Operator learning based on sparse high-dimensional approximation

Daniel Potts[*], Fabian Taubert[†]

June 7, 2024

We present a dimension-incremental method for function approximation in bounded orthonormal product bases to learn the solutions of various differential equations. Therefore, we deconstruct 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 largest absolute value. 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 deconstructed 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 [22], spectral methods [28, 2] or the famous finite element methods (FEM) [30]. 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)

[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

[27, 19, 6, 26], convolutional neural networks [8], deep operator networks [7, 4, 24] multilevel Picard approximations [29, 12] and neural operators [23, 13, 20, 25, 21], 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 [11, 10]. 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, 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.

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 $\boldsymbol{f} = (f_0, \ldots, f_{N-1}) \in \mathbb{C}^N$, 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 $T_{\boldsymbol{k}}$ being multivariate Chebyshev polynomials of dimension $N+1$, 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. Fortunately, this requirement is quite common and naturally satisfied for most real-world applications as for example stated in the sparsity of effects principle.

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

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

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

$$I = \left\{ \begin{bmatrix} * \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} * \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} * \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \ldots, \begin{bmatrix} * \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, \begin{bmatrix} * \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} * \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} * \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \ldots \right\},$$

so only containing vectors with at most a single 1 in any dimension beside the first component (which is the one corresponding to the spatial variable x). 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 [15]. 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 [9] and showed great results, even for very high dimensions.

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 [17], using point samples to detect a suitable index set I adaptively instead for such problems in this paper. 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 [17], based on the method from [18], 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 [1]. 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. After computing these coefficients, we end up with a reasonable approximation of the solution u in a similar form to (1.2), which now allows direct evaluations of the solution u for any right-hand side f, which is exactly of this type or can be well approximated by it, without the need to solve the differential equation over and over. This approach on operator learning for PDEs overcomes one of the biggest drawbacks of most machine learning techniques: the missing interpretability of the computed solution.

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 [17], which is also explained briefly. Section 3 then investigates several test examples to present the application of our method in various situations. Finally, we give a brief conclusion in Section 4.

2 Theory

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 $u \in \mathcal{U}$, all defined on the d-dimensional spatial domain $\Omega \subset \mathbb{R}^d$. Together with boundary conditions of the differential problem (2.1), the solution u can be defined on the closure $\overline{\Omega}$. Our goal is to learn the solution mapping $\mathcal{G} : \mathcal{F} \to \mathcal{U}$. In order to do so we start by assuming that right-hand side functions $f \in \mathcal{F}$ can be well approximated up to some extend by

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

with some fixed functions $A_j, j = 1, \ldots, n$, and coefficients $\boldsymbol{a} = (a_1, \ldots, a_n) \in \mathbb{C}^n$. This approximative parametrization 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 exists a fast algorithm 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

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

with $\{\Phi_{\boldsymbol{k}} : \boldsymbol{k} \in \mathbb{N}^{d+n}\}$ a bounded orthonormal product basis (BOPB) in some separable Hilbert space $\mathcal{H}$ 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 [17, Sec. 1.1] for more details on the notion of a BOPB and the corresponding domains and spaces.

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

$$S_I u(\boldsymbol{x}, \boldsymbol{a}) := \sum_{\boldsymbol{k} \in I} c_{\boldsymbol{k}} \Phi_{\boldsymbol{k}}(\boldsymbol{x}, \boldsymbol{a}),$$

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

$$S_I^{\mathcal{A}} u(\boldsymbol{x}, \boldsymbol{a}) := \sum_{\boldsymbol{k} \in I} \hat{u}_{\boldsymbol{k}} \Phi_{\boldsymbol{k}}(\boldsymbol{x}, \boldsymbol{a}), \tag{2.4}$$

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 (2.4) 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. 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.

Remark 2.1. *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*

$$u(\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{\theta}) := \sum_{\boldsymbol{k} \in \mathbb{N}^{d+n+n_\theta}} c_{\boldsymbol{k}} \Phi_{\boldsymbol{k}}(\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{\theta}) \tag{2.5}$$

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 (2.4) *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 $t \in [0, T]$ and their corresponding solution mapping $\mathcal{G} : \mathcal{F} \times [0, T] \to \mathcal{U}$. As before, we end up with the representation

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

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 (2.4)*. 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.

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

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

Note that we cannot control the boundedness constant B here since it depends on the BOPB and therefore on the space $\mathcal{H}$ for our approximation. However, 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 (2.4) a reasonable approximation of the solution u.

Algorithm

In the present work, we will use the nonlinear approximation method for high-dimensional function approximation proposed in [17] in order to receive the desired approximation (2.4). We summarize some main aspects of the dimension-incremental algorithm here and refer to [17, 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. Later, the target function g will be the solution u in $d+n$ or $d+n+n_\theta$ 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 $j = 1, \ldots, d$ 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 $\Gamma = [0, N]^d$ with a certain extension N. 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 $\mathcal{P}_{\{t\}}(\Gamma) := \{k \in \mathbb{N} \,|\, \exists \boldsymbol{k} \in \Gamma : \boldsymbol{k}_t = k\}$ for all $t = 1, \ldots, d$ by constructing a suitable cubature rule for integrals of the form

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

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 $\tilde{\boldsymbol{x}}^{(j)}, j = 1, \ldots, r$. The temporary index sets $I_{\{t\}}$ with the reasonable indices for each dimension $t = 1, \ldots, d$ are then combined to proceed in a dimension-incremental way.

