TECHNISCHE UNIVERSITÄT
CHEMNITZ

Faculty of Mathematics

Master thesis

A uniform sparse FFT approach on differential equations with random coefficients

Fabian Taubert
Born August 22, 1996 in Stollberg
Chemnitz, March 16, 2021

First Advisor: Prof. Dr. Daniel Potts
Second Advisor: Dr. Lutz Kämmerer

Contents

1 Introduction

Parametric operator equations have gained significant attention in recent years. In particular, partial differential equations with random coefficients play an important role in the study of uncertainty quantification, e.g., [5, 12, 13]. Therefore, the numerical solution of these equations and how to compute them in an efficient and reliable way has become more and more important.
In this work, we will consider partial differential equations with random coefficients of the form

$$-\nabla_{\boldsymbol{x}} \cdot (a(\boldsymbol{x}, \boldsymbol{\xi}) \nabla_{\boldsymbol{x}} u(\boldsymbol{x}, \boldsymbol{\xi})) = f(\boldsymbol{x}) \qquad \boldsymbol{x} \in D,\, \boldsymbol{\xi} \in D_{\boldsymbol{\xi}} \tag{1.1a}$$

$$u(\boldsymbol{x}, \boldsymbol{\xi}) = 0 \qquad \forall\, \boldsymbol{x} \in \partial D,\, \boldsymbol{\xi} \in D_{\boldsymbol{\xi}}, \tag{1.1b}$$

describing the diffusion characteristics of inhomogeneous materials and therefore being called diffusion equations with the random diffusion coefficients a. Here, $\boldsymbol{x} \in D$ is the spatial variable in a domain $D \subseteq \mathbb{R}^d$, where $d = 1, 2$, or 3, and $\boldsymbol{\xi} = (\xi_j)_{j=1}^{d_{\boldsymbol{\xi}}} \in D_{\boldsymbol{\xi}}$ is a high dimensional random variable with $D_{\boldsymbol{\xi}} = \bigtimes_{j=1}^{d_{\boldsymbol{\xi}}} [\alpha_j, \beta_j]$. The here considered model of the random field in the PDE is then realized by the random coefficient a of the general form

$$a(\boldsymbol{x}, \boldsymbol{\xi}) = a_0(\boldsymbol{x}) + \sum_{j=1}^{d_{\boldsymbol{\xi}}} \xi_j \, \psi_j(\boldsymbol{x}). \tag{1.2}$$

Often, the random variables $\xi_j, j = 1, ..., d_{\boldsymbol{\xi}}$, are assumed to be independent and each uniformly distributed on an interval. This model with a linear appearance of the ξ_j has been used in many recent papers, e.g., [1, 2, 6, 7, 8, 10, 17, 18, 21]. Further, [12, 13] recently also considered the modeling with periodic random variables instead, which shows also worth to be considered.
In each case, the random variable $\boldsymbol{\xi}$ is assumed to be high dimensional or even infinite dimensional. Therefore, the most common approximation methods are subject to the curse of dimensionality. Recent works commonly suggest to apply known solvers to the differential equation for several fixed $\boldsymbol{\xi}$. The quantities of interest are then computed from this set of solutions. In [4], an approach using Fourier methods was presented, that computes a complete approximate solution for one-dimensional diffusion equations with random variables, i.e., $d = 1$. While the use of the sparse FFT approach as stated in [16, 20] performs well even in high dimensions, the main advantage of this strategy can be found in the detailed characteristics of the random variables. The proposed approach is able to reveal information as the influence of each single variable ξ_j on the solution as well as the interaction between those random variables.
In this work, we present a non-intrusive approach, based on the main idea of the algorithm developed in [4]. While in [4] the sFFT was used w.r.t. the spatial variable $\boldsymbol{x}$ and the random variable $\boldsymbol{\xi}$, we will not include the spatial variable $\boldsymbol{x}$ in our Fourier approach this time. In particular, we will use the sFFT approach to compute approximations of the functions $u(\boldsymbol{x}_g, \boldsymbol{\xi})$ w.r.t. the random variable $\boldsymbol{\xi}$ for several fixed points $\boldsymbol{x}_g$ on a grid $\mathcal{T} \subset D$.

Unfortunately, applying the sFFT at each of these points $\boldsymbol{x}_g$ separately will lead to an unnecessary huge increase of the computation time, ruining one of the main advantages of our approach. Therefore, we will develop a modification of the sFFT, that avoids this problem. In particular, our so-called *uniform sFFT* combines the candidate sets between each dimension-increment and therefore uses the same sampling nodes $\boldsymbol{\xi}$ for each grid point x_g. This strategy manages to keep the number of used samples in a reasonable size, which is highly important, since each sample is one computation of the solution of the differential equation.

The main advantage of this approach is the adaptive choice of the frequency set performed by the underlying sFFT. Most of the approaches in the aforementioned works are based on tensorized polynomials or compressed sensing methods and assume the particular index set needed to be known in advance. Especially when working with certain weights to describe the index sets, slightly modified weights may result in tremendously large or way too small index sets, that are computational unfeasible or not capable of yielding a good approximation, respectively. In other words, reasonably estimating these weights is a particular challenge, which might necessitate considerable additional effort. The uniform sFFT only needs a candidate set and selects the important frequencies in this search domain on its own. Also, the size of this candidate set is not as problematic as for other approaches, since the number of used samples and the computation time suffer only mildly from larger candidate sets.

Another main advantage of our algorithm is the non-intrusive and parallelizable behavior. The uniform sFFT uses existing numerical solvers of the considered differential equation. We can use suitable, reliable and efficient solvers with no need to re-implement them. Hence, this approach works for ODEs as well as for PDEs. Further, the different samples needed in each sampling step can be computed on multiple instances. This parallelization allows to reduce the computation time even further and makes a higher number of used samples less time consuming.

The work is organized as follows: In Section 2 we restrict ourself to a one-dimensional spatial variable x and we explain the general approach in detail as well as the two ODE solvers we use in the numerical tests later on. Section 2.4 outlines the dimension-incremental method and, most important, explains the modifications made to achieve the uniform sFFT. Afterwards, we will explain how to compute quantities of interest as for example the expectation value of the solution of the differential equation. Sections 2.6 and 2.7 include several numerical tests on our method and some discussion about the used parameters and sampling strategies. In Section 3 we apply our uniform sFFT on two specific PDEs with a two-dimensional spatial variable $\boldsymbol{x}$ using a finite element PDE solver. Section 4 summarizes the findings of the numerical tests, so the advantages and disadvantages of our method and open questions.

2 The one-dimensional case

2.1 The general approach

We consider the ordinary differential equation

$$-\frac{\partial}{\partial x}\left(a(x,\boldsymbol{\xi})\frac{\partial}{\partial x}u(x,\boldsymbol{\xi})\right) = f(x) \qquad x \in [\alpha,\beta],\ \boldsymbol{\xi} \in D_{\boldsymbol{\xi}} \tag{2.1a}$$

$$u(\alpha,\boldsymbol{\xi}) = u(\beta,\boldsymbol{\xi}) = 0 \qquad \forall\, \boldsymbol{\xi} \in D_{\boldsymbol{\xi}} \tag{2.1b}$$

with homogeneous boundary conditions, where $\boldsymbol{\xi} \in \mathbb{R}^{d_{\boldsymbol{\xi}}}$ contains the random variables and x is the one-dimensional spatial variable. The diffusion coefficient $a : [\alpha,\beta] \times D_{\boldsymbol{\xi}} \to \mathbb{R}$ and the right hand side $f : [\alpha,\beta] \mapsto \mathbb{R}$ are continuous functions and assumed to be known. Here, $D_{\boldsymbol{\xi}} = \bigtimes_{j=1}^{d_{\boldsymbol{\xi}}} [\alpha_j,\beta_j]$ describes the domain of the random variables. We further assume, that

$$0 < r \le a(x,\boldsymbol{\xi}) \le R < \infty \qquad \forall (x,\boldsymbol{\xi}) \in [\alpha,\beta] \times D_{\boldsymbol{\xi}} \tag{2.2}$$

to guarantee the existence of a unique solution of (2.1) for each fixed $\boldsymbol{\xi}$. We will now treat this solution at a fixed point x_0 as a function w.r.t. the random variable $\boldsymbol{\xi}$ and want to use a Fourier approach to derive a good approximation of this function $u(x_0,\cdot)$. Unfortunately, the function w.r.t. the random variable $\boldsymbol{\xi}$ is in general not periodic. Therefore, we need to apply a suitably chosen, 1-periodic transformation function

$$\varphi : \mathbb{T}^{d_{\boldsymbol{\xi}}} \to D_{\boldsymbol{\xi}} \tag{2.3}$$

first, cf. Section 2.2. So we consider the periodization

$$\tilde{u}(x_0,\tilde{\boldsymbol{\xi}}) \coloneqq u(x_0,\varphi(\tilde{\boldsymbol{\xi}})) = u(x_0,\boldsymbol{\xi}),$$

which is then a 1-periodic function in $\tilde{\boldsymbol{\xi}}$ for each fixed x_0 and can therefore be approximated by a Fourier partial sum

$$S_{\mathrm{I}_{x_0}}[\tilde{u}(x_0,\cdot)](\tilde{\boldsymbol{\xi}}) = \sum_{\boldsymbol{k}\in\mathrm{I}_{x_0}} c_{\boldsymbol{k}}(\tilde{u}(x_0,\cdot))\, \mathrm{e}^{2\pi \mathrm{i}\boldsymbol{k}\cdot\tilde{\boldsymbol{\xi}}}. \tag{2.4}$$

Using the dimension-incremental sparse FFT approach stated in [20], we can recover a frequency set I_{x_0} and an approximation $\hat{u}_{x_0,\boldsymbol{k}}$ of the corresponding Fourier coefficients $c_{\boldsymbol{k}}(\tilde{u}(x_0,\cdot))$ with $\boldsymbol{k} \in \mathrm{I}_{x_0}$, cf. Section 2.4.1. Note, that the frequency set I_{x_0}, chosen by the algorithm, also depends on x_0. Hence, we gain an approximation

$$\hat{S}_{\mathrm{I}_{x_0}}[\tilde{u}(x_0,\cdot)](\tilde{\boldsymbol{\xi}}) = \sum_{\boldsymbol{k}\in\mathrm{I}_{x_0}} \hat{u}_{x_0,\boldsymbol{k}}\, \mathrm{e}^{2\pi \mathrm{i}\boldsymbol{k}\cdot\tilde{\boldsymbol{\xi}}}.$$

of (2.4). Then we can get back to an approximation of the non-periodized signal via

$$\hat{S}_{\mathrm{I}_{x_0}}[u(x_0,\cdot)](\boldsymbol{\xi}) \coloneqq \hat{S}_{\mathrm{I}_{x_0}}[\tilde{u}(x_0,\cdot)](\varphi^{-1}(\boldsymbol{\xi})) \qquad \forall \boldsymbol{\xi} \in D_{\boldsymbol{\xi}}, \tag{2.5}$$

with φ^{-1} being the inverse of the transformation function φ w.r.t. some interval as explained in the next section.

2.2 Periodization

To simplify notifications, we will assume

$$D_{\boldsymbol{\xi}} = \bigtimes_{j=1}^{d_{\boldsymbol{\xi}}} [-1, 1]$$

for the domain of the random variables from now on. We need to apply a suitable transformation $\varphi : \mathbb{T}^{d_{\boldsymbol{\xi}}} \to [-1, 1]^{d_{\boldsymbol{\xi}}}$ to periodize the function $u(x_0, \boldsymbol{\xi})$. Therefore, we assume, that φ acts component-wise on $\tilde{\boldsymbol{\xi}}$, i.e.,

$$\varphi(\tilde{\boldsymbol{\xi}}) = \begin{pmatrix} \varphi_1(\tilde{\xi}_1) \\ \varphi_2(\tilde{\xi}_2) \\ \vdots \\ \varphi_{d_{\boldsymbol{\xi}}}(\tilde{\xi}_{d_{\boldsymbol{\xi}}}) \end{pmatrix} = \begin{pmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_{d_{\boldsymbol{\xi}}} \end{pmatrix} = \boldsymbol{\xi}$$

and the transformation φ_j in each component $j = 1, ..., d_{\boldsymbol{\xi}}$ is the same. Further, we will assume for all $j = 1, ..., d_{\boldsymbol{\xi}}$, that

- φ_j is continuous, so $\varphi \in C(\mathbb{T}^{d_{\boldsymbol{\xi}}})$,
- $\varphi_j(0) = \varphi_j(1) = -1$ and $\varphi_j(0.5) = 1$, and thus φ is periodic with period one,
- $\varphi_j(0.5 - x) = \varphi_j(0.5 + x)$ for $x \in [0, 0.5]$,
- φ_j is strictly monotonously increasing in $(0, 0.5)$,

With this restrictions we ensure, that φ is bijective on the interval $[0, 0.5]^{d_{\boldsymbol{\xi}}}$. Hence, we can write φ^{-1} in (2.5) to denote the corresponding inverse, mapping from $[-1, 1]^{d_{\boldsymbol{\xi}}} \to [0, 0.5]^{d_{\boldsymbol{\xi}}}$. In the following, we will see three examples of possible transformation functions, stated in [4]. These periodizations are also illustrated in Figure 2.1.

Example 2.1
A simple periodization can be achieved by using the tent transformation, given by

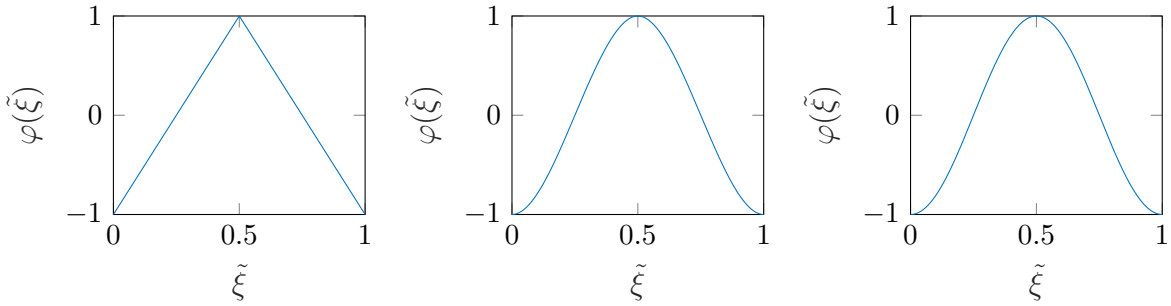
$$\begin{aligned} &\varphi : \mathbb{T}^{d_{\boldsymbol{\xi}}} \to [-1, 1]^{d_{\boldsymbol{\xi}}} \\ &\varphi_j(\tilde{\xi}_j) = 1 - |2 - 4\,\tilde{\xi}_j| \qquad\qquad \forall j = 1, ..., d_{\boldsymbol{\xi}}. \end{aligned} \tag{2.6}$$

□

Example 2.2
We can construct a two times continuously differentiable periodization by using a spline of order four, given by

$$\begin{aligned} &\varphi : \mathbb{T}^{d_{\boldsymbol{\xi}}} \to [-1, 1]^{d_{\boldsymbol{\xi}}} \\ &\varphi_j(\tilde{\xi}_j) = \begin{cases} -32\tilde{\xi}_j^{\,3} + 24\tilde{\xi}_j^{\,2} - 1 & 0 \le \tilde{\xi}_j \le 0.5 \\ 32\tilde{\xi}_j^{\,3} - 72\tilde{\xi}_j^{\,2} + 48\tilde{\xi}_j - 9 & 0.5 < \tilde{\xi}_j \le 1 \end{cases} \qquad \forall j = 1, ..., d_{\boldsymbol{\xi}}. \end{aligned} \tag{2.7}$$

□



(a) tent transform from Example 2.1 (b) spline transform from Example 2.2 (c) cosine transform from Example 2.3

Figure 2.1: The transformation functions φ from the Examples 2.1, 2.2 and 2.3 for $d_{\boldsymbol{\xi}} = 1$.

Example 2.3
The periodization

$$\begin{aligned} &\varphi : \mathbb{T}^{d_{\boldsymbol{\xi}}} \to [-1,1]^{d_{\boldsymbol{\xi}}} \\ &\varphi_j(\tilde{\xi}_j) = -\cos(2\pi\tilde{\xi}_j) \qquad\qquad \forall j = 1, ..., d_{\boldsymbol{\xi}} \end{aligned} \tag{2.8}$$

is based on the cosine function and therefore infinitely differentiable. □

The tent transformation (2.6) is a very simple transformation due to its linearity on the interval $[0, 0.5]$ and therefore φ^{-1} is also of a linear structure. This also allows for certain simplifications to be applied when used as periodization function, as we will see in Section 2.5. Unfortunately, it is not differentiable in $\tilde{\xi}_j = 0.5$, so it might be an unfavorable choice for a Fourier approach. Therefore, the usage of more smooth transformation functions φ as (2.7) or (2.8) might yield better results.

2.3 Solving the ODE

As a further simplification, we will assume, that $x \in [-1, 1]$ to simplify notations and preserve clarity from now on. In order to use the sparse FFT approach stated in Chapter 2.1, we need a method to sample, i.e., to solve the ODE

$$-\frac{\partial}{\partial x}\left(a(x, \boldsymbol{\xi})\frac{\partial}{\partial x}u(x, \boldsymbol{\xi})\right) = f(x) \qquad x \in [-1, 1],\ \boldsymbol{\xi} \in D_{\boldsymbol{\xi}} \tag{2.9a}$$

$$u(-1, \boldsymbol{\xi}) = u(1, \boldsymbol{\xi}) = 0 \qquad \forall\, \boldsymbol{\xi} \in D_{\boldsymbol{\xi}} \tag{2.9b}$$

for a fixed $\boldsymbol{\xi} \in D_{\boldsymbol{\xi}}$ and evaluate u at a given point $x_0 \in [-1, 1]$. Of course, there exist many possible approaches, to solve such a differential equation. Here, we will consider two different ways using a Fourier approach as in [4] and an already implemented numerical solver provided by Matlab®.

2.3.1 An efficient Fourier solver

Although the differential equation is of second order, it has a rather simple structure. We can solve this ODE for every fixed $\boldsymbol{\xi} \in D_{\boldsymbol{\xi}}$ by integrating from -1 to x. Further we can divide by $a(x, \boldsymbol{\xi})$, since the condition (2.2) ensures, that we won't divide by zero.

$$\begin{aligned} a(x,\boldsymbol{\xi})\frac{\partial}{\partial x}u(x,\boldsymbol{\xi}) &= -\int_{-1}^{x} f(\tau)\mathrm{d}\tau + c_1(\boldsymbol{\xi}) \\ \frac{\partial}{\partial x}u(x,\boldsymbol{\xi}) &= -\frac{1}{a(x,\boldsymbol{\xi})}\left(\int_{-1}^{x} f(\tau)\mathrm{d}\tau + c_1(\boldsymbol{\xi})\right) \\ &= -\underbrace{\frac{1}{a(x,\boldsymbol{\xi})}\int_{-1}^{x} f(\tau)\mathrm{d}\tau}_{=v_1(x,\boldsymbol{\xi})} - c_1(\boldsymbol{\xi})\ \underbrace{\frac{1}{a(x,\boldsymbol{\xi})}}_{=v_2(x,\boldsymbol{\xi})} \end{aligned}$$

A second integration on both sides now yields

$$u(x,\boldsymbol{\xi}) = -\int_{-1}^{x} v_1(\tau,\boldsymbol{\xi})\mathrm{d}\tau - c_1(\boldsymbol{\xi})\int_{-1}^{x} v_2(\tau,\boldsymbol{\xi})\mathrm{d}\tau + c_2(\boldsymbol{\xi}). \tag{2.10}$$

The integration constants $c_1(\boldsymbol{\xi})$ and $c_2(\boldsymbol{\xi})$ can be determined by using the boundary conditions (2.9b). For $x = -1$ both integrals vanish and using $u(-1, \boldsymbol{\xi}) = 0$ we obtain $c_2 = 0$ for all $\boldsymbol{\xi}$. In order to compute $c_1(\boldsymbol{\xi})$ we use $x = 1$ and the boundary condition yields

$$0 = u(1,\boldsymbol{\xi}) = -\int_{-1}^{1} v_1(\tau,\boldsymbol{\xi})\mathrm{d}\tau - c_1(\boldsymbol{\xi})\int_{-1}^{1} v_2(\tau,\boldsymbol{\xi})\mathrm{d}\tau.$$

After rearranging the terms we see, that we can compute $c_1(\boldsymbol{\xi})$ as

$$c_1(\boldsymbol{\xi}) = -\frac{\int_{-1}^{1} v_1(\tau,\boldsymbol{\xi})\mathrm{d}\tau}{\int_{-1}^{1} v_2(\tau,\boldsymbol{\xi})\mathrm{d}\tau}. \tag{2.11}$$

The final formula (2.10) now includes multiple integrations of the form

$$\int_{-1}^{x} w(\tau,\boldsymbol{\xi})\mathrm{d}\tau,$$

which we will realize by a Fourier approximation. To periodize the integrand w.r.t. the spatial variable τ, we again use the transformation function φ, which we discussed in Section 2.2. With this periodization, we can rewrite the integral as

$$h(x,\boldsymbol{\xi}) \coloneqq \int_{-1}^{x} w(\tau,\boldsymbol{\xi})\mathrm{d}\tau = \int_{\varphi^{-1}(-1)}^{\varphi^{-1}(x)} w(\varphi(\tau),\boldsymbol{\xi})\ \varphi'(\tau)\mathrm{d}\tau = \int_{0}^{\varphi^{-1}(x)} \tilde{w}(\tau,\boldsymbol{\xi})\ \varphi'(\tau)\mathrm{d}\tau, \tag{2.12}$$

where $\tilde{w}(\tau, \boldsymbol{\xi}) = w(\varphi(\tau), \boldsymbol{\xi})$. Now we approximate the integrand on the right-hand side by a Fourier partial sum:

$$S_N\left[\tilde{w}\varphi'\right](\tau, \boldsymbol{\xi}) = \sum_{k=-N}^{N-1} \hat{a}_k(\boldsymbol{\xi})\, \mathrm{e}^{2\pi \mathrm{i} k\tau} \tag{2.13a}$$

$$\hat{a}_k(\boldsymbol{\xi}) = \int_0^1 \tilde{w}(\tau, \boldsymbol{\xi})\, \varphi'(\tau)\, \mathrm{e}^{-2\pi \mathrm{i} k\tau} \mathrm{d}\tau \qquad k = -N, ..., N-1. \tag{2.13b}$$

We approximate the Fourier coefficients in the common way using the rectangle rule. Notice, that for Fourier coefficients this leads to the same formula as the trapezoidal rule would. Further, we won't use the equidistant grid $\left\{\frac{j}{M} : j = 0, ..., M-1\right\}$ to avoid using $\varphi'(0.5)$, since this value sometimes might be undefined for example when working with the tent transformation. Instead we use the shifted grid

$$\tilde{\mathcal{T}}_M = \left\{\frac{j+0.5}{M}\right\}_{j=0}^{M-1}$$

and approximate the Fourier coefficients as

$$\begin{aligned}
\hat{a}_k(\boldsymbol{\xi}) &= \int_{\frac{1}{2M}}^{1+\frac{1}{2M}} \tilde{w}(\tau, \boldsymbol{\xi})\, \varphi'(\tau)\, \mathrm{e}^{-2\pi \mathrm{i} k\tau} \mathrm{d}\tau \\
&\approx \frac{1}{M} \sum_{j=0}^{M-1} \tilde{w}\left(\frac{j+0.5}{M}, \boldsymbol{\xi}\right)\, \varphi'\left(\frac{j+0.5}{M}\right) \mathrm{e}^{-2\pi \mathrm{i} k \frac{j+0.5}{M}} \\
&= \frac{1}{M}\, \mathrm{e}^{-\pi \mathrm{i} \frac{k}{M}} \sum_{j=0}^{M-1} \tilde{w}\left(\frac{j+0.5}{M}, \boldsymbol{\xi}\right)\, \varphi'\left(\frac{j+0.5}{M}\right) \mathrm{e}^{-2\pi \mathrm{i} k \frac{j}{M}}, \quad k = -N, ..., N-1,
\end{aligned} \tag{2.14}$$

which we will compute later using an FFT of length $M = 2N$. Now we integrate the Fourier partial sum (2.13a) and have

$$\int S_N\left[\tilde{w}\varphi'\right](\tau, \boldsymbol{\xi})\, \mathrm{d}\tau = \hat{a}_0(\boldsymbol{\xi})\tau + \sum_{\substack{k=-N \\ k\neq 0}}^{N-1} \frac{\hat{a}_k(\boldsymbol{\xi})}{2\pi \mathrm{i} k} \mathrm{e}^{2\pi \mathrm{i} k\tau} + C = \hat{a}_0(\boldsymbol{\xi})\tau + \sum_{k=-N}^{N-1} \hat{b}_k(\boldsymbol{\xi}) \mathrm{e}^{2\pi \mathrm{i} k\tau} + C$$

with

$$\hat{b}_k(\boldsymbol{\xi}) = \begin{cases} 0 & k = 0 \\ \dfrac{\hat{a}_k(\boldsymbol{\xi})}{2\pi \mathrm{i} k} & k \neq 0 \end{cases} \qquad k = -N, ..., N-1.$$

