Fast Sparse Pseudo-spectral Methods for High-dimensional Problems

Vasil Velikov

Born 13th May 1988 in Dobrich, Bulgaria

15th June 2016

Master's Thesis Mathematics

Advisor: Prof. Dr. Michael Griebel

Second Advisor: Prof. Dr. Jochen Garcke

Mathematical Institute

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Contents

List of Figures

Notation and List of Symbols

n	The dimension of the grid
i, j, k, μ, ν	One-dimensional indices
$\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}, \boldsymbol{\mu}, \boldsymbol{\nu}$	Multidimensional indices
$\boldsymbol{\nu}', \boldsymbol{\nu}''$	Subparts of a multi-index $\boldsymbol{\nu}$, $\boldsymbol{\nu} = (\boldsymbol{\nu}', \boldsymbol{\nu}')$
$\lvert\boldsymbol{\nu}\rvert_1, \lvert\boldsymbol{\nu}\rvert_\infty$	Different multi-index norms
$\boldsymbol{x_i}$	Interpolation points
$I_{\boldsymbol{\nu}}, I_{\mathcal{I}}$	Interpolation operator
$\Delta_{\boldsymbol{\nu}}$	Hierarchical interpolation operator
$\boldsymbol{\sigma}(\boldsymbol{i})$	Index mapping function
$\omega_k, T_k, \mathcal{H}_k^{\alpha,\beta}$	Basis functions for the Fourier, Chebyshev and Hermite transforms
$\phi_{\boldsymbol{k}}$	Multidimensional regular basis function
$\psi_{\boldsymbol{k}}$	Multidimensional hierarchical basis function
$\hat{f}_{\boldsymbol{k}}$	Coefficients in the infinite series expansion
$\hat{f}_{\boldsymbol{k}}^{\boldsymbol{\nu}}$	Discrete regular coefficients
$\check{f}_{\boldsymbol{k}}$	Discrete hierarchical coefficients
$S_{\boldsymbol{k},\boldsymbol{\nu}}$	Index set/multiset used for the summation in the aliasing formulas
g_k	Number of interpolation points in level k
$\mathbb{T}, \mathbb{I}$	Interpolation intervals for the Fourier and Chebyshev transforms
$L^2(\mathbb{T}^n), L_w^2(\mathbb{I}^n)$	Interpolation spaces
$\mathcal{H}_w, \mathcal{H}_{mix}^{t,r}$	Weighted Sobolev space and weighted Sobolev space of mixed smoothness
$X^n, V_{\boldsymbol{\nu}}$	Infinite dimensional Hilbert space and finite-dimensional subspaces
$W_{\boldsymbol{\nu}}$	Hierarchical subspaces of X^n
$S_{\boldsymbol{\nu}}, S_{\boldsymbol{\nu}}^h$	Sets of interpolation points and hierarchical interpolation points
$\mathcal{B}_{\boldsymbol{\nu}}, \mathcal{B}_{\boldsymbol{\nu}}^h$	Sets of regular and hierarchical basis functions
$\mathcal{G}_{\boldsymbol{\nu}}, \mathcal{J}_{\boldsymbol{\nu}}$	Index set and hierarchical index set
$\mathcal{I}, \mathcal{I}_L^T, \mathcal{M}_d(\mathcal{I})$	Sets of level indices and their upper boundary in direction d
z_n, ξ_n	Elements of different Leja sequences
$\mathcal{O}, \mathcal{A}$	Old and active sets in the adaptive algorithm

1 Introduction

Numerical simulation is widely used to solve problems in chemistry, physics, statistics and other scientific fields. Many of them involve a large amount of variables which leads to the need of efficient numerical methods for high-dimensional problems. The first step of a numerical method is the discretization of the underlying function domain. The classic approach is to use a uniform mesh with mesh spacing h in each direction. This leads to a large number of nodes M of order $O(h^{-n})$, where n is the dimension of the problem. The gained approximation accuracy is of order $O(h^s)$, where s is the function's isotropic smoothness. This leads to an exponential decay of the rate of convergence with the growth of the dimension n, a problem commonly referred to as *the curse of dimensionality* [Bel61]. It limits the practical applicability of these discretization methods to only a few dimensions.

A common approach for overcoming this limitation are the so-called sparse grids [Smo63; Zen91]. The basic idea is to use a carefully chosen subset of grid nodes instead of the entire full grid. Regular sparse grids use the grid nodes based on a hyperbolic cross cutoff. The resulting subset has a number of nodes M of order $O(h^{-1}\log(h^{-1})^{n-1})$. Under the assumption that the approximated function has limited t-th mixed derivatives, it can be proven that the convergence rate is $O(h^t\log(h^{-1})^{n-1})$. This estimate greatly improves the ratio between invested computational cost and achieved approximation accuracy. The curse of dimensionality is broken up to a logarithmic term.

The classic setting of regular sparse grids can be extended in many different ways. The hyperbolic cross based selection of grid points can be generalized in order to adjust for different smoothness classes, anisotropicity and other function properties. This choice can be done a priori, based on the function class, or it can be generated with the help of an adaptive algorithm([Heg03; Gar07; GG03]). Thanks to the tensor product construct of sparse grids, it is easy to use different transforms in each direction of the problem. This way the properties of each dimension can be exploited individually. Moreover, interactions between different dimensions can be limited by simply not including grid points that correspond to them. The growth rate of the degrees of freedom can also be controlled more finely by allowing a careful selection of refinement steps in each dimension.

The purpose of this work is to examine several of the possible extensions of sparse grids and to combine them so that they can work together. From a theoretical point of view the extensions are fairly orthogonal, i.e. they can be applied together without special treatment. From an implementation point of view this is not necessarily the case. One of the goals is to implement the methods in an efficient and flexible C library that can make use of any combination of them. The library is called HCFFT and its writing, optimizing and documenting are a substantial part of this thesis.

The main extensions discussed in this text and implemented in HCFFT are:

- Mixed sparse grids, i.e. grids that use different transforms in the different directions. First, a general setting for mixed grids is presented. Thanks to the tensor product structure of sparse grids this setting is straightforward and is mostly notational. Then, a particular type of mixed Fourier-Chebyshev grids is presented and analyzed. Error bounds similar to [GH14] are derived.

- Generalized sparse grids, i.e. grids that allow slower refinement steps. Regular sparse grids use dyadic refinement in each direction. This is the only way to guarantee nestedness of the interpolation points. To achieve smaller refinement steps and consequently higher control over the degrees of freedom, it is necessary to use an alternative set of interpolation points. The Leja and weighted Leja points are the natural choice for this purpose. In the case of weighted Leja sequences, they do not only allow better control but they actually make the very existence of nested Hermite, Jacobi or Laguerre sparse grids possible.

- Adaptive grid construction, i.e. grids which are built based on the properties of the target function. An extension of the algorithm that allows more control over the grid generating process is presented. Different optimizations and common pitfalls are discussed.

The text is organized as follows. Section 2 introduces the transforms that are used for the sparse grids in this work. Apart from simple definitions, it also contains a proof of an aliasing lemma for the coefficients of mixed Fourier-Chebyshev transforms. Section 3 introduces the concept of interpolation on sparse grids in a general way, without fixing the underlying transform or the refinement steps. Section 4 presents two concrete cases of dyadic sparse grids, namely Fourier grids and Clenshaw-Curtis grids. Section 5 discusses alternative interpolation points for several different transforms. Leja sequences and several generalizations are presented. Section 6 presents the notational setting for mixed grids. It also contains an error estimate for the interpolant of mixed Fourier-Chebyshev grids. Section 7 discusses the adaptive algorithm as well as several extensions to it. Section 8 addresses some practical issues that arise while implementing the methods as well as efficient ways to solve them. Finally, section 9 demonstrates the efficiency of the methods by showing the convergence of different classic functions.

2 Preliminaries

2.1 Discrete Fourier Transform

The Discrete Fourier Transform is a broad topic covered by numerous mathematical texts(e.g. [Pey02; Boy01]). This subsection contains a brief overview of the definitions and properties that are used in this work. To a great extent its main goal is to familiarize the reader with the related notation because it often varies between different authors.

Let $\mathbb{T}$ be the interval $[0, 2\pi]$ in which the endpoints are identified. The one-dimensional Fourier basis functions are defined for every $k \in \mathbb{Z}$ with the formula

$$\omega_k(x) := e^{ikx}. \tag{2.1}$$

In the n-dimensional case the Fourier basis functions are defined as a tensor product,

$$\omega_{\boldsymbol{k}}(\boldsymbol{x}) := \Big(\bigotimes_{d=1}^{n} \omega_{k_d}\Big)(\boldsymbol{x}) = \prod_{d=1}^{n} \omega_{k_d}(x_d) \tag{2.2}$$

for every $\boldsymbol{k} \in \mathbb{Z}^n$ and $\boldsymbol{x} \in \mathbb{T}^n$. The functions $\omega_{\boldsymbol{k}}(\boldsymbol{x})$ form a basis of the space $L^2(\mathbb{T}^n)$. Thus, every function f from this space can be expanded in a Fourier series,

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{k} \in \mathbb{Z}^n} \hat{f}_{\boldsymbol{k}} \omega_{\boldsymbol{k}}(\boldsymbol{x}). \tag{2.3}$$

The Fourier series is useful for approximating the function f. This is usually achieved by truncating the series by some criteria. An alternative approach for approximating f is to do it by interpolation. Let $\boldsymbol{\nu} \in \mathbb{N}^n$ be a vector representing the number of basis functions in each dimension. The points that are used for interpolation are

$$\boldsymbol{x_i} := (x_{i_1}, \dots x_{i_n}), \quad x_{i_d} := \frac{2 i_d \pi}{\nu_d} \tag{2.4}$$

for every $i_d \in \{0, 1, \dots \nu_d - 1\}$ and every $d \in \{1, 2, \dots n\}$. The points $\boldsymbol{x_i}$ form a grid of equidistant nodes. Note that there are no points on the right boundary of $\mathbb{T}^n$. The reason for this is the periodicity of the space.

The interpolation operator $I_{\boldsymbol{\nu}}$ is now defined as a mapping

$$I_{\boldsymbol{\nu}} : L^2(\mathbb{T}^n) \to \operatorname{span}\{\omega_{\boldsymbol{\sigma(k)}} : \boldsymbol{0} \le \boldsymbol{k} < \boldsymbol{\nu}\} \tag{2.5}$$

such that for every interpolation point $\boldsymbol{x_i}$ it holds

$$I_{\boldsymbol{\nu}} f(\boldsymbol{x_i}) = f(\boldsymbol{x_i}). \tag{2.6}$$

The function σ is an indexing function whose purpose is to alternate between the positive and negative Fourier frequencies. It is defined by

$$\sigma(l) := \begin{cases} -l/2 & l \text{ is even}, \\ (l+1)/2 & l \text{ is odd}, \end{cases} \tag{2.7}$$

$$\boldsymbol{\sigma(l)} := (\sigma(l_1), \dots, \sigma(l_d)). \tag{2.8}$$

For every function $f \in L^2(\mathbb{T}^n)$ the interpolant $I_{\boldsymbol{\nu}} f$ can be expressed in a unique way as

$$I_{\boldsymbol{\nu}} f = \sum_{\mathbf{0} \le \boldsymbol{i} < \boldsymbol{\nu}} \hat{f}_{\boldsymbol{i}}^{\boldsymbol{\nu}} \omega_{\boldsymbol{\sigma}(\boldsymbol{i})}, \tag{2.9}$$

where $\hat{f}_{\boldsymbol{i}}^{\boldsymbol{\nu}}$ are the discrete Fourier coefficients. They can be computed with the formula

$$\hat{f}_{\boldsymbol{i}}^{\boldsymbol{\nu}} = \nu_1^{-1} \dots \nu_n^{-1} \sum_{\mathbf{0} \le \boldsymbol{k} < \boldsymbol{\nu}} f(\boldsymbol{x}_{\boldsymbol{k}}) \omega^*_{\boldsymbol{\sigma}(\boldsymbol{i})}(\boldsymbol{x}_{\boldsymbol{k}}). \tag{2.10}$$

This formula can easily be derived with the help of the following orthogonality property of the Fourier basis functions:

$$\sum_{\mathbf{0} \le \boldsymbol{i} < \boldsymbol{\nu}} \omega_{\boldsymbol{k}}(\boldsymbol{x}_{\boldsymbol{i}}) \omega^*_{\boldsymbol{l}}(\boldsymbol{x}_{\boldsymbol{i}}) = a_{\boldsymbol{k},\boldsymbol{l}}, \tag{2.11}$$

where

$$a_{\boldsymbol{k},\boldsymbol{l}} := \begin{cases} \nu_1 \dots \nu_n & \text{if } \boldsymbol{l} = \boldsymbol{k} + \boldsymbol{m}\boldsymbol{\nu} \text{ for some } \boldsymbol{m} \in \mathbb{Z}^n, \\ 0 & \text{otherwise.} \end{cases} \tag{2.12}$$

This identity can also be used to find a relation between the Fourier coefficients $\hat{f}_{\boldsymbol{k}}$ and the discrete Fourier coefficients $\hat{f}_{\boldsymbol{k}}^{\boldsymbol{\nu}}$.

Lemma 2.1 (Fourier aliasing formula)**.** *The following identity holds*

$$\hat{f}_{\boldsymbol{k}}^{\boldsymbol{\nu}} = \sum_{\boldsymbol{m} \in \mathbb{Z}^n} \hat{f}_{\boldsymbol{\sigma}(\boldsymbol{k}) + \boldsymbol{m}\boldsymbol{\nu}} \tag{2.13}$$

for every $\mathbf{0} \le \boldsymbol{k} < \boldsymbol{\nu}$*.*

Proof. The proof follows easily by applying equations (2.10), (2.3) and (2.11) in this order:

$$\begin{aligned} \hat{f}_{\boldsymbol{i}}^{\boldsymbol{\nu}} &= \nu_1^{-1} \dots \nu_n^{-1} \sum_{\mathbf{0} \le \boldsymbol{k} < \boldsymbol{\nu}} f(\boldsymbol{x}_{\boldsymbol{k}}) \omega^*_{\boldsymbol{\sigma}(\boldsymbol{i})}(\boldsymbol{x}_{\boldsymbol{k}}) \\ &= \nu_1^{-1} \dots \nu_n^{-1} \sum_{\mathbf{0} \le \boldsymbol{k} < \boldsymbol{\nu}} \omega^*_{\boldsymbol{\sigma}(\boldsymbol{i})}(\boldsymbol{x}_{\boldsymbol{k}}) \sum_{\boldsymbol{l} \in \mathbb{Z}^n} \hat{f}_{\boldsymbol{l}} \omega_{\boldsymbol{l}}(\boldsymbol{x}_{\boldsymbol{k}}) \\ &= \nu_1^{-1} \dots \nu_n^{-1} \sum_{\boldsymbol{l} \in \mathbb{Z}^n} \hat{f}_{\boldsymbol{l}} \sum_{\mathbf{0} \le \boldsymbol{k} < \boldsymbol{\nu}} \omega^*_{\boldsymbol{\sigma}(\boldsymbol{i})}(\boldsymbol{x}_{\boldsymbol{k}}) \omega_{\boldsymbol{l}}(\boldsymbol{x}_{\boldsymbol{k}}) \\ &= \nu_1^{-1} \dots \nu_n^{-1} \sum_{\boldsymbol{l} \in \mathbb{Z}^n} \hat{f}_{\boldsymbol{l}} a_{\boldsymbol{\sigma}(\boldsymbol{k}),\boldsymbol{l}} \\ &= \sum_{\boldsymbol{m} \in \mathbb{Z}^n} \hat{f}_{\boldsymbol{\sigma}(\boldsymbol{i}) + \boldsymbol{m}\boldsymbol{\nu}} \end{aligned} \tag{2.14}$$

□

The aliasing formula is used for the derivation of error bounds for dyadic Fourier sparse grids. Although its proof is straightforward, it serves as a good starting point for more complicated aliasing formulas that are proven later in this text. To unify the notation with them, the following alternative formulation of Lemma 2.1 will be used:

$$\hat{f}_{\boldsymbol{k}}^{\boldsymbol{\nu}} = \sum_{\boldsymbol{i} \in S_{\boldsymbol{k},\boldsymbol{\nu}}} \hat{f}_{\boldsymbol{i}}, \tag{2.15}$$

where the index set $S_{\boldsymbol{k},\boldsymbol{\nu}}$ is defined as $\{\boldsymbol{\sigma}(\boldsymbol{k}) + \boldsymbol{m}\boldsymbol{\nu} : \boldsymbol{m} \in \mathbb{Z}^n\}$.

2.2 Discrete Chebyshev Transform

The next set of basis functions which is used extensively in this work is the set of Chebyshev polynomials. Let $\mathbb{I}$ denote the closed interval $[-1, 1]$. The Chebyshev polynomials of the first kind T_n are defined for every $x \in \mathbb{I}$ and every $k \in \mathbb{N}_0$ with the formula

$$T_k(x) := \cos(k \cos^{-1}(x)). \tag{2.16}$$

They are orthogonal with respect to the weight function $w(x) = (1 - x^2)^{-1/2}$. In the n-dimensional case $T_{\boldsymbol{k}}$ is defined as a tensor product of one-dimensional basis functions,

$$T_{\boldsymbol{k}}(\boldsymbol{x}) := \Big(\bigotimes_{d=1}^{n} T_{k_d} \Big)(\boldsymbol{x}) = \prod_{d=1}^{n} T_{k_d}(x_d) \tag{2.17}$$

for every $\boldsymbol{k} \in \mathbb{N}_0^n$ and every $\boldsymbol{x} \in \mathbb{I}^n$. The Chebyshev polynomials form an orthogonal basis of the weighted space

$$L_w^2(\mathbb{I}^n) := \left\{ f : \int_{\mathbb{I}^n} f(\boldsymbol{x})^2 \boldsymbol{w}(\boldsymbol{x}) d\boldsymbol{x} < \infty \right\} \tag{2.18}$$

where $\boldsymbol{w}(\boldsymbol{x}) = \prod_{d=1}^{n} (1 - x_d^2)^{-1/2}$. Every function f in this space can be expanded in an infinite series

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{k} \in \mathbb{N}_0^n} \hat{f}_{\boldsymbol{k}} T_{\boldsymbol{k}}(\boldsymbol{x}), \tag{2.19}$$

where $\hat{f}_{\boldsymbol{k}}$ are the uniquely defined Chebyshev coefficients.

This work is focused on interpolation with Chebyshev polynomials. Let $\boldsymbol{\nu} \in \mathbb{N}_0^n$ be a vector representing the number of interpolation points in each dimension. The points that are used are the extrema of the Chebyshev polynomials, i.e.

$$\boldsymbol{x_i} := (x_{i_1}, \dots x_{i_n}), \quad x_{i_d} := \cos \frac{\pi i_d}{\nu_d} \tag{2.20}$$

for every $i_d \in \{0, 1, \dots \nu_d\}$ and every $d \in \{1, 2, \dots n\}$. Note that in this case the underlying domain is not periodic and therefore both endpoints of the interval $\mathbb{I}$ are included.

We can now define an interpolation operator $I_{\boldsymbol{\nu}}$ as a mapping

$$I_{\boldsymbol{\nu}} : L_w^2(\mathbb{I}^n) \to \operatorname{span}\{T_{\boldsymbol{k}} : \boldsymbol{0} \le \boldsymbol{k} \le \boldsymbol{\nu}\} \tag{2.21}$$

such that for every function $f \in L_w^2(\mathbb{I}^n)$ and every interpolation point $\boldsymbol{x_i}$ it holds

$$I_{\boldsymbol{\nu}} f(\boldsymbol{x_i}) = f(\boldsymbol{x_i}). \tag{2.22}$$

The result of this operator can be represented uniquely in the form

$$I_{\boldsymbol{\nu}} f = \sum_{\boldsymbol{0} \le \boldsymbol{i} \le \boldsymbol{\nu}} \hat{f}_{\boldsymbol{i}}^{\boldsymbol{\nu}} T_{\boldsymbol{i}}, \tag{2.23}$$

where $\hat{f}_{\boldsymbol{i}}^{\boldsymbol{\nu}}$ are the discrete Chebyshev coefficients. There is an explicit formula for their computation:

$$\hat{f}_{\boldsymbol{i}}^{\boldsymbol{\nu}} = 2^n \nu_1^{-1} \dots \nu_n^{-1} c_{\boldsymbol{i}}^{-1} \sum_{\boldsymbol{0} \le \boldsymbol{k} \le \boldsymbol{\nu}} c_{\boldsymbol{k}}^{-1} f(\boldsymbol{x_k}) T_{\boldsymbol{i}}(\boldsymbol{x_k}), \tag{2.24}$$

where the constants $c_{\boldsymbol{i}}$ are defined as

$$c_{\boldsymbol{i}} := \prod_{d=1}^{n} c_{i_d}, \quad c_{i_d} := \begin{cases} 2 & i_d = 0, \\ 1 & 0 < i_d < \nu_d, \\ 2 & i_d = \nu_d. \end{cases} \tag{2.25}$$

The discrete orthogonality relation

$$\sum_{\mathbf{0} \le \boldsymbol{i} \le \boldsymbol{\nu}} c_{\boldsymbol{i}}^{-1} T_{\boldsymbol{k}}(\boldsymbol{x}_{\boldsymbol{i}}) T_{\boldsymbol{l}}(\boldsymbol{x}_{\boldsymbol{i}}) = 2^{-n} \nu_1 \dots \nu_n c_{\boldsymbol{k}} \delta_{\boldsymbol{k},\boldsymbol{l}} \tag{2.26}$$

holds for every $\mathbf{0} \le \boldsymbol{k}, \boldsymbol{l} \le \boldsymbol{\nu}$, where $\delta_{\boldsymbol{k},\boldsymbol{l}} = 1$ if $\boldsymbol{l} = \boldsymbol{k}$ and equals $\delta_{\boldsymbol{k},\boldsymbol{l}} = 0$ in all other cases. Just as in the Fourier case, this formula can be used to find a relation between the Chebyshev series coefficients $\hat{f}_{\boldsymbol{k}}$ and the discrete coefficients $\hat{f}^{\boldsymbol{\nu}}_{\boldsymbol{k}}$. This relation is slightly more complicated compared to the Fourier transform. Thus it will first be proven in the one-dimensional case.

Lemma 2.2 (Chebyshev aliasing formula in one dimension)**.** *The identity*

$$\hat{f}^{\nu}_{k} = \hat{f}_k + \sum_{m=1}^{\infty} {c_k}^{-1} \hat{f}_{k+2m\nu} + \sum_{\substack{m=1 \\ -k+2m\nu > \nu}}^{\infty} {c_k}^{-1} \hat{f}_{-k+2m\nu} \tag{2.27}$$

holds for every $0 \le k \le \nu$.