In the second step, starting with $t = 2$ a new candidate set $K := (I_{\{1,\ldots,t-1\}} \times I_{\{t\}}) \cap \mathcal{P}_{\{1,\ldots,t\}}(\Gamma)$ is formed, containing now higher-dimensional indices $\boldsymbol{k} \in \mathbb{N}^{|\{1,\ldots,t\}|}$. Again, a suitable t-dimensional cubature method, e.g., multiple rank-1 lattices as in [14], is constructed and evaluated using samples of the target function g to compute the projected coefficients

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

for the indices $\boldsymbol{k} \in K$, which is the natural generalization of (2.6) to multiple dimensions $\{1, \ldots, t\}$. As before, the algorithm collects those indicies $\boldsymbol{k} \in \mathbb{N}^t$ with the sparsity s largest (in absolute value) projected coefficients $\hat{g}_{\{1,\ldots,t\},\boldsymbol{k}}$ in the temporary index set $I_{\{1,\ldots,t\}}$. 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 $\hat{g}_{\{1,\ldots,d\},\boldsymbol{k}}$ do not longer depend on a random anchor $\tilde{\boldsymbol{x}}$ at all. We finally set $I = I_{\{1,\ldots,d\}}$ and $\hat{g}_{\boldsymbol{k}} := \hat{g}_{\{1,\ldots,d\},\boldsymbol{k}}$ for all $\boldsymbol{k} \in I_{\{1,\ldots,d\}}$. 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 a black-box sampling possibility, since the 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 constructed cubature methods combined with the random anchors. We again encourage the reader to see [17] 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.

For our application of this algorithm, where we have u as the target function, the black-box sampling method is the approximation of the solution of the differential equation (2.1) for the given parameter $\boldsymbol{a}$, followed by an evaluation of this particular approximation at the spatial point $\boldsymbol{x}$. 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, since the particular properties of the differential equation are mostly dealt with the black-box sampling step, which is basically the utilization of a numerical solver for the differential equation (2.1) with fixed right-hand side f (and fixed parameters $\boldsymbol{\theta}$ for the case described in Remark 2.1). Note that while the accuracy and efficiency of the solver used obviously directly affects the accuracy and efficiency of our method, we will not investigate the properties of these solvers in more detail in this work.

Remark 2.2. *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.

3 Numerics

In this Section, we will test our approach on several test problems, such as the Poisson and heat equations and discuss the results. We show, that our approach leads to sparse index