Since we only compute approximations of the Fourier coefficients $\hat{a}_k$ in (2.14), we will also end up with approximations of the scaled coefficients $\hat{b}_k$ as well.
Note, that since φ' is symmetric, i.e., $\varphi'(0.5 - x) = -\varphi'(0.5 + x)$ for $x \in (0, 0.5)$, the Fourier coefficient $\hat{a}_0(\boldsymbol{\xi})$ vanishes. Therefore we will neglect the term $\hat{a}_0(\boldsymbol{\xi})\tau$ in the following

considerations. Finally, we rewrite the integral (2.12) as

$$\begin{aligned} h(x,\boldsymbol{\xi}) = \int_{-1}^{x} w(\tau,\boldsymbol{\xi}) d\tau &= \int_{0}^{\varphi^{-1}(x)} \tilde{w}(\tau,\boldsymbol{\xi})\ \varphi'(\tau) \mathrm{d}\tau \\ &\approx \int_{0}^{\varphi^{-1}(x)} S_N\left[\tilde{w}\varphi'\right](\tau,\boldsymbol{\xi}) \mathrm{d}\tau \\ &= \sum_{k=-N}^{N-1} \hat{b}_k(\boldsymbol{\xi}) \left(\mathrm{e}^{2\pi \mathrm{i} k \varphi^{-1}(x)} - \mathrm{e}^{0}\right) \\ &= \sum_{k=-N}^{N-1} \hat{b}_k(\boldsymbol{\xi}) \left(\mathrm{e}^{2\pi \mathrm{i} k \varphi^{-1}(x)} - 1\right). \end{aligned} \tag{2.15}$$

Further, we evaluate the first part of the sum in (2.15) with an FFT. Therefore, we use $x \in \tilde{\mathcal{T}}_M$ as in the quadrature formula (2.14) with $M = 2N$. Using these two modifications, (2.15) now reads as

$$\begin{aligned} \tilde{h}\left(\frac{j+0.5}{M}, \boldsymbol{\xi}\right) = h\left(\varphi\left(\frac{j+0.5}{M}\right), \boldsymbol{\xi}\right) &:= \int_{-1}^{\varphi\left(\frac{j+0.5}{M}\right)} w(\tau,\boldsymbol{\xi}) \mathrm{d}\tau \\ &\approx \sum_{k=-N}^{N-1} \hat{b}_k(\boldsymbol{\xi}) \left(\mathrm{e}^{2\pi \mathrm{i} k \frac{j+0.5}{M}} - 1\right) \\ &= \sum_{k=-N}^{N-1} \hat{b}_k(\boldsymbol{\xi}) \left(\mathrm{e}^{\pi \mathrm{i} \frac{k}{M}} \mathrm{e}^{2\pi \mathrm{i} k \frac{j}{M}} - 1\right) \end{aligned} \tag{2.16}$$

for all $j = 0, ..., M-1$. Note, that when plugging in $x \in \tilde{\mathcal{T}}_M$ we receive the expression $\varphi^{-1}(\varphi\left(\frac{j+0.5}{M}\right))$. This simplifies to $\frac{j+0.5}{M}$ for all $j = 0, ..., M-1$ when using the inverse of φ w.r.t. the interval $[0, 0.5]$ for $j = 0, ..., N-1$ and the inverse w.r.t. the interval $[0.5, 1]$ for $j = N, ..., M-1$.
In order to determine the integration constant $c_1(\boldsymbol{\xi})$ when solving the ODE, we need the values $\int_{-1}^{1} v_1(\tau,\boldsymbol{\xi}) d\tau$ and $\int_{-1}^{1} v_2(\tau,\boldsymbol{\xi}) d\tau$. But since we constructed our grid $\tilde{\mathcal{T}}_M$ to not contain 0.5, we don't get these values in our integration algorithm. Thus, we evaluate $\tilde{h}(0.5,\boldsymbol{\xi}) = h(1,\boldsymbol{\xi})$ separately via

$$\begin{aligned} \tilde{h}(0.5,\boldsymbol{\xi}) = h(1,\boldsymbol{\xi}) &\approx \sum_{k=-N}^{N-1} \hat{b}_k(\boldsymbol{\xi}) \left(\mathrm{e}^{\pi \mathrm{i} \frac{k}{M}} \mathrm{e}^{2\pi \mathrm{i} k \frac{1}{2}} - 1\right) \\ &= \sum_{k=-N}^{N-1} \hat{b}_k(\boldsymbol{\xi}) \left(\mathrm{e}^{\pi \mathrm{i} \frac{k}{M}} (-1)^k - 1\right). \end{aligned} \tag{2.17}$$

The final integration algorithm using matrix-vector notation and the Fast Fourier Transform is stated in Algorithm 1.
We now use this algorithm in order to compute the integrals over f, v_1 and v_2 in (2.10). With equation (2.11) we compute the constant $c_1(\boldsymbol{\xi})$, so that we can finally state the one-dimensional backward algorithm to solve the ODE (2.9), see Algorithm 2.

Algorithm 1 Integration algorithm

Input:	W	function values $\tilde{w}(\frac{j+0.5}{M}, \boldsymbol{\xi}) = w(\varphi(\frac{j+0.5}{M}), \boldsymbol{\xi})$, $j = 1, ..., M = 2N$
	Φ	function values $\varphi'(\frac{j+0.5}{M})$, $j = 1, ..., M = 2N$

$\hat{A} = \mathrm{FFT}(W \circ \Phi)$
for $k = -N, ..., N-1, k \neq 0$ **do**
$\hat{B}[k+N] = \frac{\mathrm{e}^{-\pi \mathrm{i} \frac{k}{M}}}{2\pi \mathrm{i} k M} \cdot \hat{A}[k+N]$
$\tilde{B}[k+N] = \mathrm{e}^{\pi \mathrm{i} \frac{k}{M}} \cdot \hat{B}[k+N]$
$B[k+N] = (-1)^k \tilde{B}[k+N]$
end for
$H = \mathrm{FFT}(\tilde{B}) - \mathrm{sum}(\hat{B}) \cdot \mathbb{1}$
$h = \mathrm{sum}(B) - \mathrm{sum}(\hat{B})$

Output:	H	function values $\tilde{h}(\frac{j+0.5}{M}, \boldsymbol{\xi}) = h(\varphi(\frac{j+0.5}{M}), \boldsymbol{\xi})$, cf. (2.16)
	h	function value $\tilde{h}(0.5, \boldsymbol{\xi}) = h(1, \boldsymbol{\xi})$, cf. (2.17)

Algorithm 2 One-dimensional backward algorithm

Input:	F	function values $\tilde{f}(\frac{j+0.5}{M}) = f(\varphi(\frac{j+0.5}{M}))$
	A	function values $\tilde{a}(\frac{j+0.5}{M}, \boldsymbol{\xi}) = a(\varphi(\frac{j+0.5}{M}), \boldsymbol{\xi})$
	Φ	function values $\varphi'(\frac{j+0.5}{M})$

$[Z, \sim] = \mathrm{integration}(F, \Phi)$, i.e., Algorithm 1 applied to F
$V_1 = -Z \oslash A$ [1] , $V_2 = -\mathbb{1} \oslash A$
$[I_1, i_1] = \mathrm{integration}(V_1, \Phi)$, i.e., Algorithm 1 applied to V_1
$[I_2, i_2] = \mathrm{integration}(V_2, \Phi)$, i.e., Algorithm 1 applied to V_2
$U = I_1 - \frac{i_1}{i_2} \cdot I_2$

Output:	U	approximation of the function values $\tilde{u}(\frac{j+0.5}{M}, \boldsymbol{\xi}) = u(\varphi(\frac{j+0.5}{M}), \boldsymbol{\xi})$

So we end up with an approximation of the solution $u(\cdot, \boldsymbol{\xi})$ for the fixed $\boldsymbol{\xi}$ given on the grid

$$\left\{\varphi\left(\frac{j+0.5}{M}\right)\right\}_{j=0}^{M-1} = \left\{\varphi\left(\frac{j+0.5}{M}\right)\right\}_{j=0}^{N-1} =: \mathcal{T}_N.$$

Note, that, since φ is symmetric, the points for $j = N, ..., M-1$ are equal to the first N points. Therefore it is enough to use $j = 0, ..., N-1$. Further, our approximation fulfills the boundary conditions (2.9b) by construction, so we can even work on the larger grid $\mathcal{T}_N^0 := \mathcal{T}_N \cup \{-1, 1\}$.

[1] Here, $\oslash$ is the Hadamard (or component-wise) division for vectors and matrices.

Example 2.4
We consider the ODE (2.9) with right-hand side $f(x) = 10$ and the random coefficient $a(x, \boldsymbol{\xi}) = 1$ for all $x \in [-1, 1], \boldsymbol{\xi} \in D_{\boldsymbol{\xi}}$. The exact solution to this differential equation with the zero boundary conditions (2.9b) is

$$u(x, \boldsymbol{\xi}) = -5x^2 + 5 \qquad \forall \boldsymbol{\xi} \in D_{\boldsymbol{\xi}}.$$

Since every part of the ODE is independent of $\boldsymbol{\xi}$, we will neglect it for the moment. Now we want to solve this problem with Algorithm 2. To this end, we compare the exact values $u(x)$ with the values of our approximation of the solution $\hat{u}(x)$ on every grid point $x \in \mathcal{T}_N$ in terms of a p vector norm. So our error estimation is computed as

$$\mathrm{Err}_p(N) = \left(\frac{1}{N} \sum_{x \in \mathcal{T}_N} |\hat{u}(x) - u(x)|^p \right)^{\frac{1}{p}},$$

or, for $p = \infty$, as

$$\mathrm{Err}_\infty(N) = \max_{x \in \mathcal{T}_N} |\hat{u}(x) - u(x)|.$$

Figure 2.2 illustrates this error $\mathrm{Err}_p(N)$ for $p = 1, 2, \infty$ and the different transformation functions φ presented in section 2.2. Note, that due to the simple structure of the differential equation, the approximated solution already is exact even for small N when using the infinitely differentiable cosine transformation. □

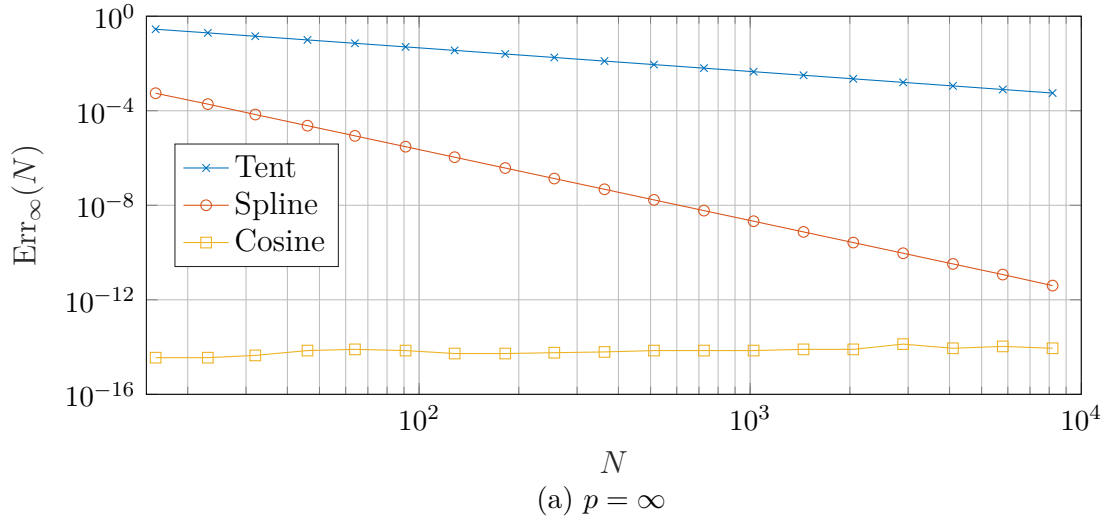
Example 2.5
We now consider a more advanced example from [4], i.e., the ODE (2.9) with right-hand side $f(x) = 10$ and the random coefficient

$$a(x, \boldsymbol{\xi}) = 4.3 + \sum_{j=1}^{10} \xi_{2j-1} \frac{\cos(j\pi x)}{j^2} + \xi_{2j} \frac{\sin(j\pi x)}{j^2},$$

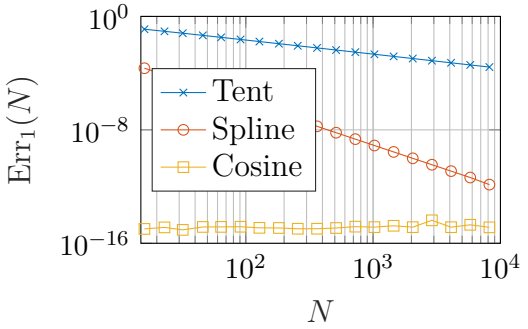
where $\boldsymbol{\xi}$ with $d_{\boldsymbol{\xi}} = 20$ are uniformly distributed, i.e., $\boldsymbol{\xi} \sim U([-1, 1]^{20})$. Further notes on this examples can be found in Section 2.6.
Again, we solve this problem with Algorithm 2 and compute the estimated error $\mathrm{Err}_p(N)$ on the grid $\mathcal{T}_N$, this time for a fixed $\boldsymbol{\xi}$. Further, we average this for a few random $\boldsymbol{\xi}^i \in D_{\boldsymbol{\xi}}$, $i = 1, ..., n_{\text{test}}$. So our error estimation is computed as

$$\mathrm{Err}_p(N) = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \begin{cases} \left(\frac{1}{N} \sum_{x \in \mathcal{T}_N} \left| \hat{u}(x, \boldsymbol{\xi}^i) - u(x, \boldsymbol{\xi}^i) \right|^p \right)^{\frac{1}{p}} & \text{if } 1 \le p < \infty, \\ \max_{x \in \mathcal{T}_N} \left| \hat{u}(x, \boldsymbol{\xi}^i) - u(x, \boldsymbol{\xi}^i) \right| & \text{if } p = \infty, \end{cases}$$

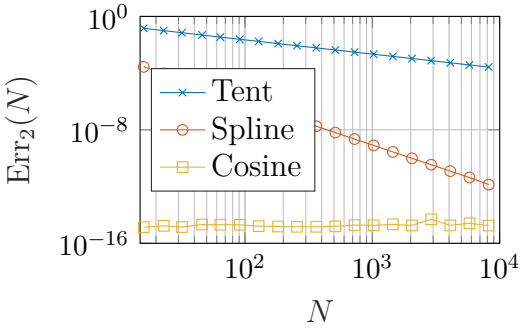
this time. Again, Figure 2.3 illustrates the estimated errors $\mathrm{Err}_p(N)$ with $n_{\text{test}} = 50$ for $p = 1, 2, \infty$ and the different transformation functions φ. □



(a) $p = \infty$



(b) $p = 1$



(c) $p = 2$

Figure 2.2: Error estimates $\mathrm{Err}_p(N)$ for Example 2.4 for the different transformation functions φ from section 2.2.

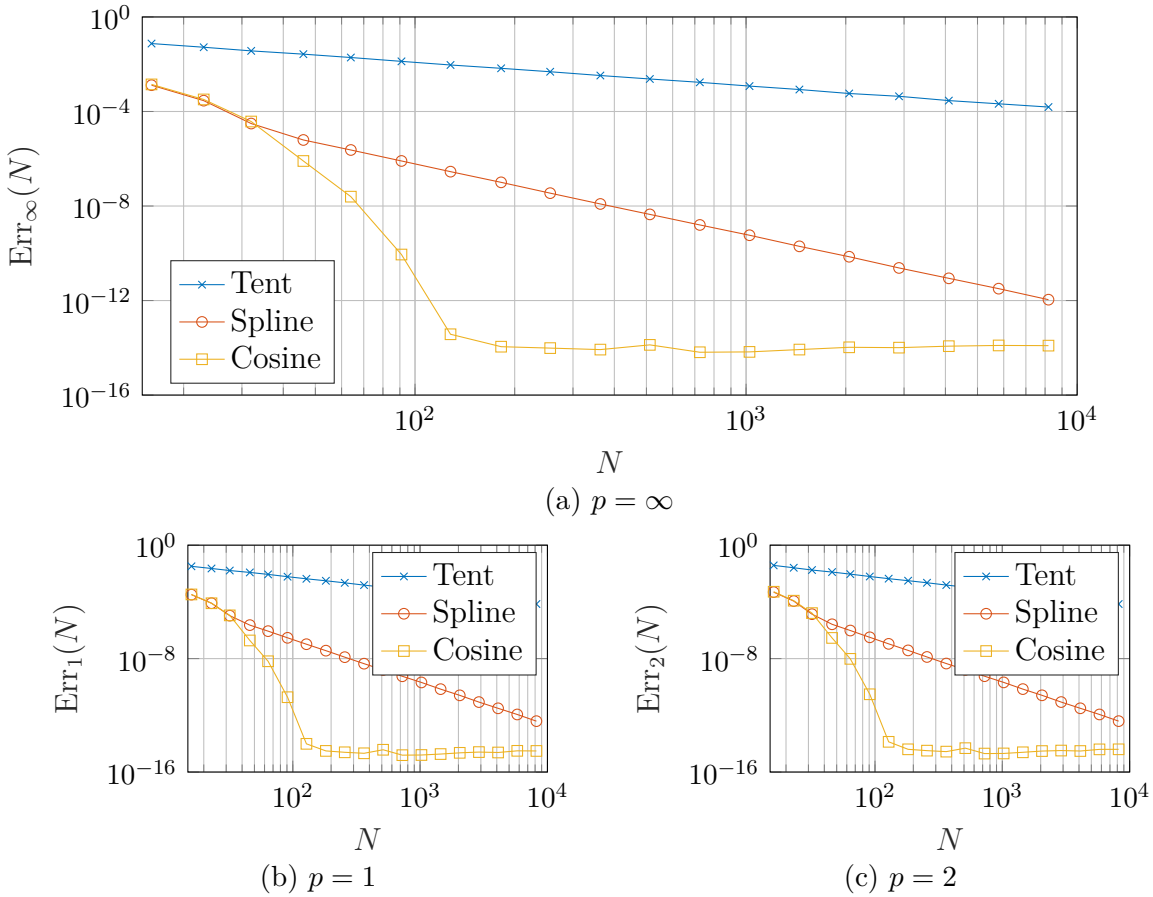
Remember, that for now, we want to evaluate this solution at a certain point x_0. If the given point x_0 already is a part of this grid, we just extract the function value $u(x_0, \boldsymbol{\xi})$ and are done. Unfortunately, most of the time this will not be the case. If x_0 lies in between the grid points x_l and x_r, we will use a straight forward linear interpolation of the form $y = mx + n$, resulting in

$$u(x_0, \boldsymbol{\xi}) = \underbrace{\frac{u(x_r, \boldsymbol{\xi}) - u(x_l, \boldsymbol{\xi})}{x_r - x_l}}_{m} x_0 + \underbrace{\left(u(x_l, \boldsymbol{\xi}) - \frac{u(x_r, \boldsymbol{\xi}) - u(x_l, \boldsymbol{\xi})}{x_r - x_l} x_l \right)}_{n}.$$

Note, that since we can use the boundary conditions (2.9b), there will always be grid points x_l, x_r to the left and right of x_0.

2.3.2 Independent numerical solvers

The second way to solve the differential equation (2.9) we will discuss is by simply using an already implemented numerical solver. We will treat the used solver as a black box



(a) $p = \infty$

(b) $p = 1$

(c) $p = 2$

Figure 2.3: Error estimations $\mathrm{Err}_p(N)$ for Example 2.5 for the different transformation functions φ from section 2.2 and $n_{\text{test}} = 50$.

algorithm. If $u(x_0, \boldsymbol{\xi})$ for the fixed x_0 is not provided by the ODE solver, we will use an easy interpolation using surrounding points to approximate it.
Although this method will probably be slower than the Fourier approach, it is easier to adapt when it comes to other kinds of ODEs or when moving forward to PDEs. We only need to choose a suitable solver for the specific differential equation and extract the desired function value of u.

Remark 2.1
Later on, we will use the function *bvp4c* provided by Matlab® to solve the ODE in our numerical tests. After inserting a system of first-order equations, the boundary conditions and an initial guess, the function approximates the solution on a grid. Further we access the required value $u(x_0, \boldsymbol{\xi})$ via interpolation with a polynomial of degree 3. We compute the interpolation by using the values of u and u' on the two surrounding grid points x_l and

x_r around x_0, which are all provided by the solver itself:

$$\begin{pmatrix} u(x_l, \boldsymbol{\xi}) \\ u'(x_l, \boldsymbol{\xi}) \\ u(x_r, \boldsymbol{\xi}) \\ u'(x_r, \boldsymbol{\xi}) \end{pmatrix} = \begin{pmatrix} x_l^3 & x_l^2 & x_l & 1 \\ 3x_l^2 & 2x_l & 1 & 0 \\ x_r^3 & x_r^2 & x_r & 1 \\ 3x_r^2 & 2x_r & 1 & 0 \end{pmatrix} \begin{pmatrix} c_3 \\ c_2 \\ c_1 \\ c_0 \end{pmatrix}$$

$$u(x_0, \boldsymbol{\xi}) = c_3 x_0^3 + c_2 x_0^2 + c_1 x_0 + c_0$$

□

2.4 The uniform sFFT

The methods from the previous section allow us to compute approximate solutions of the ODE for fixed $\boldsymbol{\xi}$. As described in Section 2.1, we can use these values $u(x_0, \boldsymbol{\xi})$ for a chosen x_0 as sampling values to recover an approximation of the function $u(x_0, \boldsymbol{\xi})$ w.r.t. the random variable $\boldsymbol{\xi}$. This can be realized by using the aforementioned so-called sFFT as presented in [20].
Using this approach, the sFFT will compute an index set I_{x_0} and the approximations $\hat{u}_{x_0,k}$ of the Fourier coefficients $c_k(\tilde{u}(x_0, \cdot))$ and with formula (2.5) we can evaluate the function $u(x_0, \boldsymbol{\xi})$ for a given x_0. However, the function is defined for every $x \in [-1, 1]$ and therefore we want to consider a set $\mathcal{T}_G$ of points $x_g \in (-1, 1)$ with $g = 1, ..., G$. So we need to call the whole algorithm for each value x_g, which significantly increases the number of used samples. Since sampling one time means solving a differential equation, a further increase in the number of samples seems to be highly inefficient. To get a better understanding, why this is a huge problem in our case and how to handle this, we roughly outline the dimension-incremental method and possible sampling strategies.

2.4.1 The dimension-incremental method

This dimension-incremental approach was presented in [20] and proceeds similarly as a dimension-incremental method for anharmonic trigonometric polynomials based on Prony's method in [19].
We consider a given search domain $\Gamma \subset \mathbb{Z}^d$, $|\Gamma| < \infty$, and our aim is to determine the non-zero Fourier coefficients $\hat{p}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathrm{I}$, of a multivariate trigonometric polynomial p and the unknown frequency set $\mathrm{I} \in \Gamma$, $|\mathrm{I}| \ll |\Gamma|$, based on samples.
First, we introduce some further notation as in [20] and assume for this explanation, that we are dealing with a multivariate trigonometric polynomial p. We denote the projection of a frequency $\boldsymbol{k} \coloneqq (k_1, ..., k_d)^\top \in \mathbb{Z}^d$ to the components $\boldsymbol{i} \coloneqq (i_i, ..., i_m) \in \{\iota \in \{1, ..., d\}^m : \iota_t \neq \iota_{t'} \text{ for } t \neq t'\}$ by $\mathcal{P}_{\boldsymbol{i}}(\boldsymbol{k}) \coloneqq (k_{i_1}, ..., k_{i_m})^\top \in \mathbb{Z}^m$. Correspondingly, we define the projection of a frequency set $\mathrm{I} \subset \mathbb{Z}^d$ to the components $\boldsymbol{i}$ by $\mathcal{P}_{\boldsymbol{i}}(\mathrm{I}) \coloneqq \{(k_{i_1}, ..., k_{i_m}) : \boldsymbol{k} \in \mathrm{I}\}$. Using these notations, the general approach is the following:

1. Determine the first components of the unknown frequency set, i.e., determine a set $\mathrm{I}^{(1)} \subseteq \mathcal{P}_1(\Gamma)$ which should be identical to the projection $\mathcal{P}_1(\operatorname{supp} \hat{p})$ or contain this projection, $\mathrm{I}^{(1)} \supseteq \mathcal{P}_1(\operatorname{supp} \hat{p})$.

2. For dimension increment step $t = 2, ..., d$, i.e., for each additional dimension:

   (a) Determine the t-th components of the unknown frequency set, i.e., determine a set $\mathrm{I}^{(t)} \subseteq \mathcal{P}_t(\Gamma)$ which should be identical to the projection $\mathcal{P}_t(\operatorname{supp} \hat{p})$ or contain this projection, $\mathrm{I}^{(t)} \supseteq \mathcal{P}_t(\operatorname{supp} \hat{p})$.

   (b) Determine a suitable sampling set $\mathcal{X}^{(1,\ldots,t)} \subset \mathbb{T}^d$, $|\mathcal{X}^{(1,\ldots,t)}| \ll |\Gamma|$, which allows to detect those frequencies from the set $(\mathrm{I}^{(1,\ldots,t-1)} \times \mathrm{I}^{(t)}) \cap \mathcal{P}_{(1,\ldots,t)}(\Gamma)$ belonging to non-zero Fourier coefficients $\hat{p}_{\boldsymbol{k}}$.

