), 2022.

[23] L. Kämmerer, D. Potts, and F. Taubert. Nonlinear approximation in bounded orthonormal product bases. *Sampl. Theory Signal Proces. Data Anal.*, 21(19), 2023.

[24] L. Kämmerer, D. Potts, and T. Volkmer. High-dimensional sparse FFT based on sampling along multiple rank-1 lattices. *Appl. Comput. Harmon. Anal.*, 51:225–257, 2021.

[25] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. Physics-informed machine learning. *Nat. Rev. Phys.*, 3:422–440, 2021.

[26] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Learning maps between function spaces with applications to PDEs. *J. Mach. Learn. Res.*, 24(89):1–97, 2023.

[27] N. B. Kovachki, S. Lanthaler, and A. M. Stuart. Chapter 9 - operator learning: Algorithms and analysis. In S. Mishra and A. Townsend, editors, *Numerical Analysis Meets Machine Learning*, volume 25 of *Handbook of Numerical Analysis*, pages 419–467. Elsevier, 2024.

[28] S. Lanthaler, A. M. Stuart, and M. Trautner. Discretization error of Fourier neural operators. *arXiv:2405.02221*, 2024.

[29] R. J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. SIAM, Philadelphia, PA, 2007.

[30] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021.

[31] G. Lin, C. Moya, and Z. Zhang. B-DeepONet: An enhanced Bayesian DeepONet for solving noisy parametric PDEs using accelerated replica exchange SGLD. *J. Comput. Phys.*, 473:111713, 2023.

[32] L. Lingsch, M. Michelis, S. M. Perera, R. K. Katzschmann, and S. Mishra. Vandermonde neural operators. *arXiv:2305.19663*, 2023.

[33] T. Luo and Q. Zhou. On Residual Minimization for PDEs: Failure of PINN, Modified Equation, and Implicit Bias. *arXiv 2310.18201*, 2023.

[34] M. Raissi, P. Perdikaris, and G. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, 378:686–707, 2019.

[35] F. Taubert. NABOPB: Nonlinear approximation in bounded orthonormal product bases (Python implementation), 2025. Version: v1.0.0, Contributor: L. Kämmerer, Link: doi.org/10.5281/zenodo.15437933.

[36] L. N. Trefethen. *Spectral Methods in* MATLAB. SIAM, Philadelphia, PA, USA, 2000.

[37] E. Weinan, M. Hutzenthaler, A. Jentzen, and T. Kruse. On multilevel Picard numerical approximations for high-dimensional nonlinear parabolic partial differential equations and high-dimensional nonlinear backward stochastic differential equations. *J. Sci. Comput.*, 79:1534–1571, 2019.

[38] C.-F. Wu and M. Hamada. *Experiments: planning, analysis and parameter design optimization*. Wiley series in probability and statistics. Wiley, Hoboken, NJ, third edition edition, 2021.

[39] J. Zech. *Sparse-Grid Approximation of High-Dimensional Parametric PDEs*. Doctoral thesis, ETH Zurich, 2018.

[40] O. C. Zienkiewicz and R. L. Taylor. *The Finite Element Method: Its Basis and Fundamentals*. Elsevier, Amsterdam, 6th edition edition, 2005.