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 $t := 1, \ldots, d$ do
     Set $I_{\{t\}} := \emptyset$.
     Compute a suitable cubature method for $\mathcal{P}_{\{t\}}(\Gamma)$.
     for $i := 1, \ldots, r$ 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 $t := 2, \ldots, d$ do
     If $t < d$, set $\tilde{r} := r$ and otherwise $\tilde{r} := 1$.
     Set $I_{\{1,\ldots,t\}} := \emptyset$.
     Construct the index set $K := (I_{\{1,\ldots,t-1\}} \times I_{\{t\}}) \cap \mathcal{P}_{\{1,\ldots,t\}}(\Gamma)$.
     Compute a suitable cubature method for $K$.
     for $i := 1, \ldots, \tilde{r}$ 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 $\hat{g}_{\{1,\ldots,t\},\boldsymbol{k}}(\tilde{\boldsymbol{x}})$ for $\boldsymbol{k} \in K$.
        Add the (up to) $s$ indices $\boldsymbol{k}$ with the largest proj. coef. $|\hat{g}_{\{1,\ldots,t\},\boldsymbol{k}}(\tilde{\boldsymbol{x}})| \geq \delta_+$ to $I_{\{1,\ldots,t\}}$.
     end for $i$
  end for $t$
(Step 3)
  Set $I := I_{\{1,\ldots,d\}}$ and $\hat{g}_{\boldsymbol{k}} := \hat{g}_{\{1,\ldots,d\},\boldsymbol{k}}$ for all $\boldsymbol{k} \in I_{\{1,\ldots,d\}}$.
Output:  $I \subset \Gamma \subset \mathbb{N}^d$           detected index set
         $(\hat{g}_{\boldsymbol{k}})_{\boldsymbol{k} \in I} \in \mathbb{C}^{|I|}$      approximated coefficients with $|\hat{g}_{\boldsymbol{k}}| \geq \delta_+$
```

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 $T_{\boldsymbol{k}}(\boldsymbol{z}) := \prod_{j=1}^{d+n} T_{k_j}(z_j) = \prod_{j=1}^{d+n} \cos(k_j \arccos(z_j))$ as used in [17] on the domain $\mathcal{D} = [-1,1]^{d+n}$. Hence, the approximation (2.4) inserting $\boldsymbol{z} := (\boldsymbol{x}, \boldsymbol{a})$ 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

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

where $\boldsymbol{x}^{(j)}$ for $j = 1, \ldots, G$ 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 MATLAB®. 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 [17, 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 [17] 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 $H_0^1((0,1))$. Further, we use as approximation space the separable Hilbert space $\mathcal{H} = L_2(\mathcal{D})$ on a domain $\mathcal{D}$. As described in Section 2, our first step is to find a suitable parametrization (2.2) of the function f, which will also be responsible for the particular domain $\mathcal{D}$ we are using. 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 $S_I^{\mathcal{A}} u$, the additional dimensions will also result in a higher sampling and computational complexity. 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, 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 in solving the differential equation and focus on the approximation of the solution u directly.

As mentioned in Remark 2.2, 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 such Fourier coefficients $\boldsymbol{a}$ during this artificial example. Overall, the final function, which we are going to approximate here, is now

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

Using the explicit formula (3.5), we get

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

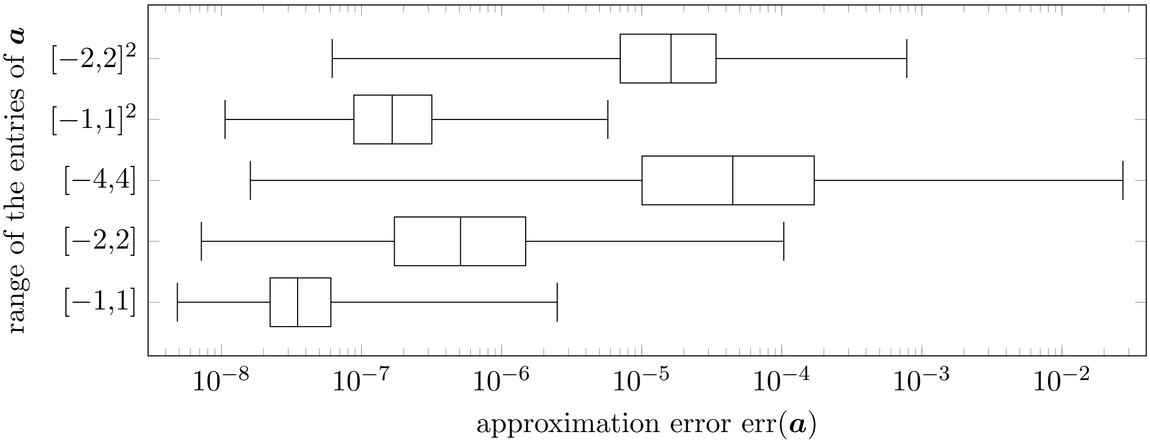