Proof. The proof follows the steps from [Pey02]. The first part is as in the Fourier case:

$$\begin{aligned} \hat{f}^{\nu}_{k} &= 2\nu^{-1} c_k^{-1} \sum_{i=0}^{\nu} c_i^{-1} f(x_i) T_k(x_i) \\ &= 2\nu^{-1} c_k^{-1} \sum_{i=0}^{\nu} c_i^{-1} T_k(x_i) \sum_{l=0}^{\infty} \hat{f}_l T_l(x_i) \\ &= 2\nu^{-1} c_k^{-1} \sum_{l=0}^{\infty} \hat{f}_l \sum_{0 \le i \le \nu} c_i^{-1} T_k(x_i) T_l(x_i) \\ &= 2\nu^{-1} c_k^{-1} \sum_{l=0}^{\infty} b_{k,l} \hat{f}_l \\ &= 2\nu^{-1} c_k^{-1} \sum_{l=0}^{\nu} b_{k,l} \hat{f}_l + 2\nu^{-1} c_k^{-1} \sum_{l=\nu+1}^{\infty} b_{k,l} \hat{f}_l, \end{aligned} \tag{2.28}$$

where the terms $b_{k,l}$ are defined as

$$b_{k,l} := \sum_{0 \le i \le \nu} c_i^{-1} T_k(x_i) T_l(x_i). \tag{2.29}$$

The key to proving the aliasing formula is simplifying this expression. For $l \le \nu$ the discrete orthogonality relation (2.26) implies

$$b_{k,l} = 2^{-1} \nu c_k \delta_{k,l} \tag{2.30}$$

which yields

$$\hat{f}^{\nu}_{k} = \hat{f}_k + 2\nu^{-1} c_k^{-1} \sum_{l=\nu+1}^{\infty} b_{k,l} \hat{f}_l. \tag{2.31}$$

Unfortunately, the Chebyshev orthogonality is not valid for $l > \nu$. Thus, in the second sum $b_{k,l}$ must be computed directly. For this purpose the following trigonometric identity is used:

$$A_p := \sum_{i=0}^{\nu} \cos \frac{pi\pi}{\nu} = \begin{cases} \nu + 1 & \text{if } p = 2m\nu \text{ for some } m \in \mathbb{Z}, \\ \frac{1+(-1)^p}{2} & \text{otherwise.} \end{cases} \tag{2.32}$$

The expression for $b_{k,l}$ can now be simplified:

$$\begin{aligned} b_{k,l} &= \sum_{i=0}^{\nu} c_i^{-1} T_k(x_i) T_l(x_i) \\ &= \sum_{i=0}^{\nu} c_i^{-1} \cos\Big(k \cos^{-1}\Big(\cos\frac{i\pi}{\nu}\Big)\Big) \cos\Big(l \cos^{-1}\Big(\cos\frac{i\pi}{\nu}\Big)\Big) \\ &= \sum_{i=0}^{\nu} c_i^{-1} \cos\frac{ki\pi}{\nu} \cos\frac{li\pi}{\nu} \\ &= \frac{1}{2} \sum_{i=0}^{\nu} c_i^{-1} \left[\cos\frac{k-l}{\nu} i\pi + \cos\frac{k+l}{\nu} i\pi \right]. \end{aligned} \tag{2.33}$$

We can now use the definition (2.25) of the constants c_i to get

$$\begin{aligned} b_{k,l} &= \frac{1}{2} \sum_{i=0}^{\nu} \Big[\cos\frac{k-l}{\nu} i\pi + \cos\frac{k+l}{\nu} i\pi \Big] - \frac{1}{4} \Big[\cos 0 + \cos 0 + \cos(k-l)\pi + \cos(k+l)\pi \Big] \\ &= \frac{1}{2} \sum_{i=0}^{\nu} \Big[\cos\frac{k-l}{\nu} i\pi + \cos\frac{k+l}{\nu} i\pi \Big] - \frac{1}{4} \Big[2 + (-1)^{k-l} + (-1)^{k+l} \Big] \\ &= \frac{1}{2} \Big[A_{k-l} + A_{k+l} - 1 - (-1)^{k+l} \Big]. \end{aligned} \tag{2.34}$$

From the definition of A_p it follows that $b_{k,l}$ is not equal to zero only if $l = k + 2m\nu$ or $l = -k + 2m\nu$ for some $m \in \mathbb{Z}$. If exactly one of these relations holds, then $b_{k,l} = \nu/2$. The other option is to have both $l = k + 2m_1\nu = -k + 2m_2\nu$. This case is only possible for $k = 0$ and then $b_{k,l} = \nu$. Therefore

$$\begin{aligned} \hat{f}_k^{\nu} &= \hat{f}_k + 2\nu^{-1} c_k^{-1} \sum_{l=\nu+1}^{\infty} b_{k,l} \hat{f}_l \\ &= \hat{f}_k + \sum_{m=1}^{\infty} {c_k}^{-1} \hat{f}_{k+2m\nu} + \sum_{\substack{m=1 \\ -k+2m\nu>\nu}}^{\infty} {c_k}^{-1} \hat{f}_{-k+2m\nu}. \end{aligned} \tag{2.35}$$

□

The aliasing formula written in this form is not particularly compact. To simplify the notation, we can define the set of all coefficient indices that are present on the right-hand side of (2.27) as follows:

$$S_{k,\nu} := \{k\} \cup \{k + 2m\nu : m \in \mathbb{N}\} \cup \{-k + 2m\nu : m \in \mathbb{N}, -k + 2m\nu > \nu\}. \tag{2.36}$$

Note that the set is actually a multiset in the case $k = 0$ because the two sums on the right-hand side coincide. In this case, the elements in $S_{k,\nu}$ have cardinality of at most 2. In all other cases, $S_{k,\nu}$ is a set. Now equation (2.27) can be written more compactly:

$$\hat{f}_k^{\nu} = \sum_{i \in S_{k,\nu}} c_{i,k} \hat{f}_i \tag{2.37}$$

where $c_{i,k} = 1$ if $i = k$ and $c_{i,k} = c_i^{-1}$ for all other values of i. It should be once again stressed that this sum runs over a multiset of indices and every term must be taken into account as many times as its cardinality.

This form is more convenient to be extended to the n-dimensional case.

Lemma 2.3 (Chebyshev aliasing formula in n dimensions)**.** *The identity*

$$\hat{f}^{\boldsymbol{\nu}}_{\boldsymbol{k}} = \sum_{\boldsymbol{i} \in S_{\boldsymbol{k},\boldsymbol{\nu}}} c_{\boldsymbol{i},\boldsymbol{k}} \hat{f}_{\boldsymbol{i}} \tag{2.38}$$

holds for every $\mathbf{0} \leq \boldsymbol{k} \leq \boldsymbol{\nu}$*, where*

$$c_{\boldsymbol{i},\boldsymbol{k}} := \prod_{d=1}^{n} c_{i_d,k_d} \tag{2.39}$$

and

$$S_{\boldsymbol{k},\boldsymbol{\nu}} = \bigtimes_{d=1}^{n} S_{k_d,\nu_d} \tag{2.40}$$

is a multiset of indices. The cardinality of each element in this multiset is at most 2^n*.*

Proof. The proof goes as in the one-dimensional case:

$$\begin{aligned}
\hat{f}^{\boldsymbol{\nu}}_{\boldsymbol{k}} &= 2^n \nu_1^{-1} \dots \nu_n^{-1} c_{\boldsymbol{k}}^{-1} \sum_{\mathbf{0} \leq \boldsymbol{i} \leq \boldsymbol{\nu}} c_{\boldsymbol{i}}^{-1} f(\boldsymbol{x}_{\boldsymbol{i}}) T_{\boldsymbol{k}}(\boldsymbol{x}_{\boldsymbol{i}}) \\
&= 2^n \nu_1^{-1} \dots \nu_n^{-1} c_{\boldsymbol{k}}^{-1} \sum_{\mathbf{0} \leq \boldsymbol{i} \leq \boldsymbol{\nu}} c_{\boldsymbol{i}}^{-1} T_{\boldsymbol{k}}(\boldsymbol{x}_{\boldsymbol{i}}) \sum_{\boldsymbol{l} \in \mathbb{N}_0^n} \hat{f}_{\boldsymbol{l}} T_{\boldsymbol{l}}(\boldsymbol{x}_{\boldsymbol{i}}) \\
&= 2^n \nu_1^{-1} \dots \nu_n^{-1} c_{\boldsymbol{k}}^{-1} \sum_{\boldsymbol{l} \in \mathbb{N}_0^n} \hat{f}_{\boldsymbol{l}} \sum_{\mathbf{0} \leq \boldsymbol{i} \leq \boldsymbol{\nu}} c_{\boldsymbol{i}}^{-1} T_{\boldsymbol{k}}(\boldsymbol{x}_{\boldsymbol{i}}) T_{\boldsymbol{l}}(\boldsymbol{x}_{\boldsymbol{i}}) \\
&= 2^n \nu_1^{-1} \dots \nu_n^{-1} c_{\boldsymbol{k}}^{-1} \sum_{\boldsymbol{l} \in \mathbb{N}_0^n} b_{\boldsymbol{k},\boldsymbol{l}} \hat{f}_{\boldsymbol{l}},
\end{aligned} \tag{2.41}$$

where

$$\begin{aligned}
b_{\boldsymbol{k},\boldsymbol{l}} &:= \sum_{\mathbf{0} \leq \boldsymbol{i} \leq \boldsymbol{\nu}} c_{\boldsymbol{i}}^{-1} T_{\boldsymbol{k}}(\boldsymbol{x}_{\boldsymbol{i}}) T_{\boldsymbol{l}}(\boldsymbol{x}_{\boldsymbol{i}}) \\
&= \prod_{d=1}^{n} \left[\sum_{i=0}^{\nu_d} c_i^{-1} T_{k_d}(x_i) T_{l_d}(x_i) \right] \\
&= \prod_{d=1}^{n} b_{k_d,l_d}
\end{aligned} \tag{2.42}$$

Hence, $b_{\boldsymbol{k},\boldsymbol{l}}$ is non-zero only for $\boldsymbol{l} = \boldsymbol{\varepsilon}\boldsymbol{k} + 2\boldsymbol{m}\boldsymbol{\nu}$ where $\boldsymbol{\varepsilon} \in \{-1, 1\}^n$. Equation (2.38) now follows easily by applying the same logic as in the one-dimensional case. Finally, the cardinality of each element in $S_{\boldsymbol{k},\boldsymbol{\nu}}$ is at most 2^n because the cardinality of each element in S_{k_d,ν_d} is at most 2. □

2.3 Mixed Fourier-Chebyshev transform

A mixed transform is an n-dimensional transform which uses different basis functions in the different dimensions. This subsection is devoted to finding explicit formulas for the coefficients of one such transform as well as an aliasing formula for the relation between continuous and discrete coefficients. The following derivations are to a great extent analogous to the non-mixed cases. However, they are worth exploring separately because the more complicated notation makes them more technical and therefore not completely straightforward.

Without loss of generality, assume that the first p dimensions of the mixed transform use Fourier basis functions and the other $n-p$ dimensions use Chebyshev basis functions. To simplify the notation, the following convention is adopted: each multi-index $\boldsymbol{i} \in \mathbb{Z}^p \times \mathbb{N}_0^{n-p}$ can be split into Fourier and Chebyshev parts, $\boldsymbol{i}'$ and $\boldsymbol{i}''$, respectively. In this splitting, $\boldsymbol{i}' \in \mathbb{Z}^p$, $\boldsymbol{i}'' \in \mathbb{N}_0^{n-p}$

and $\boldsymbol{i} = (\boldsymbol{i}', \boldsymbol{i}'')$. Furthermore, every variable, function or space that is denoted with a prime or a double prime is related to the Fourier or the Chebyshev transform, respectively.

The mixed transform can now be defined for functions on the product domain $\mathbb{T}^p \times \mathbb{I}^{n-p}$. The mixed basis functions are

$$\xi_{\boldsymbol{k}} := \omega_{\boldsymbol{k}'} \otimes T_{\boldsymbol{k}''}. \tag{2.43}$$

They are used for approximating functions in the space $L^2(\mathbb{T}^p) \times L^2_w(\mathbb{I}^{n-p})$. Every function f from this space can be expanded in a mixed infinite series,

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{k} \in \mathbb{Z}^p \times \mathbb{N}_0^{n-p}} \hat{f}_{\boldsymbol{k}} \xi_{\boldsymbol{k}}(\boldsymbol{x}). \tag{2.44}$$

The interpolation points are also easily defined as $\boldsymbol{x} := (\boldsymbol{x}', \boldsymbol{x}'')$, where $\boldsymbol{x}'$ is a p-dimensional Fourier point and $\boldsymbol{x}''$ is an $(n-p)$-dimensional Chebyshev point. For an index $\boldsymbol{\nu} \in \mathbb{N}^n$, the mixed interpolation operator is a mapping

$$I_{\boldsymbol{\nu}} : L^2(\mathbb{T}^p) \times L^2_w(\mathbb{I}^{n-p}) \to \operatorname{span}\Big\{\xi_{\boldsymbol{\sigma}(\boldsymbol{k})} : \boldsymbol{0} \le \boldsymbol{k}' < \boldsymbol{\nu}', \boldsymbol{0} \le \boldsymbol{k}'' \le \boldsymbol{\nu}''\Big\} \tag{2.45}$$

such that $I_{\boldsymbol{\nu}} f(\boldsymbol{x}) = f(\boldsymbol{x})$ for every $\boldsymbol{x}$ from the mixed product grid. In the above definition, the indexing function $\boldsymbol{\sigma}$ is defined as $\boldsymbol{\sigma}(\boldsymbol{k}) := (\boldsymbol{\sigma}'(\boldsymbol{k}), \boldsymbol{k}'')$. For each f the interpolant can be uniquely represented as

$$I_{\boldsymbol{\nu}} f = \sum_{\boldsymbol{0} \le \boldsymbol{i}' < \boldsymbol{\nu}', \boldsymbol{0} \le \boldsymbol{i}'' \le \boldsymbol{\nu}''} \hat{f}^{\boldsymbol{\nu}}_{\boldsymbol{i}} \xi_{\boldsymbol{\sigma}(\boldsymbol{i})}, \tag{2.46}$$

where $\hat{f}^{\boldsymbol{\nu}}_{\boldsymbol{i}}$ are the discrete mixed coefficients. They can be computed with the formula

$$\hat{f}^{\boldsymbol{\nu}}_{\boldsymbol{i}} = 2^{n-p} \nu_1^{-1} \dots \nu_n^{-1} c^{-1}_{\boldsymbol{i}''} \sum_{\boldsymbol{0} \le \boldsymbol{k}' < \boldsymbol{\nu}', \boldsymbol{0} \le \boldsymbol{k}'' \le \boldsymbol{\nu}''} c^{-1}_{\boldsymbol{k}''} f(\boldsymbol{x}_{\boldsymbol{k}}) \xi^*_{\boldsymbol{\sigma}(\boldsymbol{i})}(\boldsymbol{x}_{\boldsymbol{k}}). \tag{2.47}$$

The proof of this formula is very similar to the proof of the aliasing formula for the mixed transform and thus it is omitted.

Lemma 2.4 (Mixed aliasing formula)**.** *The identity*

$$\hat{f}^{\boldsymbol{\nu}}_{\boldsymbol{k}} = \sum_{\boldsymbol{i} \in S_{\boldsymbol{k},\boldsymbol{\nu}}} c_{\boldsymbol{i}'',\boldsymbol{k}''} \hat{f}_{\boldsymbol{i}} \tag{2.48}$$

holds, where

$$c_{\boldsymbol{i}'',\boldsymbol{k}''} := \prod_{d=p+1}^{n} c_{i_d,k_d} \tag{2.49}$$

and

$$S_{\boldsymbol{k},\boldsymbol{\nu}} := S'_{\boldsymbol{k}',\boldsymbol{\nu}'} \times S''_{\boldsymbol{k}'',\boldsymbol{\nu}''} \tag{2.50}$$

is a multiset of indices. The cardinality of each element in the multiset is at most 2^{n-p}*.*

Proof.

$$\begin{aligned}
\hat{f}_{\boldsymbol{i}}^{\boldsymbol{\nu}} &= 2^{n-p}\nu_1^{-1}\dots\nu_n^{-1}c_{\boldsymbol{i}''}^{-1} \sum_{\mathbf{0}\le\boldsymbol{k}'<\boldsymbol{\nu}',\mathbf{0}\le\boldsymbol{k}''\le\boldsymbol{\nu}''} c_{\boldsymbol{k}''}^{-1} f(\boldsymbol{x}_{\boldsymbol{k}})\xi^*_{\boldsymbol{\sigma}(\boldsymbol{i})}(\boldsymbol{x}_{\boldsymbol{k}}) \\
&= 2^{n-p}\nu_1^{-1}\dots\nu_n^{-1}c_{\boldsymbol{i}''}^{-1} \sum_{\mathbf{0}\le\boldsymbol{k}'<\boldsymbol{\nu}',\mathbf{0}\le\boldsymbol{k}''\le\boldsymbol{\nu}''} c_{\boldsymbol{k}''}^{-1}\xi^*_{\boldsymbol{\sigma}(\boldsymbol{i})}(\boldsymbol{x}_{\boldsymbol{k}}) \sum_{\boldsymbol{l}\in\mathbb{Z}^p\times\mathbb{N}_0^{n-p}} \hat{f}_{\boldsymbol{l}}\xi_{\boldsymbol{l}}(\boldsymbol{x}_{\boldsymbol{k}}) \\
&= 2^{n-p}\nu_1^{-1}\dots\nu_n^{-1}c_{\boldsymbol{i}''}^{-1} \sum_{\boldsymbol{l}\in\mathbb{Z}^p\times\mathbb{N}_0^{n-p}} \hat{f}_{\boldsymbol{l}} \sum_{\mathbf{0}\le\boldsymbol{k}'<\boldsymbol{\nu}',\mathbf{0}\le\boldsymbol{k}''\le\boldsymbol{\nu}''} c_{\boldsymbol{k}''}^{-1}\xi_{\boldsymbol{l}}(\boldsymbol{x}_{\boldsymbol{k}})\xi^*_{\boldsymbol{\sigma}(\boldsymbol{i})}(\boldsymbol{x}_{\boldsymbol{k}}) \\
&= 2^{n-p}\nu_1^{-1}\dots\nu_n^{-1}c_{\boldsymbol{i}''}^{-1} \sum_{\boldsymbol{l}\in\mathbb{Z}^p\times\mathbb{N}_0^{n-p}} \hat{f}_{\boldsymbol{l}} \left[\prod_{d=1}^{p}\sum_{k=0}^{\nu_d-1}\omega_{l_d}(x_k)\omega^*_{\sigma'(i_d)}(x_k)\right]\left[\prod_{d=p+1}^{n} c_k^{-1}\sum_{k=0}^{\nu_d} T_{l_d}(x_k)T_{i_d}(x_k)\right] \\
&= 2^{n-p}\nu_1^{-1}\dots\nu_n^{-1}c_{\boldsymbol{i}''}^{-1} \sum_{\boldsymbol{l}\in\mathbb{Z}^p\times\boldsymbol{\nu}_0^{n-p}} a_{\boldsymbol{\sigma}'(\boldsymbol{i}'),\boldsymbol{l}'}b_{\boldsymbol{i}'',\boldsymbol{l}''}\hat{f}_{\boldsymbol{l}},
\end{aligned} \tag{2.51}$$

where $a_{\boldsymbol{\sigma}'(\boldsymbol{i}'),\boldsymbol{l}'}$ is defined as in (2.12) and $b_{\boldsymbol{i}'',\boldsymbol{l}''}$ is defined as in (2.29). Both of them are non-zero only for indices $\boldsymbol{l}$ in the set $S'_{\boldsymbol{k}',\boldsymbol{\nu}'}\times S''_{\boldsymbol{k}'',\boldsymbol{\nu}''}$. The conclusion of the lemma follows easily by applying the formulas for the values of $a_{\boldsymbol{\sigma}'(\boldsymbol{i}'),\boldsymbol{l}'}$ and $b_{\boldsymbol{i}'',\boldsymbol{l}''}$. □

2.4 Generalized Hermite Transform

The Hermite spectral and pseudospectral methods are widely used for approximation of Gaussian-type functions on the entire real line $\mathbb{R}$. In their basic form they often provide suboptimal results because of poor resolution([Tan93]). This means that too many basis functions are required for a good approximation. The usual way to remedy this disadvantage is to generalize the typical basis functions by augmenting them with scaling and translating factors α and β. This results in an entire family of parametrized transforms.

The physical Hermite polynomials are defined with the formula

$$H_k(x) := (-1)^k e^{x^2}\frac{d^k}{dx^k}(e^{-x^2}) \tag{2.52}$$

for every $k\in\mathbb{N}_0$. They are orthogonal with respect to the weight function $w(x) := e^{-x^2}$. The generalized normalized Hermite basis functions are defined as

$$\mathcal{H}_k^{\alpha,\beta} := \left(\frac{\alpha}{2^k k!\sqrt{\pi}}\right)^{\frac{1}{2}} H_k(\alpha(x-\beta))e^{-\frac{1}{2}\alpha^2(x-\beta)^2} \tag{2.53}$$

for a scaling parameter $\alpha > 0$ and a translating parameter $\beta\in\mathbb{R}$. With this definition $\{\mathcal{H}_k^{\alpha,\beta}\}_{k\in\mathbb{N}_0}$ form an orthonormal basis of $L^2(\mathbb{R})$. This form of the basis functions is not suitable for numerical computation. In practice, the following recurrence relation is used:

$$2\alpha^2(x-\beta)\mathcal{H}_k^{\alpha,\beta}(x) = \sqrt{2\alpha^2 k}\mathcal{H}_{k-1}^{\alpha,\beta}(x) + \sqrt{2\alpha^2(k+1)}\mathcal{H}_{k+1}^{\alpha,\beta}(x). \tag{2.54}$$

For more properties of the generalized Hermite basis functions you can refer to [LY13].

In the n-dimensional case, the generalized Hermite functions are defined as

$$\mathcal{H}_{\boldsymbol{k}}^{\boldsymbol{\alpha},\boldsymbol{\beta}}(\boldsymbol{x}) := \left(\bigotimes_{d=1}^{n}\mathcal{H}_{k_d}^{\alpha_d,\beta_d}\right)(\boldsymbol{x}) = \prod_{d=1}^{n}\mathcal{H}_{k_d}^{\alpha_d,\beta_d}(x_d) \tag{2.55}$$

and they form an orthonormal basis of $L^2(\mathbb{R}^n)$. Every function f in this space can be represented in the form

$$f(\boldsymbol{x}) := \sum_{\boldsymbol{k}\in\mathbb{N}_0^n} \hat{f}_{\boldsymbol{k}} \mathcal{H}_{\boldsymbol{k}}^{\boldsymbol{\alpha},\boldsymbol{\beta}}(\boldsymbol{x}). \tag{2.56}$$

Nevertheless, the Hermite methods are typically used only for functions f with exponential decay because otherwise the convergence of the series is too slow.