   (c) Sample the trigonometric polynomial p along the nodes of the sampling set $\mathcal{X}^{(1,\ldots,t)}$.

   (d) Compute the Fourier coefficients $\tilde{\hat{p}}_{(1,\ldots,t),\boldsymbol{k}}$, $\boldsymbol{k} \in (\mathrm{I}^{(1,\ldots,t-1)} \times \mathrm{I}^{(t)}) \cap \mathcal{P}_{(1,\ldots,t)}(\Gamma)$.

   (e) Determine the non-zero Fourier coefficients from $\tilde{\hat{p}}_{(1,\ldots,t),\boldsymbol{k}}$, $\boldsymbol{k} \in (\mathrm{I}^{(1,\ldots,t-1)} \times \mathrm{I}^{(t)}) \cap \mathcal{P}_{(1,\ldots,t)}(\Gamma)$ and obtain the set $\mathrm{I}^{(1,\ldots,t)}$ of detected frequencies. The $\mathrm{I}^{(1,\ldots,t)}$ index set should be equal to the projection $\mathcal{P}_{(1,\ldots,t)}(\operatorname{supp} \hat{p})$.

3. Use the set $\mathrm{I}^{(1,\ldots,d)}$ and the computed Fourier coefficients $\tilde{\hat{p}}_{(1,\ldots,t),\boldsymbol{k}}$, $\boldsymbol{k} \in \mathrm{I}^{(1,\ldots,d)}$ as an approximation for the support $\operatorname{supp} \hat{p}$ and the Fourier coefficients $\hat{p}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \operatorname{supp} \hat{p}$.

Note, that this method can also be used for the computation of the approximately largest Fourier coefficients $\hat{f}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathrm{I}$, of a function f with suitable thresholding techniques.
The proposed approach includes the construction of a suitable sampling set in step 2b and the computation of (projected) Fourier coefficients in step 2d. Of course, there exist different methods for the realization of these steps. The algorithm should work with any sampling method, which reliably computes Fourier coefficients on a given index set. Preferable sampling sets combine the four properties:

- low oversampling factors,
- stability and thus reliability,
- efficient construction methods,
- fast Fourier transform algorithms.

In the next sections, we briefly present three different approaches stated in [15, 16, 20] for the realization of these steps based on spatial discretizations called rank-1 lattices. Further explanations on these sampling sets as well as their behavior when used in the dimension-incremental approach can be found in the referred papers.

2.4.2 Sampling strategies based on rank-1 lattices

We follow the main lines in [20] in order to present the definition of rank-1 lattices and their use as spatial discretizations when reconstructing multivariate trigonometric polynomials. As discussed in [14], we are able to exactly reconstruct the Fourier coefficients $\hat{p}_{\boldsymbol{k}}, \boldsymbol{k} \in \mathrm{I}$, of an arbitrarily chosen trigonometric polynomial $p(\boldsymbol{x}) \coloneqq \sum_{\boldsymbol{k}\in\mathrm{I}} \hat{p}_{\boldsymbol{k}}\, \mathrm{e}^{2\pi\mathrm{i}\boldsymbol{k}\cdot\boldsymbol{x}}$ with frequencies supported on a given index set $\mathrm{I} \subset \mathbb{Z}^d$ from sampling values $p(\boldsymbol{x}_j)$. Here, the sampling nodes $\boldsymbol{x}_j$, $j = 0, ..., M-1$, are the nodes of a rank-1 lattice

$$\Lambda(\boldsymbol{z}, M) \coloneqq \left\{ \frac{j}{M}\boldsymbol{z} \bmod \mathbf{1} : \, j = 0, ..., M-1 \right\} \tag{2.18}$$

with a so called generating vector $\boldsymbol{z} \in \mathbb{Z}^d$ and lattice size $M \in \mathbb{N}$.
Note, that formally the Fourier coefficients $\hat{p}_{\boldsymbol{k}}$ with frequency $\boldsymbol{k}$ are given by

$$\hat{p}_{\boldsymbol{k}} \coloneqq \int_{\boldsymbol{x}\in\mathbb{T}^d} p(\boldsymbol{x})\, \mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}\cdot\boldsymbol{x}} \mathrm{d}\boldsymbol{x}.$$

Applying the rank-1 lattice rule with rank-1 lattice $\Lambda(\boldsymbol{z}, M)$ as in (2.18) yields

$$\begin{aligned}
\hat{p}_{\boldsymbol{k}} &\approx \frac{1}{M} \sum_{j=0}^{M-1} p(\boldsymbol{x}_j)\, \mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}\cdot\boldsymbol{x}_j} = \frac{1}{M} \sum_{j=0}^{M-1} p\left(\frac{j}{M}\boldsymbol{z}\right) \mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}\cdot\boldsymbol{z}/M} \\
&= \frac{1}{M} \sum_{j=0}^{M-1} \sum_{\boldsymbol{k}'\in\mathrm{I}} \hat{p}_{\boldsymbol{k}'}\, \mathrm{e}^{2\pi\mathrm{i}j\boldsymbol{k}'\cdot\boldsymbol{z}/M}\, \mathrm{e}^{-2\pi\mathrm{i}\boldsymbol{k}\cdot\boldsymbol{z}/M} \\
&= \sum_{\boldsymbol{k}'\in\mathrm{I}} \hat{p}_{\boldsymbol{k}'} \frac{1}{M} \sum_{j=0}^{M-1} \mathrm{e}^{2\pi\mathrm{i}j(\boldsymbol{k}'-\boldsymbol{k})\cdot\boldsymbol{z}/M} \\
&= \sum_{\boldsymbol{k}'\in\mathrm{I}} \hat{p}_{\boldsymbol{k}'}\, \delta_0((\boldsymbol{k}'-\boldsymbol{k})\cdot\boldsymbol{z} \bmod M).
\end{aligned}$$

This means, the rank-1 lattice rule is exact if and only if

$$\boldsymbol{k}\cdot\boldsymbol{z} \not\equiv \boldsymbol{k}'\cdot\boldsymbol{z} \,(\mathrm{mod}\ M) \quad \forall \boldsymbol{k}, \boldsymbol{k}' \in \mathrm{I}, \boldsymbol{k} \neq \boldsymbol{k}'.$$

Introducing the difference set $\mathcal{D}(\mathrm{I}) \coloneqq \{\boldsymbol{k} - \boldsymbol{k}' : \boldsymbol{k}, \boldsymbol{k}' \in \mathrm{I}\}$ for the frequency set I, we can rewrite this condition as

$$\boldsymbol{m}\cdot\boldsymbol{z} \not\equiv 0 \,(\mathrm{mod}\ M) \quad \forall \boldsymbol{m} \in \mathcal{D}(\mathrm{I}) \setminus \{0\},$$

which we call reconstruction property from now on. A rank-1 lattice $\Lambda(\boldsymbol{z}, M)$ will be called reconstructing rank-1 lattice $\Lambda(\boldsymbol{z}, M, \mathrm{I})$ for the frequency set I, if the reconstruction property is fulfilled.
So while we can evaluate an arbitrarily chosen multivariate trigonometric polynomial p with frequencies supported on the frequency set I at all nodes of an arbitrary rank-1 lattice $\Lambda(\boldsymbol{z}, M)$, the unique reconstruction of the Fourier coefficients $\hat{p}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathrm{I}$, of $\boldsymbol{p}$ requires sampling values on a reconstructing rank-1 lattice $\Lambda(\boldsymbol{z}, M, \mathrm{I})$. Both computations can

be efficiently done in $\mathcal{O}(M \log M + d|\mathrm{I}|)$ arithmetic operations, i.e., we only need a one-dimensional fast Fourier transform and the scalar products $\boldsymbol{k} \cdot \boldsymbol{z}$, $\boldsymbol{k} \in \mathrm{I}$, to rearrange the results.
In [20, Theorem 2.1] it is shown that for a given frequency set I there always exists a suitable prime rank-1 lattice size M and generating vector $\boldsymbol{z}$, such that $\Lambda(\boldsymbol{z}, M) = \Lambda(\boldsymbol{z}, M, \mathrm{I})$ is a reconstructing rank-1 lattice for I. Further, there exist upper bounds on the size of the rank-1 lattice M and the arithmetic operations needed to construct the generating vector $\boldsymbol{z}$.
While single rank-1 lattices provide a perfectly stable, reliable and efficient way to reconstruct Fourier coefficients, the bottleneck of this approach is the relatively large number of samples M up to $M \gtrsim |\mathrm{I}|^2$ and the expensive construction of the generating vector $\boldsymbol{z}$. In [16] so-called multiple rank-1 lattices

$$\Lambda(\boldsymbol{z}_1, M_1, ..., \boldsymbol{z}_L, M_L) := \bigcup_{\ell=1}^{L} \Lambda(\boldsymbol{z}_\ell, M_\ell),$$

consisting of L rank-1 lattices $\Lambda(\boldsymbol{z}_\ell, M_\ell)$, $\ell = 1, ..., L$, are used as spatial discretizations. Again, a reconstruction property is needed to ensure, that the multiple rank-1 lattice allows for the reconstruction of all Fourier coefficients $\hat{p}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathrm{I}$. The reconstruction property we will use reads as

$$\boldsymbol{k} \cdot \boldsymbol{z}_\ell \not\equiv \boldsymbol{k}' \cdot \boldsymbol{z}_\ell \mod M_\ell \quad \text{for all } \boldsymbol{k} \in \mathrm{I}_\ell,\ \boldsymbol{k}' \in \mathrm{I},\ \boldsymbol{k} \neq \boldsymbol{k}',\ \cup_{\ell=1}^{L} \mathrm{I}_\ell = \mathrm{I}.$$

The corresponding approach chooses L rank-1 lattices with size $M_\ell \sim |\mathrm{I}|$ and random generating vectors $\boldsymbol{z}_\ell \in [1, M_\ell - 1]^d \cap \mathbb{Z}^d$. Then we reconstruct every Fourier coefficient $\hat{p}_{\boldsymbol{k}}$ belonging to a non-aliasing frequency $\boldsymbol{k} \in \mathrm{I}_\ell$ for each of the L rank-1 lattices. With a certain success probability $1 - \delta$, each frequency $\boldsymbol{k} \in \mathrm{I}$ is non-aliasing in at least one rank-1 lattice $\Lambda(\boldsymbol{z}_\ell, M_\ell)$ and therefore every Fourier coefficient $\hat{p}_{\boldsymbol{k}}$, $\boldsymbol{k} \in \mathrm{I}$, can be reconstructed.
The number of required rank-1 lattices L can be bounded by $\mathcal{O}(\log |\mathrm{I}| - \log \delta)$ with the failure probability $\delta \in (0, 1)$. Therefore, the total number M of samples used in this approach is bounded by $\mathcal{O}(|\mathrm{I}|(\log |\mathrm{I}| - \log \delta))$. Further, the complexity of the construction of the multiple rank-1 lattice and the reconstruction of the Fourier coefficients reads as $\mathcal{O}(M \log M + L(d + \log |\mathrm{I}|)|\mathrm{I}|)$, cf. [16]. Note, that these sample and arithmetic complexities are a lot better than for single rank-1 lattices, since $M \lesssim |\mathrm{I}| \log |\mathrm{I}|$ holds for a fixed failure probability.
The third approach we want to consider uses random rank-1 lattices to reduce the number of samples even more. This method was stated in [15] and assumes, that there are only a few active frequencies $\boldsymbol{k} \in \mathrm{I}^*$ in the index set I, while all the other Fourier coefficients $\hat{p}_{\boldsymbol{k}}, \boldsymbol{k} \in \mathrm{I} \setminus \mathrm{I}^*$ are zero. Therefore, it only uses L rank-1 lattices with size $M_\ell \sim |\mathrm{I}^*|$ and randomly chosen generating vectors $\boldsymbol{z}_\ell$ for the reconstruction. Since there will occur aliasing effects, the algorithm counts for fixed frequency $\boldsymbol{k} \in \mathrm{I}$ how many of the computed coefficients $\hat{p}_{\boldsymbol{k}}^{(\ell)}$, $\ell = 1, ..., L$ are non-zero. If the count is above a threshold νL, ν typically chosen to be $1/2$, the frequency $\boldsymbol{k}$ is kept, as it is likely to be part of the active frequencies $\boldsymbol{k} \in \mathrm{I}^*$. If the fraction is below this threshold, one discards the frequency $\boldsymbol{k}$ since the few

	samples	arithmetic operations
single R1L	$\mathcal{O}(dr^3s^2N)$	$\mathcal{O}(dr^3s^3 + dr^3s^2N \log^{\mathcal{O}(1)}(...))$
multiple R1L	$\mathcal{O}(dr^2sN \log^{\mathcal{O}(1)}(...))$	$\mathcal{O}(d^2r^2sN \log^{\mathcal{O}(1)}(...))$
random R1L	$\mathcal{O}(drs \log^{\mathcal{O}(1)}(...))$	$\mathcal{O}(d^2rsN \log^{\mathcal{O}(1)}(...))$

Table 1: Sampling and arithmetic complexities of the sFFT approach when using different sampling strategies based on rank-1 lattices.

non-zero $\hat{p}_{\boldsymbol{k}}^{(\ell)}$ most likely are aliasing effects that show up. Choosing L odd and using the threshold $\nu = 1/2$ allows for the additional computation of the unknown Fourier coefficients by

$$\hat{p}_{\boldsymbol{k}} := \text{median}\left\{\text{Re}\left(\hat{p}_{\boldsymbol{k}}^{(\ell)}\right) : \ell = 1, ..., L\right\} + \mathrm{i} \cdot \text{median}\left\{\text{Im}\left(\hat{p}_{\boldsymbol{k}}^{(\ell)}\right) : \ell = 1, ..., L\right\}$$

Since the number of rank-1 lattices L again behaves like $\mathcal{O}(\log|\mathrm{I}| - \log\delta)$ for the failure probability δ, we end up with a total number of samples $M \lesssim \mathcal{O}(|\mathrm{I}^*|(\log|\mathrm{I}| - \log\delta))$. The computational complexity $\mathcal{O}(M \log M + Ld|\mathrm{I}|)$ is similar to the complexity for multiple rank-1 lattices, but also benefits from the lower number M of samples used.
Overall, rank-1 lattices provide efficient and reliable sampling strategies which can be used in the dimension-incremental method. While single rank-1 lattices are perfectly stable and do not depend on a failure probability, multiple and random rank-1 lattices may reduce the number of samples and arithmetic operations drastically. Therefore, we will consider all of these three different approaches in our numerical experiments. A short and simplified summary of the sampling and arithmetic complexities for the sFFT using the different sampling strategies is given in Table 1.
Up to now, we have an algorithm that computes an approximation of the solution $u(x_g, \boldsymbol{\xi})$ for a single x_g. Considering a whole set of points $x_g \in \mathcal{T}_G$, we have to call the existing method G times, which ends up with unnecessary many samples. Therefore, we now modify the dimension-incremental method, such that we can work on the grid $\mathcal{T}_G$ and one call of the algorithm computes approximations of all the Fourier coefficients $c_k(\tilde{u}(x_g, \cdot))$ for each $g = 1, ..., G$, including a clever usage of the sampling nodes $\boldsymbol{\xi}$.

2.4.3 Modifying the sFFT

As we saw in the previous section, the dimension-incremental method based on rank-1 lattice sampling provides an efficient and reliable way to approximate high-dimensional functions based on the fast Fourier transform. Especially the sampling complexity, which is particularly important in our application, behaves way better than for other possible methods, cf. [15, Table 1.1].
Matching our expectations, the computation time of the sampling step 2c still outweigh all the other steps of the sFFT by a lot already for small input parameters as Table 2 indicates. Especially, when using the slower numerical solver, the large number of sampling points leads to a significant increase in computation time. The following modifications provide

Steps	1	2a	2b	2c	total	# of samples
numerical solver	26.4512	2.7511	3843.9731	0.0919	3873.5796	367835
Fourier solver ($N = 2^8$, φ: spline)	55.8121	3.7617	948.3809	0.1406	1008.3482	342850

Table 2: Computation time of the sFFT steps for $\Gamma = [-8, 8]^{20}, s = 100, r = 5$, using single rank-1 lattices and different differential equation solvers, in seconds, averaged over 20 runs each.

a possibility to compute approximations of the Fourier coefficients $c_k(\tilde{u}(x_g, \cdot))$ for each $g = 1, ..., G$ without applying the sFFT algorithm explained above for each x_g separately and therefore without increasing the number of used samples by the factor G.
We force the dimension-incremental method to select a frequency set I containing the s approximately largest Fourier coefficients $c_k(\tilde{u}(x, \cdot))$ for each x_g in the equidistant grid $\mathcal{T}_G$. Therefore, we compute the set of detected frequencies $\mathrm{I}_{x_g}^{(1,...,t)}$ for each x_g in each dimension-increment t, but after that we form the union of these sets $\bigcup_{g=1}^{G} \mathrm{I}_{x_g}^{(1,...,t)}$, which will be the set of detected frequencies $\mathrm{I}^{(1,...,t)}$ that is given to the next dimension-incremental step. So we start each iteration with a larger frequency candidate set $(\mathrm{I}^{(1,...,t-1)} \times \mathrm{I}^{(t)}) \cap \mathcal{P}_{(1,...,t)}(\Gamma)$, which is suitable for all x_g from $\mathcal{T}_G$. This way, the chosen sampling set $\mathcal{X}^{(1,...,t)}$ is the same for each x_g and we can take advantage of the fact, that our differential equation solvers from section 2.3 can evaluate their solutions $u(x, \boldsymbol{\xi})$ for a given $\boldsymbol{\xi}$ for multiple values of x on a grid in $[-1, 1]$. So we only need to solve the differential equation once for each sampling point $\boldsymbol{\xi}$ and still get all the sampling values $\tilde{u}(x_g, \boldsymbol{\xi})$ for all x_g in our grid $\mathcal{T}_G$. Note, that this also holds for the one-dimensional detections in steps 1 and 2a since the used sampling sets here do not depend on the particular function $u(x_g, \cdot)$ and therefore are the same for each x_g. Also note, that we will probably still need to interpolate since the solution grids from 2.3 might differ from the grid $\mathcal{T}_G$. Obviously, the larger candidate sets $(\mathrm{I}^{(1,...,t-1)} \times \mathrm{I}^{(t)}) \cap \mathcal{P}_{(1,...,t)}(\Gamma)$ will also result in larger sampling sets, but even if only a few indices of the detected frequencies $\mathrm{I}_{x_g}^{(1,...,t)}$ are the same for different $x_g \in \mathcal{T}_G$, we already safe time compared to dealing with slightly smaller candidate sets for each x_g separately. We could also think of further thresholding methods to cut the number of frequencies back to the sparsity s at the end of each dimension-incremental step or at least at the end of the whole algorithm. In this work, we will not do this, but take a look at the total number of frequencies in the output of the algorithm in relation to the sparsity s.
We will call this modified version of the sFFT the *uniform sFFT* or short *usFFT* from now on, where *uniform* is meant w.r.t. a discrete set of points, e.g., $\mathcal{T}_G$. The full method is stated in Algorithm 3. Note, that the notations in Algorithm 3 are taken from the original works on the sFFT [20], [16] and [15] to set the focus on the modifications and therefore slightly differ from our here used notations.

Algorithm 3 The uniform sFFT on a grid $\{\chi_g : g = 1, ..., G\}$

Input:	$\Gamma \subset \mathbb{Z}^d$	search space in frequency domain, candidate set for $I = \operatorname{supp} \hat{p}$
	$p(\circ, \circ)$	trigonometric polynomial p as black box (function handle) (Here: $p(\chi, \boldsymbol{x}) = u(x, \boldsymbol{\xi})$)
	$\mathcal{T}_G$	grid points χ_g, $g = 1, ..., G$
	$s, s_{\text{local}} \in \mathbb{N}$	sparsity parameter, $s \leq s_{\text{local}}$
	Algorithm A	**efficient algorithm** A that guarantees the identification of the frequency support of each s_{local}-sparse trigonometric polynomial w.h.p., cf. Section 2.4.2, and computes the Fourier coeffcients
	$\theta \in \mathbb{R}^+$	absolute threshold
	$r \in \mathbb{N}$	number of detection iterations

(Step 1 & 2a) [Single frequency component identification]

for $t := 1, \ldots, d$ **do**

Set $K_t := \max(\mathcal{P}_t(\Gamma)) - \min(\mathcal{P}_t(\Gamma)) + 1$, $I^{(t)} := \emptyset, I_g^{(t)} := \emptyset \ \forall g = 1, \ldots, G$.

for $i := 1, \ldots, r$ **do**

Choose $x'_j \in \mathbb{T}$, $j \in \{1, \ldots, d\} \setminus \{t\}$ uniformly at random.

Set $\boldsymbol{x}^{(\ell)} := \left(x_1^{(\ell)}, \ldots, x_d^{(\ell)}\right)^\top$, $x_j^{(\ell)} := \begin{cases} \ell/K_t, & j = t, \\ x'_j, & j \neq t, \end{cases}$ for all $\ell = 0, \ldots, K_t - 1$.

for $g := 1, \ldots, G$ **do**

Compute $\tilde{\hat{p}}_{t,k_t,g} := \frac{1}{K_t} \sum_{\ell=0}^{K_t-1} p\left(\chi_g, \boldsymbol{x}^{(\ell)}\right) \mathrm{e}^{-2\pi \mathrm{i} \ell k_t / K_t}$, $k_t \in \mathcal{P}_t(\Gamma)$, via FFT.

Set $I_g^{(t)} := I_g^{(t)} \cup \{k_t \in \mathcal{P}_t(\Gamma) \colon \tilde{\hat{p}}_{t,k_t,g}$ is among the largest s_{local} (in absolute value) elements of $\{\tilde{\hat{p}}_{t,j,g}\}_{j \in \mathcal{P}_t(\Gamma)}$ and $|\tilde{\hat{p}}_{t,k_t,g}| \geq \theta\}$.

end for g

Set $I^{(t)} := I^{(t)} \cup I_g^{(t)}$.

end for i

end for t

(Step 2) [Coupling frequency components identification]

for $t := 2, \ldots, d$ **do**

If $t < d$, set $\tilde{r} := r$ and $\tilde{s} := s_{\text{local}}$, otherwise $\tilde{r} := 1$ and $\tilde{s} := s$.

Set $I^{(1,\ldots,t)} := \emptyset$ and $I_g^{(1,\ldots,t)} := \emptyset \ \forall g = 1, \ldots, G$.

for $i := 1, \ldots, \tilde{r}$ **do**

Choose components $x'_{t+1}, \ldots, x'_d \in \mathbb{T}$ of sampling nodes uniformly at random.

(Step 2b)

Generate a sampling set $\mathcal{X} \subset \mathbb{T}^t$ for $J_t := (I^{(1,\ldots,t-1)} \times I^{(t)}) \cap \mathcal{P}_{(1,\ldots,t)}(\Gamma)$ that allows for the application of Algorithm A. Set $\mathcal{X}_{t,i} := \{\boldsymbol{x} := (\tilde{\boldsymbol{x}}, x'_{t+1}, \ldots, x'_d) \colon \tilde{\boldsymbol{x}} \in \mathcal{X}\} \subset \mathbb{T}^d$.

(Step 2c)

Sample p along the nodes of the sampling set $\mathcal{X}_{t,i}$ for every χ_g.

for $g := 1, \ldots, G$ **do**

(Step 2d)

Apply Algorithm A to obtain the support $\tilde{J}_{t,i.g} \subset J_t$, $|\tilde{J}_{t,i.g}| \leq \tilde{s}$, of frequencies belonging to the at most $\tilde{s}$ largest Fourier coefficients, each larger than θ in absolute value, using the sampling values $p(\chi_g, \boldsymbol{x}_j)$, $\boldsymbol{x}_j \in \mathcal{X}_{t,i}$.

Algorithm 3 continued.