Figure 3.1: The relative approximation error $\mathrm{err}(\boldsymbol{a})$ for 10000 randomly drawn $\boldsymbol{a}$ when using the Fourier series parametrization. The box-and-whisker plots show the median, the first and the second quartile as well as the maximal and minimal error observed. The five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

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 dependence of u on $\boldsymbol{a}$ such as the tent-transform, cf. [3]. 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 $d = 10$. Further, we choose the sparsity $s = 1000$ and the extension $N = 64$ of the search space Γ.

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 satisfying. 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

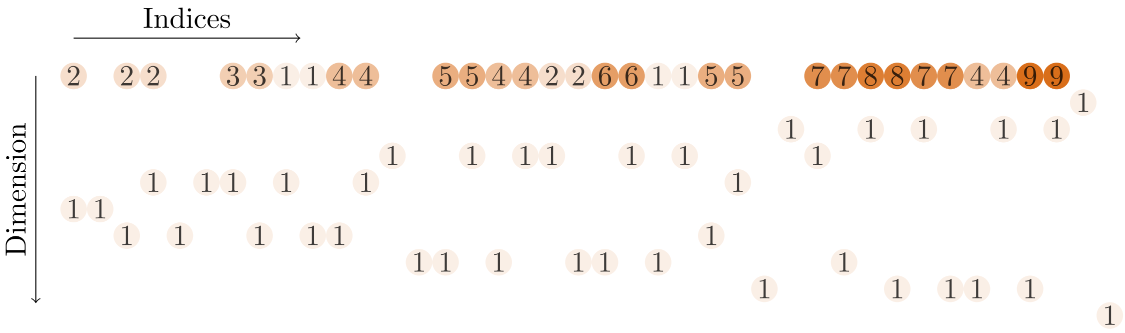


Figure 3.2: An abstract visualization of the first 40 indices $\boldsymbol{k}$ detected when using the Fourier series parametrization. The most left column contains the index $\boldsymbol{k}$ corresponding to the largest (in absolute value) basis coefficient $\hat{u}_{\boldsymbol{k}}$, the second column 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.

the restricted ones from the training setting.

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

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

with

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

and

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

being the indices mentioned above. Finally, we would expect no couplings between the different Fourier coefficients $a_{-4}, \ldots, a_4$ 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 such 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 [15] as before in Algorithm 1. This way, we only need about 650000 *samples to approximate all the basis coefficients $u_{\boldsymbol{k}}, \boldsymbol{k} \in I$, while our full algorithm in just* 10 *dimensions already needed around* 800000 *samples for this simple example. Overall, we used less than* 1.5 *million samples here, which 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 goal, 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 [16] and was the main motivation for the method proposed there. The relative approximation errors range from 10^{-8} to at most 10^{-7}, 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

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

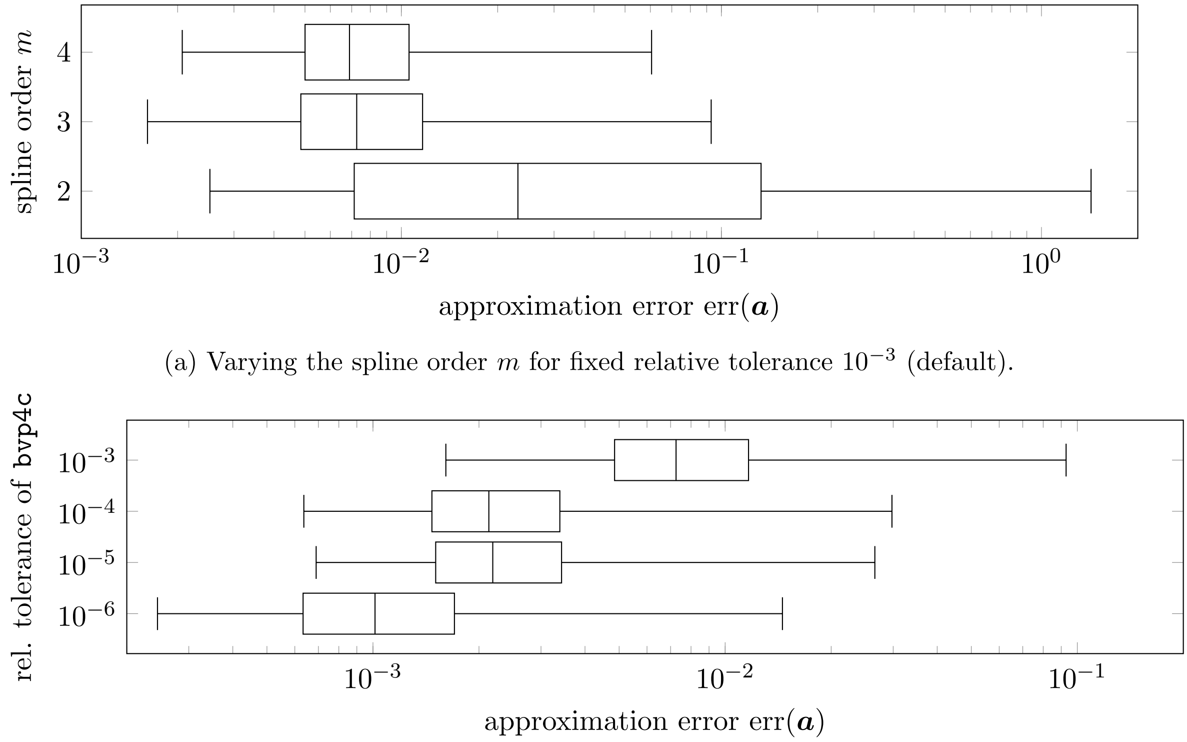
We use the additional index ℓ to indicate, that we scaled and shifted them w.r.t. the interval $[0, 1]$ and the desired amount of B-splines n, such that their peaks are equidistantly spaced along the interval and each spline overlaps $m - 1$ neighboring splines in each direction.

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 `bvp4c` of MATLAB® here, which is capable of integrating (systems of) differential equations $y' = f(x, y)$ with boundary conditions. As mentioned in Remark 2.2, 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 $\tilde{u}(\tilde{x}, \boldsymbol{a}) := u(\mathcal{T}\tilde{x}, \boldsymbol{a}) = u(\frac{1}{2}(\tilde{x} + 1), \boldsymbol{a})$.

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.