Let $\boldsymbol{\nu} \in \mathbb{N}^n$ and let $\{\boldsymbol{x}_{\boldsymbol{k}} : \mathbf{0} \le \boldsymbol{k} \le \boldsymbol{\nu}\}$ be the interpolation points. The interpolation operator $I_{\boldsymbol{\nu}}$ is defined as the mapping

$$I_{\boldsymbol{\nu}} : L^2(\mathbb{R}^n) \to \operatorname{span}\{\mathcal{H}_{\boldsymbol{k}}^{\boldsymbol{\alpha},\boldsymbol{\beta}} : \mathbf{0} \le \boldsymbol{k} \le \boldsymbol{\nu}\} \tag{2.57}$$

which interpolates every function f in every interpolation point $\boldsymbol{x}$. The result of the operator is

$$I_{\boldsymbol{\nu}} f = \sum_{\mathbf{0}\le\boldsymbol{k}\le\boldsymbol{\nu}} \hat{f}_{\boldsymbol{k},\boldsymbol{\nu}} \mathcal{H}_{\boldsymbol{k}}^{\boldsymbol{\alpha},\boldsymbol{\beta}}, \tag{2.58}$$

where $\hat{f}_{\boldsymbol{k},\boldsymbol{\nu}}$ are the discrete Hermite coefficients. The typical choice of interpolation points $\{\boldsymbol{x}_{\boldsymbol{k}} : \mathbf{0} \le \boldsymbol{k} \le \boldsymbol{\nu}\}$ is $\boldsymbol{x}_{\boldsymbol{k}} := (\gamma_{k_1}, \dots \gamma_{k_n})$, where $\{\gamma_j\}_{j=0}^N$ are the roots of the polynomial H_{N+1}. While this set of points provides good approximation results, it suffers from a limitation that renders it unusable for interpolatory sparse grids. The discussion of finding a proper set of interpolation points is postponed to subsection 5.3.

As already mentioned, the choice of the transform parameters is crucial for the efficiency of the method. While the choice of β is usually obvious, finding the best value for α is a much more complicated task. A detailed discussion of this topic can be found in [Tan93]. The effect of α and β on the form of the basis functions can be seen in Figure 1.

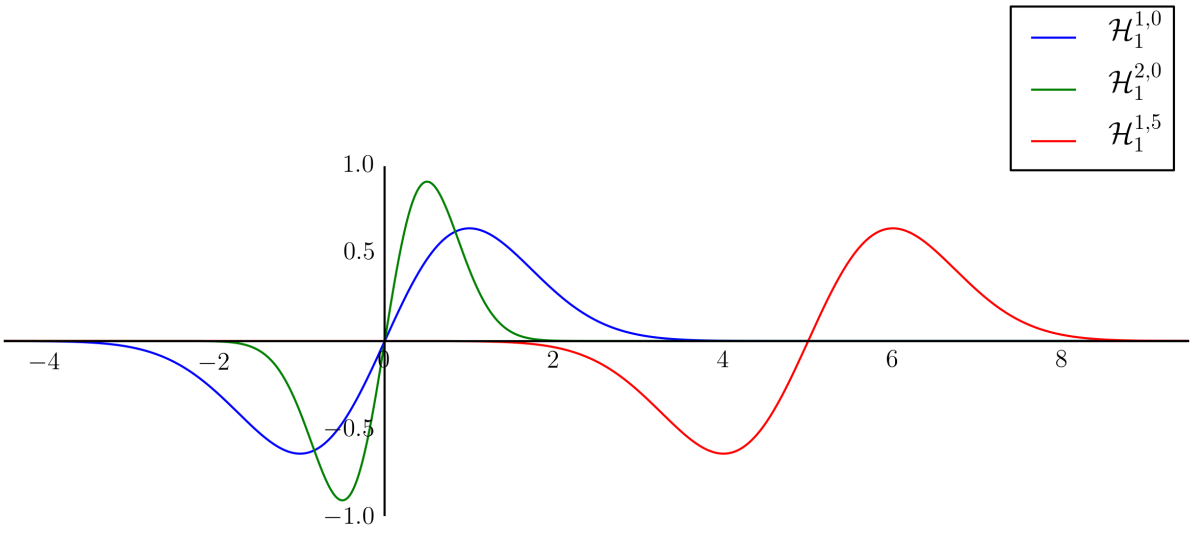


Figure 1: Effect of the parameters α and β on the shape and location of the basis function.

3 Generalized Sparse Grids

Sparse grids can be introduced in many different ways with a varying degree of complexity, rigorousness and extensibility. Most texts introduce them in the context of a specific transform - Zenger [Zen91] explores their properties for a basis of piecewise linear functions, Hallatschek [Hal92] focuses on the Fourier transform and Barthelmann, Novak and Ritter [BNR00] develop the theory for polynomial interpolation. One of the conveniences of working with sparse grids is that they can be used with a large variety of transforms. One of the goals of this thesis is to demonstrate precisely this. Thus, an introduction that is bound to a specific transform would not be appropriate. The approach adopted in this text is to introduce sparse grids in a general setting, abstracting from the underlying transform. All results that are derived in this section are valid for all interpolatory sparse grids. The theory follows closely [Hal92] and [GH14]. The notation is mostly borrowed from [GH14].

3.1 Interpolation in one dimension

Consider an infinite-dimensional Hilbert space $X = \{f : I \to \mathbb{C}\}$ of one-dimensional functions, where I is an arbitrary finite or infinite subinterval of $\mathbb{R}$. Consider also a sequence of linearly independent functions $\phi_n \in X$ for every $n \in \mathbb{N}_0$. For an arbitrary increasing sequence of natural numbers g_n, we can define the index set

$$\mathcal{G}_n := \{0, 1, \dots g_n - 1\}, \tag{3.1}$$

the set of basis functions

$$\mathcal{B}_n := \{\phi_i : i \in \mathcal{G}_n\}, \tag{3.2}$$

and the finite-dimensional subspaces

$$V_n := \operatorname{span}\{\mathcal{B}_n\}. \tag{3.3}$$

These definitions imply that the spaces V_n form an increasing sequence, i.e. $V_{n-1} \subset V_n$ for every $n \in \mathbb{N}$.

The goal of this subsection is to define an interpolation operator I_n that maps every function $f \in X$ to an approximation function $I_n f \in V_n$. In order to define I_n, we need a sequence of interpolation points $x_n \in I$, $n \in \mathbb{N}_0$. The interpolation operator I_n is the unique operator $I_n : X \to V_n$ such that

$$I_n f(x) = f(x) \tag{3.4}$$

for every $x \in S_n := \{x_i : i \in \mathcal{G}_n\}$. For a fixed function $f \in X$, the result of the operator I_n can be represented as a linear combination of the basis functions of V_n,

$$I_n f = \sum_{i \in \mathcal{G}_n} \hat{f}_{i,n} \phi_i. \tag{3.5}$$

The coefficients $\hat{f}_{i,n}$ are uniquely defined as long as certain conditions are satisfied. More precisely, the interpolation conditions given by equation (3.4) form a system of g_n linear equations for the g_n coefficients. Therefore, a unique solution exists if and only if the matrix of the system

is invertible,

$$A_{g_n-1} := \begin{pmatrix} \phi_0(x_0) & \dots & \phi_{g_n-1}(x_0) \\ \vdots & \ddots & \vdots \\ \phi_0(x_{g_n-1}) & \dots & \phi_{g_n-1}(x_{g_n-1}) \end{pmatrix}. \tag{3.6}$$

If this is true then the coefficients can be computed with the formula

$$\begin{pmatrix} \hat{f}_{0,n} \\ \vdots \\ \hat{f}_{g_n-1,n} \end{pmatrix} = A_{g_n-1}^{-1} \begin{pmatrix} f(x_0) \\ \vdots \\ f(x_{g_n-1}) \end{pmatrix}. \tag{3.7}$$

Finally, to make subsequent definitions easier, define $V_{-1} := \{0\}$, $\mathcal{B}_{-1} := \{\}$, $\mathcal{G}_{-1} = \{\}$, $S_{-1} := \{\}$ and $I_{-1} : X \to V_{-1}, f \mapsto 0$.

3.2 Interpolation in multiple dimensions

We can now move on to the concept of interpolation in multiple dimensions. The operator from the previous subsection can easily be extended by using a tensor product approach. Consider functions f that belong to the tensor product space $X^n = X \otimes \cdots \otimes X$. For a multi-index $\boldsymbol{\nu} \in \mathbb{N}_0^n$ the multidimensional equivalents of the index sets, basis functions, basis function spaces, approximation spaces and collocation points are defined as follows:

$$\mathcal{G}_{\boldsymbol{\nu}} := \bigtimes_{d=1}^{n} \mathcal{G}_{\nu_d}, \tag{3.8}$$

$$\phi_{\boldsymbol{\nu}}(\boldsymbol{x}) := \Big(\bigotimes_{d=1}^{n} \phi_{\nu_d}\Big)(\boldsymbol{x}) = \prod_{d=1}^{n} \phi_{\nu_d}(x_{\nu_d}), \tag{3.9}$$

$$\mathcal{B}_{\boldsymbol{\nu}} := \{\phi_{\boldsymbol{\mu}} : \boldsymbol{\mu} \in \mathcal{G}_{\boldsymbol{\nu}}\}, \tag{3.10}$$

$$V_{\boldsymbol{\nu}} := \bigotimes_{d=1}^{n} V_{\nu_d} = \operatorname{span}\{\mathcal{B}_{\boldsymbol{\nu}}\}, \tag{3.11}$$

$$S_{\boldsymbol{\nu}} := \bigtimes_{d=1}^{n} S_{\nu_d}. \tag{3.12}$$

The interpolation operator is a mapping from X^n to $V_{\boldsymbol{\nu}}$ and is also defined as a tensor product of one-dimensional operators,

$$I_{\boldsymbol{\nu}} = \bigotimes_{d=1}^{n} I_{\nu_d}. \tag{3.13}$$

With this definition, $I_{\boldsymbol{\nu}}$ interpolates a function $f \in X^n$ in all points $\boldsymbol{x} \in S_{\boldsymbol{\nu}}$. The result of the interpolation can be expressed as a linear combination of multidimensional basis functions:

$$I_{\boldsymbol{\nu}} f = \sum_{\boldsymbol{i} \in \mathcal{G}_{\boldsymbol{\nu}}} \hat{f}_{\boldsymbol{i},\boldsymbol{\nu}} \phi_{\boldsymbol{i}}, \tag{3.14}$$

where $\hat{f}_{\boldsymbol{i},\boldsymbol{\nu}}$ are coefficients that are uniquely defined by the interpolation properties of $I_{\boldsymbol{\nu}}$.

Computing $I_{\boldsymbol{\nu}} f$ is equivalent to finding the coefficients $\hat{f}_{\boldsymbol{i},\boldsymbol{\nu}}$. In the one-dimensional case this is straightforward and is achieved by applying equation (3.7). The complexity of this procedure in the general case is $O(g_\nu^3)$ because it is equivalent to a matrix inversion problem. For specific transforms and sequences g_ν, however, this can be done in a much more efficient way and

the complexity can get as low as $O(g_\nu \log g_\nu)$(e.g. the Fourier transform). This stems from the fact that the matrix A_ν has a very specific structure. Using this approach directly in the multidimensional case is impractical. For dimension $n > 1$, the system of equations is

$$\sum_{\boldsymbol{i} \in \mathcal{G}_{\boldsymbol{\nu}}} \hat{f}_{i,\boldsymbol{\nu}} \phi_{\boldsymbol{i}}(\boldsymbol{x}) = f(\boldsymbol{x}) \tag{3.15}$$

for $\boldsymbol{x} \in S_{\boldsymbol{\nu}}$ has $|S_{\boldsymbol{\nu}}| = g_{\nu_1} \dots g_{\nu_n}$ unknowns. This means that the complexity of solving it by directly inverting its matrix is $O(g_{\nu_1}^3 \dots g_{\nu_n}^3)$. Therefore, this approach is computationally infeasible for even small values of n.

The usual way to tackle this problem is to use an iterative approach to find the interpolation coefficients. The tensor product structure allows us to do this in a very efficient way. The basic idea is to perform a one-dimensional transform in every dimension of the problem. To make this more precise, fix a grid level $\boldsymbol{\nu}$ and define the parametrized coefficients $\hat{f}^i_{\boldsymbol{\alpha}}(x_{i+1}, \dots x_n)$ for an i-dimensional index $\boldsymbol{\alpha} < (\nu_1, \dots \nu_i)$. For $i = 0$, the index $\boldsymbol{\alpha}$ can be omitted since it has zero entries. For the base case, $i = 0$ $\hat{f}^0$ is defined as $\hat{f}^0 := f$. For subsequent values of i, the coefficients are defined iteratively with the equation

$$I_{\nu_{i+1}} \hat{f}^i_{\boldsymbol{\alpha}}(x_{i+1}, \dots x_n) = \sum_{j \in \mathcal{G}_{\nu_i}} \hat{f}^{i+1}_{(\boldsymbol{\alpha},j)}(x_{i+2}, \dots x_n) \phi_j(x_{i+1}). \tag{3.16}$$

Starting from $\hat{f}^0$ and moving up the dimensions, we eventually get the coefficients $\hat{f}^n_{\boldsymbol{\alpha}}$, $\boldsymbol{\alpha} \in \mathcal{G}_{\boldsymbol{\nu}}$. The goal is to prove that these coefficients are equal to the coefficients that are computed with the naive approach, i.e. $\hat{f}^n_{\boldsymbol{\alpha}} = \hat{f}_{\boldsymbol{\alpha},\boldsymbol{\nu}}$. To achieve this, consider the operator

$$I'_{\boldsymbol{\nu}} f(\boldsymbol{x}) = \sum_{\boldsymbol{i} \in \mathcal{G}_{\boldsymbol{\nu}}} \hat{f}^n_{\boldsymbol{i}} \phi_{\boldsymbol{i}}(\boldsymbol{x}). \tag{3.17}$$

Because of the uniqueness of the interpolation operator, proving that $I'_{\boldsymbol{\nu}} f(\boldsymbol{x}) = f(\boldsymbol{x})$ for every $\boldsymbol{x} \in S_{\boldsymbol{\nu}}$ will prove the desired result. This can be achieved by induction over n. For convenience, the following notation is used:

$$\boldsymbol{\nu} = (\nu_1, \nu_2, \dots \nu_d) = (\boldsymbol{\nu}', \nu_d), \tag{3.18}$$

$$\boldsymbol{i} = (i_1, i_2, \dots i_d) = (\boldsymbol{i}', i_d), \tag{3.19}$$

$$\boldsymbol{x} = (x_1, x_2, \dots x_d) = (\boldsymbol{x}', x_d). \tag{3.20}$$

Consider first the base case $n = 1$:

$$I'_{\boldsymbol{\nu}} f(\boldsymbol{x}) = \sum_{\boldsymbol{i} \in \mathcal{G}_{\boldsymbol{\nu}}} \hat{f}^n_{\boldsymbol{i}} \phi_{\boldsymbol{i}}(\boldsymbol{x}) = \sum_{i_1 \in \mathcal{G}_{\nu_1}} \hat{f}^1_{i_1} \phi_{i_1}(x_1) = I_{\nu_1} \hat{f}^0(x_1) = I_{\nu_1} f(x_1) = f(x_1) \tag{3.21}$$

for every $x_1 \in S_{\nu_1}$. Hence, for $n = 1$ the operator $I'_{\boldsymbol{\nu}}$ interpolates f in all points from $S_{\boldsymbol{\nu}}$.

Now fix some $n > 1$ and assume that the statement is true for all values less than n:

$$\begin{aligned}
I'_{\boldsymbol{\nu}} f(\boldsymbol{x}) &= \sum_{\boldsymbol{i} \in \mathcal{G}_{\boldsymbol{\nu}}} \hat{f}^n_{\boldsymbol{i}} \phi_{\boldsymbol{i}}(\boldsymbol{x}) \\
&\overset{(1)}{=} \sum_{\boldsymbol{i}' \in \mathcal{G}_{\boldsymbol{\nu}'}} \sum_{i_n \in \mathcal{G}_{\nu_n}} \hat{f}^n_{(\boldsymbol{i}', i_n)} \phi_{\boldsymbol{i}'}(\boldsymbol{x}') \phi_{i_n}(x_n) \\
&\overset{(2)}{=} \sum_{\boldsymbol{i}' \in \mathcal{G}_{\boldsymbol{\nu}'}} \phi_{\boldsymbol{i}'}(\boldsymbol{x}') \sum_{i_n \in \mathcal{G}_{\nu_n}} \hat{f}^n_{(\boldsymbol{i}', i_n)} \phi_{i_n}(x_n) \\
&\overset{(3)}{=} \sum_{\boldsymbol{i}' \in \mathcal{G}_{\boldsymbol{\nu}'}} \phi_{\boldsymbol{i}'}(\boldsymbol{x}') I_{\nu_n} \hat{f}^{n-1}_{\boldsymbol{i}'}(x_n) \\
&\overset{(4)}{=} I_{\nu_n} \sum_{\boldsymbol{i}' \in \mathcal{G}_{\boldsymbol{\nu}'}} \hat{f}^{n-1}_{\boldsymbol{i}'}(x_n) \phi_{\boldsymbol{i}'}(\boldsymbol{x}') \\
&\overset{(5)}{=} I_{\nu_n} f(\boldsymbol{x}', x_n) \\
&\overset{(6)}{=} f(\boldsymbol{x})
\end{aligned} \tag{3.22}$$

In the calculation above, step (1) is achieved by splitting the sum into two sums, the first one running over the first $n-1$ dimensions and the second one running over the last dimension. Step (2) is a simple reordering of the terms. Step (3) uses the definition of the parametrized coefficients (3.16) with $i = n-1$. Step (4) exploits the linearity of the operator I_{ν_n} and the fact that it is applied only on the last variable. Step (5) uses the induction hypothesis for $n-1$. Step (6) uses the interpolation property of the one-dimensional operator I_{ν_n}.

This proves that the operator $I'_{\boldsymbol{\nu}}$ is indeed equivalent to $I_{\boldsymbol{\nu}}$ and in the same time provides an efficient iterative algorithm for its computation. The overall computational complexity of the procedure is $O(g_{\nu_1} \dots g_{\nu_n}(g^2_{\nu_1} + \dots + g^2_{\nu_n}))$ which is a significant improvement over the naive $O(g^3_{\nu_1} \dots g^3_{\nu_n})$. For transforms that allow fast computation of the one-dimensional interpolation coefficients, the complexity is reduced even more. In the Fourier case it is equal to $O(g_{\nu_1} \dots g_{\nu_n} \log(g_{\nu_1} \dots g_{\nu_n}))$.

3.3 General interpolation operator

The main problem with the interpolation operators $I_{\boldsymbol{\nu}}$ constructed in the previous section is that they work exclusively on product grids(also known as full grids). The number of points in a full grid grows exponentially with the dimension n. This leads to infeasibility of the interpolation problem when the dimension increases.

The goal of this subsection is to define an approach for the construction of interpolation operators that work on a more flexible set of grids. Starting from the one-dimensional interpolation operator I_n, the difference operators Δ_n are defined as

$$\Delta_n := I_n - I_{n-1} : X \mapsto V_n, \tag{3.23}$$

$$\Delta_{\boldsymbol{\nu}} := \Delta_{\nu_1} \otimes \dots \otimes \Delta_{\nu_n} : X^n \mapsto V_{\boldsymbol{\nu}}. \tag{3.24}$$

The following relation between $I_{\boldsymbol{\nu}}$ and the difference operators holds:

$$\begin{aligned} I_{\boldsymbol{\nu}} &= I_{\nu_1} \otimes \cdots \otimes I_{\nu_d} \\ &= (\sum_{i_1=0}^{\nu_1} \Delta_{i_1}) \otimes \cdots \otimes (\sum_{i_d=0}^{\nu_d} \Delta_{i_d}) \\ &= \sum_{\mathbf{0} \le \boldsymbol{i} \le \boldsymbol{\nu}} \Delta_{\boldsymbol{i}}. \end{aligned} \tag{3.25}$$

From now on, we will only be interested in functions $f \in X^n$ for which $\Delta_{\boldsymbol{\nu}} f$ are summable and the corresponding series converges pointwise to f, i.e. for which

$$f = \sum_{\boldsymbol{\nu} \in \mathbb{N}_0^n} \Delta_{\boldsymbol{\nu}} f. \tag{3.26}$$

The hierarchical set of collocation points $S_{\boldsymbol{\nu}}^h$ is defined as

$$S_n^h := S_n \setminus S_{n-1}, \tag{3.27}$$

$$S_{\boldsymbol{\nu}}^h := S_{\nu_1}^h \times \cdots \times S_{\nu_n}^h = S_{\boldsymbol{\nu}} \setminus \Big(\bigcup_{\boldsymbol{\mu} \le \boldsymbol{\nu}, \boldsymbol{\mu} \ne \boldsymbol{\nu}} S_{\boldsymbol{\mu}} \Big). \tag{3.28}$$

The sets $S_{\boldsymbol{\nu}}^h$ are disjoint for different values of $\boldsymbol{\nu}$. The same observation holds for the hierarchical set of indices

$$\mathcal{J}_n := \mathcal{G}_n \setminus \mathcal{G}_{n-1}, \tag{3.29}$$

$$\mathcal{J}_{\boldsymbol{\nu}} := \mathcal{J}_{\nu_1} \times \cdots \times \mathcal{J}_{\nu_n} = \mathcal{G}_{\boldsymbol{\nu}} \setminus \Big(\bigcup_{\boldsymbol{\mu} \le \boldsymbol{\nu}, \boldsymbol{\mu} \ne \boldsymbol{\nu}} \mathcal{G}_{\boldsymbol{\mu}} \Big). \tag{3.30}$$

Consider now an arbitrary index set $\mathcal{I} \subset \mathbb{N}_0^n$ and define the operator

$$I_{\mathcal{I}} f := \sum_{\boldsymbol{\nu} \in \mathcal{I}} \Delta_{\boldsymbol{\nu}} f \tag{3.31}$$

and the interpolation points

$$S_{\mathcal{I}} := \bigcup_{\boldsymbol{\nu} \in \mathcal{I}} S_{\boldsymbol{\nu}}^h. \tag{3.32}$$

If $\mathcal{I} = \{\boldsymbol{\mu} \in \mathbb{N}_0^n : \mathbf{0} \le \boldsymbol{\mu} \le \boldsymbol{\nu}\}$ then the operators $I_{\mathcal{I}}$ and $I_{\boldsymbol{\nu}}$ are equivalent. Therefore, $I_{\mathcal{I}}$ is an interpolating operator on the set $S_{\mathcal{I}}$. In the general case, it is important to find what are the minimal conditions that $\mathcal{I}$ must satisfy in order to guarantee that the operator $I_{\mathcal{I}}$ is interpolatory on $S_{\mathcal{I}}$.