(Step 2e)
Set $I_g^{(1,\dots,t)} := I_g^{(1,\dots,t)} \cup \tilde{J}_{t,i,g}$.
end for g
Set $I^{(1,\dots,t)} := I^{(1,\dots,t)} \cup I_g^{(1,\dots,t)}$.
end for i
end for t
(Step 3) [Computation of Fourier coefficients]
Generate a sampling set $\mathcal{X} \subset \mathbb{T}^d$ for $I^{(1,\dots,d)}$ such that the corresponding Fourier matrix $\boldsymbol{A}(\mathcal{X}, I^{(1,\dots,d)})$ is of full column rank and its pseudoinverse can be applied **efficiently**.
for $g := 1, \dots, G$ **do**
Compute the corresponding Fourier coefficients $\left(\tilde{\hat{p}}_{(1,\dots,d),\boldsymbol{k},g}\right)_{\boldsymbol{k}\in I^{(1,\dots,d)}}$.
end for g
Set $\tilde{I} := I^{(1,\dots,d)}$
Output: $\tilde{I} \subset \Gamma \subset \mathbb{Z}^d$ index set of detected frequencies
$\tilde{\hat{\boldsymbol{p}}}_g \in \mathbb{C}^{|\tilde{I}|}$ corresponding Fourier coefficients, $|\tilde{\hat{p}}_{(1,\dots,d),\boldsymbol{k},g}| \geq \theta$ for all χ_g

2.5 Expectation value of the approximation

With Algorithm 3 we can compute an approximated solution $\breve{u}$ of the ODE (2.1). Often, we are also interested in further quantities of interest as for example the nth moments of the solution. A general approach on the computation of these quantities can be found in [4]. In this work, we only consider the expectation value

$$\mathbb{E}(\breve{u}(x,\cdot)) \coloneqq \int_{D_{\boldsymbol{\xi}}} \breve{u}(x,\boldsymbol{\xi})\,\mathrm{d}\mu(\boldsymbol{\xi}) = \int_{D_{\boldsymbol{\xi}}} \breve{u}(x,\boldsymbol{\xi})\,p(\boldsymbol{\xi})\mathrm{d}\boldsymbol{\xi},$$

where p is the probability density function of the random variable vector $\boldsymbol{\xi}$.
The evaluation of the formula above using our approximation $\breve{u}$ obviously depends on the used periodization φ and the distribution of the random variable $\boldsymbol{\xi}$. In Section 2.6, we will only work with uniformly distributed $\boldsymbol{\xi}$, i.e., $\boldsymbol{\xi} \sim U([-1,1]^{d_{\boldsymbol{\xi}}})$. Further, we will only use the tent transform, cf. Example 2.1, as periodization function on our random variables $\boldsymbol{\xi}$. This allows us to realize the computation of the expectation value in an easy way, following the main idea in [3]. As stated in Section 2.1, our approximation $\breve{u}(x_0,\boldsymbol{\xi})$ at a fixed point x_0 reads as

$$\begin{aligned}\breve{u}(x_0,\boldsymbol{\xi}) \coloneqq \hat{S}_{\mathrm{I}_{x_0}}[u(x_0,\cdot)](\boldsymbol{\xi}) &= \hat{S}_{\mathrm{I}_{x_0}}[\tilde{u}(x_0,\cdot)](\varphi^{-1}(\boldsymbol{\xi})) \\ &= \sum_{\boldsymbol{k}\in\mathrm{I}_{x_0}} \hat{u}_{x_0,\boldsymbol{k}}\,\mathrm{e}^{2\pi\mathrm{i}\boldsymbol{k}\cdot(\varphi^{-1}(\boldsymbol{\xi}))} \qquad \forall\,\boldsymbol{\xi}\in D_{\boldsymbol{\xi}} = [-1,1]^{d_{\boldsymbol{\xi}}}.\end{aligned}$$

Using the inverse of the tent transform φ^{-1} w.r.t. the interval $[0, 0.5]$, we get

$$\begin{aligned}\breve{u}(x_0, \boldsymbol{\xi}) &= \sum_{\boldsymbol{k} \in \mathrm{I}_{x_0}} \hat{u}_{x_0,\boldsymbol{k}} \, \mathrm{e}^{2\pi \mathrm{i} \boldsymbol{k} \cdot \frac{\boldsymbol{\xi}+1}{4}} \\ &= \sum_{\boldsymbol{k} \in \mathrm{I}_{x_0}} \hat{u}_{x_0,\boldsymbol{k}} \, \mathrm{e}^{\pi \mathrm{i} \boldsymbol{k} \cdot \frac{\boldsymbol{\xi}+1}{2}} \qquad \forall \, \boldsymbol{\xi} \in D_{\boldsymbol{\xi}} = [-1, 1]^{d_{\boldsymbol{\xi}}}.\end{aligned}$$

Therefore and since the probability density function of the uniform distribution is just a constant, we have

$$\begin{aligned}\mathbb{E}(\breve{u}(x_0, \cdot)) &= 2^{-d_{\boldsymbol{\xi}}} \int_{[-1,1]^{d_{\boldsymbol{\xi}}}} \breve{u}(x_0, \boldsymbol{\xi}) \, \mathrm{d}\boldsymbol{\xi} \\ &= 2^{-d_{\boldsymbol{\xi}}} \int_{[-1,1]^{d_{\boldsymbol{\xi}}}} \sum_{\boldsymbol{k} \in \mathrm{I}_{x_0}} \hat{u}_{x_0,\boldsymbol{k}} \, \mathrm{e}^{\pi \mathrm{i} \boldsymbol{k} \cdot \frac{\boldsymbol{\xi}+1}{2}} \, \mathrm{d}\boldsymbol{\xi} \\ &= 2^{-d_{\boldsymbol{\xi}}} \sum_{\boldsymbol{k} \in \mathrm{I}_{x_0}} \hat{u}_{x_0,\boldsymbol{k}} \int_{[-1,1]^{d_{\boldsymbol{\xi}}}} \mathrm{e}^{\pi \mathrm{i} \boldsymbol{k} \cdot \frac{\boldsymbol{\xi}+1}{2}} \, \mathrm{d}\boldsymbol{\xi} \\ &= 2^{-d_{\boldsymbol{\xi}}} \sum_{\boldsymbol{k} \in \mathrm{I}_{x_0}} c_{\boldsymbol{k}} \, \hat{u}_{x_0,\boldsymbol{k}}, \end{aligned} \tag{2.19}$$

with

$$c_{\boldsymbol{k}} = \prod_{j=1}^{d_{\boldsymbol{\xi}}} c_{k_j} \qquad \text{and} \qquad c_{k_j} = \begin{cases} \frac{-4}{\pi \mathrm{i} k_j} & k_j \equiv 1 \mod 2 \\ 2 & k_j = 0 \\ 0 & \text{else.} \end{cases} \tag{2.20}$$

This formula now allows us to evaluate the expectation value of our approximated solution $\breve{u}$ by simply summing up the computed Fourier coefficients with a certain scaling. A simliar approach for the computation of the variance of our solution could be derived as in [3], but will not be considered here.

2.6 Numerical results

For the numerical tests on our algorithm, we use a differential equation from [4]. We consider the boundary-value problem

$$-\frac{\partial}{\partial x}\left(a(x, \boldsymbol{\xi}) \frac{\partial}{\partial x} u(x, \boldsymbol{\xi})\right) = 10 \qquad x \in [-1, 1], \, \boldsymbol{\xi} \in \bigtimes_{j=1}^{20} [-1, 1] \tag{2.21a}$$

$$u(-1, \boldsymbol{\xi}) = u(1, \boldsymbol{\xi}) = 0 \qquad \forall \, \boldsymbol{\xi} \in \bigtimes_{j=1}^{20} [-1, 1] \tag{2.21b}$$

with the random coefficient

$$a(x, \boldsymbol{\xi}) = a_0 + \sum_{j=1}^{d_{\boldsymbol{\xi}}/2} \xi_{2j-1} \frac{\cos(j\pi x)}{j^{\gamma}} + \xi_{2j} \frac{\sin(j\pi x)}{j^{\gamma}}, \tag{2.22}$$

ρ	N	s	θ	r
I	16	250	$1 \cdot 10^{-12}$	5
II	32	250	$1 \cdot 10^{-12}$	5
III	16	500	$1 \cdot 10^{-12}$	5
IV	32	500	$1 \cdot 10^{-12}$	5
V	64	500	$1 \cdot 10^{-12}$	5
VI	32	1000	$1 \cdot 10^{-12}$	5
VII	64	1000	$1 \cdot 10^{-12}$	5
VIII	32	2000	$1 \cdot 10^{-12}$	5
IX	64	2000	$1 \cdot 10^{-12}$	5

Table 3: Parameter settings for the numerical tests of Algorithm 3 with $d_{\boldsymbol{\xi}} = 20$.

where $\gamma \in \mathbb{R}, \gamma > 1, a_0 \in \mathbb{R}, a_0 > 2\zeta(\gamma), d_{\boldsymbol{\xi}} \in 2\mathbb{N}$, and ζ denotes the Riemann zeta function. Further, we also choose $a_0 = 4.3, \gamma = 2, d_{\boldsymbol{\xi}} = 20$ and $\boldsymbol{\xi}$ uniformly distributed, i.e., $\boldsymbol{\xi} \sim U([-1,1]^{d_{\boldsymbol{\xi}}})$. Note, that (2.2) is fulfilled, since $a \in [a_0 - 2\zeta(\gamma), a_0 + 2\zeta(\gamma)]$ and for our parameters this means $a \in [4.3 - \pi^2/3, 4.3 + \pi^2/3]$.
As already mentioned in Section 2.5, we only consider the tent transform as periodization function on the random variables $\boldsymbol{\xi}$ in our approach, because it is one of the easiest and straight forward periodizations one could think of. Further, the choice seems to be unfavorable in terms of approximation quality because of the lack of smoothness, such that other periodizations will probably lead to even smaller errors than the ones we will see in the following examples.
We will use the notation $\breve{u}^{\rho}_{\dagger}$ for our approximated solution, where $\dagger = sR1L, mR1L, rR1L$ characterizes the different kinds of our algorithm using single, multiple or random rank-1 lattices and ρ denotes the used uniform sFFT parameters, which can be found in Table 3. Note, that Algorithm 3 always works with the full grid $\hat{G}_N^{d_{\boldsymbol{\xi}}} \coloneqq \{-N, ..., N\}^{d_{\boldsymbol{\xi}}}$ as search space Γ.
Further, all of our numerical tests will consider the equidistant grid

$$\mathcal{T}_{199}^{+} \coloneqq \left\{ x_g = \frac{g}{100} - 1 \right\}_{g=0}^{200},$$

consisting of the 199 inner grid points $\mathcal{T}_{199}$ and the boundary points $x_0 = -1$ and $x_{200} = 1$.

Remark 2.2
One important difference mentioned in Section 2.4.3 is the number of Fourier coefficients in the output. While the sFFT will only output the sparsity s largest Fourier coefficients, our modification of this algorithm will output an unknown number of Fourier coefficients, but includes the sparsity s largest for each inner grid point $x_g \in \mathcal{T}_{199}$. Our numerical tests show, that the total number of Fourier coefficients in the output of the uniform sFFT is roughly $2s$, e.g., for sparsity $s = 500$ the algorithm detected amounts like $1013, 1021, 1027$

or 1033 Fourier coefficients. Some random tests for larger grid sizes like T_{999} also showed this behavior. Note, that for other input parameters than ours, significantly larger or smaller grid sizes or other differential equations, this behavior might not hold. □

First, we will now take a look at the number of samples used in our algorithm using the different parameters and rank-1 lattice approaches. Note, that the following Examples always work with the ODE solver given in Remark 2.1. The Fourier solver from Section 2.3.1 will be considered in Example 2.10 separately.

Example 2.6
Table 4 shows the number of used samples for the different input parameters ρ of our algorithm when using the different sampling strategies from Section 2.4.2 and the ODE solver from Section 2.3.2. Note, that these number of samples were summed up over all $d = 20$ dimensions and $r = 5$ detection iterations. Further, the amount of considered frequency candidates is also given. Since the frequency candidate set is the same for each detection iteration r, these are obviously not summed up over $r = 5$ but again over the $d = 20$ dimensions.
The number of frequency candidates is nearly the same for same parameters ρ when using different sampling strategies. This is not surprising, since it only depends on the size of the index sets of detected frequencies $\mathrm{I}^{(1,\dots,t-1)}$ and $\mathrm{I}^{(t)}$, which do not depend on the used sampling strategy, cf. Section 2.4.3 and Remark 2.2.
The number of used samples for multiple rank-1 lattices is larger than for single rank-1 lattices. This is, because our single rank-1 lattices probably behave way better than the worst case bounds given in Section 2.4.2. Also, the relative difference between single and multiple rank-1 lattices decays, when increasing the sparsity s and the extension of the full grid N. So for larger s and N single rank-1 lattices will probably use more samples, matching our expectations on the asymptotic behavior.
As expected by the results in Section 2.4.2, random rank-1 lattices seem to be superior in terms of used samples, especially when it comes to large extensions N. Doubling the extension N and therefore the number of frequency candidates, results in an increase in the amount of used samples by roughly the same factor for single and multiple rank-1 lattices. Random rank-1 lattices do not show this behavior, increasing the extension N only results in a small increase in the amount of sampling nodes used. We also see the better behavior of random rank-1 lattices for larger sparsities s compared to single rank-1 lattices. Therefore, the random rank-1 lattice approach really outperforms the other approaches especially for large input parameters ρ by a lot. □

Further, we want to test the precision of the uniform sFFT. To this end, as in [4], we first compare our approximations of the solution pointwise against a suitable approximation of the true solution of (2.21).

	I $s = 250, N = 16$	II $s = 250, N = 32$	III $s = 500, N = 16$	IV $s = 500, N = 32$	V $s = 500, N = 64$
single R1L					
samples	$5.35 \cdot 10^6$	$1.17 \cdot 10^7$	$1.35 \cdot 10^7$	$3.21 \cdot 10^7$	$6.91 \cdot 10^7$
freq. cand.	$2.83 \cdot 10^5$	$5.63 \cdot 10^5$	$5.63 \cdot 10^5$	$1.17 \cdot 10^6$	$2.28 \cdot 10^6$
multiple R1L					
samples	$2.27 \cdot 10^7$	$4.17 \cdot 10^7$	$4.74 \cdot 10^7$	$1.01 \cdot 10^8$	$1.87 \cdot 10^8$
freq. cand.	$2.96 \cdot 10^5$	$5.57 \cdot 10^5$	$5.74 \cdot 10^5$	$1.13 \cdot 10^6$	$2.20 \cdot 10^6$
random R1L					
samples	$6.14 \cdot 10^6$	$6.48 \cdot 10^6$	$1.29 \cdot 10^7$	$1.44 \cdot 10^7$	$1.43 \cdot 10^7$
freq. cand.	$2.97 \cdot 10^5$	$5.63 \cdot 10^5$	$5.87 \cdot 10^5$	$1.13 \cdot 10^6$	$2.21 \cdot 10^6$

	VI $s = 1000, N = 32$	VII $s = 1000, N = 64$	VIII $s = 2000, N = 32$	IX $s = 2000, N = 64$
single R1L				
samples	$9.78 \cdot 10^7$	$2.11 \cdot 10^8$	$3.00 \cdot 10^8$	$6.59 \cdot 10^8$
freq. cand.	$2.35 \cdot 10^6$	$4.87 \cdot 10^6$	$4.67 \cdot 10^6$	$9.40 \cdot 10^6$
multiple R1L				
samples	$2.14 \cdot 10^8$	$4.47 \cdot 10^8$	$4.85 \cdot 10^8$	$9.98 \cdot 10^8$
freq. cand.	$2.38 \cdot 10^6$	$4.64 \cdot 10^6$	$4.68 \cdot 10^6$	$9.46 \cdot 10^6$
random R1L				
samples	$2.88 \cdot 10^7$	$3.04 \cdot 10^7$	$5.94 \cdot 10^7$	$6.21 \cdot 10^7$
freq. cand.	$2.41 \cdot 10^6$	$4.87 \cdot 10^6$	$5.14 \cdot 10^6$	$1.04 \cdot 10^7$

Table 4: Number of used samples and considered frequency candidates over all dimension increments and detection iterations for different parameters with $d_{\boldsymbol{\xi}} = 20$.

Example 2.7
We compute an averaged error by calculating the pointwise difference of our approximated solution $\breve{u}^{\rho}_{\dagger}(x_g, \boldsymbol{\xi}^i)$ and a suitable approximation of the true solution $\breve{u}(x_g, \boldsymbol{\xi}^i)$ for each point $x_g \in \mathcal{T}^+_{199}$. The approximation $\breve{u}$ is computed as given in (2.10) via numerical integration using an relative error tolerance of 10^{-6}. So we end up with

$$\mathrm{Err}^{\rho}_{\dagger}(x_g) \coloneqq \frac{1}{n_{\mathrm{test}}} \sum_{i=1}^{n_{\mathrm{test}}} |\breve{u}(x_g, \boldsymbol{\xi}^i) - \breve{u}^{\rho}_{\dagger}(x_g, \boldsymbol{\xi}^i)|. \tag{2.23}$$

Figure 2.4 illustrates this averaged pointwise error for $n_{\mathrm{test}} = 10000$ when using single rank-1 lattices. The green line shows the averaged function values $\breve{u}(x_g, \boldsymbol{\xi}^i)$ for comparison, since we are only working with absolute errors. The shown parameter settings ρ use different sparsities s and the same extension $N = 32$.
The error is smaller for higher sparsities s and therefore more frequencies $\boldsymbol{k}$ and Fourier coefficients $c_{\boldsymbol{k}}(\tilde{u}(x_g, \cdot))$ used in the Fourier partial sum (2.5). The same holds for the shape

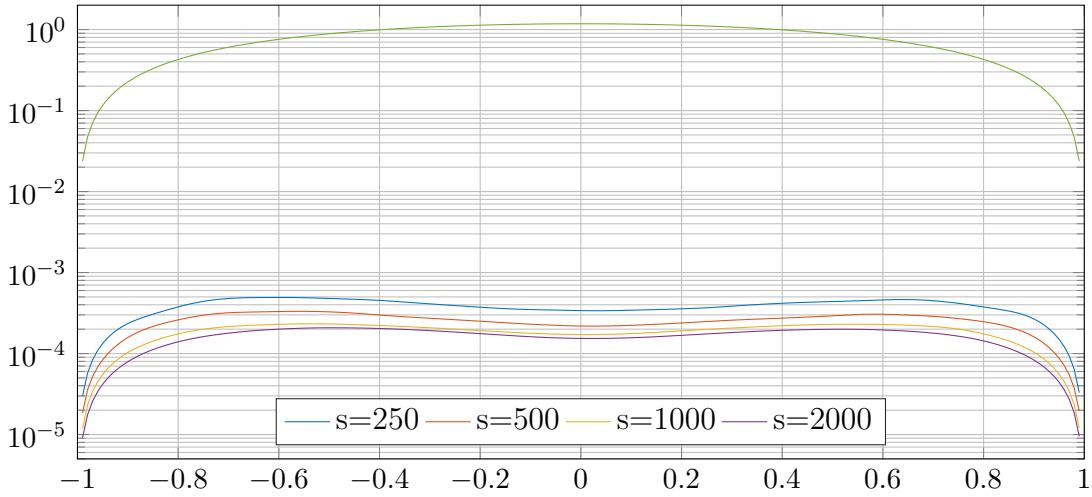
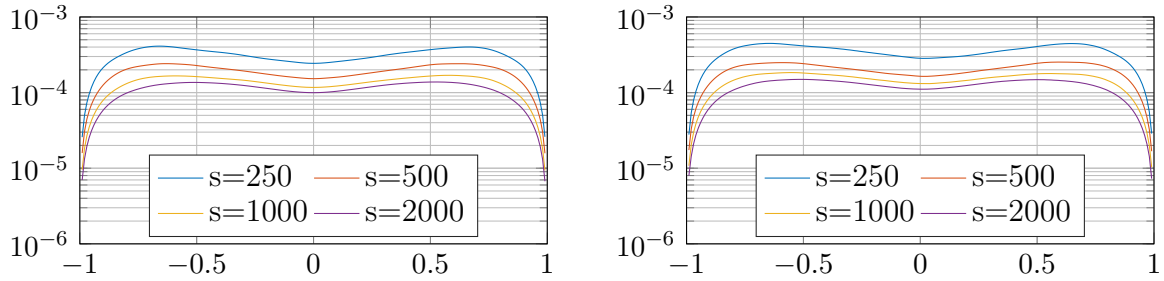


Figure 2.4: Averaged pointwise error $\mathrm{Err}^{\rho}_{\mathrm{sR1L}}$ with $n_{\mathrm{test}} = 10000$ when using single rank-1 lattices for parameter settings ρ = II, IV, VI, VIII, so with $N = 32$ and different sparsities s. The green line is the function value $\breve{u}(x_g, \boldsymbol{\xi}^i)$, averaged over the $\boldsymbol{\xi}^i$.

of the graph, the boundary conditions lead to a fast decay of the error when coming closer to the boundary.
The error decreases by a lot when using for example $s = 500$ instead of $s = 250$. This confirms, that for small sparsities s there are still large Fourier coefficients $c_{\boldsymbol{k}}$ the algorithm does not detect. On the other hand, for $s = 1000$ and $s = 2000$ we see, that even the sparsity increase by 1000 only results in a relatively small error decrease. Therefore, we expect, that we really detected the most important frequencies $\boldsymbol{k}$ in our search domain $\Gamma = \hat{G}^{20}_{32}$ and a further increase of the sparsity s will result in even smaller error decreases, since it only detects new frequencies $\boldsymbol{k}$ belonging to even smaller Fourier coefficients $c_{\boldsymbol{k}}$.
The same behavior shows when using multiple or random rank-1 lattices as sampling strategies as we can see in Figure 2.5. The errors seem to be in the same order of magnitude as for single rank-1 lattices. We will compare the errors when using the different rank-1 lattice sampling strategies soon. But first, we take a look at the role of the extension N and therefore the search space Γ.
In Figure 2.6 the same error as in Figure 2.4 is shown, but this time for fixed sparsity $s = 500$ and different extensions N. We see, that our approximation suffers, if we choose the extension N too small. In fact, working with $N = 32$ or $N = 64$ allows the algorithm to detect large Fourier coefficients in a much bigger search domain Γ. For $N = 16$, our algorithm simply ignores some large Fourier coefficients outside of $[-16, 16]^{20}$ due the restriction of the search domain Γ.
The increase of N from 32 to 64 for sparsity $s = 500$ did result in a relatively small error decrease. For this small sparsity, most of the s largest Fourier coefficients are already



(a) Multiple rank-1 lattices.

(b) Random rank-1 lattices.

Figure 2.5: Averaged pointwise error $\mathrm{Err}^{\rho}_{\mathrm{sR1L}}$ with $n_{\mathrm{test}} = 10000$ when using multiple and random rank-1 lattices for parameter settings $\rho =$ III, IV, V, so with $s = 500$ and different extensions N.

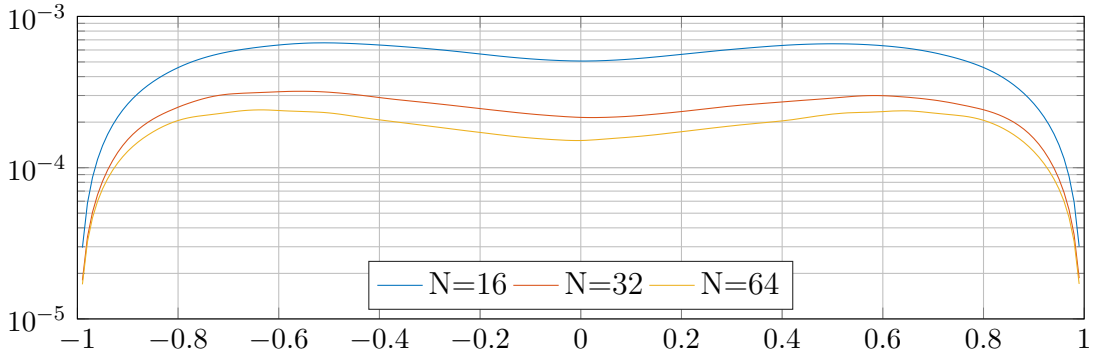


Figure 2.6: Averaged pointwise error $\mathrm{Err}^{\rho}_{\mathrm{sR1L}}$ with $n_{\mathrm{test}} = 10000$ when using single rank-1 lattices for parameter settings $\rho =$ III, IV, V, so with $s = 500$ and different extensions N.

located in $[-32, 32]^{20}$. For larger sparsities, e.g., $s = 2000$, we obviously want to use larger search domains Γ, as we see in Figure 2.7. This time, the larger extension $N = 64$ really results in a lot of different detected frequencies belonging to larger Fourier coefficients and therefore increasing the accuracy of our approximation. This behavior is matching our theoretical expectations and reminds us to choose large enough extension N when working with high sparsities s.

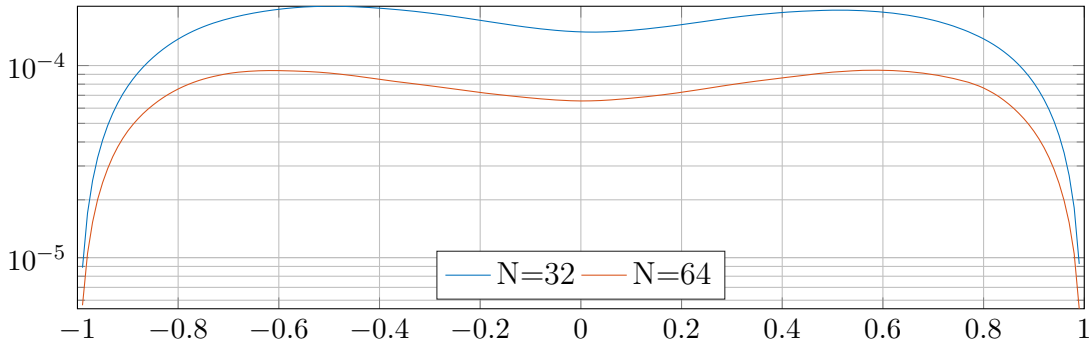


Figure 2.7: Averaged pointwise error $\mathrm{Err}^{\rho}_{\mathrm{sR1L}}$ with $n_{\mathrm{test}} = 10000$ when using single rank-1 lattices for parameter settings $\rho =$ VIII and IX , so with $s = 2000$ and different extensions N.