The approximation error shown in Figure 3.3 is derived by evaluating our approximation as well as the solution given by the `bvp4c` on 1000 equidistant spatial points and considering the respective ℓ_2-error $\mathrm{err}(\boldsymbol{a})$. Note, that we forced `bvp4c` to use a relative tolerance of 10^{-9} here in order to get reasonable values for the error estimation. 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^{-3} of the `bvp4c` function, which we used for these tests. Obviously, a higher accuracy of the underlying differential equation solver leads to a higher accuracy of our method, which can be observed in Figure 3.3b. Here, we fixed the spline order $m = 3$ and varied the relative tolerance of the `bvp4c` function. As mentioned before, we will not go into further detail about the properties of the differential



(a) Varying the spline order m for fixed relative tolerance 10^{-3} (default).

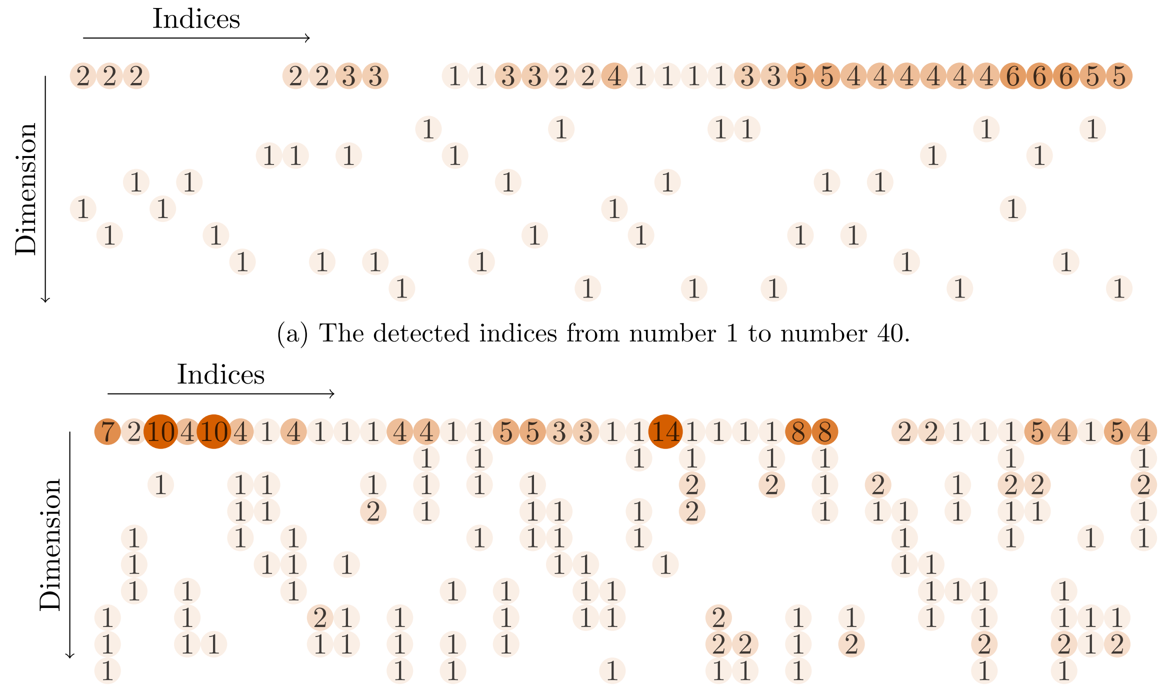
(b) Varying the relative tolerance for fixed spline order $m = 3$.

Figure 3.3: The relative approximation error $\mathrm{err}(\boldsymbol{a})$ for 10000 randomly drawn $\boldsymbol{a}$ for different choices of the spline order m and the relative tolerance of the solver function `bvp4c`. The box-and-whisker plots show the median, the first and the second quartile as well as the maximal and minimal error observed.

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 exemplary shown in Figure 3.4b, there appear higher-dimensional couplings and also values greater than 1 besides the spatial dimension. The intuitive guess, that the coupling B-spline coefficients a_ℓ are (almost) always neighboring each other, can also be observed. 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 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.



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

(b) The detected indices from number 101 to number 140.

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 $a_{-4}, \dots, a_4$ from top to bottom in this order. Zeros are neglected to preserve clarity.

As before, these are mainly caused by numerical errors and could be reduced by choosing the search space Γ more restrictive.

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 continuous 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 [26, 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 [26] 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 space $\mathcal{H} = L_2(\mathcal{D})$, the 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

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

with

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

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

Normally, we would proceed by choosing a suitable differential equation solver for our problem to insert into our dimension-incremental method. However, for the most common classical solvers, we detected similar results as for the PINNs in [26]. Obviously, since our algorithm is directly using the results of these solvers to compute approximation of u, we can not use these solvers with their incorrect solutions here. Therefore, we choose the same strategy as in Section 3.1.1 and directly use the analytical solution to test our method. Note 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.

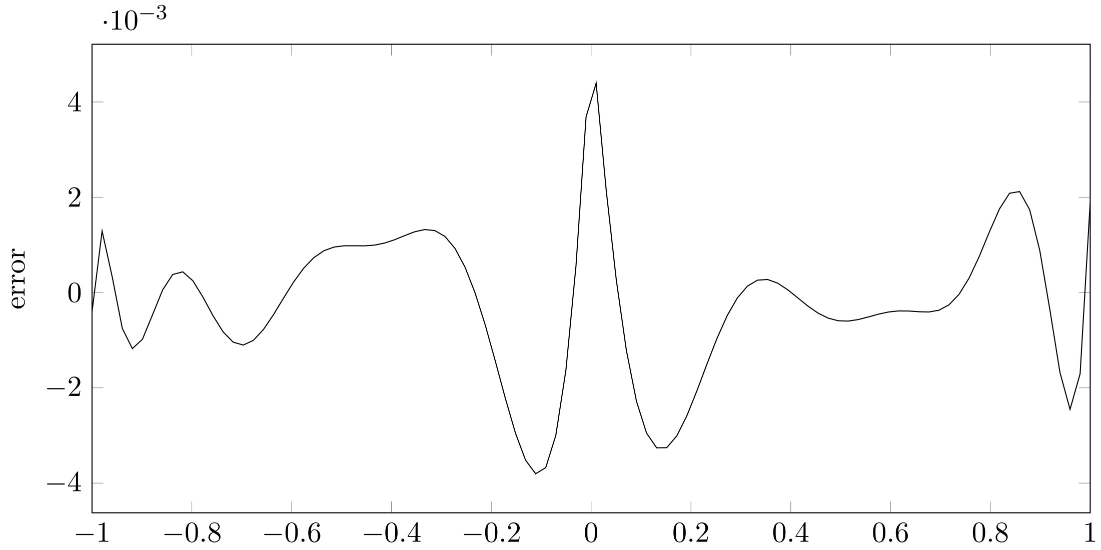


Figure 3.5: The (absolute) pointwise approximation error of our approximation when using the right-hand side (3.8) compared to the exact solution (3.9).

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 $s = 1000$ again with the extension $N = 64$ for this test example.

Obviously, since we are using an analytical solution to train our algorithm as we already did in Secion 3.1.1, the results are rather good. 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.