First, note that for every point $x \in S_{n-1}$ it holds

$$\Delta_n f(x) = I_n f(x) - I_{n-1} f(x) = f(x) - f(x) = 0 \tag{3.33}$$

because both I_n and I_{n-1} are interpolating operators on S_{n-1}. In multiple dimensions this relation becomes

$$\Delta_{\boldsymbol{\nu}} f(\boldsymbol{x}) = (\Delta_{n_1} \otimes \cdots \otimes (I_{\nu_i} - I_{\nu_i - 1}) \otimes \cdots \otimes \Delta_{\nu_n}) f(\boldsymbol{x}) = 0 \tag{3.34}$$

if $x_i \in S_{\nu_i - 1}$ for any $i = 1, \ldots n$. Therefore, among the points $S_{\boldsymbol{\nu}}$ the operator $\Delta_{\boldsymbol{\nu}}$ is non-zero only on the subset $S_{\boldsymbol{\nu}}^h$.

According to equation (3.26), for every point $\boldsymbol{x} \in I^n$ we have

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{\nu} \in \mathbb{N}_0^n} \Delta_{\boldsymbol{\nu}} f(\boldsymbol{x}). \tag{3.35}$$

This relation also holds for every $\boldsymbol{x} \in S_{\mathcal{I}} \subset I^n$. The goal is to find under what conditions this identity remains valid if the sum on the right-hand side runs only over the set $\mathcal{I}$. This would be true if all terms $\Delta_{\boldsymbol{\nu}} f$ for $\boldsymbol{\nu} \in \mathbb{N}_0^n \setminus \mathcal{I}$ are equal to zero when evaluated at any point $\boldsymbol{x} \in S_{\mathcal{I}}$. Fix one such point $\boldsymbol{x}$ and let $S_{\boldsymbol{\mu}}^h$, $\boldsymbol{\mu} \in \mathcal{I}$, be the unique hierarchical set to which $\boldsymbol{x}$ belongs. Consider now an arbitrary index $\boldsymbol{\nu} \in \mathbb{N}_0^n$. If $\nu_i > \mu_i$ for some $i = 1, \dots n$ then the corresponding difference operator evaluates to zero at the point $\boldsymbol{x}$, i.e. $\Delta_{\boldsymbol{\nu}} f(\boldsymbol{x}) = 0$. This follows from equation (3.34). Hence, in the infinite sum (3.35) for a fixed point $\boldsymbol{x} \in S_{\boldsymbol{\mu}}^h$ all terms with $\boldsymbol{\nu} \not\leq \boldsymbol{\mu}$ are zero. This leads to the following definition. The set $\mathcal{I}$ is called *admissible* if the following condition holds:

$$\{\boldsymbol{\nu} \in \mathbb{N}_0^n : \boldsymbol{\nu} \leq \boldsymbol{\mu}\} \subseteq \mathcal{I} \tag{3.36}$$

for every $\boldsymbol{\mu} \in \mathcal{I}$. If the set $\mathcal{I}$ is admissible, then the corresponding operator $I_{\mathcal{I}}$ is interpolatory for all points in $S_{\mathcal{I}}$. From now on, all index sets $\mathcal{I}$ in this text are assumed to have this property.

For such a set $\mathcal{I}$, the set of functions $V_{\mathcal{I}}$ is defined as

$$V_{\mathcal{I}} := \sum_{\boldsymbol{\nu} \in \mathcal{I}} V_{\boldsymbol{\nu}}. \tag{3.37}$$

It is clear that $\mathrm{Im}(I_{\mathcal{I}}) \subseteq V_{\mathcal{I}}$. The functions $\phi_{\boldsymbol{\nu}}$ for $\boldsymbol{\nu} \in \mathcal{I}$ form a basis of $V_{\mathcal{I}}$. Therefore, it follows that $\dim(V_{\mathcal{I}}) = |S_{\mathcal{I}}|$. Fix an index $\boldsymbol{\nu} \in \mathcal{I}$ and consider the image of the function $\phi_{\boldsymbol{\nu}}$ under the operator $I_{\mathcal{I}}$. Since $I_{\mathcal{I}}$ is interpolatory on $S_{\mathcal{I}}$, we know that $I_{\mathcal{I}} \phi_{\boldsymbol{\nu}}(\boldsymbol{x}) = \phi_{\boldsymbol{\nu}}(\boldsymbol{x})$ for every $\boldsymbol{x} \in S_{\mathcal{I}}$. This means that the image of $\phi_{\boldsymbol{\nu}}$ is $\phi_{\boldsymbol{\nu}}$ itself. Since every function in $V_{\mathcal{I}}$ can be represented as a finite linear combination of basis functions $\phi_{\boldsymbol{\nu}}$, it follows that $I_{\mathcal{I}}$ maps every function $f \in V_{\mathcal{I}}$ to itself. Therefore, $I_{\mathcal{I}}$ is a projection and $\mathrm{Im}(I_{\mathcal{I}}) = V_{\mathcal{I}}$.

3.4 The hierarchical basis

In the previous subsection it was proven that an interpolatory operator can be constructed for an arbitrary admissible index set $\mathcal{I}$. The next step is to provide an efficient method for computing this operator. Without such a method its practical value is very limited.

Subsection 3.2 described a procedure for computing efficiently the operator $I_{\boldsymbol{\nu}}$. The proposed method relied heavily on the tensor product structure of the full grid. Unfortunately, this method cannot be applied directly for a general admissible set $\mathcal{I}$ because this structure is no longer present. This is why a slightly different approach is needed to tackle this problem. To achieve this, the properties of the difference operators $\Delta_{\boldsymbol{\nu}}$ must first be explored in greater depth.

Consider the image of the operator Δ_n and denote it as $W_n := \mathrm{Im}(\Delta_n) \subseteq V_n$. The intersection of W_{n_1} and W_{n_2} is equal to $\{0\}$ for each $n_1 \neq n_2$. To prove this, assume that $n_1 < n_2$ and choose a function $\phi \in W_{n_1} \cap W_{n_2}$. Since $\phi \in W_{n_2}$, it follows that $\phi(x) = 0$ for each $x \in S_{n_2 - 1}$. Since $n_1 < n_2$, it also follows that $S_{n_1} \subseteq S_{n_2 - 1}$ and therefore $\phi(x) = 0$ for each $x \in S_{n_1}$. Hence, the function ϕ is identically equal to 0.

This observation is easily extended to the multidimensional case: $W_{\boldsymbol{\nu_1}} \cap W_{\boldsymbol{\nu_2}} = \{0\}$ for each $\boldsymbol{\nu_1} \neq \boldsymbol{\nu_2}$, where

$$W_{\boldsymbol{\nu}} = \bigotimes_{d=1}^{n} W_{\nu_i}. \tag{3.38}$$

Since $I_{\mathcal{I}} = \sum_{\boldsymbol{\nu}\in\boldsymbol{\mathcal{I}}} \Delta_{\boldsymbol{\nu}}$ it follows that $V_{\mathcal{I}}$ can be represented as a direct sum of the sets $W_{\boldsymbol{\nu}}$:

$$V_{\mathcal{I}} = \sum_{\boldsymbol{\nu}\in\mathcal{I}} V_{\boldsymbol{\nu}} = \bigoplus_{\boldsymbol{\nu}\in\mathcal{I}} W_{\boldsymbol{\nu}}. \tag{3.39}$$

It also holds that $V_{\boldsymbol{\nu}_1} \cap W_{\boldsymbol{\nu}_2} = \{\mathbf{0}\}$ for every $\boldsymbol{\nu_2} \in \mathbb{N}_0^n$ and every $\boldsymbol{\nu}_2 \not\leq \boldsymbol{\nu}_1$.

A sequence of basis function sets $\mathcal{B}_n^h$, $n \in \mathbb{N}_0 \cup \{-1\}$, is called a hierarchical basis system if the following conditions are satisfied:

1. The sequence is increasing, i.e. $\mathcal{B}_{n-1}^h \subset \mathcal{B}_n^h$ for every $n \in \mathbb{N}_0$.
2. $\mathcal{B}_n^h$ is a basis of V_n for $n \in \mathbb{N}_0 \cup \{-1\}$.
3. $\mathcal{B}_n^h \setminus \mathcal{B}_{n-1}^h$ is a basis of W_n for $n \in \mathbb{N}_0$.

Condition 3 has an important consequence: for every hierarchical basis function $\psi \in \mathcal{B}_n^h \setminus \mathcal{B}_{n-1}^h$ and every point $x \in S_{n-1}$ it holds $\psi(x) = 0$. This follows from the fact that $\psi \in W_n$, the definition of W_n and equation (3.33). More importantly, this consequence is actually equivalent to condition 3. To prove this, assume that every basis function $\psi \in \mathcal{B}_n^h \setminus \mathcal{B}_{n-1}^h$ satisfies $\psi(x) = 0$ for every $x \in S_{n-1}$. According to equation (3.39) in one dimension, the function ψ can be represented as $\psi = v + w$, where $v \in V_{n-1}$ and $w \in W_n$. Based on the assumption and on equation (3.33), it follows that $v(x) = 0$ for every $x \in S_{n-1}$ and therefore v is identically zero. Hence, each basis function $\psi \in \mathcal{B}_n^h \setminus \mathcal{B}_{n-1}^h$ belongs to W_n. The conclusion follows from the fact that the number of elements in $\mathcal{B}_n^h \setminus \mathcal{B}_{n-1}^h$ and the dimension of W_n are both equal to $|J_n|$. Condition 3 can now be rephrased:

3′. For every $n \in \mathbb{N}_0$, $\psi \in \mathcal{B}_n^h \setminus \mathcal{B}_{n-1}^h$ and $x \in S_{n-1}$ it holds $\psi(x) = 0$.

This is the key property of the hierarchical basis that makes the fast computation of the operator $I_{\mathcal{I}}$ possible. While the basis functions $\mathcal{B}_n$ satisfy conditions 1 and 2, they are not guaranteed to satisfy condition 3. This is why the hierarchical basis must be defined additionally.

The sets $\mathcal{B}_n^h$ can now be written as $\mathcal{B}_n^h = \{\psi_i : i \in \mathcal{G}_n\}$, where ψ_i are the hierarchical basis functions. Condition 3′ means that $\psi_i(x_j) = 0$ if $i \in \mathcal{J}_n$ and $j \in \mathcal{G}_{n-1}$ for some $n \in \mathbb{N}$. In the multidimensional case, it follows that $\psi_{\boldsymbol{i}}(\boldsymbol{x_j}) = 0$ for every $\boldsymbol{i} \in \mathcal{J}_{\boldsymbol{\nu}}$ and $\boldsymbol{j} \in \mathcal{G}_{\boldsymbol{\nu}} \setminus \mathcal{J}_{\boldsymbol{\nu}}$.

Since both $\mathcal{B}_n$ and $\mathcal{B}_n^h$ span V_n, the image $I_n f$ of every function $f \in X$ has two representations, corresponding to these two basis sets:

$$I_n f = \sum_{i\in\mathcal{G}_n} \hat{f}_{i,n}\phi_i, \tag{3.40}$$

$$\check{I}_n f = \sum_{i\in\mathcal{G}_n} \check{f}_{i,n}\psi_i. \tag{3.41}$$

In fact, it is easy to prove that the coefficients $\check{f}_{i,n}$ depend only on the first index i. To see this, consider the hierarchical representation of $\Delta_{n+1} f$:

$$\Delta_{n+1} f = \check{I}_{n+1} f - \check{I}_n f = \sum_{i\in\mathcal{G}_n} (\check{f}_{i,n+1} - \check{f}_{i,n})\psi_i + \sum_{i\in\mathcal{J}_{n+1}} \check{f}_{i,n+1}\psi_i = u + v. \tag{3.42}$$

The functions u and v belong to the sets V_n and W_{n+1}, respectively. If u is non-zero then the sum $u + v$ would belong to the set $V_{n+1} \setminus W_{n+1}$. However, this is not true because by definition

$\Delta_{n+1} f \in W_{n+1}$. This means that $u \equiv 0$ and all terms in the first summation in equation (3.42) are equal to zero, i.e. $\check{f}_{i,n} = \check{f}_{i,n+1}$ for every $i \in \mathcal{G}_n$. Therefore $\check{f}_{i,n}$ does not depend on n. Thanks to this observation, equation (3.41) can be rewritten as

$$I_n f = \sum_{i \in \mathcal{G}_n} \check{f}_i \psi_i. \tag{3.43}$$

The procedure of transforming the regular representation of $I_n f$ to its hierarchical representation $\check{I}_n f$ is called hierarchization. The inverse procedure is called dehierarchization. These two operations play an important role in the sparse grid theory. They are equivalent to transforming the vector of coefficients $\hat{f}_{i,n}$ to $\check{f}_i$ and vice versa. In the most general case, this can be done with a matrix multiplication which has quadratic complexity.

First, consider the question of existence of such a hierarchical basis. For each $j \in \mathbb{N}_0$, ψ_j is a linear combination of the functions $\{\phi_0, \dots \phi_j\}$:

$$\psi_j = \sum_{k=0}^{j} c_{j,k} \phi_k. \tag{3.44}$$

Assume that $j \in J_n$. According to condition $3'$, for each interpolation point x_i, $i \in G_{n-1}$, the following should hold:

$$\psi_j(x_i) = \sum_{k=0}^{j} c_{j,k} \phi_k(x_i) = 0. \tag{3.45}$$

These conditions form a system of g_{n-1} equations with $j+1$ unknown variables:

$$\begin{pmatrix} \phi_0(x_0) & \dots & \phi_j(x_0) \\ \vdots & \ddots & \vdots \\ \phi_0(x_{g_{n-1}-1}) & \dots & \phi_j(x_{g_{n-1}-1}) \end{pmatrix} \begin{pmatrix} c_{j,0} \\ \vdots \\ c_{j,j} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{3.46}$$

This system has a solution space with dimension $j+1-g_{n-1}$. Each of the solutions corresponds to one possible choice for a hierarchical basis function ψ_j. To fix a specific one, it is required to add $j+1-g_{n-1}$ more equations to the system:

$$\begin{pmatrix} \phi_0(x_{g_{n-1}}) & \dots & \phi_j(x_{g_{n-1}}) \\ \vdots & \ddots & \vdots \\ \phi_0(x_j) & \dots & \phi_j(x_j) \end{pmatrix} \begin{pmatrix} c_{j,0} \\ \vdots \\ c_{j,j} \end{pmatrix} = \begin{pmatrix} d_{g_{n-1},j} \\ \vdots \\ d_{j,j} \end{pmatrix}, \tag{3.47}$$

where the vector on the right-hand side is arbitrary non-zero $(j+1-g_{n-1})$-dimensional vector. For some transforms, there is a specific choice of this vector that makes the computations significantly easier. Good examples are the Fourier and Chebyshev transforms. For others, there are no obvious advantages of choosing one vector over another one. In these cases one can simply choose the vector $d_{i,j} = \delta_{i,j}$. In fact, this choice has certain computational advantages that will be discussed below.

The full square system of equations for $c_{j,k}$ can now be written as

$$A_{g_n-1} \boldsymbol{c}_j = \boldsymbol{d}_j, \tag{3.48}$$

where $\boldsymbol{c}_j = (c_{j,0}, \dots c_{j,j}, 0, \dots 0)^T$ and $\boldsymbol{d}_j = (0, \dots 0, d_{g_{n-1}}, \dots d_j, 0, \dots 0)^T$ are g_n-dimensional vectors whose last entries are zeroes. Define the $g_n \times g_n$ matrix C_{g_n-1} as

$$C_{g_n-1} := \begin{pmatrix} c_{0,0} & \dots & c_{0,g_n-1} \\ \vdots & \ddots & \vdots \\ c_{g_n-1,0} & \dots & c_{g_n-1,g_n-1} \end{pmatrix}. \tag{3.49}$$

This is simply the change of basis matrix between the regular and the hierarchical basis. If $\Phi_j := (\phi_0, \dots \phi_j)^T$ and $\Psi_j := (\psi_0, \dots \psi_j)^T$ then

$$\Psi_{g_n-1} = C_{g_n-1}\Phi_{g_n-1}. \tag{3.50}$$

The interpolation operator in the regular basis form can be written as

$$I_n f = \sum_{i \in \mathcal{G}_n} \hat{f}_{i,n}\phi_i = \hat{F}^T_{g_n-1}\Phi_{g_n-1}, \tag{3.51}$$

where $\hat{F}_{g_n-1} = (\hat{f}_{0,n}, \dots \hat{f}_{g_n-1,n})^T$. In analogy, the hierarchical form is

$$\check{I}_n f = \sum_{i \in \mathcal{G}_n} \check{f}_i\psi_i = \check{F}^T_{g_n-1}\Psi_{g_n-1}, \tag{3.52}$$

where $\check{F}_{g_n-1} = (\check{f}_0, \dots \check{f}_{g_n-1})^T$. Using equation (3.50), we deduce the following general equations for hierarchization

$$\hat{F}_{g_n-1} = C^{-T}_{g_n-1}\check{F}_{g_n-1} \tag{3.53}$$

and dehierarchization

$$\check{F}_{g_n-1} = C^T_{g_n-1}\hat{F}_{g_n-1}. \tag{3.54}$$

As mentioned earlier, in the general case these operations have complexity $\mathrm{O}(g_n^2)$. For certain transforms the vectors $\boldsymbol{d}_j$ can be chosen in such a way that the matrices C and C^{-1} are sparse. In these cases, the complexity can become as small as linear. In the general case it is reasonable to choose the $\boldsymbol{d}_j = \boldsymbol{e}_j$. With this choice, equation (3.48) becomes

$$A_{g_n-1}C_{g_n-1} = I_{g_n}. \tag{3.55}$$

Therefore, once we have computed the matrices A_{g_n-1} and their inverses, it is no longer necessary to compute C_{g_n-1} and $C^{-1}_{g_n-1}$ separately. This typically saves a lot of computational time.

As a conclusion of this subsection, we will prove the following lemma.

Lemma 3.1. *Assume that for every $i \in \mathcal{J}_n$, ψ_i is the only hierarchical basis function that includes ϕ_i in its representation (3.44). Assume also that the coefficient in front of ϕ_i in this representation is* 1*. Then $\check{f}_i = \hat{f}_{i,n}$ for every $i \in \mathcal{J}_n$.*

Proof. The assumption in the statement of the lemma simply means that the i-th row of the matrix $C^T_{g_n-1}$ contains a single non-zero entry in position i and this entry is equal to 1. Equation (3.54) yields the desired result. □

3.5 Computing the general interpolation operator

Subsection 3.2 described a procedure for the efficient computation of the operator $I_{\boldsymbol{\nu}}$. The computation relied heavily on the tensor product structure of the operator and the underlying grid. This structure is no longer present for the general operator $I_{\mathcal{I}}$ and therefore the same method cannot be applied directly. Fortunately, this problem can be circumvented by exploiting the convenient properties of the hierarchical operator $\check{I}_{\mathcal{I}}$. Working in the hierarchical basis allows the application of an almost identical efficient method.

Just as in tensor product case, we will define an iterative approach for finding the hierarchical coefficients $\check{f}_{\boldsymbol{\alpha}}$. For $i = 0, 1 \dots n$ define the parametrized coefficient functions $\check{f}^i_{\boldsymbol{\alpha}}(x_{i+1}, \dots x_n)$ with the equations

$$\check{f}^0(\boldsymbol{x}) = f(\boldsymbol{x}), \tag{3.56}$$

$$\check{I}_{\nu_{i+1}} \check{f}^i_{\boldsymbol{\alpha}}(x_{i+1}, \dots x_n) = \sum_{j \in \mathcal{G}_{\nu_i}} \check{f}^{i+1}_{(\boldsymbol{\alpha},j)}(x_{i+2}, \dots x_n)\phi_j(x_{i+1}). \tag{3.57}$$

The goal is to prove that $\check{f}_{\boldsymbol{\alpha}} = \check{f}^n_{\boldsymbol{\alpha}}$. This is equivalent to proving that the operators $\check{I}_{\mathcal{I}}$ and $\check{I}'_{\mathcal{I}}$ are identical, where

$$\check{I}'_{\mathcal{I}} f := \sum_{\boldsymbol{\mu} \in \mathcal{I}} \sum_{\boldsymbol{i} \in \mathcal{J}_{\boldsymbol{\mu}}} \check{f}^n_{\boldsymbol{i}} \psi_{\boldsymbol{i}}(\boldsymbol{x}). \tag{3.58}$$

To prove this, it is enough to verify that $\check{I}_{\mathcal{I}} f(\boldsymbol{x}) = \check{I}'_{\mathcal{I}} f(\boldsymbol{x})$ for every $\boldsymbol{x} \in S_{\mathcal{I}}$. This can be done in two steps.

First, assume that the set $\mathcal{I}$ has a product structure, i.e. $\mathcal{I} = \{\boldsymbol{\mu} \in \mathbb{N}_0^n : \boldsymbol{\mu} \leq \boldsymbol{\nu}\}$ for some index $\boldsymbol{\nu}$. This case can be proven by induction over the dimension n. The proof is completely identical to the one in subsection 3.2 and thus will be omitted.

Assume now that $\mathcal{I}$ no longer has a product structure and choose a point $\boldsymbol{x} \in S_{\mathcal{I}}$. Because of the hierarchical decomposition of $S_{\mathcal{I}}$ (3.32), there exist a unique index $\boldsymbol{\nu} \in \mathcal{I}$ such that $\boldsymbol{x} \in S^h_{\boldsymbol{\nu}}$. The value of $\check{I}'_{\mathcal{I}} f$ at the point $\boldsymbol{x}$ is

$$\check{I}'_{\mathcal{I}} f(\boldsymbol{x}) = \sum_{\boldsymbol{\mu} \in \mathcal{I}} \sum_{\boldsymbol{i} \in J_{\boldsymbol{\mu}}} \check{f}^n_{\boldsymbol{i}} \psi_{\boldsymbol{i}}(\boldsymbol{x}). \tag{3.59}$$

The key observation here is that $\psi_{\boldsymbol{i}}(\boldsymbol{x}) = 0$ for all indices $\boldsymbol{i} \in \mathcal{J}_{\boldsymbol{\mu}}$ with $\boldsymbol{\mu} \not\leq \boldsymbol{\nu}$. Discarding these terms leads to the equation

$$\check{I}'_{\mathcal{I}} f(\boldsymbol{x}) = \sum_{\boldsymbol{\mu} \leq \boldsymbol{\nu}} \sum_{\boldsymbol{i} \in \mathcal{J}_{\boldsymbol{\mu}}} \check{f}^n_{\boldsymbol{i}} \psi_{\boldsymbol{i}}(\boldsymbol{x}) = \check{I}'_{\{\boldsymbol{\mu} \leq \boldsymbol{\nu}\}} f(\boldsymbol{x}) = f(\boldsymbol{x}). \tag{3.60}$$

The last equality holds because $\{\boldsymbol{\mu} \leq \boldsymbol{\nu}\}$ is a product set and the statement is already proven for it.