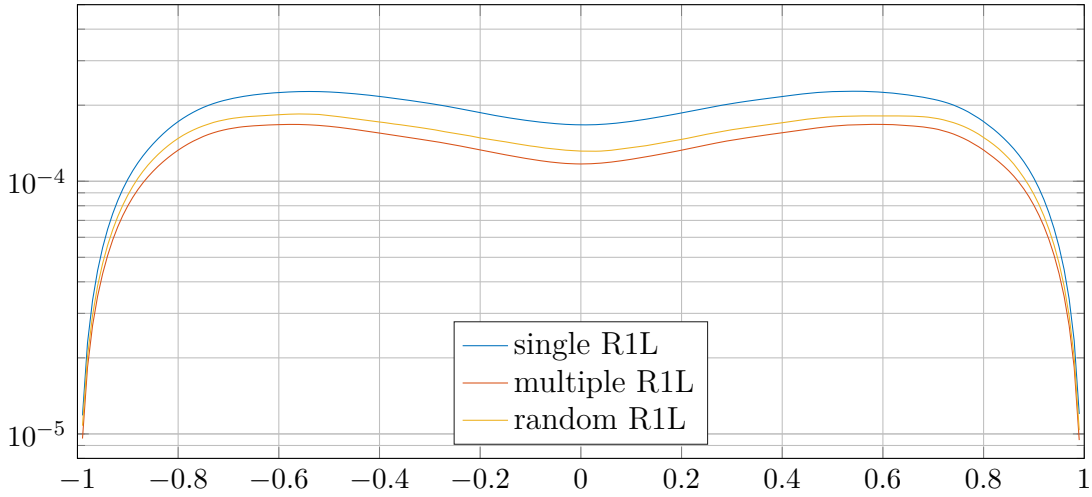


Figure 2.8: Averaged pointwise error $\mathrm{Err}^{\rho}_{\mathrm{sR1L}}$ with $n_{\mathrm{test}} = 10000$ when using different rank-1 lattice methods for parameter setting $\rho =$ VI, so with $s = 1000$ and $N = 32$.

In Example 2.6 we saw, that the chosen sampling strategy strongly impacts the number of used samples. Figure 2.8 shows the averaged pointwise error $\mathrm{Err}^{\rho}_{\dagger}$ for $s = 1000$ and $N = 32$ and the different sampling strategies. While the errors are very similar, we still see, that multiple and random rank-1 lattices seem to outperform the single rank-1 lattice approach. For multiple rank-1 lattices, we could argue by the larger total number of samples used, since therefore the approximation contains more information about the real solution. Unfortunately, this argumentation will not hold when explaining the behavior for random rank-1 lattices, since a significantly smaller number of samples was used there, cf. Table 4. Another possible explanation is the way each sampling strategy deals with aliasing: Multiple and random rank-1 lattices use averaging and median arguments when computing the

Fourier coefficients, cf. Section 2.4.2 and the given references therein. It seems, that these methods tend to reduce the occurring aliasing errors and therefore increase the accuracy of the whole approximation. □

Up to now, we considered averaged pointwise errors. In other words, we saw the error we can expect in general when using our algorithm for arbitrarily chosen $\boldsymbol{\xi}$. On the other hand, there might be some worst case scenarios, where the particular $\boldsymbol{\xi}$ leads to a worse approximation with a way larger pointwise error. Therefore, we now take a look at the largest error occurring for each grid point x_g for a set of n_{test} randomly chosen $\boldsymbol{\xi}$. This error can be considered as an approximate upper bound on the pointwise error for any $\boldsymbol{\xi}$ and therefore on the approximation error in such worst case scenarios.

Example 2.8
We consider the maximum pointwise error

$$\text{MaxErr}_{\dagger}^{\rho}(x_g) \coloneqq \max_{i=1,\dots,n_{\text{test}}} |\breve{u}(x_g, \boldsymbol{\xi}^i) - \breve{u}_{\dagger}^{\rho}(x_g, \boldsymbol{\xi}^i)|,$$

which is the largest error occurring for all n_{test} randomly chosen $\boldsymbol{\xi}$. Note, that the particular choice of the random parameter $\boldsymbol{\xi}$ impacts the differential equation and therefore the accuracy of our approximation. The maximum error $\text{MaxErr}_{\dagger}^{\rho}$ only considers the worst approximation $\breve{u}_{\dagger}^{\rho}(x_g, \boldsymbol{\xi}^i)$ occurring for a certain $\boldsymbol{\xi}^i$ for each x_g separately.

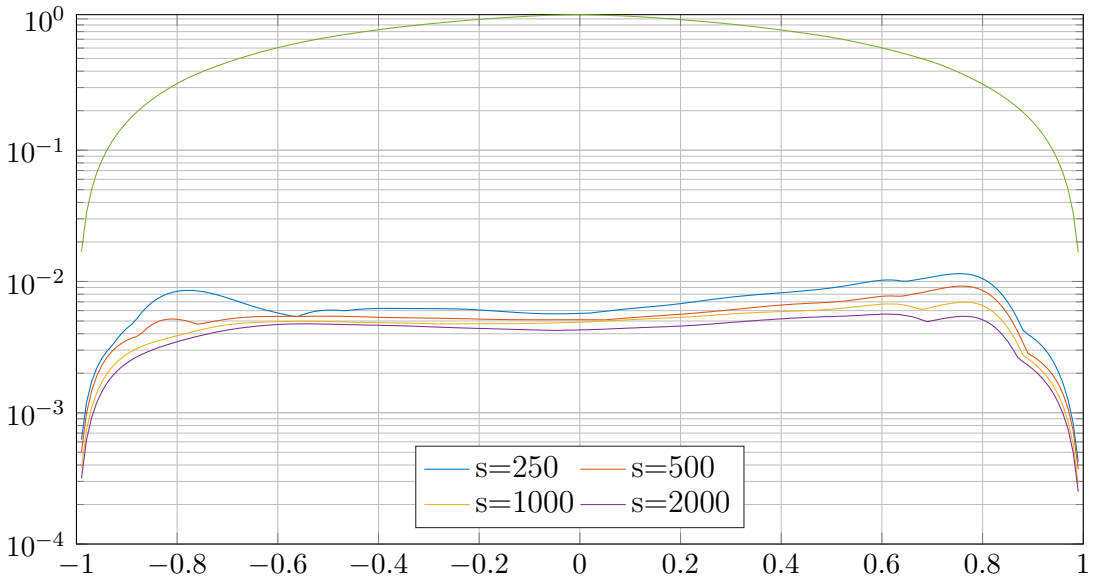


Figure 2.9: Maximum pointwise error $\text{MaxErr}_{\text{sR1L}}^{\rho}$ with $n_{\text{test}} = 100000$ when using single rank-1 lattices for parameter settings $\rho =$ II, IV, VI, VIII, so with $N = 32$ and different sparsities s. The green line is the smallest function value $\breve{u}(x_g, \boldsymbol{\xi}^i)$ occuring for each x_g w.r.t. $\boldsymbol{\xi}^i$.

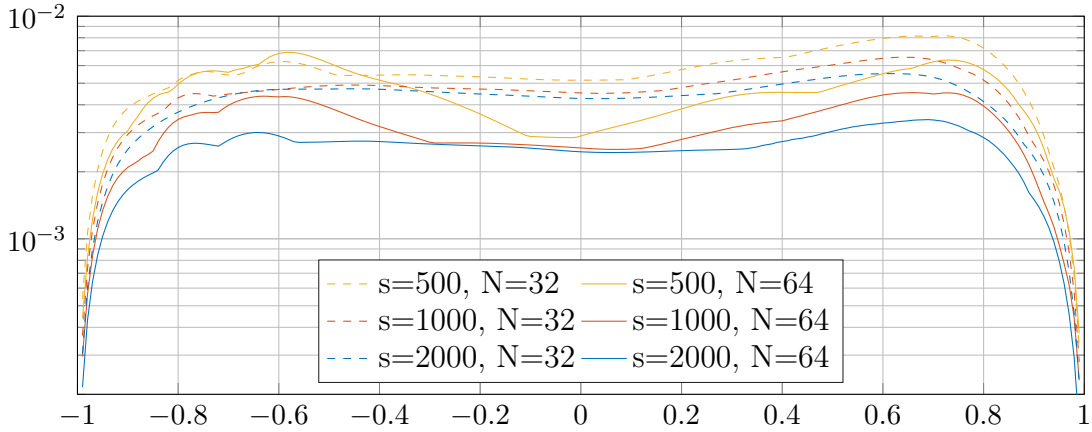


Figure 2.10: Maximum pointwise error $\mathrm{MaxErr}^{\rho}_{\mathrm{sR1L}}$ with $n_{\mathrm{test}} = 100000$ when using single rank-1 lattices for all parameter settings from $\rho = \mathrm{IV}$ to $\rho = \mathrm{IX}$.

Figure 2.9 shows these errors for the same settings as in Figure 2.4. We increased the number n_{test} by a factor of 10 to receive a smoother error plot. We see, that the errors are slightly larger than before. So while the averaged error tends to be smaller than 10^{-3}, certain fixed random variables $\boldsymbol{\xi}$ may lead to errors up to 10^{-2}. As for the averaged errors we also notice, that the error is larger near the boundary around $x = -0.7$ and $x = 0.7$ than at $x = 0$. The increase of the sparsity s still reduces the error, but not as much as before in Example 2.7.
A closer look at the maximum error $\mathrm{MaxErr}^{\rho}_{\dagger}$ for different pairings of s and N is given in Figure 2.10. First, we take a look at the area around $x = 0$. There it shows, that for $N = 32$ the larger sparsities show nearly no increase in the approximation accuracy. The solution at these grid points x_g seems to be very sparse in the search domain $\Gamma = [-32, 32]^{20}$, such that the sparsity $s = 500$ already covers most of the important Fourier coefficients. A similar observation can be seen for $N = 64$, but with smaller errors due to the larger search domain Γ.
Near $x = -0.7$ and $x = 0.7$, the behavior is slightly different. Especially at the left side $x = -0.7$, a larger search domain Γ shows nearly no impact for the sparsities $s = 500$ and $s = 1000$. The structure of the solution at these points seems to be way more complicated and we really need a lot of Fourier coefficients to achieve a good approximation.
In Figure 2.11 we make the same observation as before in Figure 2.8. The approximations when using the different sampling strategies lead to nearly the same errors. Again, only single rank-1 lattices seem to be slightly worse than multiple and random rank-1 lattices, probably for the reason already stated above in Example 2.7. □

Since we know from Section 2.5 how to compute the expectation value of our approximated solution, we can use it as another possibility to test the accuracy of our algorithm.

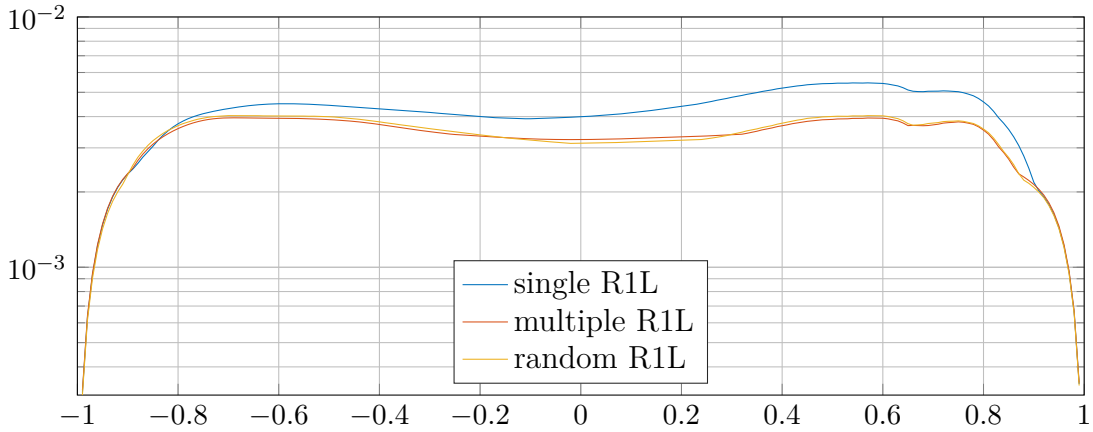


Figure 2.11: Maximum pointwise error $\mathrm{MaxErr}_\dagger^\rho$ with $n_{\text{test}} = 100000$ when using different sampling strategies for the parameter setting $\rho = \text{VIII}$, i.e., with $s = 2000$ and $N = 32$.

Example 2.9

We use the approach from Section 2.5, i.e., the formula (2.19) with the $c_{\boldsymbol{k}}$ as defined in (2.20). As a comparison, we compute the Monte-Carlo approximation of the expectation value by

$$\overline{u_{n_{\mathrm{MC}}}(x_g)} \coloneqq \frac{1}{n_{\mathrm{MC}}} \sum_{i=1}^{n_{\mathrm{MC}}} \breve{u}(x_g, \boldsymbol{\xi}^i)$$

for each fixed x_g. The error at these points is then computed by

$$\mathrm{Res}_\dagger^\rho(x_g) \coloneqq |\overline{u_{n_{\mathrm{MC}}}(x_g)} - \mathbb{E}\breve{u}_\dagger^\rho(x_g, \cdot)|.$$

In the following computations, we used $n_{\mathrm{MC}} = 10^7$, since we noticed the Monte-Carlo approximation to be inaccurate for smaller n_{MC} in our tests. Note, that especially for large sparsities s and extensions N, which are leading to a smaller error, as we saw in Example 2.7, we want an even larger number of samples n_{MC}. Otherwise, the error estimation $\mathrm{Res}_\dagger^\rho$ suffers from the missing precision of our Monte-Carlo approximation and is therefore no reliable error estimator.

Figure 2.12 illustrates this error for $\rho = \text{VI}$ when using random rank-1 lattices. As mentioned earlier, we use numerical integration with an relative error tolerance of 10^{-6} to compute the approximation of the true solution $\breve{u}$. Combined with the randomness when drawing the $\boldsymbol{\xi}$, the non-smooth structure of our error can be explained. In detail, the Monte-Carlo approximation tends to oscillate around the computed expectation value of our approximation of the solution. Nevertheless, the error tends to be very small compared to the function values, matching our pointwise results from Example 2.7.

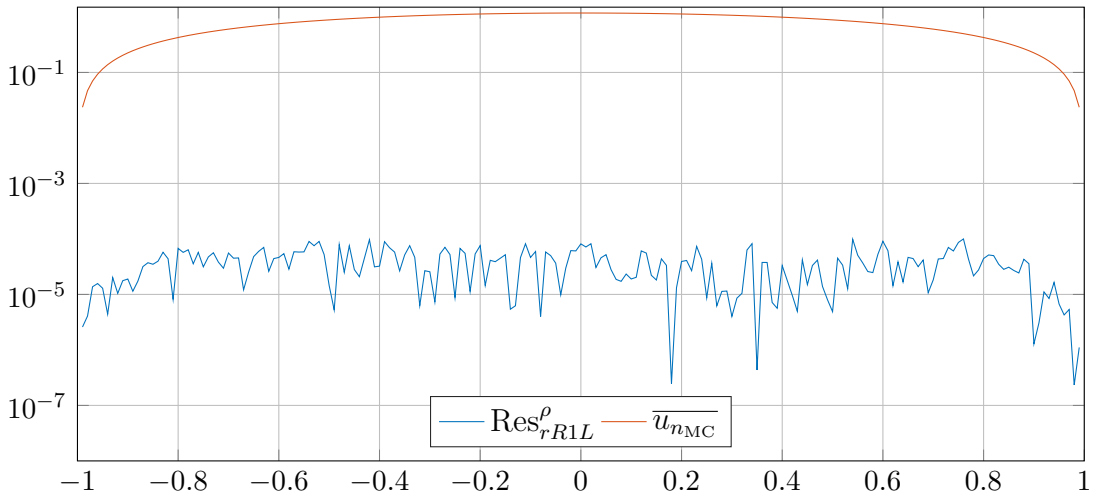


Figure 2.12: Expectation value error $\mathrm{Res}^{\rho}_{rR1L}$ with $n_{\mathrm{MC}} = 10^7$ when using random rank-1 lattices for parameter setting $\rho = \mathrm{VI}$, so with $N = 32$ and sparsity $s = 1000$.

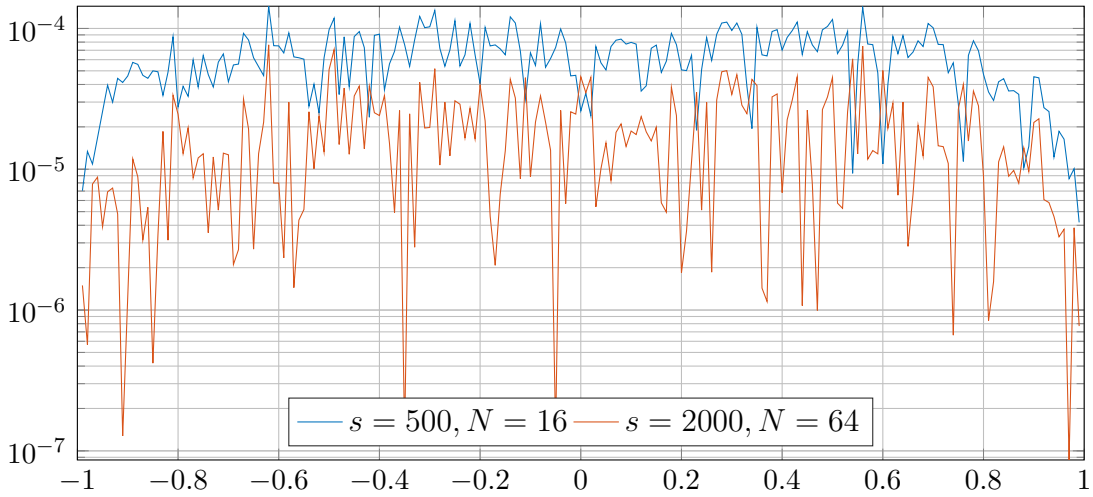


Figure 2.13: Expectation value error $\mathrm{Res}^{\rho}_{rR1L}$ with $n_{\mathrm{MC}} = 10^7$ when using random rank-1 lattices for parameter setting $\rho = \mathrm{III}$ and IX.

The non-smooth structure of $\mathrm{Res}^{\rho}_{\dagger}$ makes it difficult to compare these errors for different parameter settings ρ and sampling strategies $\dagger$. Figure 2.13 shows the error for two very different parameter settings $\rho = \mathrm{III}$ and $\rho = \mathrm{IX}$. Therefore, the difference of the results is large enough so they do not overlap each other too often and a tendency is visible. So again we can see, that the larger sparsity $s = 2000$ combined with the larger extension $N = 64$ results in a better approximation. This is obviously not surprising, but again a good confirmation of our former results and theoretical expectations. □

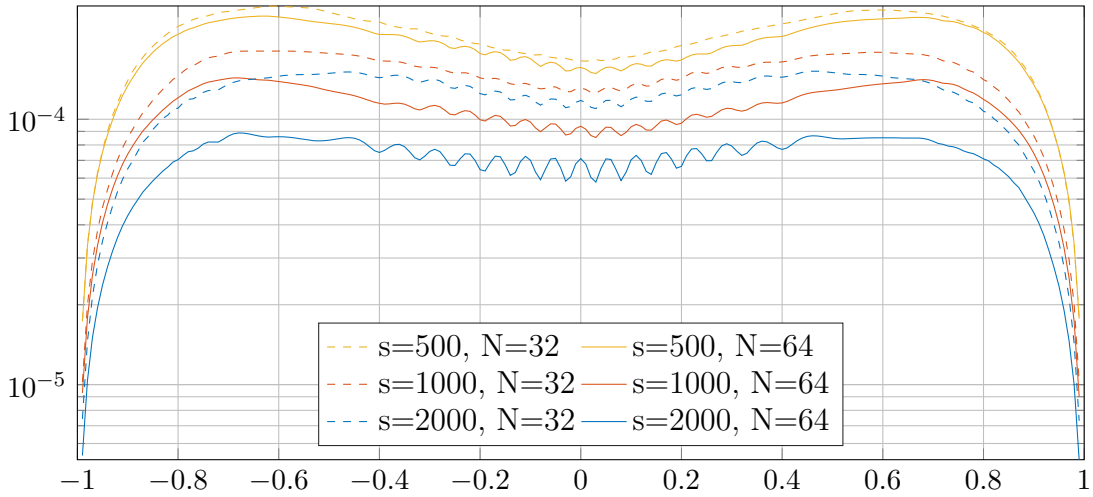
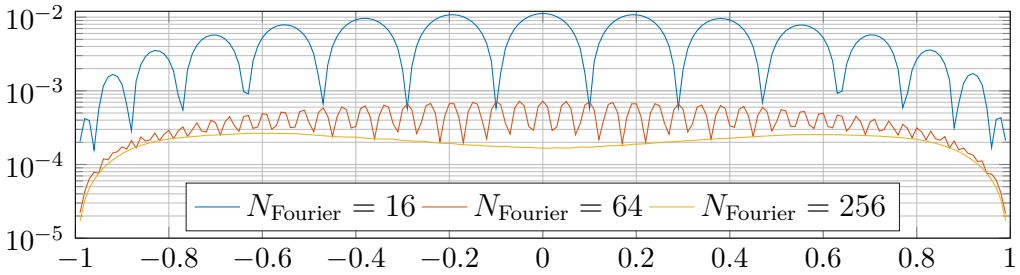


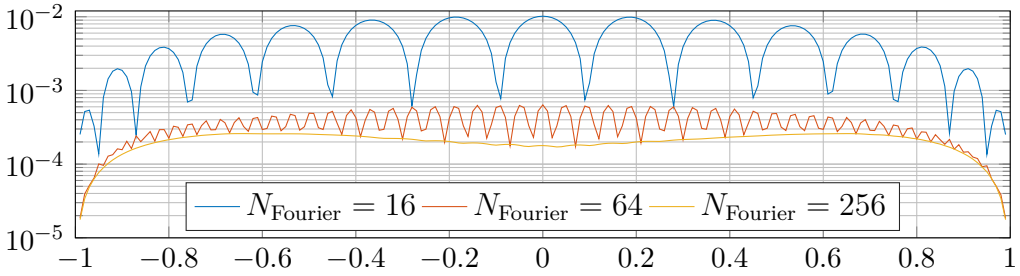
Figure 2.14: Averaged pointwise error $\mathrm{Err}^{\rho}_{\mathrm{rR1L}}$ with $n_{\mathrm{test}} = 10000$ when using the Fourier solver based on the cosine periodization with $N_{\mathrm{Fourier}} = 256$ for all parameter settings from $\rho = \mathrm{IV}$ to $\rho = \mathrm{IX}$.

Up to now, we have only worked with the numerical solver explained in Remark 2.1. The following example uses the Fourier method proposed in Section 2.3.1 instead.

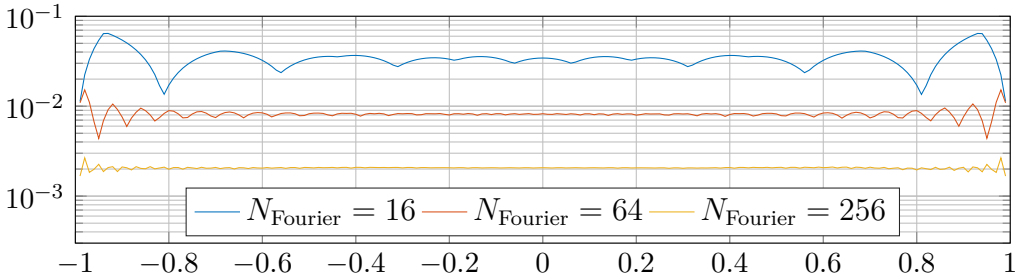
Example 2.10
Again, we consider the averaged pointwise error $\mathrm{Err}^{\rho}_{\dagger}(x_g)$ as defined in (2.23). This time, Algorithm 2 is used to compute the sampling values in our uniform sFFT. To avoid confusion at this point, we now use the notation N_{Fourier} for the number of Fourier coefficients used in the Fourier solver, cf. Section 2.3.1, and keep the variable N as the extension of the full grid $\hat{G}^{d_{\boldsymbol{\xi}}}_{N}$. Further, the following tests always use the random rank-1 lattice approach as sampling strategy, cf. Section 2.7.
In Figure 2.14 the Fourier solver was used with the cosine function, cf. Example 2.3, and $N_{\mathrm{Fourier}} = 256$. In Example 2.5 we saw, that for these choices the Fourier method solves the differential equation with high precision. Therefore, it is not surprising, that the results now are very similar to Example 2.7. The magnitude of the averaged pointwise Error $\mathrm{Err}^{\rho}_{\mathrm{rR1L}}$ is the same as before and we also see the same behavior as before when using different sparsities s and extensions N. One noticeable difference is the Fourier typical oscillation of the error plot around $x = 0$, especially for large sparsities s.
By now, we tested the accuracy of our uniform sFFT based on the corresponding parameters and sampling strategies. While the non-intrusive algorithm allows for different ODE solvers to be used inside the uniform sFFT, one could ask for the influence of the precision of this solver on the final result. Now we can take a look at this influence by changing the parameters of our Fourier solver. In Figure 2.3 we saw, how the precision of our solver changes when working with different periodization functions φ and different N_{Fourier}. Figure 2.15 illustrates the results for the different periodizations and three different N_{Fourier}. Note, that these periodization mappings are only used w.r.t. the spatial variable x for the



(a) Cosine periodization.



(b) Spline periodization.



(c) Tent periodization.

Figure 2.15: Averaged pointwise error $\text{Err}^{\rho}_{\text{rR1L}}$ with $n_{\text{test}} = 10000$ when using the different periodization functions φ and different N_{Fourier} for parameter setting $\rho = \text{IV}$, so with $s = 500$ and $N = 32$.