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 again detected perfectly, neglecting any entries larger than 1 in these dimensions already in Step 1 of Algorithm 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 equation with homogeneous Dirichlet boundary conditions is given by

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

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

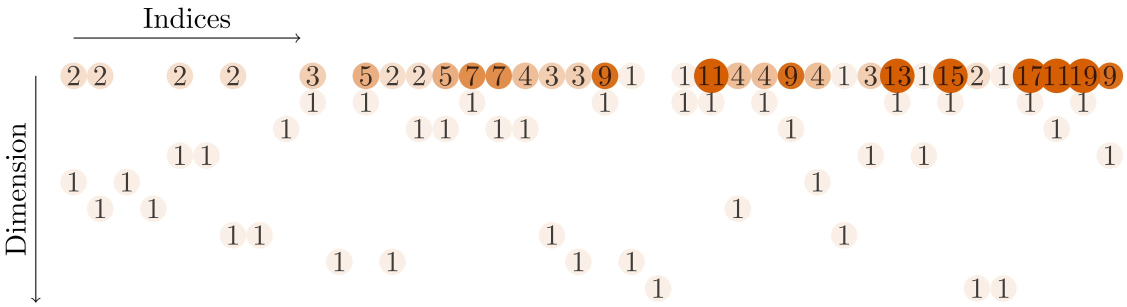


Figure 3.6: An abstract visualization of the first 40 indices $\boldsymbol{k}$ detected for the piece-wise continuous differential equation example. The indices $\boldsymbol{k}$ are ordered by the absolute values of their corresponding approximated coefficients $\hat{u}_{\boldsymbol{k}}$ in descending 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.

spaces used in Section 3.1. Also, the approximation space is again $\mathcal{H} = L_2(\mathcal{D})$ 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.

We use the Partial Differential Equation Toolbox™ of MATLAB® to solve the PDE (3.12) for given $\boldsymbol{a}$ this time. Briefly explained, we insert the domain Ω of the differential problem as well as the particular structure of the equation and the initial and boundary conditions. Then, the differential equation is solved using a Finite Element Method (FEM). For a more detailed explanation, we refer to the manual of the respective toolbox. For the Finite Element mesh, we used the generation parameter $H_{\max} = 0.05$, resulting in a mesh with 1893 spatial nodes (of which 81 nodes are on the boundary $\partial\Omega$ of our domain). 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 $J := \{-1, 0, 1\}^2$ 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.

Figure 3.7 illustrates the relative approximation error $\mathrm{err}(\boldsymbol{a})$ 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^{-4}, 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.

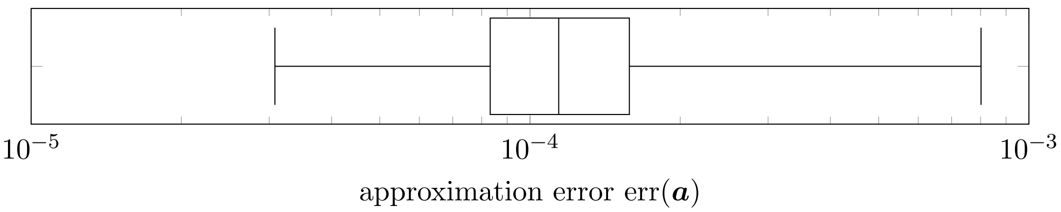


Figure 3.7: The relative approximation error err($\boldsymbol{a}$) for 10000 randomly drawn $\boldsymbol{a}$ for the two-dimensional Poisson equation example. The box-and-whisker plots show the median, the first and the second quartile as well as the maximal and minimal error observed.

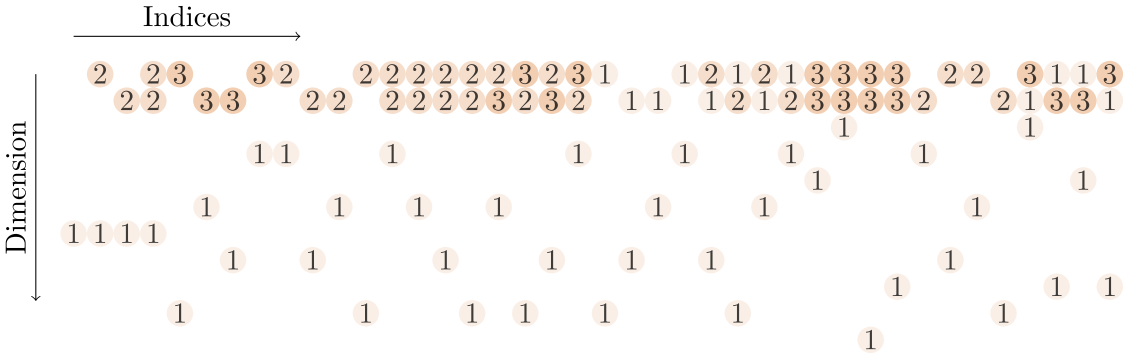


Figure 3.8: An abstract visualization of the first 40 indices $\boldsymbol{k}$ detected for the two-dimensional Poisson equation example. The indices $\boldsymbol{k}$ are ordered by the absolute values of their corresponding approximated coefficients $\hat{u}_{\boldsymbol{k}}$ in descending order from 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.

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.

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.

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

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 [16], we will only work with an affine random coefficient a here. In more detail, we consider the particular example from [5, Sec. 11], where we have for $n_{\boldsymbol{y}} = 20$ the affine coefficient

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

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

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

Here, $c > 0$ is a constant and $\mu > 1$ the decay rate. Further, $m_1(j)$ and $m_2(j)$ are defined as

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