This method allows the computation of the hierarchical coefficients by applying a series of one-dimensional transforms in each direction. However, usually we are interested in finding the regular basis coefficients. There are multiple reasons to prefer the regular basis representation over the hierarchical one. The regular basis usually has some meaning and its coefficients can be interpreted in certain ways. The case is different with the hierarchical basis. Its basis functions have no meaning outside the context of sparse grids. Therefore, the hierarchical coefficients cannot be interpreted as easily as the regular ones. Another reason is that the regular basis functions are much faster to compute. Each hierarchical basis function is computed as a linear combination of up to a linear number of regular basis functions. Therefore, evaluating the interpolant at a given point in its hierarchical form is several times slower compared to evaluating the regular interpolant.

Because of these reasons we also need an efficient method for finding the regular coefficients from the hierarchical ones. This can be achieved with a similar iterative approach over the dimension n. Define the coefficients $c^k_{\boldsymbol{i}}$ for $\boldsymbol{i} \in \bigcup_{\boldsymbol{\mu} \in \mathcal{I}} \mathcal{J}_{\boldsymbol{\mu}}$ and $k = 0, \dots n$ as

$$I_{\mathcal{I}} f = \sum_{\boldsymbol{\mu} \in \mathcal{I}} \sum_{\boldsymbol{i} \in \mathcal{J}_{\boldsymbol{\mu}}} c^k_{\boldsymbol{i}} (\psi_{\boldsymbol{i}'} \otimes \phi_{\boldsymbol{i}''}), \tag{3.61}$$

where $\boldsymbol{i} = (\boldsymbol{i}', \boldsymbol{i}'')$, $\boldsymbol{i}' \in \mathbb{N}_0^{n-k}$ and $\boldsymbol{i}'' \in \mathbb{N}_0^k$. From this definition, it follows that $c_{\boldsymbol{i}}^0 = \check{f}_{\boldsymbol{i}}$ and $c_{\boldsymbol{i}}^n = \hat{f}_{i,\boldsymbol{\mu}}$. The goal is to find an efficient way to transform the coefficients $c_{\boldsymbol{i}}^k$ to $c_{\boldsymbol{i}}^{k+1}$. It will be demonstrated how to do this for $k = 0$. For other values of k, the only difference is the more complicated notation. First, we need a few definitions:

$$\boldsymbol{\mu} = (\boldsymbol{\mu}', \mu_n), \boldsymbol{\mu}' \in \mathbb{N}_0^{n-1}, \mu_n \in \mathbb{N}_0, \tag{3.62}$$

$$\mathcal{I}' = \{\boldsymbol{\mu}' : \exists \boldsymbol{\mu} \in \mathcal{I} \text{ such that } \boldsymbol{\mu} = (\boldsymbol{\mu}', \mu_n)\}, \tag{3.63}$$

$$h(\boldsymbol{\mu}') = \max\{\mu_n : \exists \boldsymbol{\mu} \in \mathcal{I} \text{ such that } \boldsymbol{\mu} = (\boldsymbol{\mu}', \mu_n)\}. \tag{3.64}$$

The set $\mathcal{I}'$ is simply the projection for the set $\mathcal{I}$ along the n-th dimension. The function $h(\boldsymbol{\mu}')$ is a height function which denotes the height of the set $\mathcal{I}$ at the point $\boldsymbol{\mu}'$. Note that the admissibility property of $\mathcal{I}$ implies that $(\boldsymbol{\mu}', \mu_n) \in \mathcal{I}$ for every $\boldsymbol{\mu}' \in \mathcal{I}$ and $\mu_n = 0, \dots h(\boldsymbol{\mu}')$. We are now ready to proceed to the first iteration step:

$$\begin{aligned}
I_{\mathcal{I}} f &= \sum_{\boldsymbol{\mu} \in \mathcal{I}} \sum_{\boldsymbol{i} \in \mathcal{J}_{\boldsymbol{\mu}}} \check{f}_{\boldsymbol{i}} \psi_{\boldsymbol{i}} \\
&= \sum_{\boldsymbol{\mu} \in \mathcal{I}} \sum_{\boldsymbol{i} \in \mathcal{J}_{\boldsymbol{\mu}}} c_{\boldsymbol{i}}^0 \psi_{\boldsymbol{i}} \\
&\overset{(1)}{=} \sum_{\boldsymbol{\mu}' \in \mathcal{I}'} \sum_{\mu_n=0}^{h(\boldsymbol{\mu}')} \sum_{\boldsymbol{i}' \in \mathcal{J}_{\boldsymbol{\mu}'}} \sum_{i_n \in \mathcal{J}_{\mu_n}} c_{\boldsymbol{i}}^0 (\psi_{\boldsymbol{i}'} \otimes \psi_{i_n}) \\
&\overset{(2)}{=} \sum_{\boldsymbol{\mu}' \in \mathcal{I}'} \sum_{\boldsymbol{i}' \in \mathcal{J}_{\boldsymbol{\mu}'}} \psi_{\boldsymbol{i}'} \otimes \Big(\sum_{\mu_n=0}^{h(\boldsymbol{\mu}')} \sum_{i_n \in \mathcal{J}_{\mu_n}} c_{\boldsymbol{i}}^0 \psi_{i_n} \Big) \\
&\overset{(3)}{=} \sum_{\boldsymbol{\mu}' \in \mathcal{I}'} \sum_{\boldsymbol{i}' \in \mathcal{J}_{\boldsymbol{\mu}'}} \psi_{\boldsymbol{i}'} \otimes \Big(\sum_{i_n \in \mathcal{G}_{h(\mu_n)}} c_{\boldsymbol{i}}^0 \psi_{i_n} \Big) \\
&\overset{(4)}{=} \sum_{\boldsymbol{\mu}' \in \mathcal{I}'} \sum_{\boldsymbol{i}' \in \mathcal{J}_{\boldsymbol{\mu}'}} \psi_{\boldsymbol{i}'} \otimes \Big(\sum_{i_n \in \mathcal{G}_{h(\mu_n)}} c_{\boldsymbol{i}}^1 \phi_{i_n} \Big) \\
&\overset{(5)}{=} \sum_{\boldsymbol{\mu} \in \mathcal{I}} \sum_{\boldsymbol{i} \in \mathcal{J}_{\boldsymbol{\mu}}} c_{\boldsymbol{i}}^1 (\psi_{\boldsymbol{i}'} \otimes \phi_{i_n}).
\end{aligned} \tag{3.65}$$

In the calculation above, step (1) is achieved by splitting the summation into one summation over the dimensions $1, \dots n-1$ and a second summation over dimension n. Note that because of the non-product structure of $\mathcal{I}$, the inner sum runs over a different set of numbers for each $\boldsymbol{\mu}' \in \mathcal{I}$. Step (2) comes from reordering the summations. Step (3) uses the definition of $\mathcal{J}_n$ and $\mathcal{G}_n$. Step (4) is the crucial step since it applies a one-dimensional dehierarchization on the sum in the parenthesis. Finally, step (5) simply reverts back the summation to its original form. The conclusion of this computation is that the coefficients $\boldsymbol{c}_{\boldsymbol{i}}^0$ can be transformed to $\boldsymbol{c}_{\boldsymbol{i}}^1$ by applying one-dimensional dehierarchizations on a subsets of coefficients, corresponding to each projected index $\boldsymbol{\mu}' \in \mathcal{I}$.

3.6 Algorithms

The previous subsection described two iterative methods for finding the hierarchical interpolant and for multidimensional dehierarchization. It is useful to see how these methods can be written in algorithmic form. In the algorithms below, $\boldsymbol{u}$ denotes a vector of values

$$\boldsymbol{u} := \{u_{\boldsymbol{i}} : \boldsymbol{\mu} \in \mathcal{I}, \boldsymbol{i} \in \mathcal{J}_{\boldsymbol{\mu}}\} \tag{3.66}$$

corresponding to each interpolation point. The upper boundary of $\mathcal{I}$ can also be denoted as

$$\mathcal{M}_d(\mathcal{I}) := \{\boldsymbol{\mu} \in \mathcal{I} : \boldsymbol{\mu} + \boldsymbol{e}_d \notin \mathcal{I}\}, \tag{3.67}$$

where $\boldsymbol{e}_d$ is the standard basis vector in direction d. If the height function in direction d is defined as

$$h_d(\boldsymbol{\mu}', \boldsymbol{\mu}'') := \max\{\mu_d : \exists \boldsymbol{\mu} \in \mathcal{I} \text{ such that } \boldsymbol{\mu} = (\boldsymbol{\mu}', \mu_d, \boldsymbol{\mu}''), \boldsymbol{\mu}' \in \mathbb{N}_0^{d-1}\}, \tag{3.68}$$

then the set $\mathcal{M}_d(\mathcal{I})$ can also be defined as

$$\mathcal{M}_d(\mathcal{I}) = \{(\boldsymbol{\mu}', h_d(\boldsymbol{\mu}', \boldsymbol{\mu}''), \boldsymbol{\mu}'') : (\boldsymbol{\mu}', \mu_d, \boldsymbol{\mu}'') \in \mathcal{I} \text{ and } \boldsymbol{\mu}' \in \mathbb{N}_0^{d-1}\}. \tag{3.69}$$

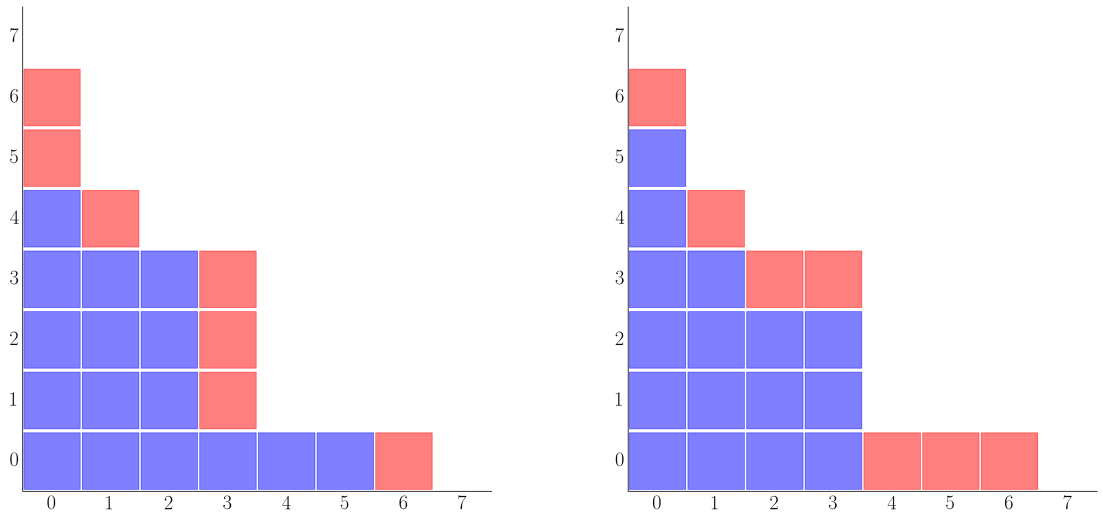


Figure 2: The upper boundary of $\mathcal{I}$ in both dimensions. Each index in $\mathcal{I}$ is depicted with a square. Red squares are in the upper boundary set. Left image shows $\mathcal{M}_1(\mathcal{I})$ and right image shows $\mathcal{M}_2(\mathcal{I})$

Furthermore, assume that we have the following one-dimensional algorithms. All of them work on a one-dimensional vector $\boldsymbol{v}$ and work in-place:

- TRANSFORM($\boldsymbol{v}$) – Transforms a function value vector $\boldsymbol{v}$ into the corresponding vector of regular coefficients.
- INVERSE_TRANSFORM($\boldsymbol{v}$) – Transforms a regular coefficient vector $\boldsymbol{v}$ into the corresponding function value vector.
- HIERARCHIZE($\boldsymbol{v}$) – Transforms regular coefficients into hierarchical coefficients.
- DEHIERARCHIZE($\boldsymbol{v}$) – Transforms hierarchical coefficients into regular coefficients.

In the general case, these algorithms are simply implemented as the matrix multiplications given in (3.7), (3.53) and (3.54).

We are finally ready to write the multidimensional algorithms.

Algorithm 1 Transforms the vector $\boldsymbol{u}$ of function values into a vector of hierarchical coefficients.

procedure *N_TRANSFORM*
  for $d = 1, \dots n$ **do**
    for $\boldsymbol{\mu} \in \mathcal{M}_d(\mathcal{I})$ **do**
      for $\boldsymbol{i}' \in \mathcal{J}_{\mu_1} \times \cdots \times \mathcal{J}_{\mu_{d-1}}, \boldsymbol{i}'' \in \mathcal{J}_{\mu_{d+1}} \times \cdots \times \mathcal{J}_{\mu_n}$ **do**
        $TRANSFORM(u_{(\boldsymbol{i}',0,\boldsymbol{i}'')}, \dots, u_{(\boldsymbol{i}',g_{\mu_d}-1,\boldsymbol{i}'')})$
        $HIERARCHIZE(u_{(\boldsymbol{i}',0,\boldsymbol{i}'')}, \dots, u_{(\boldsymbol{i}',g_{\mu_d}-1,\boldsymbol{i}'')})$
      end for
    end for
  end for
end procedure

Algorithm 2 Transforms the hierarchical coefficients vector $\boldsymbol{u}$ into a vector of function values.

procedure *INVERSE_N_TRANSFORM*
  for $d = n, \dots 1$ **do**
    for $\boldsymbol{\mu} \in \mathcal{M}_d(\mathcal{I})$ **do**
      for $\boldsymbol{i}' \in \mathcal{J}_{\mu_1} \times \cdots \times \mathcal{J}_{\mu_{d-1}}, \boldsymbol{i}'' \in \mathcal{J}_{\mu_{d+1}} \times \cdots \times \mathcal{J}_{\mu_n}$ **do**
        $DEHIERARCHIZE(u_{(\boldsymbol{i}',0,\boldsymbol{i}'')}, \dots, u_{(\boldsymbol{i}',g_{\mu_d}-1,\boldsymbol{i}'')})$
        $INVERSE\_TRANSFORM(u_{(\boldsymbol{i}',0,\boldsymbol{i}'')}, \dots, u_{(\boldsymbol{i}',g_{\mu_d}-1,\boldsymbol{i}'')})$
      end for
    end for
  end for
end procedure

Algorithm 3 Transforms the vector $\boldsymbol{u}$ of regular coefficients into a vector of hierarchical coefficients.

procedure *N_HIERARCHIZE*
  for $d = 1, \dots n$ **do**
    for $\boldsymbol{\mu} \in \mathcal{M}_d(\mathcal{I})$ **do**
      for $\boldsymbol{i}' \in \mathcal{J}_{\mu_1} \times \cdots \times \mathcal{J}_{\mu_{d-1}}, \boldsymbol{i}'' \in \mathcal{J}_{\mu_{d+1}} \times \cdots \times \mathcal{J}_{\mu_n}$ **do**
        $HIERARCHIZE(u_{(\boldsymbol{i}',0,\boldsymbol{i}'')}, \dots, u_{(\boldsymbol{i}',g_{\mu_d}-1,\boldsymbol{i}'')})$
      end for
    end for
  end for
end procedure

Algorithm 4 Transforms the vector $\boldsymbol{u}$ of hierarchical coefficients into a vector of regular coefficients.

1: **procedure** *N_DEHIERARCHIZE*
2: **for** $d = n, \dots 1$ **do**
3: **for** $\boldsymbol{\mu} \in \mathcal{M}_d(\mathcal{I})$ **do**
4: **for** $\boldsymbol{i}' \in \mathcal{J}_{\mu_1} \times \cdots \times \mathcal{J}_{\mu_{d-1}}, \boldsymbol{i}'' \in \mathcal{J}_{\mu_{d+1}} \times \cdots \times \mathcal{J}_{\mu_n}$ **do**
5: $DEHIERARCHIZE(u_{(\boldsymbol{i}',0,\boldsymbol{i}'')}, \dots, u_{(\boldsymbol{i}',g_{\mu_d}-1,\boldsymbol{i}'')})$
6: **end for**
7: **end for**
8: **end for**
9: **end procedure**

4 Dyadic Sparse Grids

The primary goal of the previous section was to explore the interpolatory properties of operators on sparse grids. Apart from purely theoretical results, it also included efficient algorithms for the computation of these operators. While these questions are interesting on their own, they fail to demonstrate the importance of sparse grids, namely their approximation properties. The main reasons to compute an interpolant on a sparse grid is to use it for computation of approximate function values outside the grid. To this moment, this question has not been addressed in this text. The reason is that section 3 aimed to introduce spare grids in their most general setting and deriving error bounds in such a setting is not possible. Therefore, in order to get error estimates for the interpolant, it is required to work with concrete grids. This section is devoted to several such grids.

To transform the general definition of a sparse grid from the previous section into a concrete one, the following information should be provided:

- The function space X
- The interpolation interval I and the interpolation points x_n
- The number of grid points per level g_n
- The regular basis functions ϕ_n
- The hierarchical basis functions ψ_n

Additionally, if the transform allows it, the general quadratic procedures *TRANSFORM*, *INVERSE_TRANSFORM*, *HIERARCHIZE* and *DEHIERARCHIZE* should be replaced with faster alternatives, tailored to the specific type of transform.

Finally, if we wish to derive error bounds for the interpolant, the space X^n is usually too big to allow any useful results. This is why the error estimates are derived for functions f in a subspace $Y \in X^n$ which does not necessarily have a product structure.

4.1 Dyadic Fourier Sparse Grids

4.1.1 Definition

Fourier sparse grids are based on the Fourier transform and thus are used for approximation of periodic functions. The interpolation interval I is the 1-torus $\mathbb{T}$ and the function space X is $L^2(\mathbb{T})$. The basis functions are defined as

$$\phi_n(x) = \omega_{\sigma(n)}(x) = e^{i\sigma(n)x}. \tag{4.1}$$

As we know from subsection 2.1, the points that are used for the Fourier transform with g_n nodes are $\{2k\pi/g_n : k = 0, 1, \dots g_n - 1\}$. This specific choice of points has many benefits for the accuracy and efficiency of the transform. This is why, it is best to use these points for the Fourier sparse grid. Since this text focuses only on nested sparse grids, this means that the set of points $\{2k\pi/g_n : k = 0, 1, \dots g_n - 1\}$ must be a subset of $\{2k\pi/g_{n+1} : k = 0, 1, \dots g_{n+1} - 1\}$

for every n. It is easy to see that setting $g_n = 2^n$ satisfies this requirement. With this choice of grid points and level sizes, it becomes possible to use the fast Fourier transform and its inverse for the 1-dimensional algorithms. Therefore, the general functions *TRANSFORM* and *INVERSE_TRANSFORM* with quadratic complexity can be replaced with the functions *FFT* and *IFFT*, which have complexity $O(g_n \log g_n)$.

The next important question is the choice of hierarchical basis functions. As discussed in subsection 3.4, there is an infinite space of choices for this basis. However in the case of dyadic Fourier grids, one specific choice leads to a change of basis matrix which is very sparse. The hierarchical basis functions are defined as

$$\psi_n = \begin{cases} \phi_0 & n = 0, \\ \phi_n - \phi_{2^l - n - 1} & 2^{l-1} \leq n \leq 2^l - 1. \end{cases} \tag{4.2}$$

The change of basis matrices $C_{g_n - 1}$ for the first few values of n are

$$\begin{pmatrix} 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.3}$$

The inverse matrices are also sparse:

$$\begin{pmatrix} 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.4}$$

This sparsity can be exploited to implement faster hierarchization and dehierarchization algorithms. More precisely, the complexity of the specialized versions of the *HIERARCHIZE* and *DEHIERARCHIZE* functions is $O(g_n)$, compared to the general case $O(g_n^2)$. One way to implement these functions is to use regular sparse matrix multiplication. Another and more efficient way is to avoid the matrix multiplication whatsoever and to replace it with iteration over the grid levels.

Algorithm 5 Hierarchization of a coefficient vector $\boldsymbol{a}$ of length g_n.

```
1: procedure FOURIER_HIERARCHIZE
2:     for l = n downto 1 do
3:         for k = 2^l − 1 downto 2^(l−1) do
4:             a_(2^l − 1 − k) += a_k
5:         end for
6:     end for
7: end procedure
```

Algorithm 6 Dehierarchization of a coefficient vector $\boldsymbol{a}$ of length g_n.

```
procedure FOURIER_DEHIERARCHIZE
    for l = 1 to n do
        for k = 2^{l-1} to 2^l - 1 do
            a_{2^l - 1 - k} -= a_k
        end for
    end for
end procedure
```

The definition of the hierarchical basis in this form has another important consequence. The hierarchical basis functions satisfy the requirements of Lemma 3.1. Therefore, $\check{f}_{\boldsymbol{k}} = \hat{f}_{\boldsymbol{k},\boldsymbol{\nu}}$ for every $\boldsymbol{k} \in \mathcal{J}_{\boldsymbol{\nu}}$. Combining this result with the aliasing formula (2.1) yields an aliasing formula for the hierarchical coefficients.

Lemma 4.1. *The hierarchical Fourier coefficients satisfy the identity*

$$\check{f}_{\boldsymbol{k}} = \sum_{\boldsymbol{m} \in \mathbb{Z}^n} \hat{f}_{\sigma(\boldsymbol{k}) + \boldsymbol{m} g_{\boldsymbol{\nu}}} = \sum_{\boldsymbol{i} \in S_{\boldsymbol{k}, g_{\boldsymbol{\nu}}}} \hat{f}_{\boldsymbol{i}} \tag{4.5}$$

for every index $\boldsymbol{k} \in \mathcal{J}_{\boldsymbol{\nu}}$.

4.1.2 Choice of index set

In the previous section, we defined the admissibility property of the index set $\mathcal{I}$ which is required for the interpolatory properties of the sparse grid operator. This definition is too broad to be able to derive any useful estimates for it. This is why we need to define a smaller family of index sets that will be used for Fourier sparse grids. The following parametrized index set, introduced in [GK00], can be used:

$$\mathcal{I}_L^T := \{\boldsymbol{l} : |\boldsymbol{l}|_1 - T|\boldsymbol{l}|_\infty \leq (1-T)L\}, \quad T < 1. \tag{4.6}$$

Note that $\mathcal{I}_L^{-\infty}$ and $\mathcal{I}_L^0$ correspond to full grids and conventional sparse grids, respectively.

With this choice of index set, one can prove the following bound for the number of points in the sparse grid.