ODE solver. As mentioned before, the uniform sFFT here only uses the tent transform on the random variables $\boldsymbol{\xi}$ every time.

We notice, that the error for each pair of φ and N_{Fourier}, the errors are smaller at the Fourier nodes. The larger errors between these nodes is because of the linear interpolation to compute the missing sampling values there. Obviously, this effect is significantly larger when using less Fourier nodes, so a smaller N_{Fourier}. On the other hand, near the Fourier nodes, the magnitude of the errors for the spline and cosine periodizations for $N_{\text{Fourier}} = 64$ and $N_{\text{Fourier}} = 256$ are nearly the same. It seems like the precision of the solver is already good enough for the spline function with $N_{\text{Fourier}} = 64$, such that a further increase of N_{Fourier} or working with the better cosine periodization, does not result in a different approximation up to the better interpolation behavior for larger N_{Fourier} as explained above. For $N_{\text{Fourier}} = 16$, the error of our solver in Example 2.5 is in the same magnitude as the

error of our final result in Figure 2.15 near the Fourier nodes. It seems, that here the precision of our solver is limiting our approximation. On the other hand, for the larger N_{Fourier}, the solver works precise enough while the precision of the uniform sFFT in general is limiting the results.
Using the tent transformation here as periodization function seems to achieve way worse results. The non-smooth periodization, leading to a very slow error decrease as seen in Example 2.5, results in heavily disturbed sampling values for the uniform sFFT, such that the final approximation is nearly useless, especially for small N_{Fourier}. □

2.7 Discussion of the numerical results

First of all, the results in our numerical tests in Section 2.6 matched our theoretical expectations on the dimension-incremental algorithm in general, cf. Section 2.4.1.
The sparsity s as well as the extension N of the full grid $\hat{G}_N^{d_{\boldsymbol{\xi}}} \coloneqq \{-N, ..., N\}^{d_{\boldsymbol{\xi}}}$ and therefore the size of our search space Γ influence the output of our algorithm significantly. We saw, that in general the solution $u(x, \boldsymbol{\xi})$ can be approximated really well already with small sparsities as for example $s = 1000$ when choosing a suitable extension N.
Further, as discussed in Example 2.8, the errors might be slightly larger in some cases. So while in general we will expect errors as in Example 2.7, we always have to keep in mind, that there is a certain spread due to the randomness and so there are also cases for certain $\boldsymbol{\xi}$, where the approximation of our solution behaves slightly worse with errors as seen in e.g., Figure 2.10.
The Fourier solver developed in Section 2.3.1 works very well compared to the numerical solver. For large enough N_{Fourier} and a smooth periodization, the accuracy of the solver is good enough and results in similar approximations. Note, that for large sparsities s and extensions N we also need to choose a larger N_{Fourier} and/or a smoother periodization. Otherwise, the precision of the Fourier solver is the limiting factor again and further increases in the uniform sFFT parameters will show no effect. In general, we would recommend using very smooth periodizations as for example the cosine transformation (2.8). Non-smooth periodizations require a very large N_{Fourier} to work reliably. This results in a huge increase of the computation time of the solver and therefore the whole sampling step of our uniform sFFT.
The different sampling strategies, proposed in Section 2.4.2 and the given references therein, had a large impact on our Algorithm 3. The number of used samples in the uniform sFFT is an important value. While we worked with two different ODE solvers given in Section 2.3, the algorithm itself is non-intrusive and one may use any suitable solver on the differential equation to compute the required sampling values. Each sample in our algorithm is then one application of this differential equation solver. Even though these executions are parallelizable, this will be the most expensive step of our algorithm for almost every interesting differential equation we want to solve, as already mentioned in Section 2.4.3.
In Example 2.6 we saw, that random rank-1 lattices outperform single and multiple rank-1 lattices in terms of used samples by a lot. So the usage of this sampling strategy seems

to be superior and results in the fastest overall computation time in the sampling step as well as all other steps of the dimension-incremental algorithm, cf. Section 2.4.2 and Table 1. Further, the approximation errors occurring when using random rank-1 lattices do not suffer from this lack of information as one could expect. The numerical examples showed, that the errors for multiple and random rank-1 lattices are nearly the same and are also slightly smaller than for single rank-1 lattices.
Overall, random rank-1 lattices seems to be the superior sampling strategy used in our algorithm in many ways. The usage of single and multiple rank-1 lattices is way more expensive and does not result in significantly better approximations. Therefore, in the following part of this work, we will only consider Algorithm 3 with random rank-1 lattices as used sampling strategy.

3 The two-dimensional case

As already stated in the previous section, the uniform sFFT is not limited to one-dimensional spatial variables x. Therefore, we now proceed to partial differential equations with $\boldsymbol{x} \in D$, where D is assumed to be a bounded Lipschitz domain $D \subset \mathbb{R}^2$. So we now consider the second order divergence form elliptic Dirichlet problem

$$-\nabla_{\boldsymbol{x}} \cdot (a(\boldsymbol{x}, \boldsymbol{\xi}) \nabla_{\boldsymbol{x}} u(\boldsymbol{x}, \boldsymbol{\xi})) = f(\boldsymbol{x}), \qquad \boldsymbol{x} \in D,\, \boldsymbol{\xi} \in D_{\boldsymbol{\xi}} \tag{3.1a}$$

$$u(\boldsymbol{x}, \boldsymbol{\xi}) = 0, \qquad \forall\, \boldsymbol{x} \in \partial D,\, \boldsymbol{\xi} \in D_{\boldsymbol{\xi}}, \tag{3.1b}$$

with random coefficient $a(\boldsymbol{x}, \boldsymbol{\xi})$. The weak formulation of (3.1) reads: Given $f \in H^{-1}(D)$, for every $\boldsymbol{\xi} \in D_{\boldsymbol{\xi}}$ find $u(\cdot, \boldsymbol{\xi}) \in H_0^1(D)$, s.t.

$$\int_D a(\boldsymbol{x}, \boldsymbol{\xi}) \nabla_{\boldsymbol{x}} u(\boldsymbol{x}, \boldsymbol{\xi}) \cdot \nabla_{\boldsymbol{x}} v(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} = \int_D f(\boldsymbol{x}) v(\boldsymbol{x}) \mathrm{d}\boldsymbol{x}, \qquad \forall v \in H_0^1(D). \tag{3.2}$$

Again, we ask the random coefficient $a(\boldsymbol{x}, \boldsymbol{\xi})$ to fulfill the uniform ellipticity assumption

$$0 < r \leq a(\boldsymbol{x}, \boldsymbol{\xi}) \leq R < \infty \qquad \text{for almost all } \boldsymbol{x} \in D, \text{ for all } \boldsymbol{\xi} \in D_{\boldsymbol{\xi}} \tag{3.3}$$

to ensure well-posedness of the problem (3.1). Further and more general notes on the formulation of the PDE and its well-posedness can be found in [9] or [5].

3.1 The uniform sFFT on finite elements

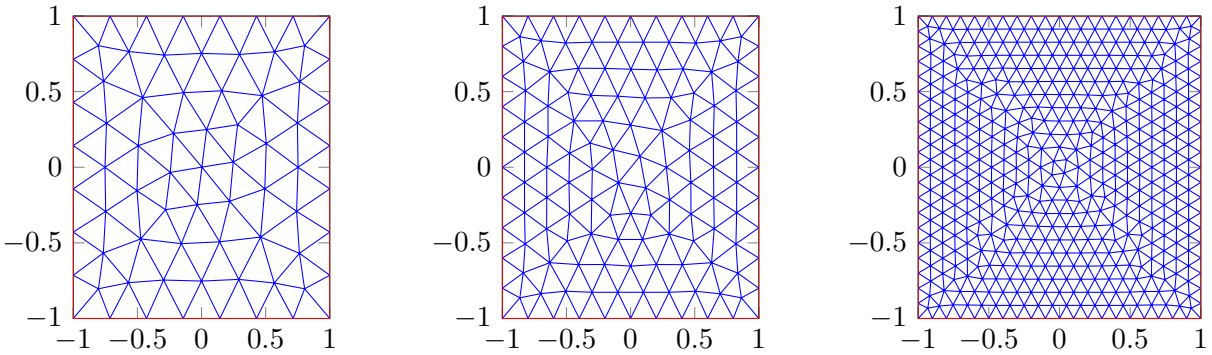
As in Section 2, we assume $D_{\boldsymbol{\xi}} = [-1, 1]^{d_{\boldsymbol{\xi}}}$ from now on. We will perform some minor changes on our algorithm before using it on the two-dimensional problem. Most important, we need a suitable method to solve the PDE (3.1) for a given $\boldsymbol{\xi}$. Obviously, as in Section 2.3, any suitable solver can be used to do so due to the non-intrusive approach of the uniform sFFT.

Here, we will only consider a similar approach as in Section 2.3.2, using an FEM. To this end we work with the Partial Differential Equation Toolbox™ provided by Matlab®. It provides a numerical solution of the PDE using finite element analysis, i.e., we end up with a finite element mesh for the given domain D and fixing $\boldsymbol{\xi}$, the solver provides the values $u(\boldsymbol{x}, \boldsymbol{\xi})$ for each node $\boldsymbol{x}$ of the mesh. Since the finite element mesh only depends on the domain D and certain refinement parameters, we can ensure to use the same mesh for each call of the PDE solver. Therefore, it seems reasonable to work with the nodes of this mesh in the uniform sFFT instead of using an equidistant grid on the domain D as in Section 2.4. Note, that only inner nodes $\boldsymbol{x} \notin \partial D$ are considered in the uniform sFFT. The boundary condition (3.1b) ensures, that all Fourier coefficients for boundary nodes $\boldsymbol{x} \in \partial D$ are zero, since the solution $u(\boldsymbol{x}, \boldsymbol{\xi})$ is constant zero w.r.t. the variable $\boldsymbol{\xi}$ at these nodes.

As in the one-dimensional case, we denote the set of points the uniform sFFT is working on, i.e., all the inner nodes of the FE mesh, as $\mathcal{T}_G$ with G being the number of inner nodes. The full FE mesh including the boundary nodes is accordingly denoted by $\mathcal{T}_G^+$.

$H_{\max}$	# of elements	# of nodes	# of inner nodes
0.3	116	261	205
0.2	228	497	417
0.15	394	841	737
0.1	906	1893	1733
0.075	1652	3413	3197
0.05	3656	7473	7153

Table 5: Properties of the generated finite element mesh on $D = [-1, 1]^2$ for different $H_{\max}$.



(a) $H_{\max} = 0.3$ (b) $H_{\max} = 0.2$ (c) $H_{\max} = 0.1$

Figure 3.1: Generated finite element mesh on $D = [-1, 1]^2$ for different $H_{\max}$.

Example 3.1
We consider the domain $D = [-1, 1]^2$. We use the refinement parameter $H_{\max}$, specifying the target maximum mesh edge length. It is an approximate upper bound on the mesh edge lengths and therefore a smaller value results in a more refined mesh.
Table 5 shows the numbers of elements and nodes of a finite element mesh using different values for $H_{\max}$. Figure 3.1 illustrates some of these meshes. □

Remark 3.1
If another suitable method is used to solve the PDE (3.1), one maybe need to rethink this particular grid strategy for $\boldsymbol{x}$. For example, as in Section 2.3, one could choose a grid in advance and receive the corresponding values $u(\boldsymbol{x}, \boldsymbol{\xi})$ via a suitable interpolation. This strategy tends to be very general and can be used for most solvers, but obviously a more specific approach fitting the particular solver might be favorable to avoid error increase by the interpolation as seen in Example 2.10 and also end up with faster computation times. □

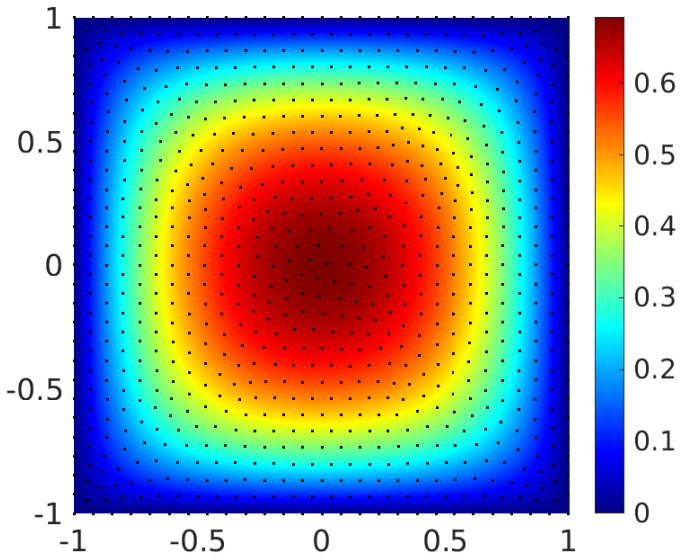


Figure 3.2: The Monte-Carlo approximation of the expectation value of the solution of the PDE using the random coefficient (3.4) and $n_{\mathrm{MC}} = 10^7$. The black dots are the nodes of the FE mesh, on which the solution is approximated. The error estimate is interpolated between these points for a better visualization.

3.2 Numerical results

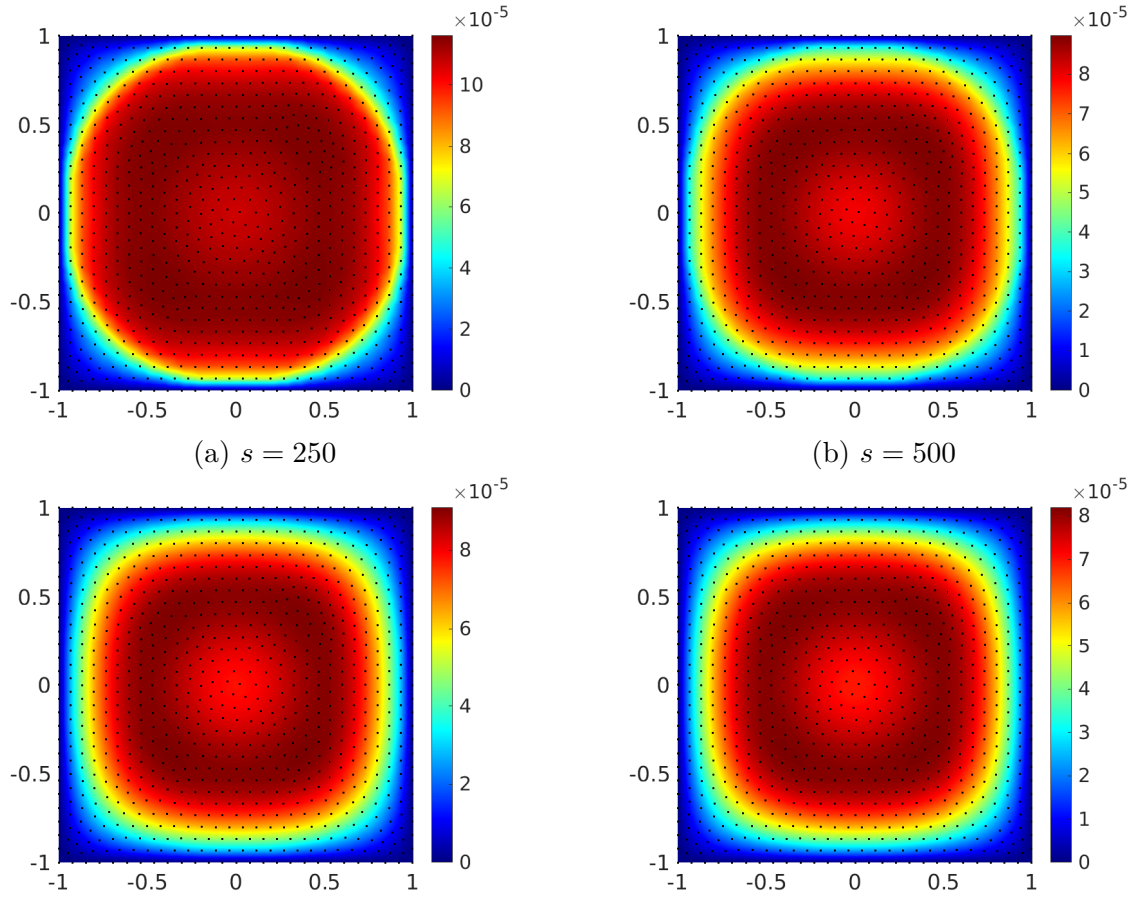
For the numerical results on our uniform sFFT, we consider two examples using slightly different random coefficients $a(\boldsymbol{x}, \boldsymbol{\xi})$. Both random coefficients are of the general form $a(\boldsymbol{x}, \boldsymbol{\xi}) = a_0(\boldsymbol{x}) + \sum \xi_j a_j(\boldsymbol{x})$. In the first example, the spatial variable $\boldsymbol{x}$ appears only inside an Euclidean norm. Therefore, we will call this $a(\boldsymbol{x}, \boldsymbol{\xi})$ a radial random coefficient. In the second example, the coefficient functions $a_j(\boldsymbol{x})$ are built by a product of cosine functions in each spatial dimension x_i, $i = 1, 2$.

3.2.1 A radial random coefficient

We consider the PDE (3.1) with the domain $D = [-1, 1]^2$ and right-hand side $f \equiv 10$. As in [5, Section 6], we use the random coefficient

$$a(\boldsymbol{x}, \boldsymbol{\xi}) = a_0(\boldsymbol{x}) + \sum_{j=1}^{d_{\boldsymbol{\xi}}} \boldsymbol{\xi}_j \frac{\cos(\pi j \|\boldsymbol{x}\|)}{j^{\mu}}, \qquad \boldsymbol{x} \in [-1, 1]^2,\ \boldsymbol{\xi} \in [-1, 1]^{d_{\boldsymbol{\xi}}} \tag{3.4}$$

with parameters $\mu = 2$, $a_0 \equiv 4.3$ and $d_{\boldsymbol{\xi}} = 20$ as before, fulfilling the uniform ellipticity assumption (3.3). Again, we assume the random variable $\boldsymbol{\xi}$ to be uniformly distributed, i.e., $\boldsymbol{\xi} \sim U([-1, 1]^{d_{\boldsymbol{\xi}}})$. Figure 3.2 shows the Monte-Carlo approximation of the expectation value of the solution to reveal the magnitude of the solution for comparison to the absolute errors, which we see in the following examples.



(a) $s = 250$ (b) $s = 1000$

(c) $s = 1000$ (d) $s = 2000$

Figure 3.3: Averaged pointwise error Err^{ρ} with $n_{\text{test}} = 10000$, $H_{\max} = 0.15$ and parameter settings ρ =II, IV, VI and VIII, so with different sparsities s and extension $N = 32$.

Example 3.2

We consider the averaged pointwise error $\mathrm{Err}^{\rho}(\boldsymbol{x}_g)$ as in Example 2.7, i.e.,

$$\mathrm{Err}^{\rho}(\boldsymbol{x}_g) \coloneqq \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} |\breve{u}(\boldsymbol{x}_g, \boldsymbol{\xi}^i) - \breve{u}^{\rho}(\boldsymbol{x}_g, \boldsymbol{\xi}^i)|. \tag{3.5}$$

Note, that $\boldsymbol{x}_g \in [-1, 1]^2, g = 1, ..., G$, are now the inner nodes T_G of our finite element mesh as explained in Section 3.1. We choose $H_{\max} = 0.15$ for now, so we end up with $G = 737$ spatial nodes in our uniform sFFT. We use the same parameter settings ρ as for the one-dimensional case, cf. Table 3, the random rank-1 lattice approach as sampling strategy for the reasons stated in Section 2.7, and we choose $n_{\text{test}} = 10000$ as in the one-dimensional examples.
Figure 3.3 shows the averaged pointwise error Err^{ρ} for different sparsities s and the extension $N = 32$. While we only evaluate the approximation error at the nodes of the FE mesh

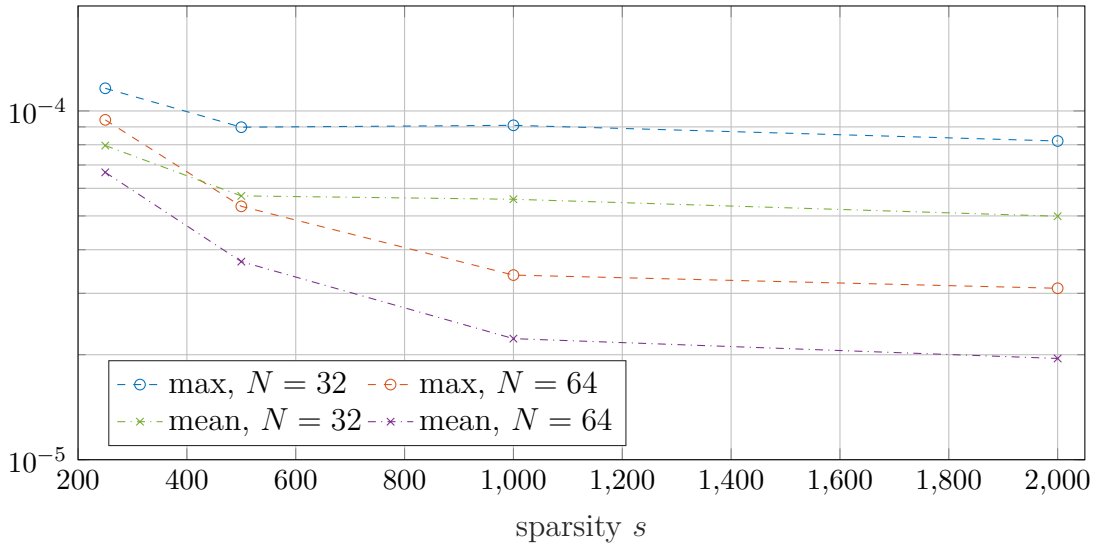
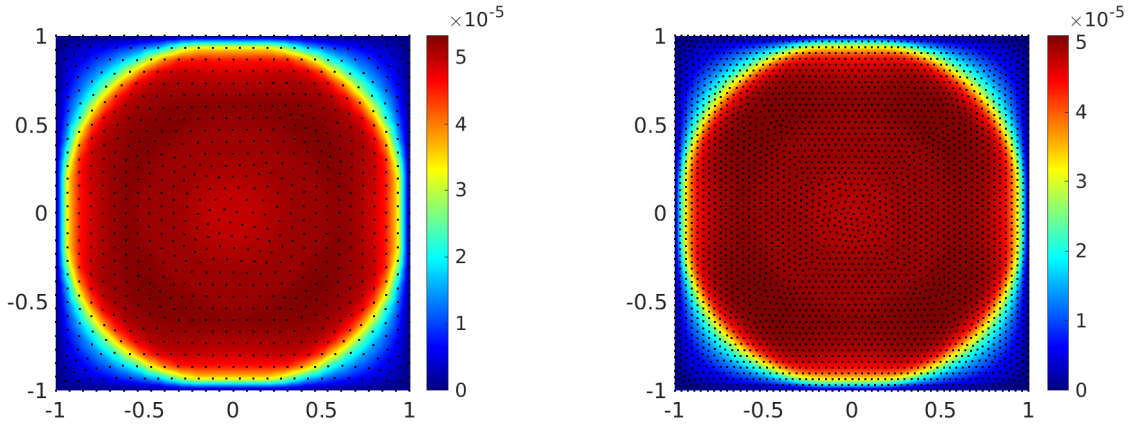


Figure 3.4: The maximum and mean value of the averaged pointwise errors Err^{ρ} with $n_{\text{test}} = 10000$, $H_{\max} = 0.15$ for different sparsities s and two different extensions N.

T_G^+, i.e., the black dots in the plots, we use an interpolation based on these non-equidistant data points T_G^+ provided by Matlab® to visualize the results on the whole domain $[-1, 1]^2$. The approximation error seems to show the same behavior as in the one-dimensional test: It increases very fast when moving away from the boundary and takes its highest values in a ring around the center. We saw the similar effect in Example 2.7, where the error was largest around $x = \pm 0.7$ and slightly smaller in the center $x = 0$. The magnitude of the approximation errors is already very good. Using $s = 2000$, we achieve a precision of roughly $8 \cdot 10^{-5}$, which is very small compared to the range of the solution as seen in Figure 3.2. Unfortunately, the error plots look very similar for $s = 500$, $s = 1000$ and $s = 2000$, indicating, that our extension N is too small and therefore no better approximation can be achieved when using more and more frequencies.
Figure 3.4 illustrates this behavior in another way. Here, we see the maximum and mean value of the averaged pointwise errors $\mathrm{Err}^{\rho}_{\text{rR1L}}$ w.r.t. all nodes $\boldsymbol{x}_g \in \mathcal{T}_G^+$, i.e.,

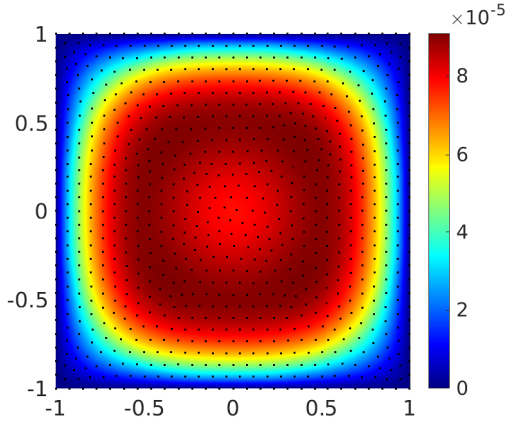
$$\max_{\boldsymbol{x}_g \in \mathcal{T}_G^+} \mathrm{Err}^{\rho}(\boldsymbol{x}_g) \quad \text{and} \quad \frac{1}{|\mathcal{T}_G^+|} \sum_{\boldsymbol{x}_g \in \mathcal{T}_G^+} \mathrm{Err}^{\rho}(\boldsymbol{x}_g). \tag{3.6}$$

Note, that we now also used the expansion $N = 64$ of the candidate set $\Gamma = [-N, N]^{d_{\boldsymbol{\xi}}}$ together with sparsity $s = 250$ for comparison reasons, which were not part of the original parameter settings shown in Table 3. For $N = 32$ we clearly see the stagnation in both the maximum and the mean value. On the other hand, the red and purple lines show the errors for $N = 64$. These errors are smaller than for $N = 32$, as we would expect. Moreover, we also seem to observe a beginning stagnation for larger sparsities s, but this stagnation starts for larger s compared to the case $N = 32$.