with $k(j) := \lfloor -1/2 + \sqrt{1/4 + 2j} \rfloor$. For some explicit values of $m_1(j), m_2(j)$ and $k(j)$ as well as more details on this differential problem, see [5]. 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 [16, Sec. 4.3] using a slightly different approach. Therein, we discretize the spatial domain $\Omega = [0, 1]^2$ and compute approximations like (2.5) with $\boldsymbol{\theta} := \boldsymbol{y}$ 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.*

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 [16]. 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 $\mathcal{T}\boldsymbol{x} = \frac{1}{2}(\boldsymbol{x} + \boldsymbol{1})$. 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 the Partial Differential Equation Toolbox™ of MATLAB® to solve the differential equation. Note that we choose the generation parameter $H_{\max} = 0.075$ this time in order to coincide with the choice from [16].

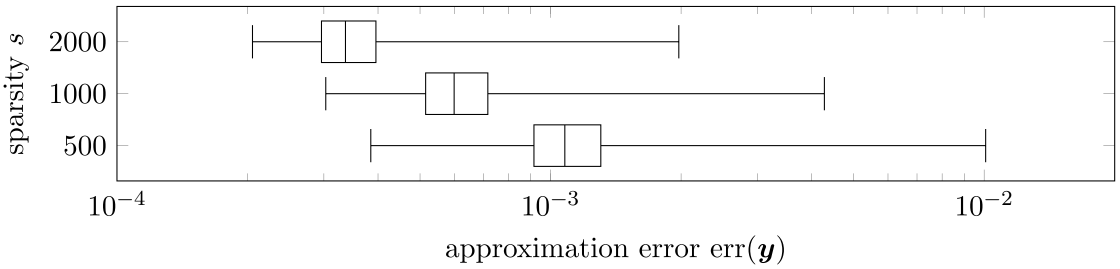


Figure 3.9: The relative approximation error $\mathrm{err}(\boldsymbol{y})$ for 10000 randomly drawn variables $\boldsymbol{y}$ for the diffusion equation example for different sparsities s. The box-and-whisker plots show the median, the first and the second quartile as well as the maximal and minimal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

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.

Figure 3.9 illustrates the relative approximation error $\mathrm{err}(\boldsymbol{y})$ 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 [16, 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 [16] by a factor of maximum 2. 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.

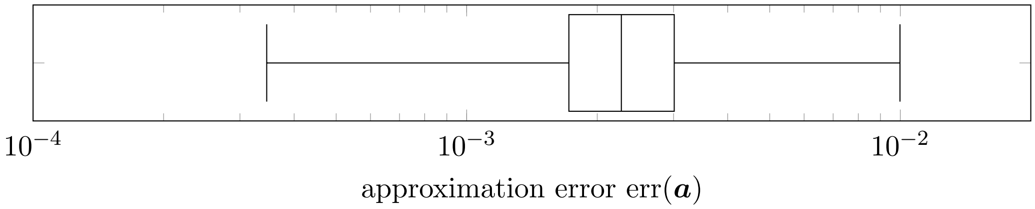


Figure 3.10: The relative approximation error $\operatorname{err}(\boldsymbol{a})$ for 10000 randomly drawn $\boldsymbol{a}$ for the heat equation example. The box-and-whisker plots show the median, the first and the second quartile as well as the maximal and minimal error observed.

3.5 Heat equation

Our final example is the heat equation in one dimension with homogeneous boundary conditions, i.e.

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

This time-dependent differential equation has no classical right-hand side as in the previous examples and our theoretical part in Section 2, but the initial condition $u(x, 0) = f(x)$. Hence, we are interested in parametrizing the function f, describing the initial state of the system at the time $t = 0$.

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

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

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 $t \geq 0$, we set the final time $T = 1$. Further, we set the length $L = 1$ and the diffusivity constant $\alpha = 0.25$. Since we consider the heat equation with zero boundary conditions, we again use $\mathcal{U} = H_0^1(\Omega)$ for this differential operator and thus the function space $\mathcal{F} = H^{-1}(\Omega)$ for the initial condition f. 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 2.1.

We parametrize the function f by truncating the sum (3.16) to n terms. Similar to Section 3.1.1, we restrict the coefficients $a_\ell \in [-1, 1]^n$ and transform both the spatial and time variable by $\mathcal{T}x = \frac{1}{2}(x+1)$ and $\mathcal{T}t = \frac{1}{2}(t+1)$. The differential equation is solved using the MATLAB® function `pdepe`, which is based on the method of lines. 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.