Lemma 4.2. *The dimension of the sparse grid space* $V_{\mathcal{I}_L^T}$ *is limited by*

$$|V_{\mathcal{I}_L^T}| = \sum_{\boldsymbol{l} \in \mathcal{I}_L^T} 2^{|\boldsymbol{l}|_1} \lesssim \begin{cases} 2^L & 0 < T < 1, \\ 2^L L^{n-1} & T = 0, \\ 2^{L \frac{T-1}{T/n-1}} & T < 0, \\ 2^{Ln} & T = -\infty, \end{cases} \tag{4.7}$$

where n *is the number of dimensions,* $L \in \mathbb{N}_0$ *and* $T < 1$.

Proof. A proof of this lemma can be found in [GK00]. □

On several occasions in this and the previous sections, we discussed the complexity of the one-dimensional algorithms. We also discussed some presumably efficient algorithms for computations on sparse grids. However, up until now we have not shown any concrete results about the computational complexity of these algorithms. Lemma 4.3 fills this gap.

Lemma 4.3. *Let $\mathcal{T}[\mathcal{I}]$ denote the computational complexity of Algorithm 1 in which $TRANSFORM$ and $HIERARCHIZE$ are replaced with FFT and $FOURIER\_HIERARCHIZE$. Then the following upper bound holds:*

$$\mathcal{T}[\mathcal{I}_L^T] \lesssim nL \sum_{l \in \mathcal{I}_L^T} 2^{|l|_1} \lesssim \begin{cases} nL2^L & 0 < T < 1, \\ nL2^L L^{n-1} & T = 0, \\ nL2^{L\frac{T-1}{T/n-1}} & T < 0, \\ nL2^{Ln} & T = \infty. \end{cases} \tag{4.8}$$

Proof. A proof of a generalized version of this lemma can be seen in Lemma 6.7. □

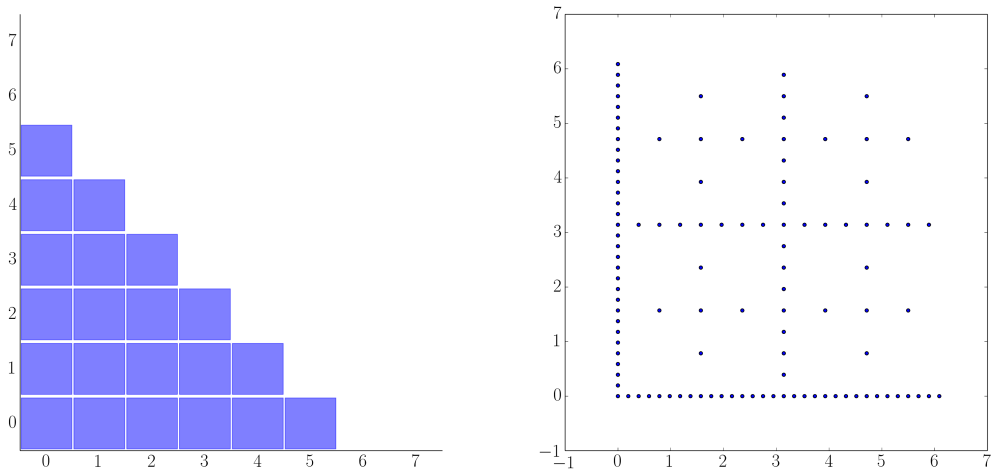


Figure 3: Index set $\mathcal{I}_5^0$(left) and the corresponding Fourier sparse grid(right).

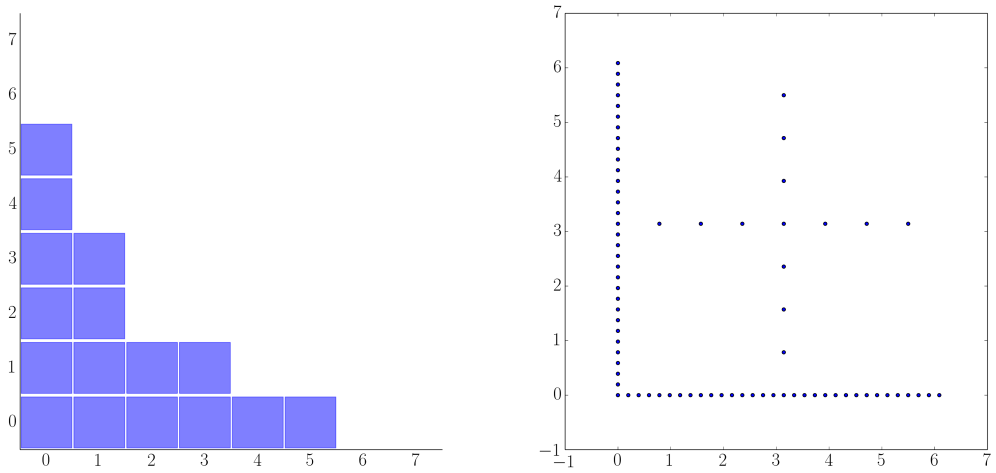


Figure 4: Index set $\mathcal{I}_5^{0.4}$(left) and the corresponding Fourier sparse grid(right).

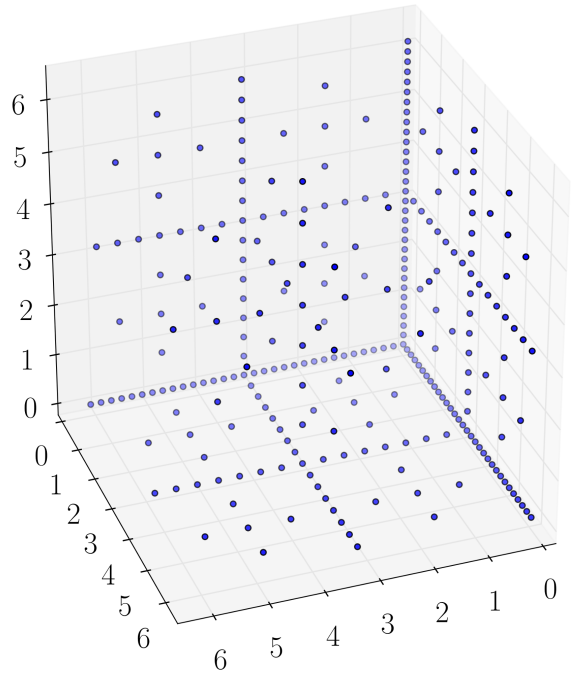


Figure 5: 3D Fourier sparse grid generated for the index set $\mathcal{I}_5^0$.

4.1.3 Approximation error of the interpolant

The next step in the analysis of Fourier grids is to find error bounds for the sparse grid interpolant. Since the space $L^2(\mathbb{T}^n)$ is too big for any useful result, we will focus on a subset of functions which are sufficiently smooth. More precisely, let $w : \mathbb{Z}^n \to \mathbb{R}_+$ be a continuous positive weight function. We can define the weighted Sobolev space $\mathcal{H}_w(\mathbb{T}^n)$ by measuring the decay of the weighted Fourier coefficients of a function:

$$\mathcal{H}_w(\mathbb{T}^n) := \left\{ f(\boldsymbol{x}) = \sum_{\boldsymbol{k}\in\mathbb{Z}^n} \hat{f}_{\boldsymbol{k}}\omega_{\boldsymbol{k}}(\boldsymbol{x}) : ||f||_w := \sqrt{\sum_{\boldsymbol{k}\in\mathbb{Z}^n} w(\boldsymbol{k})^2 |\hat{f}_{\boldsymbol{k}}|^2} < \infty \right\}. \tag{4.9}$$

Two specific choice of weight functions are commonly used. The first one is $w(\boldsymbol{k}) = \lambda_{iso}(\boldsymbol{k})^r = (1 + |\boldsymbol{k}|_\infty)^r$ and it corresponds to the isotropic Sobolev space $\mathcal{H}^r$, as defined in [Ada75]. The other choice is $w(\boldsymbol{k}) = \lambda_{mix}(\boldsymbol{k})^t = \prod_{d=1}^n (1 + |k_d|)^t$ and it corresponds to the Sobolev space with dominated mixed smoothness $\mathcal{H}^t_{mix}$, as defined in [ST87]. Error bounds for Fourier grids can be derived in a space that generalizes these two concepts, i.e. a generalized Sobolev space of dominating mixed smoothness [GK00]:

$$\mathcal{H}^{t,r}_{mix}(\mathbb{T}^n) := \left\{ f(\boldsymbol{x}) = \sum_{\boldsymbol{k}\in\mathbb{Z}^n} \hat{f}_{\boldsymbol{k}}\omega_{\boldsymbol{k}}(\boldsymbol{x}) : ||f||_{\mathcal{H}^{t,r}_{mix}} := \sqrt{\sum_{\boldsymbol{k}\in\mathbb{Z}^n} \lambda_{mix}(\boldsymbol{k})^{2t}\lambda_{iso}(\boldsymbol{k})^{2r} |\hat{f}_{\boldsymbol{k}}|^2} < \infty \right\}. \tag{4.10}$$

The following lemma can now be proven:

Lemma 4.4. *Let $f \in \mathcal{H}^t_{mix}$ have pointwise convergent Fourier series, $L \in \mathbb{N}_0$, $T < 1$, $r < t$ and $t > 1/2$. Then the following holds:*

$$||f - I_{\mathcal{I}_L^T} f||_{\mathcal{H}^r} \lesssim \begin{cases} 2^{-((t-r)+(Tt-r)\frac{n-1}{n-T})} L^{n-1} ||f||_{\mathcal{H}^t_{mix}} & T \geq r/t, \\ 2^{-(t-r)L} ||f||_{\mathcal{H}^t_{mix}} & T < r/t. \end{cases} \tag{4.11}$$

Proof. A proof of a generalized version of this lemma can be seen in Lemma 6.3. □

This error bound can be transformed to a bound with respect to the degrees of freedom of the grid, instead of the grid level L.

Lemma 4.5. *Let $f \in \mathcal{H}^t_{mix}$ have a pointwise convergent Fourier series, $L \in \mathbb{N}_0$, $0 < r < t$, $t > 1/2$ and $0 < T < r/t$. Then the following holds:*

$$||f - I_{\mathcal{I}_L^T} f||_{\mathcal{H}^r} \lesssim M^{-(t-r)} ||f||_{\mathcal{H}^t_{mix}}, \tag{4.12}$$

where M denotes the number of degrees of freedom in the grid, $M := |V_{\mathcal{I}_L^T}|$.

4.2 Dyadic Polynomial Sparse Grids

Using polynomial basis functions for sparse grids is a natural choice because of the vast amount of theory that exists for regular polynomial interpolation. However, the choice of a particular set of basis functions is not that obvious. While any set of linearly independent polynomials can be used, not all choices are good, just as in the one-dimensional case. For example, one can construct a sparse grid with monomial basis. However, this is in general a bad idea, because of the ill-conditioned Vandermonde matrix and the lack of an efficient hierarchical basis. The Lagrange polynomials solve the first problems as they eliminate the need for solving a system of interpolation conditions. However, the problem with the hierarchical basis still exists.

The polynomial basis of choice in this text is the set of Chebyshev polynomials. The Chebyshev transform, as introduced in subsection 2.2, does not suffer from problems with the condition number of the transformation matrix. A hierarchical basis exists which is as efficient as in the Fourier case. Moreover, because of the tight relation between the Chebyshev and the Fourier transforms, there exist efficient algorithms for the transform and its inverse, at least in the case of dyadic grids.

It is important to stress that the problems of the alternative polynomial basis sets are entirely of practical nature. The ill-conditioning of the transformation matrix is problematic because of the limitations of floating point arithmetic in computers. The lack of a good hierarchical basis is even less important because it only affects the complexity of the computation of the interpolation operator. The same is valid for the existence of the fast transforms and its inverse. From a point of view of the theoretical error bounds, these problems have no effect whatsoever. Once the interpolation interval I, the index set $\mathcal{I}$ and the interpolation points $\boldsymbol{x_\nu}$ are fixed, the interpolant $I_{\mathcal{I}} f$ is always the same, no matter what basis functions are used for its representation and no matter how efficient it is to compute. Therefore, one can think of polynomial sparse grids from two different points of view. A practical one, that is used to implement them numerically, and a theoretical one, that is used for their error analysis.

4.2.1 Definition

The most natural way to define Chebyshev sparse grids is to follow the same approach as in the Fourier case. This results in the so-called Clenshaw-Curtis grids which are examined in more detail in [BNR00]. The interpolation interval $\mathbb{I}$ is the closed interval $[-1, 1]$ and the function

space X is $L^2_\omega(\mathbb{I})$. The basis functions are defined as the Chebyshev polynomials,

$$\phi_n(x) := T_n(x) = \cos(n \cos^{-1} x). \tag{4.13}$$

The number of grid points per level g_n could be set to $2^n + 1$ and the interpolation points for each level could be $\{\cos(k\pi/2^n) : k = 0, \dots 2^n\}$. This way we could construct a valid nested sparse grid that fits in the general framework from section 3.

This definition, however, suffers from a serious problem and it is caused by the fact that $g_0 = 2$. From the admissibility property it follows that every index set $\mathcal{I}$ contains the index $\mathbf{0}$. This means that every sparse grid of dimension n contains the grid $S_{\mathbf{0}}$ and thus has at least g_0^n degrees of freedom. Hence, the number of degrees of freedom in the most simple sparse grid grows exponentially with the dimension. This limits the applicability of such grids to problems with at most $10 - 15$ dimensions. This is clearly undesirable. Therefore, although in theory g_0 can have any value, in practice it should always be set to 1.

This requirement leads to the following definitions for the number of grid points

$$g_n := \begin{cases} 1 & \text{for } n = 0, \\ 2^n + 1 & \text{for } n \geq 1, \end{cases} \tag{4.14}$$

and the interpolation points

$$S_n := \begin{cases} \{0\} & \text{for } n = 0, \\ \{\cos(k\pi/2^n) : k = 0, \dots 2^n\} & \text{for } n \geq 1. \end{cases} \tag{4.15}$$

The sets S_n are nested and thus suitable for the definition of sparse grids. Just as in the Fourier case, it is possible to use the fast Chebyshev transform and its inverse instead of the general functions *TRANSFORM* and *INVERSE_TRANSFORM*. The fast algorithms have complexity $O(g_n \log g_n)$.

The hierarchical basis functions can now be defined as

$$\psi_n := \begin{cases} \phi_n & \text{for } n = 0, 1, \\ \phi_2 + \phi_0 & \text{for } n = 2, \\ \phi_n - \phi_{2^l - n} & \text{for } 2^{l-1} + 1 \leq n \leq 2^l, l \geq 2. \end{cases} \tag{4.16}$$

Again, this choice leads to sparse change of basis matrices $C_{g_n - 1}$:

$$\begin{pmatrix} 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.17}$$

and their inverses:

$$\begin{pmatrix} 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.18}$$

Exactly as in the Fourier case, the specific structure of the matrices can be exploited to get hierarchization and dehierarchization algorithms with lower complexity than the general case. Note that a direct sparse matrix multiplication leads to a $O(g_n \log g_n)$ complexity. Algorithms 7 and 8, however, are linear. Note also that because of the special definition of ψ_2, the algorithms have to handle a_0 separately.

Algorithm 7 Hierarchization of a coefficient vector $\boldsymbol{a}$ of length g_n.

```
1: procedure CHEBYSHEV_HIERARCHIZE
2:     for l = n downto 2 do
3:         for k = 2^l downto 2^{l-1} + 1 do
4:             a_{2^l - k} += a_k
5:         end for
6:     end for
7:     a_0 -= a_2
8: end procedure
```

Algorithm 8 Dehierarchization of a coefficient vector $\boldsymbol{a}$ of length g_n.

```
1: procedure CHEBYSHEV_DEHIERARCHIZE
2:     a_0 += a_2
3:     for l = 2 to n do
4:         for k = 2^{l-1} + 1 to 2^l do
5:             a_{2^l - k} -= a_k
6:         end for
7:     end for
8: end procedure
```

This choice of hierarchical basis functions also satisfies the requirements of Lemma 3.1. Therefore, $\check{f}_{\boldsymbol{k}} = \hat{f}_{\boldsymbol{k},\boldsymbol{\nu}}$ for every $\boldsymbol{k} \in \mathcal{J}_{\boldsymbol{\nu}}$ and every $\boldsymbol{\nu} \in \mathbb{N}_0^n$. By using Lemma 2.3, this leads to the following aliasing formula for the hierarchical coefficients:

Lemma 4.6. *The hierarchical Chebyshev coefficients satisfy the identity*

$$\check{f}_{\boldsymbol{k}} = \sum_{\boldsymbol{i} \in S_{\boldsymbol{k},g_{\boldsymbol{\nu}}}} c_{\boldsymbol{i},\boldsymbol{k}} \hat{f}_{\boldsymbol{i}}, \tag{4.19}$$

where $c_{\boldsymbol{i},\boldsymbol{k}}$ and $S_{\boldsymbol{k},\boldsymbol{N}}$ are defined as in Lemma 2.3.

4.2.2 Choice of index set

Consider now dyadic Chebyshev sparse grids for index sets $\mathcal{I}_L^T$ in the form defined in (4.6). Unsurprisingly, the same lemmata as in the Fourier case hold for the degrees of freedom of the sparse grid and for the computational complexity of the interpolant. The reason is that the setting of both types of grids is almost the same. The number of grid points per level differ by at most 1. The computational complexity of the transform, inverse transform, hierarchization and dehierarchization is the same in both cases. Therefore, it is easy to see that the number of grid points and the computational effort have the same rate of growth with respect to the parameters T and L. The statements of Lemma 4.2 and Lemma 4.3 will not be repeated here because they are completely identical.

4.2.3 Approximation error of the interpolant

For the error analysis of the interpolant, we will limit the discussion to the index set $\mathcal{I}_L^0$. As in the Fourier case, the function space $L_w^2(\mathbb{I}^n)$ is too big for the derivation of error bounds. The results in this subsection are limited to functions from the function space

$$F_n^k := \{f : \mathbb{I}^n \to \mathbb{R} : D^{\boldsymbol{\alpha}} f \text{ exists and is continuous for every for } |\boldsymbol{\alpha}|_\infty \leq k\} \tag{4.20}$$

with the norm

$$||f||_{F_n^k} := \max\{||D^\alpha f||_\infty : |\boldsymbol{\alpha}|_\infty \leq k\}. \tag{4.21}$$

The following estimate is proven in [BNR00].

Lemma 4.7. *For every function $f \in F_n^k$ it holds*

$$||f - I_{\mathcal{I}_L^0} f||_\infty \lesssim M^{-k} (\log M)^{(k+2)(n-1)+1} ||f||_{F_n^k} \tag{4.22}$$

where M is the number of grid points.

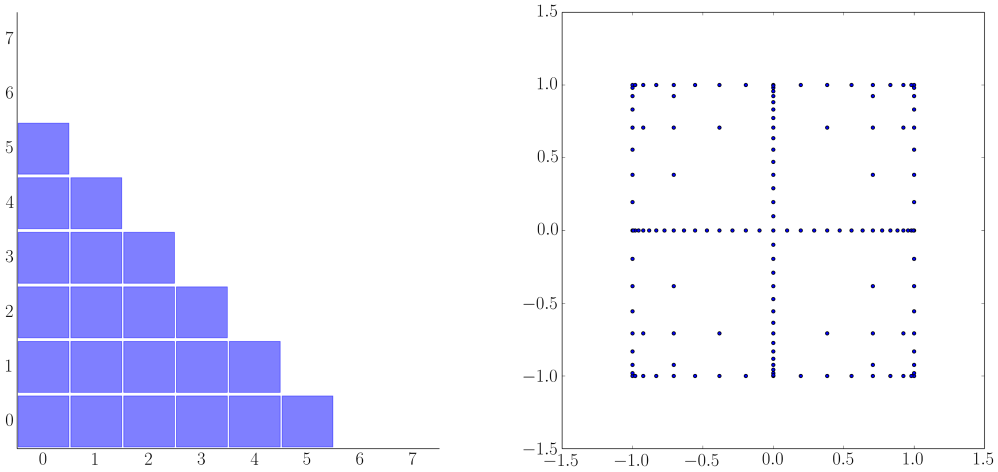


Figure 6: Index set $\mathcal{I}_5^0$(left) and the corresponding Chebyshev sparse grid(right).

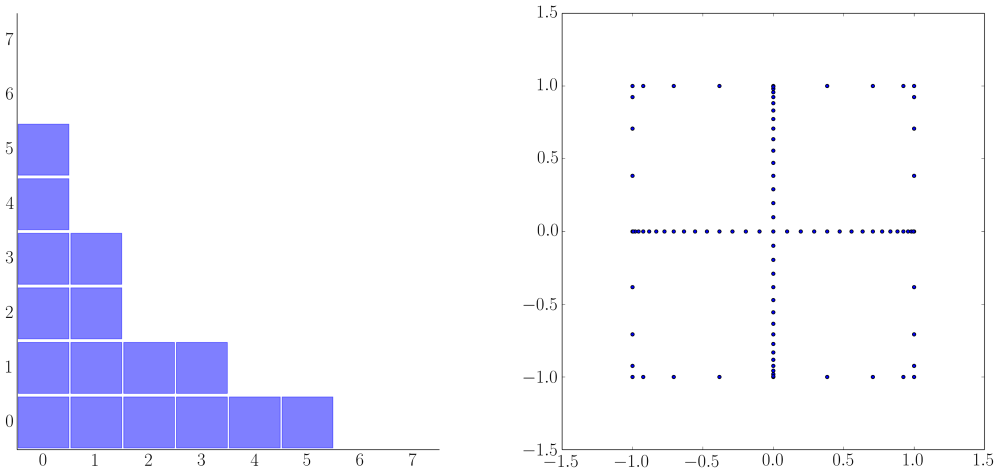


Figure 7: Index set $\mathcal{I}_5^{0.4}$(left) and the corresponding Chebyshev sparse grid(right).

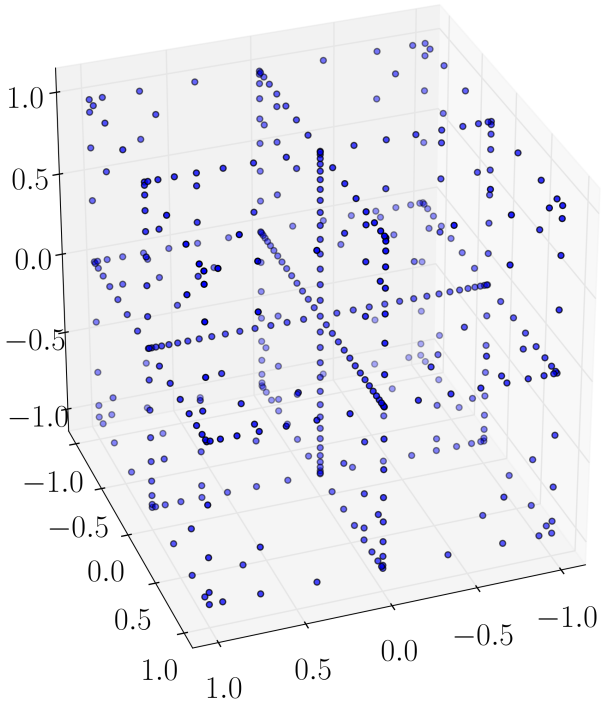


Figure 8: 3D Chebyshev sparse grid generated for the index set $\mathcal{I}_5^0$.