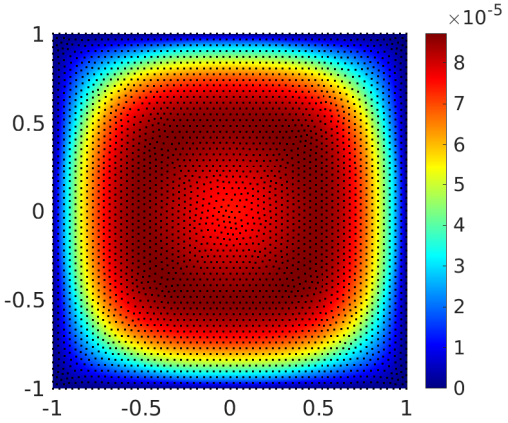


(a) $s = 500$, $N = 64$, $H_{\max} = 0.15$

(b) $s = 500$, $N = 64$, $H_{\max} = 0.075$



(c) $s = 1000$, $N = 32$, $H_{\max} = 0.15$



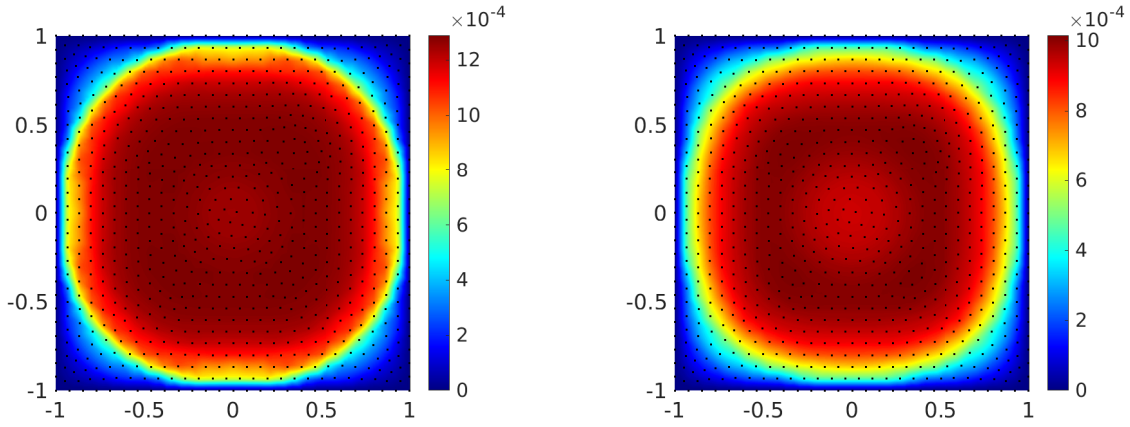
(d) $s = 1000$, $N = 32$, $H_{\max} = 0.075$

Figure 3.5: Averaged pointwise error Err^{ρ} with $n_{\text{test}} = 10000$, parameter settings $\rho =$V and VI and two different $H_{\max}$.

Up to now, we have worked with a FE mesh of $G = 737$ inner nodes using the parameter $H_{\max} = 0.15$. Figure 3.5 illustrates the difference when using a finer mesh with $G = 3197$ inner nodes, i.e., $H_{\max} = 0.075$. We see, that for our particular example the finer mesh does not achieve a significant decrease of the error, neither for the parameter setting $\rho =$ V nor $\rho =$ VI. We end up with more points, where we evaluated the solution, but the error size and the structure of the error is nearly the same in both cases. Accordingly, it will not be necessary to further refine the FE mesh for our current example. □

Remark 3.2

Note, that for other choices of the random coefficient $a(\boldsymbol{x}, \boldsymbol{\xi})$ or other kinds of PDEs the used FE mesh might not be fine enough. For example, if the solution is not very smooth at some areas in the domain or possess very large derivatives there, it might be necessary to refine the FE mesh in advance on the whole domain or at least on these particular areas, to ensure a good approximation. Also, if we work with other domains than $[-1, 1]^2$ as for



(a) $s = 500$ (b) $s = 1000$

Figure 3.6: Maximum pointwise error MaxErr^ρ with $n_{\text{test}} = 10000$, $H_{\max} = 0.15$ and parameter settings ρ =V and VII, so with different sparsities s and extension $N = 64$.

example an L-shaped domain, we might also have to pay additional attention to certain areas and the FE mesh there. □

In the one-dimensional examples we also studied the maximum pointwise errors $\mathrm{MaxErr}_\dagger^\rho$ to achieve an approximation on the upper bound of the approximation error. The following example does the same for our current PDE example, since the spread of the $\boldsymbol{\xi}$ will obviously yield also larger errors than the ones shown in Example 3.2 in some cases.

Example 3.3
As in Example 2.8, we now consider the maximum pointwise error

$$\mathrm{MaxErr}^\rho(\boldsymbol{x}_g) \coloneqq \max_{i=1,\dots,n_{\text{test}}} |\breve{u}(\boldsymbol{x}_g, \boldsymbol{\xi}^i) - \breve{u}^\rho(\boldsymbol{x}_g, \boldsymbol{\xi}^i)|. \tag{3.7}$$

Figure 3.6 shows this error for the extension $N = 64$ and the sparsities $s = 500$ and $s = 1000$. The general structure of the approximation error is the same as before up to some small deviations, e.g., at $\boldsymbol{x} = (\pm 0.8, 0)^\top$.
Some more insight on the actual magnitude of the errors is given in Figure 3.7. There, we see the maximum and mean values of the averaged pointwise errors Err^ρ as explained in (3.6) and the same quantities for the maximum pointwise errors MaxErr^ρ. While the maximum value of $\mathrm{MaxErr}^\rho(\boldsymbol{x}_g)$ at any grid point $\boldsymbol{x}_g$, so the largest possible error for these parameters for any $\boldsymbol{x}_g$ and any $\boldsymbol{\xi}^i$, seems to stagnate around $1 \cdot 10^{-3}$, the maximum value of $\mathrm{Err}^\rho(\boldsymbol{x}_g)$ is about 30 times smaller.
Another observation we also made here and in Figure 3.4 is the small difference between the maximum and mean value for both errors Err^ρ and MaxErr^ρ. Hence, the largest value of these errors at any grid point $\boldsymbol{x}_g$ is not that much larger than the error in general. So there are no particular peaks, where the approximation is significantly worse than on the

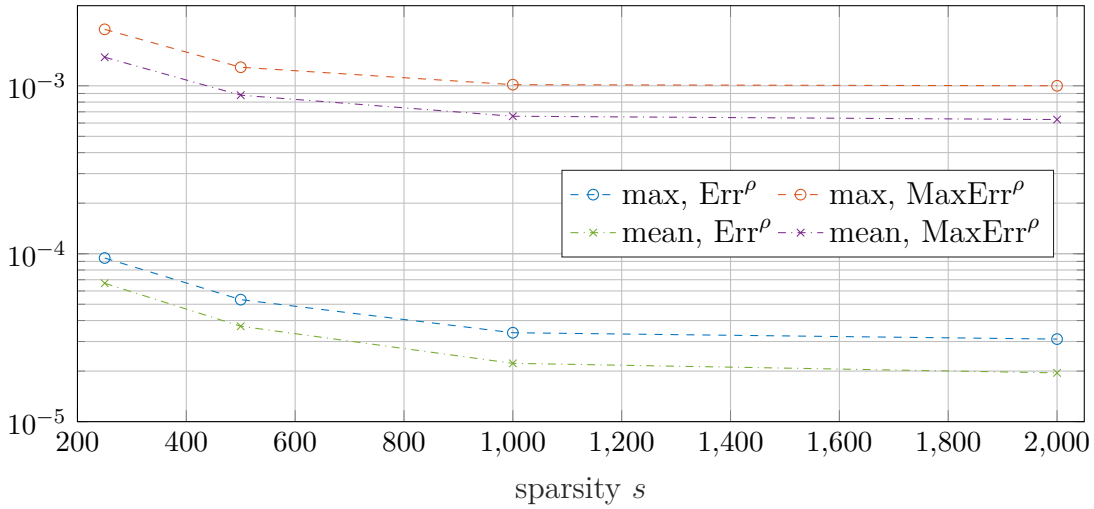


Figure 3.7: The maximum and mean value of the maximum pointwise errors MaxErr^ρ and the averaged pointwise errors Err^ρ with $n_{\text{test}} = 10000$, $H_{\max} = 0.15$ for different sparsities s and extension $N = 64$.

rest of the domain. We also see this in the colored plots, where the red color illustrates the respectively largest errors but is also dominating the whole plot anyway. Here, we want to stress on the fact, that the mean value of both errors is computed for all mesh nodes T_G^+, i.e., the averaging also includes all boundary nodes, where the approximation error is zero anyway. Therefore, the mean value is even smaller than the one computed by averaging only over the inner nodes T_G, where we actually computed our approximations. □

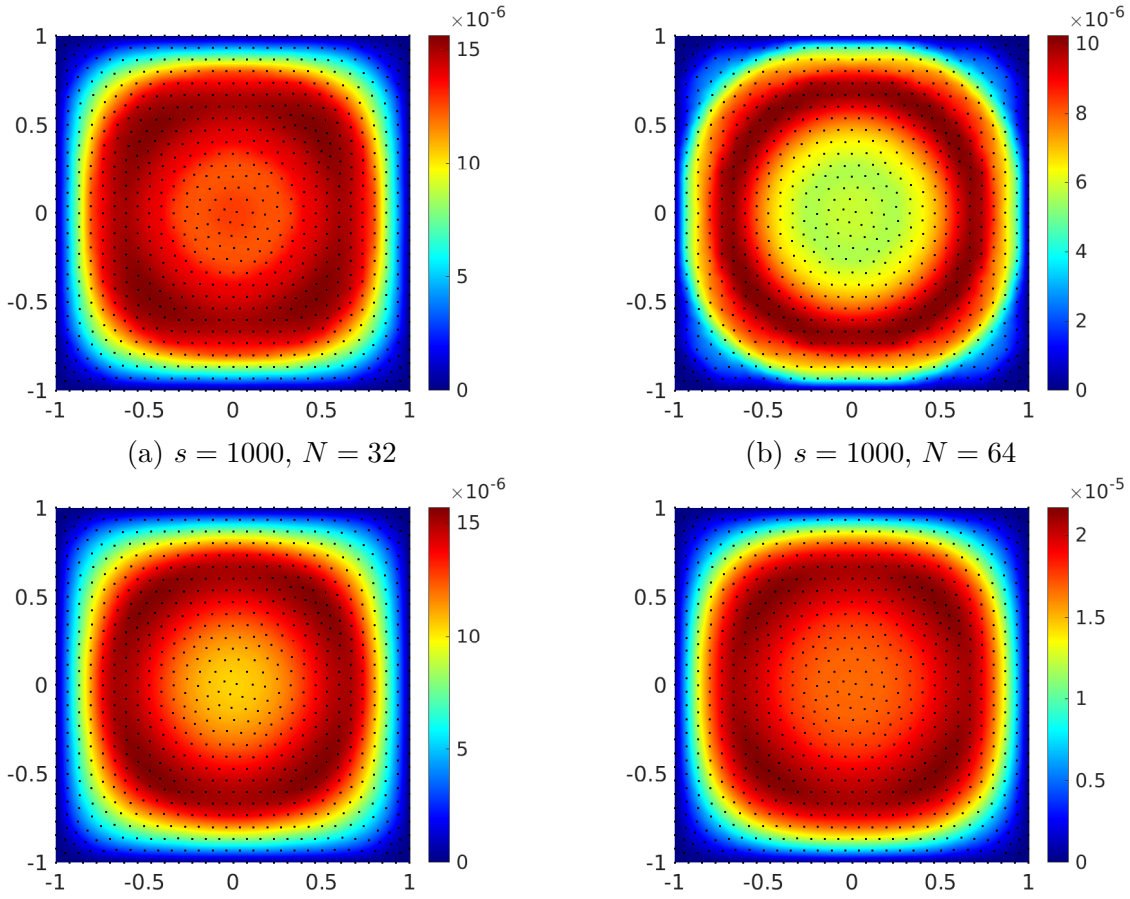
In Section 2.5 we explained, how to compute moments of the solution in general and gave an easy formula for the computation of the expectation value under certain assumptions. Since these assumptions still hold and the given approach is independent of the dimension of the spatial variable $\boldsymbol{x}$, we can use the same formula (2.19) now for the estimation of the expectation value of our two-dimensional approximation.

Example 3.4
We consider an estimation of the expectation value of our approximation $\mathbb{E}\breve{u}^\rho(\boldsymbol{x}_g, \cdot)$ using the formula (2.19) with the $c_{\boldsymbol{k}}$ as defined in (2.20). Again, we compare these estimations with the Monte-Carlo approximation of the expectation value, i.e., with

$$\overline{u_{n_{\mathrm{MC}}}(\boldsymbol{x}_g)} \coloneqq \frac{1}{n_{\mathrm{MC}}} \sum_{i=1}^{n_{\mathrm{MC}}} \breve{u}(\boldsymbol{x}_g, \boldsymbol{\xi}^i)$$

for each $\boldsymbol{x}_g \in \mathcal{T}_G$. The error is then again computed by

$$\mathrm{Res}^\rho(\boldsymbol{x}_g) \coloneqq |\overline{u_{n_{\mathrm{MC}}}(\boldsymbol{x}_g)} - \mathbb{E}\breve{u}^\rho(\boldsymbol{x}_g, \cdot)|. \tag{3.8}$$



(a) $s = 1000$, $N = 32$ (b) $s = 1000$, $N = 64$

(c) $s = 2000$, $N = 32$ (d) $s = 2000$, $N = 64$

Figure 3.8: Expectation value error Res^{ρ} with $n_{\mathrm{MC}} = 10^7$ and $H_{\max} = 0.15$ for the parameter settings $\rho =$ VI, VII, VIII and IX.

Figure 3.8 illustrates this error Res^{ρ} for different parameter settings ρ. The MC approximation used $n_{\mathrm{MC}} = 10^7$ samples and was already shown in Figure 3.2.
The magnitude of these errors is, as expected, slightly smaller than for the averaged pointwise errors. Surprisingly, the setting $\rho =$ IX, i.e., sparsity $s = 2000$ and extension $N = 64$, yields slightly worse results than the settings with smaller parameters. A possible explanation for this can be found in the stagnation for large sparsities, which we have seen in Figures 3.4 and 3.7. The increase from $s = 1000$ to $s = 2000$ showed nearly no effect in each case, i.e., the additional detected frequencies and the corresponding Fourier coefficients did not influence the approximation in a significant way. While these additional terms result in no harm but also nearly no improvement for the case $N = 32$ now, the error Res really suffers from these small deviations in the approximation. So this is another indicator, that we may need to use larger search domains, i.e., larger extensions N, when working with this example for these high sparsities s. □

Remark 3.3
In Section 2.4.3 we mentioned, that the uniform sFFT detects more than sparsity s frequencies due to the union of the sets of detected frequencies in each dimension-increment. For the one-dimensional example, the algorithm terminated with roughly $2s$ Fourier coefficients, cf. Remark 2.2.
However, the current example, i.e. the PDE (3.1) with the random coefficient (3.4), shows a similar behaviour. The amount of detected frequencies in our tests is $c \cdot s$, with $c \in [1.7, 1.9]$ for the different sparsities s, extensions N and the two used refinement parameters $H_{\max}$. So the factor c tends to be even smaller now than for the one-dimensional example, where we found c to be slightly larger than 2 most of the time. □

3.2.2 A random coefficient from a cosine product space

Remember, that we are talking about random coefficients of the general form

$$a(\boldsymbol{x}, \boldsymbol{\xi}) = a_0(\boldsymbol{x}) + \sum_{j=1}^{d_{\boldsymbol{\xi}}} \xi_j \, a_j(\boldsymbol{x}).$$

In the previous section, the spatial variable $\boldsymbol{x}$ appeared only inside of an Euclidean norm in the coefficients $a_j(\boldsymbol{x})$ and therefore also in the whole random coefficient $a(\boldsymbol{x}, \boldsymbol{\xi})$. We now test our uniform sFFT on another example, where the two components x_1 and x_2 appear separately. Therefore, we use an example from [9], where the coefficients $a_j(\boldsymbol{x})$ in the sum are chosen to be

$$a_j(\boldsymbol{x}) \coloneqq \frac{\bar{\alpha}}{j^{\mu}} \cos(2\pi\beta_1(j)\, x_1) \cos(2\pi\beta_2(j)\, x_2) \tag{3.9}$$

with $\mu > 1$ and some $0 < \bar{\alpha} < 1/\zeta(\mu)$, where ζ denotes the Riemann zeta function. We choose $\mu = 2$ to receive a similar decay as in (3.4). Further, we work with the choices $\bar{\alpha} = 0.9/\zeta(2) \approx 0.547$ and $a_0 \equiv 1$ to ensure the well-posedness of the problem. Finally, the β_i are defined as

$$\beta_1(j) \coloneqq j - \frac{k(j)(k(j)+1)}{2} \quad \text{and} \quad \beta_2(j) \coloneqq k(j) - \beta_1(j)$$

with $k(j) \coloneqq \lfloor -1/2 + \sqrt{1/4 + 2j} \rfloor$. Further notes on these coefficients and the parameter choices can be found in [9, Section 11]. The random variable $\boldsymbol{\xi}$ is still assumed to be uniformly distributed in $d_{\boldsymbol{\xi}} = 20$ dimensions as before. The influence of the non-radial structure of the random coefficient $a(\boldsymbol{x}, \boldsymbol{\xi})$ can be seen in Figure 3.9.

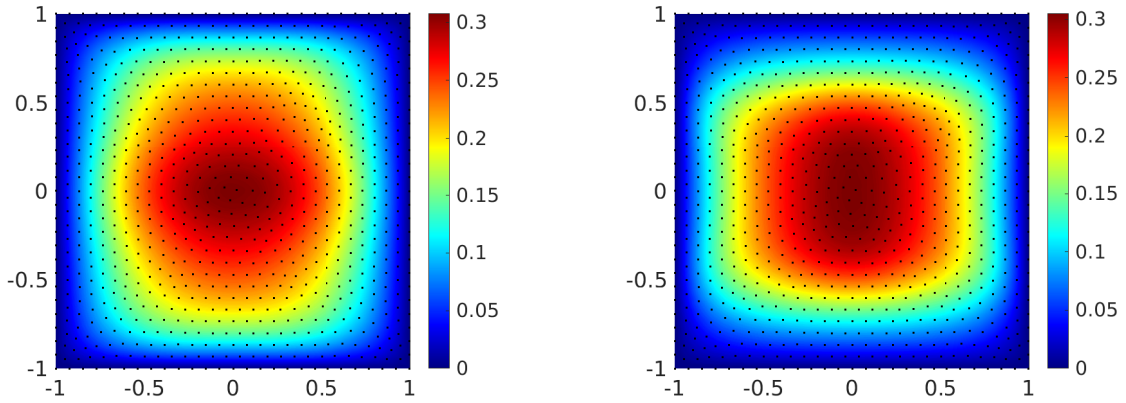
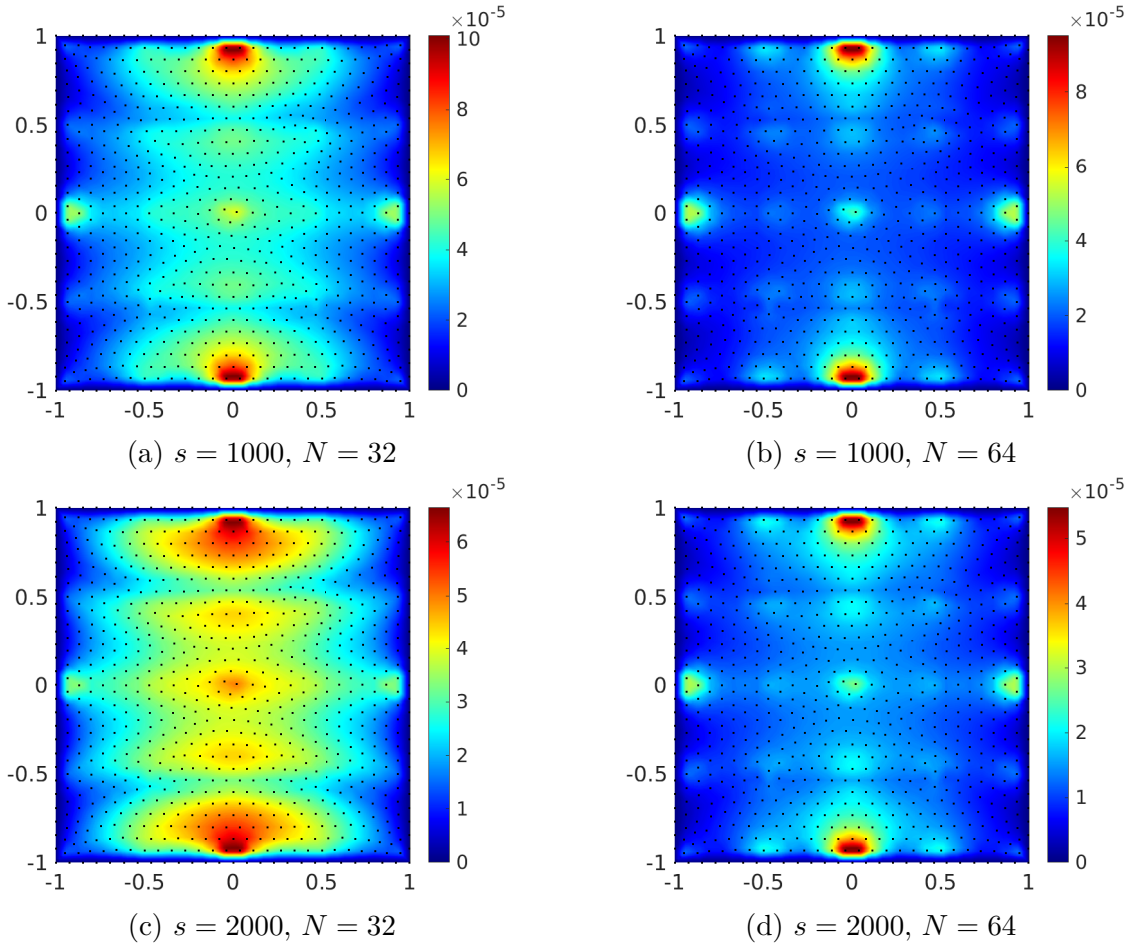


Figure 3.9: Two solutions of the PDE (3.1) using the random coefficient $a(\boldsymbol{x}, \boldsymbol{\xi})$ with (3.9) for different, randomly chosen $\boldsymbol{\xi}$.