Figure 3.10 shows the relative approximation error $\operatorname{err}(\boldsymbol{a})$ computed for 10000 randomly drawn coefficients $\boldsymbol{a}$. The error is computed similarly to (3.2) for 100 equidistant nodes in

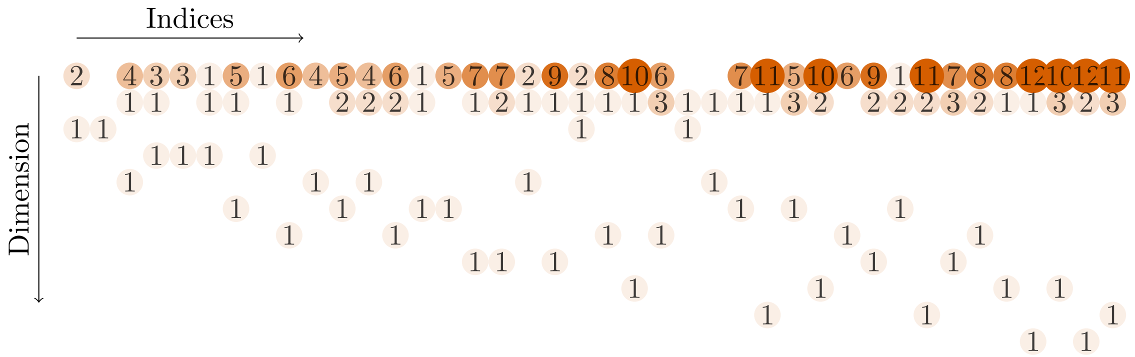


Figure 3.11: An abstract visualization of the first 40 indices $\boldsymbol{k}$ detected for the heat equation example. The indices $\boldsymbol{k}$ are ordered by the absolute values of their corresponding approximated coefficients $\hat{u}_{\boldsymbol{k}}$ in descending order from left to right. The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

space and time each and using the exact solution (3.15) as comparison. The average error size is about 10^{-3} for this example. Our numerical tests showed, that the diffusivity constant α has a tremendous impact on the accuracy of the approximation. The rapid decay of the solution w.r.t. the time t for larger α increases the difficulty of this problem and would force our method to use a much higher resolution w.r.t. the time t than only $N = 64$.

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 a_ℓ appear only linearly and separated from each other in the sum in (3.15). 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 t. However, the maximal entry in the first dimension happens to be 22 while the second dimension has 64 (matching our parameter choice $N = 64$ for the search space Γ) as largest entry. This again confirms, that the time-dependent exponential term in (3.15) is the main difficulty in this approximation problem.

4 Conclusion

We presented an approach that uses the dimension-incremental algorithm from [17] 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, 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 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.

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

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

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

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

[10] T. G. Grossmann, U. J. Komorowska, J. Latz, and C.-B. Schönlieb. Can physics-informed neural networks beat the finite element method? *arXiv:2302.04107*, 2023.

[11] E. Hubert and M. F. Singer. Three ways to solve partial differential equations with neural networks — a review. *GAMM-Mitteilungen*, 44:e202100006, 2021.

[12] 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.

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

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

[15] L. Kämmerer. Constructing efficient spatial discretizations of spans of multivariate Chebyshev polynomials. *arXiv:2406.03281*, 2024.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

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

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

[23] 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.

[24] 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.

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

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

[27] 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.

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

[29] 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.

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