5 Leja Sequences

The sparse grids that were considered in section 4 have a very specific dyadic structure. This structure brings important advantages. First, the construction of these grids is very natural. The reason for this is the nestedness of the classic Fourier and Chebyshev interpolation nodes for the appropriate number of grid points per level. The second major advantage of these grids is the possibility to use fast algorithms for the one-dimensional transforms and for the hierarchization and dehierarchization procedures. These advantages, however, come at a cost which is expressed in the lack of flexibility in the sparse grid construction. This rigidity stems from the fact that the number of grid points per level must be doubled on every step in order to keep the nestedness of the grid points. A flexible sparse grids framework should possess the quality of *granularity*, i.e. the increase in the number of grid points after a single refinement step should be reasonable. Ideally, the control over the degrees of freedom in the grid should be as big as possible. Clearly, in the case of dyadic grids this control is very limited because each level has a fixed number of interpolation points. Although this is acceptable for small values of the dimension n, the issue worsens for high-dimensional problems (see Figure 9).

	$n=1$	$n=5$	$n=10$	$n=20$
$L=1$	2	6	11	21
$L=2$	4	26	76	251
$L=3$	8	96	416	2231
$L=4$	16	321	1966	16356
$L=5$	32	1002	8378	104380
$L=6$	64	2972	33028	599020
$L=7$	128	8472	122468	3158460

Figure 9: Number of grid points in a dyadic Fourier grid with dimension n and index set $\mathcal{I}_L^0$

To circumvent this problem, finer control over the number of grid points per level should somehow be allowed . In theory, it is possible to construct sparse grids with an arbitrary sequence g_n(although it is strongly advisable to keep $g_0 = 1$). In practice, an arbitrary sequence g_n requires a suitable sequence of sets of interpolation points S_n. While this choice is obvious for the dyadic grids from section 4, this is not at all the case for arbitrary g_n. The goal of this section is to provide means of finding *good* sets of interpolation points S_n for arbitrary sequence $\{g_n\}$. By *good* we mean that the corresponding one-dimensional interpolation operator on these points has a slowly increasing Lebesgue constant. Fourier sparse grids that utilize such a sequence have been discussed in length in [Mat14] and thus they will not be considered here. This section focuses on finding such sequences for Chebyshev and Hermite sparse grids.

Finally, it should be kept in mind that the flexibility of arbitrary grids also comes at a cost. The reason is that the specific advantages of dyadic sparse grids are lost. More precisely, the optimal properties of the classic interpolation nodes cannot be counted on because these nodes are not the ones used for the arbitrary grid. There are also no fast algorithms for arbitrary number of points for any type of transform. Thus the construction of an arbitrary grid takes more computational time compared to a dyadic one. As always, there is a trade-off between flexibility and optimality. The applicability of dyadic and arbitrary grids depends on the specific

problem. Both of them can be either suitable or not for the problem at hand.

5.1 Classic Leja sequences

As explained in section 4.2, the theoretical results obtained for one polynomial basis are often also valid for other sets of polynomial basis functions. Although we are interested in Chebyshev sparse grids, this subsection focuses on general polynomial interpolation. Finding a good sequence of nodes for polynomial interpolation is a challenging problem with a long history. One possible set of interpolation points that is guaranteed to be good is the sequence of Fekete points [Fek23]. They are defined as the points that maximize the corresponding Vandermonde determinant. The problem with them is that their computation is extremely costly because it requires the solving of a multivariate optimization problem. Analytical formulas are known in only a few types of domains.

An alternative set of points is the set of Leja points [Lej57]. They provide a good compromise between computational efficiency and approximation accuracy. Their computation also involves the solution of a sequence of optimization problems but in this case the problems have a single variable.

Definition 5.1. A Leja sequence on the set $[-1, 1]$ is a sequence of points $\{z_n\}_{n\in\mathbb{N}_0}$ defined by $z_0 = 0$ and

$$z_n \in \underset{z\in[-1,1]}{\arg\max} \left\{ \prod_{k=0}^{n-1} |z_k - z| \right\} \tag{5.1}$$

for $n \geq 1$.

A Leja sequence can be defined for a general compact subset $E \subset \mathbb{C}$ and an arbitrary initial point $z_0 \in E$. Applying the maximum principle on the maximization problem implies that all Leja points belong to the boundary of E, i.e. $z_n \in \partial E$ for $n \in \mathbb{N}_0$. Of course, in our limited case this does not mean anything because the set is the same as its boundary, $\partial[-1, 1] = [-1, 1]$. The second and more important observation is that the maximization problem often has several solutions. For this reason, it is not possible to define the Leja points with equality in Equation (5.1). The compactness of the set $[-1, 1]$(and E in the general case) guarantees that the maximization problem has at least one solution. Therefore, at least one Leja sequence exists but it might not be unique. In our case, all possible Leja sequences have the same properties regardless of which maximization point is chosen on every step.

The properties of classic Leja sequences are derived with the help of potential theory which is out of the scope of this work. A detailed introduction can be found in [DM04]. Here we only list the results that are relevant to the application of the sequence to sparse grids.

Lemma 5.1. *The Leja points are asymptotically distributed as the Chebyshev nodes. Assume that $N \in \mathbb{N}$, $\xi_{k,N}$ are the Chebyshev nodes, z_n are the Leja points and δ is the Dirac delta function. Then*

$$\lim_{n\to\infty} \nu_N^C = \lim_{n\to\infty} \nu_N^L \tag{5.2}$$

in a week sense, where

$$\nu_N^C = \frac{1}{N} \sum_{k=0}^{N} \delta(\xi_{k,N}) \tag{5.3}$$

and

$$\nu_N^L = \frac{1}{N}\sum_{k=0}^{N}\delta(z_n) \tag{5.4}$$

The lemma states that the Leja sequence distributes in the interval $[-1, 1]$ exactly as the Chebyshev nodes(see Figure 10). This hints that the sequence might have good approximation properties because the set of Chebyshev nodes does. The Lebesgue constant for interpolation on the Chebyshev nodes is of order $O(\log n)$. Unfortunately, such a strong result does not exist for the Leja sequences. In fact, it is only theoretically proven that the growth of the corresponding Lebesgue constant is subexponential.

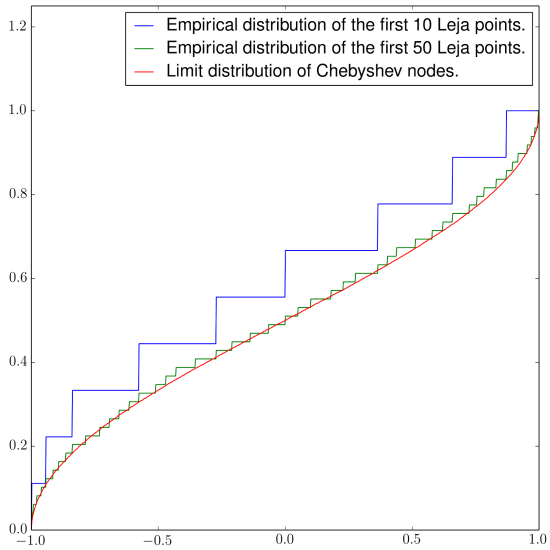


Figure 10: Empirical distribution of Leja points vs limit distribution of Chebyshev nodes.

Lemma 5.2. *Let Λ_n be the Lebesgue constant that corresponds to the interpolation operator at the Leja points $\{z_0, \dots z_n\}$. Then the following holds:*

$$\lim_{n\to\infty}(\Lambda_n)^{1/n} = 1. \tag{5.5}$$

Proof. A proof of this result can be found in [TT10]. □

This bound is too weak to justify the practical usage of Leja sequences. However, numerical evidence exists that the asymptotic behaviour of the Lebesgue constant Λ_n is actually linear instead of exponential ([Chk13]). This is the reason that in spite of the lack of theoretical support the Leja sequences are often used in practice.

5.2 $\mathcal{R}$-Leja sequences

The previous subsection discussed polynomial interpolation on Leja sequences. This approach has some important disadvantages. The Leja sequence still must be computed numerically. Even though the computation is relatively fast, it is easy to see how this can be a problem if we need to compute the first several thousand Leja points. The second disadvantage is the fact

that there is no proof that the Lebesgue constant grows linearly. This subsection considers a variation of the Leja sequence that solves both of these problems at the cost of a slight decrease in the approximation properties.

Definition 5.2. A Leja sequence on the unit disk $D := \{z \in \mathbb{C} : |z| \leq 1\}$ is a sequence of points $\{z_n\}_{n\in\mathbb{N}_0}$ defined by $z_0 := 1$ and

$$z_n \in \arg\max_{z\in D} \left\{ \prod_{k=0}^{n-1} |z_k - z| \right\} \tag{5.6}$$

for $n \geq 1$.

The same remarks as the ones after Definition 5.1 are valid for Leja sequences on the unit disk. All points belong to the boundary of the disk(i.e. $|z_n| = 1$), the sequence is guaranteed to exist and it is not uniquely defined. Using this definition, we can finally define the $\mathcal{R}$-Leja sequence.

Definition 5.3. An $\mathcal{R}$-Leja sequence is a sequence $\{\xi_n\}_{n\in\mathbb{N}_0}$, $\xi_n \in [-1, 1]$, defined by $\xi_0 := 1$ and

$$\xi_n := \text{Real}(z_m), \quad m := \min\left\{k \in \mathbb{N}_0 : \text{Real}(z_k) \notin \{\xi_0, \dots \xi_{n-1}\}\right\} \tag{5.7}$$

where $\{z_n\}_{n\in\mathbb{N}_0}$ is a Leja sequence on the unit disk.

This definition means that an $\mathcal{R}$-Leja sequence is generated by projecting the points from a Leja sequence on the unit disk to the $[-1, 1]$ interval and removing the duplicates. Since Leja sequences are not unique, it follows that the $\mathcal{R}$-Leja sequences are also not unique. There is, however, one instance of these sequences that is particularly important.

Lemma 5.3. *The sequence $\{\xi_n\}_{n\in\mathbb{N}_0}$ defined by*

$$\xi_n := \cos\left(\pi \sum_{l=0}^{s} k_l 2^{-l}\right) \tag{5.8}$$

where k_l are the digits in the binary representation of k, $k = \sum_{l=0}^{s} k_l 2^l$, is an $\mathcal{R}$-Leja sequence.

Proof. A proof of this result can be found in [CM11]. □

This lemma provides a closed formula for the points of the sequence. It is interesting to note that the first $2^n + 1$ nodes of this sequence coincide with the set of Chebyshev nodes for $N = 2^n$. This means that the $\mathcal{R}$-Leja sequence simply provides a specific ordering of the same nodes that are used for dyadic interpolation.

The closed formula solves one of the problems related to the classic Leja sequences. What is left is to prove a bound for the Lebesgue constant for interpolation on these points. The next lemma addresses this issue.

Lemma 5.4. *For each $n \in \mathbb{N}$ the Lebesgue constant Λ_n of the $\mathcal{R}$-Leja sequence from Lemma 5.3 satisfied*

$$\Lambda_n \lesssim 8\sqrt{2}(n+1)^2 \tag{5.9}$$

Proof. A proof of this lemma can be found in [CC15]. □

This means that Λ_n is of order $O(n^2)$. This bound is slightly worse than $O(n)$ but it has the benefit of being mathematically proven. In practice, however, classic Leja sequences usually outperform the $\mathcal{R}$-Leja sequence. Therefore, if one can neglect the lack of proven bound and the higher computational cost, then Leja sequences are usually the better choice.

5.3 Weighted Leja Sequences

In the previous subsections Leja sequences were considered as a method for overcoming the limitations inherent to dyadic Chebyshev grids. Nonetheless, Chebyshev grids still exist even without the Leja sequences. The case is different when it comes down to the Hermite transform. As mentioned in subsection 2.4, the classic nodes for interpolation with Hermite basis functions are the Gauss-Hermite nodes. The problem with them is that there is no nestedness relation between the interpolation points. Thus, constructing a sparse grid with Hermite basis functions is not as straightforward as in the dyadic Chebyshev and Fourier cases. One way to achieve this is to drop the nestedness condition on the sparse grid. The result is a non-nested Hermite sparse grid([LY13]). This type of grid has the limitation that the corresponding operator is no longer interpolatory. Another problem is that it contains a much higher number of points because the lack of nestedness prevents higher grid levels to reuse points from lower levels. Therefore this construction is not suitable for the purposes of this work.

The alternative way is to find a modified Leja sequence that can provide interpolation points for the Hermite sparse grid. The definition given here is based on the work in [NJ14].

Definition 5.4. Let $w(z) = e^{-z^2}$. A weighted Leja sequence on the set $\mathbb{R}$ is a sequence of points $\{z_n\}_{n\in\mathbb{N}_0}$ defined by $z_0 = 0$ and

$$z_n \in \arg\max_{z\in\mathbb{R}} \left\{ \sqrt{w(z)} \prod_{k=0}^{n-1} |z_k - z| \right\} \tag{5.10}$$

for $n \geq 1$.

This recursive definition is similar to the one for the classic Leja sequence. The main difference is that it incorporates the weight function in the optimization problem. The weighted Leja sequence is not unique because the optimization problems might have several maximizers. On the other hand, it is guaranteed to have at least one maximizer because of the exponential decay of the weight function and the polynomial growth of the product term:

$$\lim_{|z|\to\infty} \sqrt{w(z)} \prod_{k=0}^{n-1} |z_k - z| = 0. \tag{5.11}$$

The reason for the specific way in which the weight function is included in the optimization problem is the following result, proven in [NJ14].

Lemma 5.5. *The weighted Leja sequence is asymptotically distributed as the w-Gauss quadrature nodes. Assume that $N \in \mathbb{N}$, $\xi_{k,N}$ are the w-Gauss nodes, z_n are the weighted Leja points and δ is the Dirac delta function. Then*

$$\lim_{n\to\infty} \nu_N^G = \lim_{n\to\infty} \nu_N^L \tag{5.12}$$

in a week sense, where

$$\nu_N^G = \frac{1}{N}\sum_{k=0}^{N} \delta(N^{-1/2}\xi_{k,N}) \tag{5.13}$$

and

$$\nu_N^L = \frac{1}{N}\sum_{k=0}^{N} \delta(N^{-1/2} z_n) \tag{5.14}$$

Therefore, just as in the unweighted case, w-Leja sequences distribute in the same way as the classic nodes for interpolation with Hermite functions. The limit distribution of the w-Gauss nodes can be found in [Gaw87] . This does not guarantee that these points are good and there are currently no theoretical results about the growth rate of the resulting Lebesgue constant. However, numerical results suggest that the sequence is a good choice for the construction of sparse grids.

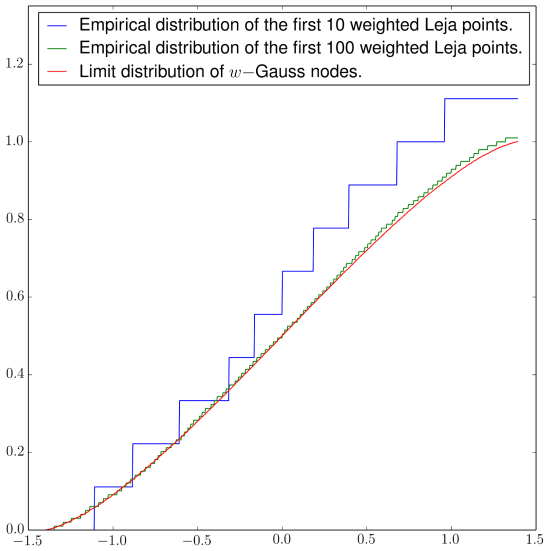


Figure 11: Empirical distribution of weighted Leja points vs limit distribution of w-Gauss nodes.

The described weighted Leja sequence is good for approximation with non-parametrized Hermite basis functions, i.e. $(\alpha, \beta) = (1, 0)$. For the general case of the Hermite transform, one way to find suitable interpolation points is to modify the weight in the definition of the sequence (5.4). This, however, is unnecessary because of the following simple observation. The parameters α and β simply scale and translate the Hermite basis functions. Therefore, it makes sense to define the interpolation points for the parametrized Hermite transform by simply scaling and translating the Leja points for the base case $(\alpha, \beta) = (1, 0)$. The resulting parametrized weighted Leja sequence is $\{z_n^{\alpha,\beta} := z_n/\alpha + \beta\}$. The direct computation of the sequence with a parametrized weight function yields the same points.

As a conclusion, it should be noted that definition 5.4 can also be extended for other weight functions. This is done in [NJ14] for the generalized Jacobi and generalized Laguerre weights. The resulting Leja sequences provide good nodes for Jacobi and Laguerre sparse grids. Both of these types of sparse grids are available in HCFFT.

5.4 Hermite Sparse Grids

Parametrized Hermite sparse grids are used for interpolation of functions from the space $X := L^2(\mathbb{R}^n)$. In practice, only function which have exponential decay are considered because otherwise the convergence is too slow. The interpolation interval is the entire real line, i.e. $I = \mathbb{R}$. The basis functions are

$$\phi_n(x) := \mathcal{H}_n^{\alpha,\beta}(x). \tag{5.15}$$

The weighted Leja sequence from definition 5.4 is used to provide interpolation points. This choice does not limit the number of grid points per level. Moreover, a fast Hermite transform does not exist. Thus, there are no limitations on the size of the grid levels because there is no way to benefit from such a limitation. Therefore, any increasing sequence g_n can be used. There is also no obvious way to define a hierarchical basis which produces a sparse change of basis matrix. The reason is that the interpolation points are computed numerically. Thus, the general definition of the hierarchical basis from subsection 3.4 is used. This means that the algorithms *TRANSFORM*, *INVERSE_TRANSFORM*, *HIERARCHIZE* and *DEHIERARCHIZE* have quadratic complexity. As a result, the computational complexity for the computation of the sparse grid operator is higher compared to the case of dyadic Fourier and Chebyshev grids. The following lemma measures exactly how much slower the algorithms is.

Lemma 5.6. *Let $\mathcal{T}[\mathcal{I}]$ denote the computational complexity of Algorithm 1 for a Hermite sparse grid with $g_n := 2^n$. Then the following upper bound holds:*

$$\mathcal{T}[\mathcal{I}_L^T] \lesssim n2^L \sum_{\boldsymbol{l}\in\mathcal{I}_L^T} 2^{|\boldsymbol{l}|_1} \lesssim \begin{cases} n2^{2L} & 0 < T < 1 \\ n2^{2L}L^{n-1} & T = 0 \\ n2^{L\left(\frac{T-1}{T/n-1}+1\right)} & T < 0 \\ n2^{L(n+1)} & T = \infty \end{cases} \tag{5.16}$$

Proof. For a general admissible set $\mathcal{I}$ we have

$$\begin{aligned} \mathcal{T}[\mathcal{I}_L^T] &\lesssim \sum_{d=1}^{n} \sum_{\boldsymbol{l}\in\mathcal{M}_d(\mathcal{I})} 2^{2l_d} 2^{|l|_1 - l_d} \\ &= \sum_{d=1}^{n} \sum_{\boldsymbol{l}\in\mathcal{M}_d(\mathcal{I})} 2^{l_d} 2^{|l|_1} \\ &= 2^{l_{max}} \sum_{d=1}^{n} \sum_{\boldsymbol{l}\in\mathcal{M}_d(\mathcal{I})} 2^{|l|_1} \\ &\leq n2^{l_{max}} \sum_{\boldsymbol{l}\in\mathcal{I}} 2^{|l|_1} \end{aligned} \tag{5.17}$$

where $l_{max} := \max_{\boldsymbol{l}\in\mathcal{I}} |\boldsymbol{l}|_\infty$. For the set $\mathcal{I}_L^T$, we can calculate $l_{max} = L$. Applying the inequality from Lemma 4.2 yields the desired result.

□

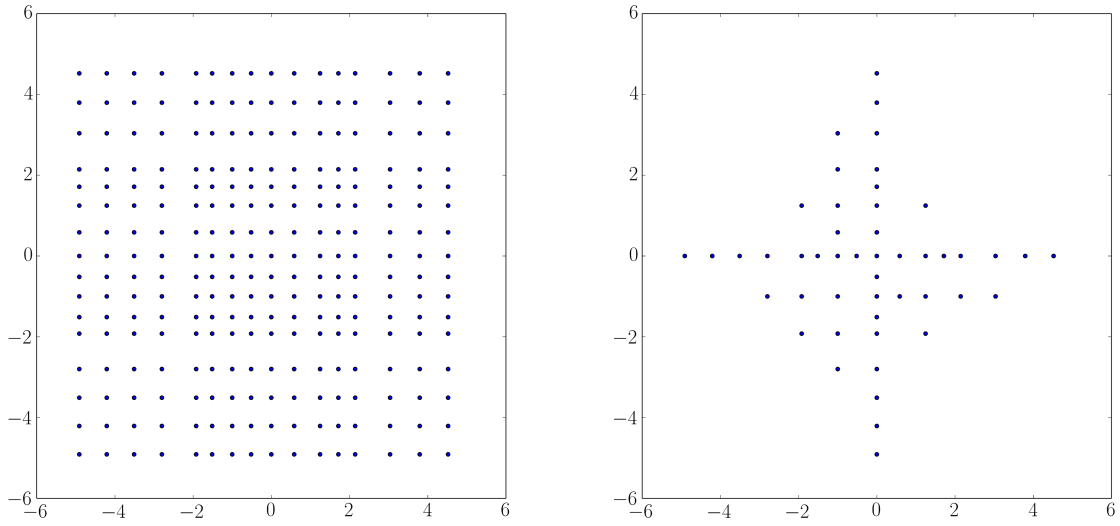


Figure 12: 2D Hermite grid with index set $\mathcal{I}_4^{-\infty}$(left) and with index set $\mathcal{I}_4^0$(right).