(a) $s = 1000$, $N = 32$

(b) $s = 1000$, $N = 64$

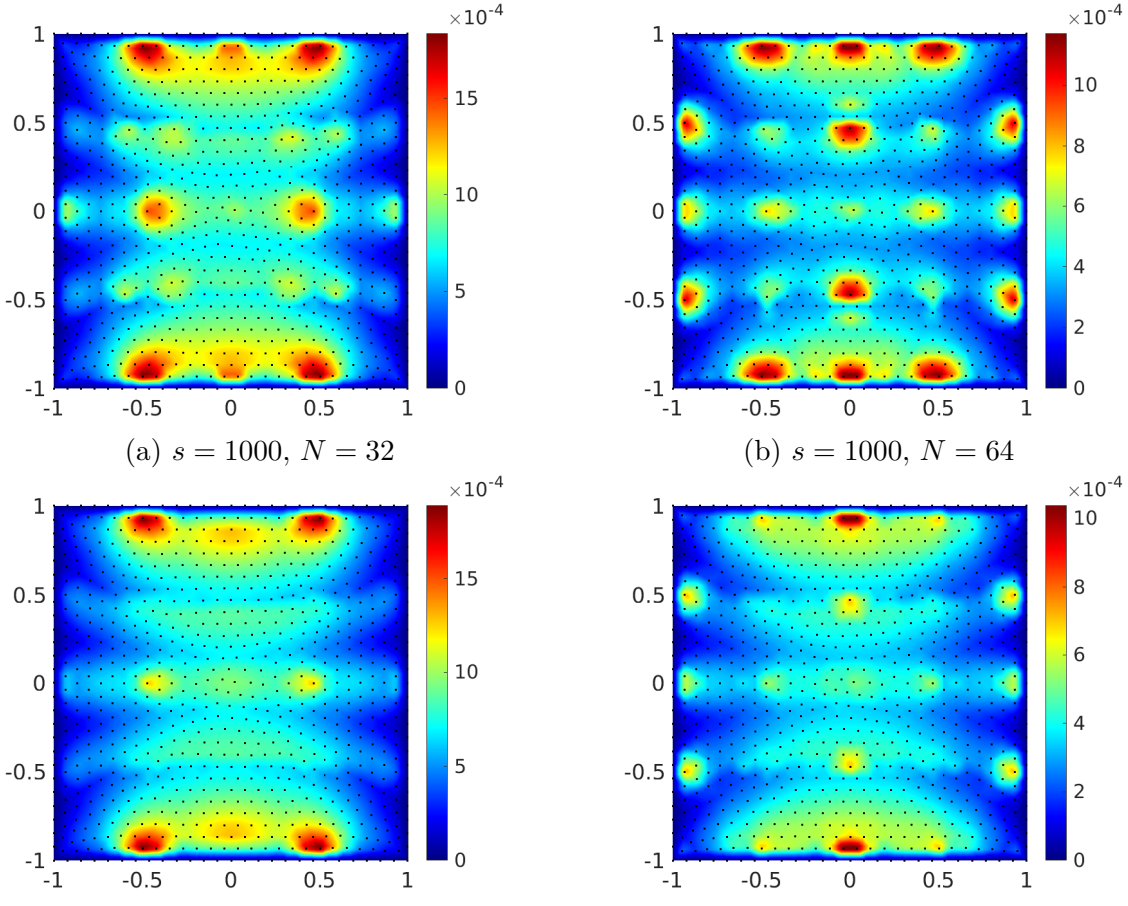
(c) $s = 2000$, $N = 32$

(d) $s = 2000$, $N = 64$

Figure 3.10: Averaged pointwise error Err^{ρ} with $n_{\mathrm{test}} = 10000$, $H_{\max} = 0.15$ and parameter settings $\rho =$ VI - IX.

Example 3.5

Figure 3.10 again illustrates the averaged pointwise error Err^{ρ} as given in (3.5). First of all, we notice a completely different structure of the approximation error than before. There are certain peaks, where the error Err^{ρ} is visibly larger than on the rest of the domain. The highest peaks are very close to the boundary around $(0, \pm 1)$ and $(\pm 1, 0)$. So we see, that the non-radial structure of the random coefficient $a(\boldsymbol{x}, \boldsymbol{\xi})$ results in solutions, that behave completely different w.r.t. the random variable $\boldsymbol{\xi}$.
The magnitude of the approximation error Err^{ρ} behaves similar w.r.t. the parameters s and N as in Section 3.2.1. Especially off the peaks, the approximation is very good, resulting in a small error on most parts of the domain. □



(a) $s = 1000$, $N = 32$ (b) $s = 1000$, $N = 64$

(c) $s = 2000$, $N = 32$ (d) $s = 2000$, $N = 64$

Figure 3.11: Maximum pointwise error MaxErr^{ρ} with $n_{\mathrm{test}} = 10000$, $H_{\max} = 0.15$ and parameter settings $\rho =$ VI - IX.

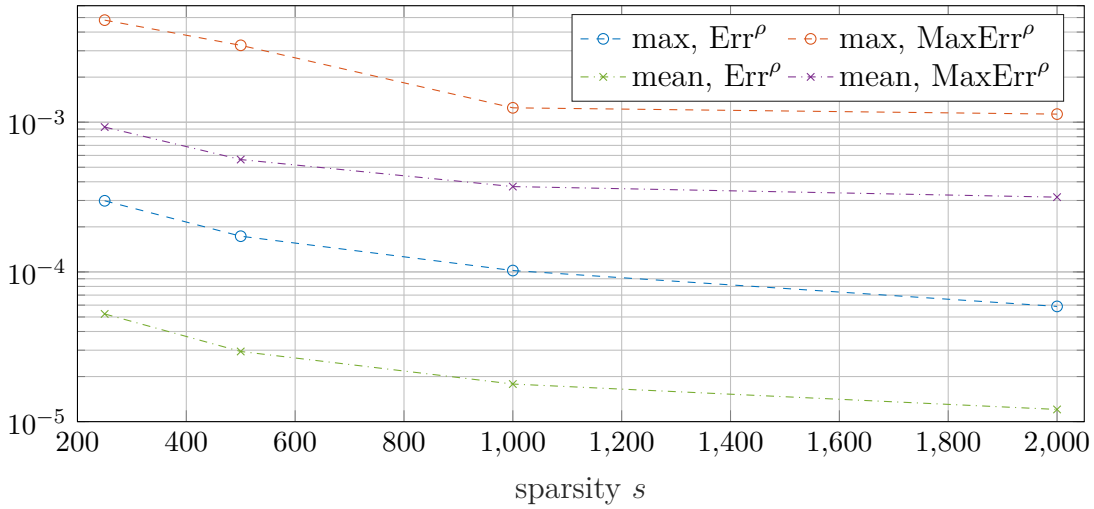


Figure 3.12: The maximum and mean value of the maximum pointwise errors MaxErr^ρ and the averaged pointwise errors Err^ρ with $n_{\text{test}} = 10000$, $H_{\max} = 0.15$ for different sparsities s and extension $N = 64$.

Example 3.6
We consider the maximum pointwise error MaxErr^ρ as given in (3.7). We see a similiar behavior as for the averaged pointwise error Err^ρ. There we saw the four larger error peaks around $(0, \pm 1)$ and $(\pm 1, 0)$ and several small peaks, especially in the case $N = 64$. In the plots of MaxErr^ρ, we see all of these peaks more clearly. They are roughly located at the intersections of the lines $x_i = 0$ and $x_i = \pm 0.5$ with $i = 1, 2$. Nevertheless, the largest peaks appear near the boundaries $x_2 = \pm 1$, which is similar to our observation in Example 3.5. When increasing the sparsity from $s = 1000$ to $s = 2000$, the maximum value of the errors does not change that much, but the approximations off the main peaks gets way better and results in smaller errors on most parts of the domain. It remains to be investigated, if a refinement of the FE mesh, especially near those peaks, can also reduce the errors even further.
Figure 3.12 again compares the maximum and averaged values of the maximum pointwise errors MaxErr^ρ and the averaged pointwise errors Err^ρ w.r.t. the spatial nodes $\boldsymbol{x}_g$. We see, that the maximum values are much larger than the averaged values of both error estimations. This is matching our observations, that the error at the observed peaks is way larger than on the rest of the domain.
We notice, that the relative difference between MaxErr^ρ and Err^ρ is very similar to the one we observed in Example 3.3. So the overall structure of the random coefficient might be a little more difficult, but the influence of the random variable $\boldsymbol{\xi}$ and the spread resulting from the randomness behave very similar. □

Finally, we take a look at the approximation of the expectation value of this solution as well.

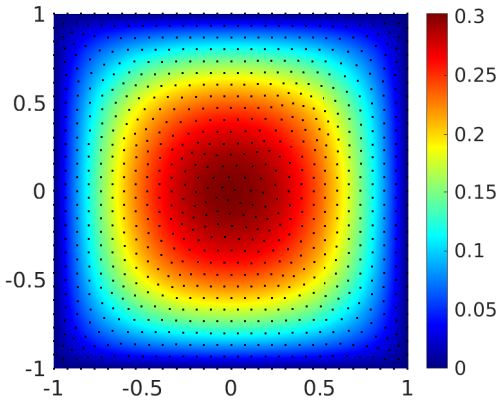
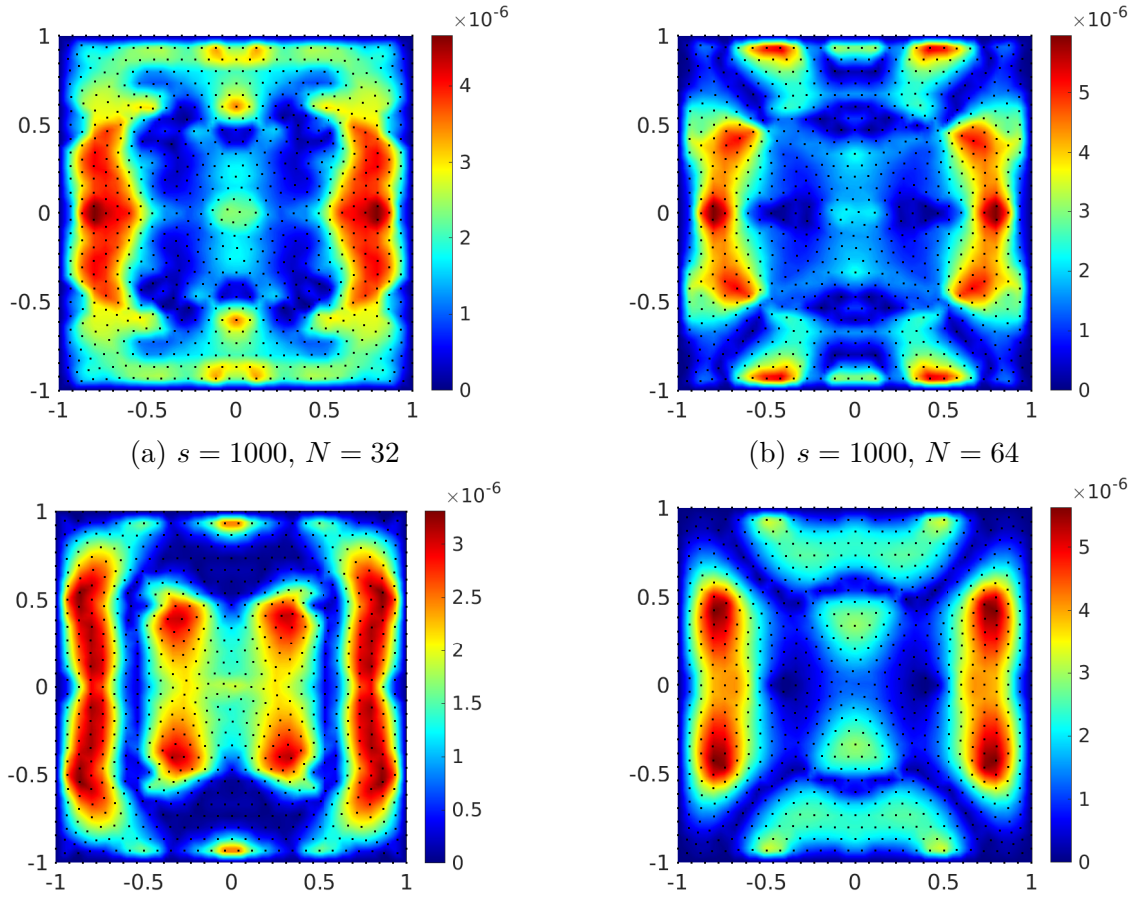


Figure 3.13: The Monte-Carlo approximation of the expectation value of the solution of the PDE using the random coefficient $a(\boldsymbol{x}, \boldsymbol{\xi})$ with (3.9) and $n_{\mathrm{MC}} = 10^7$.

Example 3.7

We consider the error estimation Res^{ρ}, cf. (3.8), as before in Example 3.4. Again, we used $n_{\mathrm{MC}} = 10^7$ samples to compute the MC approximation $\overline{u_{n_{\mathrm{MC}}}}$ shown in Figure 3.13.
We observe a different structure of the expectation value error Res^{ρ} in Figure 3.14 than for the pointwise errors before. The peaks near $(0, \pm 1)$, that we saw in the previous examples, are more inconspicuous and especially for the case $N = 64$ they nearly vanished. The smaller peaks near $(\pm 0.5, \pm 1)$ are still visible for $N = 64$. On the other hand, the areas around $x_1 = \pm 0.8$ now yield the largest approximation errors in each case. These areas are also not really matching the smaller peaks we noticed in the former considerations. This observation could suggest, that at these areas the pointwise approximation error is more biased. Also, we need to consider, that our comparison value is not the exact expectation value but a Monte-Carlo approximation. Therefore, smaller errors in this Monte-Carlo approximation could also be a reason for the observed behavior.
Nevertheless, the error estimation Res^{ρ} is very small, so we are already working with a good approximation of the expectation value. Surprisingly, the maximum value of this error estimation grows when moving from $N = 32$ to $N = 64$. Moreover, this increase of the error size only applies to some of the red regions, where the error was largest anyway. On the other hand, many other areas seem to yield a better approximation, e.g., for $s = 2000$ the red and yellow areas with larger errors around $x_1 = \pm 0.35$ seem to disappear completely in the case $N = 64$. The increase of the search domain Γ leads to the detection of different frequencies in our uniform sFFT. In Figure 3.12 we saw, that for this particular PDE and our parameters the stagnation did not start yet or at least not as much as in Section 3.2.1, cf. Figure 3.7. It seems, that the new frequencies manage to yield a better approximation for some areas as the ones around $x_1 = \pm 0.35$, but also a slightly worse approximation for some other areas, especially around $x_1 = \pm 0.8$. This again indicates, that currently the sparsity s is the restricting parameter and should be chosen a little bit larger for these extensions N. The increase of the maximum value of the approximation



(a) $s = 1000$, $N = 32$

(b) $s = 1000$, $N = 64$

(c) $s = 2000$, $N = 32$

(d) $s = 2000$, $N = 64$

Figure 3.14: Expectation value error Res^{ρ} with $n_{\text{MC}} = 10^7$ and $H_{\max} = 0.15$ for the parameter settings $\rho =$ VI, VII, VIII and IX.

error Res^{ρ} is about $2 \cdot 10^{-6}$. Comparing the mean values of the errors w.r.t. the spatial nodes x_g, we observe slightly larger values for $N = 64$ compared to $N = 32$. The difference is in the order of magnitude of 10^{-7}. □

Remark 3.4

As in Remark 3.3, we finally take a look at the total number of frequencies detected for the PDE with the random coefficient as defined in Section 3.2.2.

We denote the amount of detected frequencies in our tests with $c \cdot s$ again. The factor c ranges from 2.46 to 2.54 this time. This larger c is a result of the more complicated structure of the random coefficient $a(\boldsymbol{x}, \boldsymbol{\xi})$. The different functions $u(\boldsymbol{x}_g, \cdot)$, $g = 1, ..., G$, are not as similar as for the random coefficient defined in (3.4). Therefore, the detected index sets in each dimension-increment of our algorithm share some less frequencies, resulting in a larger number of total frequencies when joining them. □

4 Summary & Outlook

We show, that the uniform sFFT is a powerful tool for solving differential equations with random coefficients. Our numerical tests showed, that even for small sparsities and search domain parameters good approximations can be achieved. Moreover, we noticed, that we need to choose these parameters carefully, since too large sparsities s show no effect when the extension N and therefore the search domain Γ is too small and vice versa. This is one example for a basic property of the sFFT that still holds true for our uniform sFFT. Other properties and the influence of other sFFT parameters, as for example the number of detection iterations r, will probably also have similar effects as in the sFFT, but were not tested and studied in our examples. Further, we want to stress on the fact that our algorithm automatically reveals detailed characteristics of the random variables. These information, e.g., the influence of each random variable ξ_j on the solution or the interactions between these random variables, are very meaningful and can be computed by our algorithm with reasonable computational cost.
One of the most important steps in our approach is the periodization of the random variable $\boldsymbol{\xi}$. We presented some general assumptions on suitable periodizations and three particular periodization mappings in Section 2.2, but the tent transformation was the only periodization mapping used w.r.t. the random variable $\boldsymbol{\xi}$ in our numerical tests. Although this choice simplifies some formulas and computations, e.g., the computation of the moments of the solution, there might be better alternatives due to the lack of smoothness of the tent transform. Therefore, smoother periodization mappings are highly interesting and should be subject of further tests and research on this algorithm.
In this work, we only considered the diffusion equation with random coefficients $a(\boldsymbol{x}, \boldsymbol{\xi})$ of a particular form and on the domain $D = [-1, 1]^d$, $d = 1, 2$. As mentioned before, our algorithm is not restricted to these particular assumptions due to the non-intrusive approach. Approximations on more complicated domains, e.g., circular or L-shaped domains, only require a suitable PDE solver and can easily be studied. Another interesting topic is the type of the random coefficient $a(\boldsymbol{x}, \boldsymbol{\xi})$. Here, we used the linear model $a(\boldsymbol{x}, \boldsymbol{\xi}) = a_0(\boldsymbol{x}) + \sum \xi_j a_j(\boldsymbol{x})$. Other models, e.g., log-normal random coefficients as in [11] or periodic random coefficients as in [13], should be studied with this algorithm as well. The influence of the random variables ξ_j and interactions between them can be revealed by our algorithm and give deeper insight on the used models.
A crucial point in the uniform sFFT is the union of the detected frequencies in each dimension-increment. We saw, that these frequencies were very similar for the different grid points $\boldsymbol{x}_g$ and therefore resulted in only a small increase in the number of output frequencies. It remains to be investigated, if this increase in the amount of detected frequencies can be bounded under certain assumptions. So for example, is it possible to prove that the uniform sFFT detects the same frequencies in each dimension-increment up to a certain variation if the functions $u(\boldsymbol{x}_g, \cdot)$ for the different grid points $\boldsymbol{x}_g$ are known to be in certain function spaces? Such theoretical results are not shown yet, but will be very important to gain a better understanding of the advantageous behavior of our algorithm.
Further, the uniform sFFT can be generalized on functionals. In this work, we considered

the evaluation of the solution u on certain points $\boldsymbol{x}_g$. In other words, we used the evaluation functionals $e_{\boldsymbol{x}_g}$ with $e_{\boldsymbol{x}_g}(u(\boldsymbol{\xi})) \coloneqq u(\boldsymbol{x}_g, \boldsymbol{\xi})$. We can generalize this approach by using other and more complicated functionals F_k on the solution u instead. As before, we have to take care that these functionals F_g are of a similar structure to end up with a reasonable union of the detected frequencies in the dimension-incremental steps. Then, we end up with approximations of all the F_g with one use of the uniform sFFT. Additionally, we then may also investigate the detailed information on the influence of the random variable $\boldsymbol{\xi}$ on these functionals.
The number of used samples and the computational cost of our uniform sFFT are significantly smaller than for applying the sFFT G times separately, i.e., at each point $\boldsymbol{x}_g \in \mathcal{T}_G$. Nevertheless, any further decrease of these attributes comes in handy. Especially the number of used samples plays an important role, since the call of the used differential equation solver is very expensive in general. Therefore, the majority of the computation time of our uniform sFFT is used in the sampling step. Hence, the application of very efficient differential equation solvers is highly recommended. We already reduced the number of used samples in this work by using random rank-1 lattices instead of single or multiple rank-1 lattices, cf. Sections 2.4.2 and 2.7. Further methods to decrease the number of sampling nodes without increasing the approximation error too much are desirable. And since the call of the underlying PDE solver can be parallelized, the usage of multiple workers also reduces the computation time by a significant amount if available.

References

[1] M. Bachmayr, A. Cohen, and W. Dahmen. "Parametric PDEs: Sparse or Low-Rank Approximations?" In: *IMA Journal of Numerical Analysis* 38 (July 2016). DOI: 10.1093/imanum/drx052.

[2] M. Bachmayr, A. Cohen, and G. Migliorati. "Representations of Gaussian Random Fields and Approximation of Elliptic PDEs with Lognormal Coefficients". In: *Journal of Fourier Analysis and Applications* 24 (June 2018). DOI: 10.1007/s00041-017-9539-5.

[3] M. Bochmann. "Hochdimensionale Fourier-Methoden für Differentialgleichungen mit zufälligen Koeffizienten". Bachelor Thesis. TU Chemnitz, 2017.

[4] M. Bochmann, L. Kämmerer, and D. Potts. "A sparse FFT approach for ODE with random coefficients". In: *Advances in Computational Mathematics* 46 (July 2020). DOI: 10.1007/s10444-020-09807-w.

[5] J.-L. Bouchot, H. Rauhut, and C. Schwab. *Multi-level Compressed Sensing Petrov-Galerkin discretization of high-dimensional parametric PDEs*. Dec. 2017. arXiv: 1701.01671v2 [math.NA].

[6] A. Cohen, R. DeVore, and C. Schwab. "Convergence Rates of Best N-term Galerkin Approximations for a Class of Elliptic sPDEs". In: *Foundations of Computational Mathematics* 10 (Dec. 2010), pp. 615–646. DOI: 10.1007/s10208-010-9072-2.

[7] J. Dick, F. Kuo, Q. Le Gia, and C. Schwab. "Multilevel Higher Order QMC Petrov–Galerkin Discretization for Affine Parametric Operator Equations". In: *SIAM Journal on Numerical Analysis* 54 (Jan. 2016), pp. 2541–2568. DOI: 10.1137/16M1078690.

[8] J. Dick, Q. Le Gia, and C. Schwab. "Higher Order Quasi–Monte Carlo Integration for Holomorphic, Parametric Operator Equations". In: *SIAM/ASA Journal on Uncertainty Quantification* 4 (Sept. 2014). DOI: 10.1137/140985913.

[9] M. Eigel, C. Gittelson, C. Schwab, and E. Zander. "Adaptive stochastic Galerkin FEM". In: *Computer Methods in Applied Mechanics and Engineering* 270 (Mar. 2014), pp. 247–269. DOI: 10.1016/j.cma.2013.11.015.

[10] R. Gantner, L. Herrmann, and C. Schwab. "Multilevel QMC with Product Weights for Affine-Parametric, Elliptic PDEs". In: *Contemporary Computational Mathematics - A Celebration of the 80th Birthday of Ian Sloan* (May 2018), pp. 373–405. DOI: 10.1007/978-3-319-72456-0_18.

[11] I. Graham, F. Kuo, J. Nichols, R. Scheichl, C. Schwab, and I. Sloan. "Quasi-Monte Carlo finite element methods for elliptic PDEs with log-normal random coefficients". In: *SAM Report, ETH Zurich* 2013-14 (May 2013). DOI: 10.1007/s00211-014-0689-y.

[12] V. Kaarnioja, Y. Kazashi, F. Kuo, F. Nobile, and I. Sloan. *Fast approximation by periodic kernel-based lattice-point interpolation with application in uncertainty quantification*. July 2020. arXiv: 2007.06367 [math.NA].

[13] V. Kaarnioja, F. Kuo, and I. Sloan. "Uncertainty Quantification Using Periodic Random Variables". In: *SIAM Journal on Numerical Analysis* 58 (Jan. 2020), pp. 1068–1091. DOI: 10.1137/19M1262796.

[14] L. Kämmerer. "Reconstructing Multivariate Trigonometric Polynomials from Samples Along Rank-1 Lattices". In: *Approximation Theory XIV: San Antonio 2013* (Jan. 2014), pp. 255–271. DOI: 10.1007/978-3-319-06404-8_14.

[15] L. Kämmerer, F. Krahmer, and T. Volkmer. *A sample efficient sparse FFT for arbitrary frequency candidate sets in high dimensions*. June 2020. arXiv: 2006.13053 [math.NA].

[16] L. Kämmerer, D. Potts, and T. Volkmer. "High-dimensional sparse FFT based on sampling along multiple rank-1 lattices". In: *Applied and Computational Harmonic Analysis* 51 (Nov. 2017). DOI: 10.1016/j.acha.2020.11.002.

[17] F. Kuo, C. Schwab, and I. Sloan. "Multi-level Quasi-Monte Carlo Finite Element Methods for a Class of Elliptic PDEs with Random Coefficients". In: *Foundations of Computational Mathematics* 15 (Jan. 2015), pp. 411–449. DOI: 10.1007/s10208-014-9237-5.

[18] F. Kuo, C. Schwab, and I. Sloan. "Quasi-Monte Carlo Finite Element Methods for a Class of Elliptic Partial Differential Equations with Random Coefficients". In: *SIAM Journal on Numerical Analysis* 50 (Jan. 2012), pp. 3351–3374. DOI: 10.1137/110845537.

[19] D. Potts and M. Tasche. "Parameter estimation for multivariate exponential sums". In: *Electronic Transactions on Numerical Analysis* 40 (Jan. 2013), pp. 204–224.

[20] D. Potts and T. Volkmer. "Sparse high-dimensional FFT based on rank-1 lattice sampling". In: *Applied and Computational Harmonic Analysis* 41 (June 2015). DOI: 10.1016/j.acha.2015.05.002.

[21] C. Schwab. "QMC Galerkin Discretization of Parametric Operator Equations". In: *Springer Proceedings in Mathematics and Statistics* 65 (Jan. 2013), pp. 613–629. DOI: 10.1007/978-3-642-41095-6_32.

TECHNISCHE UNIVERSITÄT CHEMNITZ

Studentenservice – Zentrales Prüfungsamt

Selbstständigkeitserklärung

Name: Vorname: geb. am: Matr.-Nr.:	<u>Bitte beachten:</u> 1. Bitte binden Sie dieses Blatt am Ende Ihrer Arbeit ein.

Selbstständigkeitserklärung*

Ich erkläre gegenüber der Technischen Universität Chemnitz, dass ich die vorliegende selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Datum: ……. Unterschrift: ……

* Statement of Authorship

I hereby certify to the Technische Universität Chemnitz that this thesis is all my own work and uses no external material other than that acknowledged in the text.

This work contains no plagiarism and all sentences or passages directly quoted from other people's work or including content derived from such work have been specifically credited to the authors and sources.

This paper has neither been submitted in the same or a similar form to any other examiner nor for the award of any other degree, nor has it previously been published.