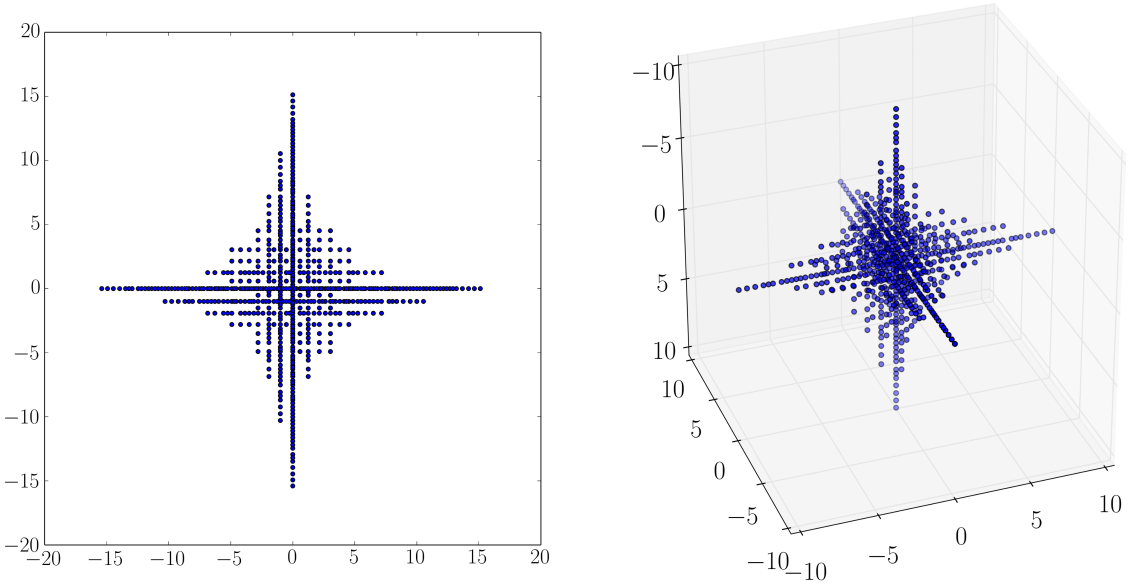


Figure 13: Hermite grid with index set $\mathcal{I}_7^0$ in two dimensions(left) and with index set $\mathcal{I}_6^0$ in three dimensions(right).

6 Mixed Sparse Grids

Section 3 introduced sparse grids in a very general setting. However, this definition is still not flexible enough for many practical problems. One of the main reasons for this lack of flexibility is the requirement to use the same type of transform in every dimension of the problem. Many functions arising from real-life problems are asymmetric and the different variables have very different nature and therefore different properties. It is often unrealistic to expect that a single type of transform would be able to represent accurately all features of such a function. Using the appropriate transform for each dimension leads to better convergence rates. In some cases, the improvement in convergence is so big that the problem is practically unsolvable without it.

The goal of this section is to lift the limitation imposed in section 3 by introducing and analysing mixed sparse grids, i.e. sparse grids that can utilize different types of transforms in the different dimensions. Both the interpolation and the approximation properties of the mixed sparse grid operator are discussed.

6.1 General definition

Thanks to the tensor-product structure of sparse grids, allowing the use of different transforms in different dimensions is fairly straightforward. In fact, it is mostly a matter of enriching the notation used for non-mixed grids. This subsection presents such a notation and briefly demonstrates how the results and properties of sparse grids are translated into the mixed case.

To define mixed sparse grids, we need to provide different one-dimensional parameters for each dimension. Each one-dimensional object is defined in n different versions by adding a superscript (d) to it, where d is the corresponding dimension index. Therefore, to define a general mixed grid the following information should be provided for $d = 1, \dots n$:

- Interpolation interval $I^{(d)}$ and function space $X^{(d)}$
- Number of grid points per level $g_k^{(d)}$
- Interpolation points $S_k^{(d)} := \{x_i^{(d)} : i \in \mathcal{G}_k^{(d)}\}$
- Regular basis functions $\phi_k^{(d)}$
- Hierarchical basis functions $\psi_k^{(d)}$

All other one-dimensional objects are defined in terms of the objects in this list and their definition is completely identical to the one given in section 3. This is why it will be omitted here.

The multidimensional objects are now defined exactly as in the non-mixed case with the addition of a superscript for the appropriate dimension to each one-dimensional object. For example, the mixed basis functions are defined as

$$\phi_{\boldsymbol{\nu}}(\boldsymbol{x}) := \Big(\bigotimes_{d=1}^{n} \phi_{\nu_d}^{(d)} \Big)(\boldsymbol{x}) = \prod_{d=1}^{n} \phi_{\nu_d}^{(d)}(x_{\nu_d}), \tag{6.1}$$

the hierarchical set of indices for level $\boldsymbol{\nu}$ as

$$\mathcal{J}_{\boldsymbol{\nu}} := \mathcal{J}_{\nu_1}^{(1)} \times \cdots \times \mathcal{J}_{\nu_n}^{(n)}, \tag{6.2}$$

and the difference operators as

$$\Delta_{\boldsymbol{\nu}} := \Delta_{\nu_1}^{(1)} \otimes \cdots \otimes \Delta_{\nu_n}^{(n)} : (X^{(1)} \times \cdots \times X^{(n)}) \mapsto V_{\boldsymbol{\nu}}. \tag{6.3}$$

The rest of the multidimensional objects ($\mathcal{B}_{\boldsymbol{\nu}}^h$, $\mathcal{G}_{\boldsymbol{\nu}}$, $\psi_{\boldsymbol{\nu}}$, etc) are defined analogously and thus their definitions are also omitted.

Finally, the mixed interpolation operator $I_{\mathcal{I}}$ is defined again as in the non-mixed case,

$$I_{\mathcal{I}} f := \sum_{\boldsymbol{\nu} \in \mathcal{I}} \Delta_{\boldsymbol{\nu}} f. \tag{6.4}$$

This operator has the same properties as the operator on non-mixed grids. The most important one of them, namely that the operator is indeed an interpolation operator, is proven in the next lemma.

Lemma 6.1. *For any admissible index set $\mathcal{I}$ the operator $I_{\mathcal{I}}$ interpolates every function $f \in X^{(1)} \times \cdots \times X^{(n)}$ on the set of points $S_{\mathcal{I}}$.*

Proof. Choose a point $\boldsymbol{x} \in S_{\mathcal{I}}$ and let $\boldsymbol{\nu} \in \mathcal{I}$ be the unique index such that $\boldsymbol{x} \in S_{\boldsymbol{\nu}}^h$. The interpolant's value at the point $\boldsymbol{x}$ is

$$I_{\mathcal{I}} f(\boldsymbol{x}) = \sum_{\boldsymbol{\mu} \in \mathcal{I}} \Delta_{\boldsymbol{\mu}} f(\boldsymbol{x}). \tag{6.5}$$

We can now use the fact that $\Delta_{\boldsymbol{\mu}} f(\boldsymbol{x}) = 0$ for all indices $\boldsymbol{\mu} \not\leq \boldsymbol{\nu}$. Removing these terms from the sum yields

$$\begin{aligned} I_{\mathcal{I}} f(\boldsymbol{x}) &= \sum_{\boldsymbol{\mu} \in \mathcal{I}} \Delta_{\boldsymbol{\mu}} f(\boldsymbol{x}) \\ &= \sum_{\mathbf{0} \leq \boldsymbol{\mu} \leq \boldsymbol{\nu}} \Delta_{\boldsymbol{\mu}} f(\boldsymbol{x}) \\ &= I_{\boldsymbol{\nu}} f(\boldsymbol{x}) \\ &= (I_{\nu_1}^{(1)} \otimes \cdots \otimes I_{\nu_n}^{(n)}) f(\boldsymbol{x}) \\ &= f(\boldsymbol{x}). \end{aligned} \tag{6.6}$$

The last equality follows from the properties of tensor product interpolation operators on full grids. □

We will also need the following generalization of Lemma 3.1.

Lemma 6.2. *Assume that the conditions in Lemma 3.1 are satisfied for the basis functions in every dimension of the mixed grid. Then for every $\boldsymbol{\nu} \in \mathbb{N}_0^n$ and every $\boldsymbol{k} \in \mathcal{J}_{\boldsymbol{\nu}}$ we have $\check{f}_{\boldsymbol{k}} = \hat{f}_{\boldsymbol{k},\boldsymbol{\nu}}$.*

The algorithms for the computation of the mixed interpolation operator and its inverse, and those for hierarchization and dehierarchization are straightforward generalizations of the non-mixed versions. For example, the interpolation operator can be computed with Algorithm 9.

Algorithm 9 Transforms the vector $\boldsymbol{u}$ of function values into a vector of hierarchical coefficients.

procedure *N_MIXED_TRANSFORM*
  for $d = 1, \dots n$ **do**
    for $\boldsymbol{\mu} \in \mathcal{M}_d(\mathcal{I})$ **do**
      for $\boldsymbol{i}' \in \mathcal{J}_{\mu_1}^{(1)} \times \dots \mathcal{J}_{\mu_{d-1}}^{(d-1)}, \boldsymbol{i}'' \in \mathcal{J}_{\mu_{d+1}}^{(d+1)} \times \dots \mathcal{J}_{\mu_n}^{(n)}$ **do**
        $TRANSFORM^{(d)}(u_{(\boldsymbol{i}',0,\boldsymbol{i}'')}, \dots, u_{(\boldsymbol{i}',g_{\mu_d}^{(d)}-1,\boldsymbol{i}'')})$
        $HIERARCHIZE^{(d)}(u_{(\boldsymbol{i}',0,\boldsymbol{i}'')}, \dots, u_{(\boldsymbol{i}',g_{\mu_d}^{(d)}-1,\boldsymbol{i}'')})$
      end for
    end for
  end for
end procedure

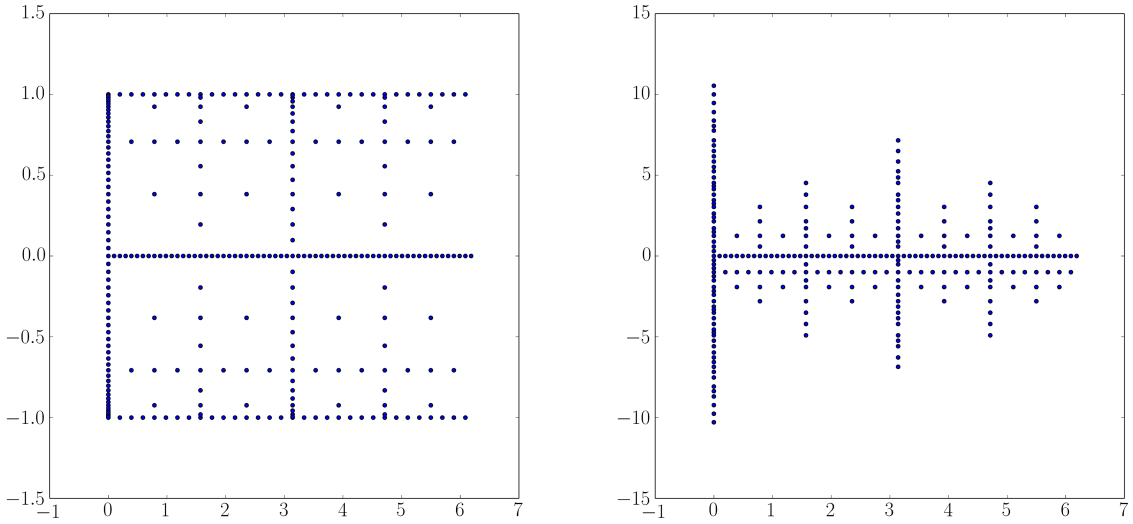


Figure 14: Mixed Fourier-Chebyshev grid with index set $\mathcal{I}_6^0$(left) and mixed Fourier-Hermite grid with index set $\mathcal{I}_6^0$(right).

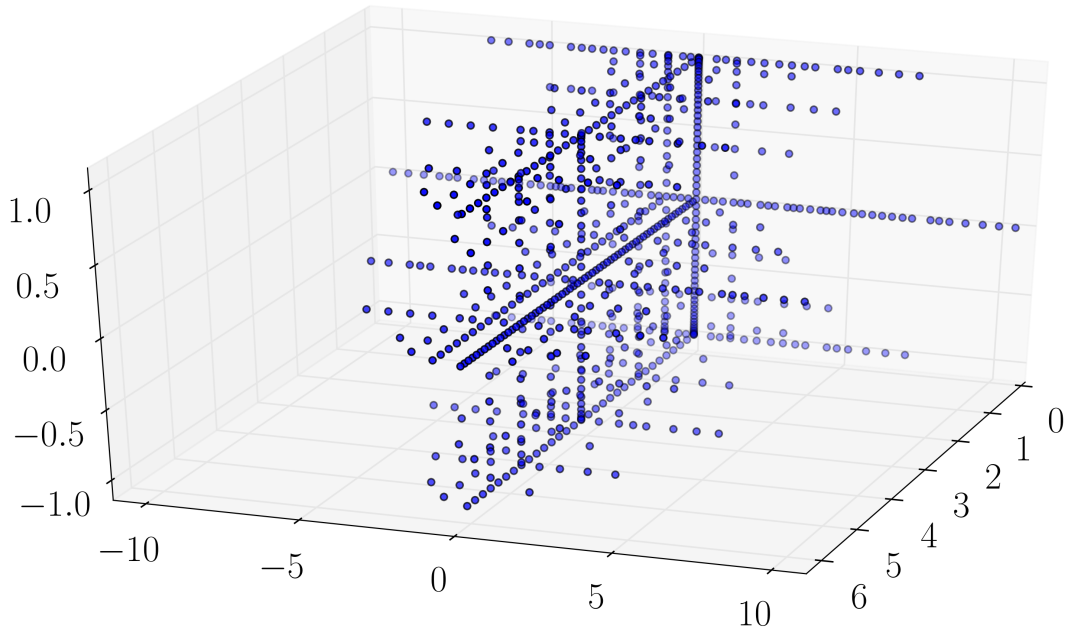


Figure 15: Mixed Fourier-Hermite-Chebyshev grid with index set $\mathcal{I}_6^0$.

6.2 Approximation error in Fourier-Chebyshev grids

The setting from the previous subsection guarantees that the sparse grid operator has the same interpolatory properties as the operator on non-mixed grids. However, it does not explore the approximation accuracy of the interpolant. Exactly as in the case of non-mixed sparse grids, the reason is that in order to derive error bounds for the interpolant we need to work with concrete grids. This is why the current subsection is devoted to defining and analyzing one particular type of mixed grids, namely the mixed Fourier-Chebyshev grid.

Let n be the dimension of the problem and $p \in \{0, 1, \dots n\}$. Without loss of generality, assume that the Fourier transform is used for the first p dimensions of the sparse grid and the Chebyhsev transform is used for the other $n - p$. To simplify the subsequent derivations, the following notation convention is established. Functions, sets, constants, indices, and coefficients that correspond to the Fourier transform(or equivalently, to the first p dimensions) are marked with a single prime, e.g. ϕ'_n, $\mathcal{B}'_n$ and $\mathcal{J}'_n$. Chebyshev related objects are marked with a double prime, e.g. ϕ''_n, $\mathcal{B}''_n$ and $\mathcal{J}''_n$. Objects that do not have any additional superscript denote mixed objects and can be split into Fourier and Chebyshev parts. The following examples illustrate the idea:

$$\boldsymbol{\nu} = (\boldsymbol{\nu}', \boldsymbol{\nu}''), \text{ where } \boldsymbol{\nu} \in \mathbb{Z}^p \times \mathbb{N}_0^{n-p}, \boldsymbol{\nu}' \in \mathbb{Z}^p \text{ and } \boldsymbol{\nu}'' \in \mathbb{N}_0^{n-p}, \tag{6.7}$$

$$\phi_{\boldsymbol{\nu}} = \phi'_{\boldsymbol{\nu}'} \otimes \phi''_{\boldsymbol{\nu}''}, \tag{6.8}$$

$$\mathcal{J}_{\boldsymbol{\nu}} = \mathcal{J}'_{\boldsymbol{\nu}'} \times \mathcal{J}''_{\boldsymbol{\nu}''}. \tag{6.9}$$

The Fourier-Chebyshev interpolant is defined for functions in the space $L^2(\mathbb{T}^p) \times L^2_w(\mathbb{I}^{n-p})$. This function space, however, is too big for the derivation of error bounds because it contains functions which are not sufficiently smooth. The error analysis is therefore limited to functions

from the weighted Sobolev space

$$\mathcal{H}_w(\mathbb{T}^p \times \mathbb{I}^{n-p}) := \left\{ f(\boldsymbol{x}) = \sum_{\boldsymbol{k} \in \mathbb{Z}^p \times \mathbb{N}_0^{n-p}} \hat{f}_{\boldsymbol{k}} \omega_{\boldsymbol{k}'}(\boldsymbol{x}') T_{\boldsymbol{k}''}(\boldsymbol{x}'') : ||f||_w < \infty \right\}, \tag{6.10}$$

where

$$||f||_w^2 := \sum_{\boldsymbol{k} \in \mathbb{Z}^p \times \mathbb{N}_0^{n-p}} w(\boldsymbol{k})^2 |\hat{f}_{\boldsymbol{k}}|^2 \tag{6.11}$$

and $w : \mathbb{Z}^p \times \mathbb{N}_0^{n-p} \to \mathbb{R}_+$ is a positive weight function. We will derive bounds for weight functions in the form $w = w^{t,r} := \lambda_{mix}^t \lambda_{iso}^r$. For such a weight function, the corresponding Sobolev space is denoted with $\mathcal{H}_{mix}^{t,r}(\mathbb{T}^p \times \mathbb{I}^{n-p})$. If the parameter r is equal to zero, then the space is denoted with $\mathcal{H}_{mix}^t(\mathbb{T}^p \times \mathbb{I}^{n-p})$. If r is equal to zero, the notation becomes $\mathcal{H}^r(\mathbb{T}^p \times \mathbb{I}^{n-p})$.

The goal of this subsection is to prove the following lemma:

Lemma 6.3. *Let $f \in \mathcal{H}_{mix}^t$ have pointwise convergent mixed series, $L \in \mathbb{N}_0$, $T < 1$, $r < t$ and $t > 1/2$. Then the following holds:*

$$||f - I_{\mathcal{I}_L^T} f||_{\mathcal{H}^r} \lesssim \begin{cases} 2^{-((t-r)+(Tt-r)\frac{n-1}{n-T})} L^{n-1} ||f||_{\mathcal{H}_{mix}^t} & T \geq r/t \\ 2^{-(t-r)L} ||f||_{\mathcal{H}_{mix}^t} & T < r/t \end{cases} \tag{6.12}$$

The proof of this lemma is rather lengthy and is partitioned into several other lemmata. First, we need the following general aliasing formula.

Lemma 6.4. *The hierarchical mixed coefficients satisfy the identity*

$$\check{f}_{\boldsymbol{k}} = \sum_{\boldsymbol{i} \in S_{\boldsymbol{k},\boldsymbol{g}_{\boldsymbol{l}}}} c_{\boldsymbol{i}'',\boldsymbol{k}''} \hat{f}_{\boldsymbol{i}} \tag{6.13}$$

for every $\boldsymbol{l} \in \mathbb{N}_0^n$ and every $\boldsymbol{k} \in \mathcal{J}_{\boldsymbol{l}}$, where $c_{\boldsymbol{i}'',\boldsymbol{k}''}$ and $S_{\boldsymbol{k},\boldsymbol{g}_{\boldsymbol{l}}}$ are defined as in Lemma 2.4.

Proof. The mixed sparse grid uses the Fourier and Chebyshev hierarchical basis functions as defined in section 4. This means that the conditions of Lemma 6.2 are satisfied. Hence for every $\boldsymbol{k} \in \mathcal{J}_{\boldsymbol{l}}$ we have $\check{f}_{\boldsymbol{k}} = \hat{f}_{\boldsymbol{k},\boldsymbol{l}}$. Combining this with the mixed aliasing formula from Lemma 2.4 yields the desired result. □

We also need the following basic inequality:

Lemma 6.5. *For $L \in \mathbb{N}_0$, $T < 1$ and $t \geq 0$ it holds*

$$\sum_{\boldsymbol{l} \in \mathbb{N}_0^n \setminus \mathcal{I}_L^T} 2^{-t|\boldsymbol{l}|_1 + r|\boldsymbol{l}|_\infty} \leq \begin{cases} 2^{-\left((t-r)+(Tt-r)\frac{n-1}{n-T}\right)L} L^{n-1} & \text{for } T \geq r/t \\ 2^{-(t-r)L} & \text{for } T < r/t \end{cases} \tag{6.14}$$

Proof. The proof of this lemma can be found in [Kna00]. □

Let us define the following indexing functions:

$$\mu'_{v,l}(j) := \begin{cases} j & \text{for } v = 0, \\ -1 & \text{for } v = 1 \text{ and } l = 0, \\ 2^l - 1 - j & \text{for } v = 1 \text{ and } l \geq 1, \end{cases} \tag{6.15}$$

$$\mu''_{v,l}(j) := \begin{cases} j & \text{for } v = 0, \\ -1 & \text{for } v = 1 \text{ and } l = 0, \\ 2^l - j & \text{for } v = 1 \text{ and } l \geq 1, \end{cases} \tag{6.16}$$

$$\boldsymbol{\mu}_{\boldsymbol{v},\boldsymbol{l}}(\boldsymbol{j}) := (\mu'_{v_1,l_1}(j_1), \dots \mu'_{v_p,l_p}(j_p), \mu''_{v_{p+1},l_{p+1}}(j_{p+1}), \dots \mu''_{v_n,l_n}(j_n)). \tag{6.17}$$

The purpose of these indexing functions is to simply allow us to write the hierarchical basis functions more conveniently. The following relations hold:

$$\psi'_j = \phi'_{\mu_{0,l}}(j) - \phi'_{\mu_{1,l}}(j), \tag{6.18}$$

$$\psi''_j = \phi''_{\mu_{0,l}}(j) - \phi''_{\mu_{1,l}}(j), \tag{6.19}$$

for every $l \in \mathbb{N}_0$ and $j \in \mathcal{J}'_l$ or $j \in \mathcal{J}''_l$, respectively (recall that both ϕ'_{-1} and ϕ''_{-1} were defined as equal to 0). By using these indexing functions the statement of the next lemma can be written more compactly.

Lemma 6.6. *Let $\tilde{w} = \lambda^r_{iso}$ and $w = \lambda^t_{mix}$. Then the inequality*

$$\tilde{w}(\boldsymbol{\sigma}(\boldsymbol{\mu}_{\boldsymbol{v},\boldsymbol{l}}(\boldsymbol{j})))^2 \sum_{\boldsymbol{m} \in S_{\boldsymbol{j},\boldsymbol{g}_{\boldsymbol{l}}}} |w(\boldsymbol{m})|^{-2} \leq C^2 2^{-2t|\boldsymbol{l}|_1 + 2r|\boldsymbol{l}|_\infty} \tag{6.20}$$

holds for every $\boldsymbol{l} \in \mathbb{N}_0^n$, $\boldsymbol{j} \in \mathcal{J}_{\boldsymbol{l}}$, $\boldsymbol{v} \in \{0,1\}^n$, $\boldsymbol{l} \geq \boldsymbol{v}$ and a constant C which does not depend on $\boldsymbol{j}$ and $\boldsymbol{v}$.

Proof. We will first find a bound for the sum

$$\sum_{\boldsymbol{m} \in S_{\boldsymbol{j},g_{\boldsymbol{l}}}} |w(\boldsymbol{m})|^{-2} = \sum_{\boldsymbol{m} \in S_{\boldsymbol{j},\boldsymbol{g}_{\boldsymbol{l}}}} \prod_{d=1}^{n} (1 + |m_d|)^{-2t}. \tag{6.21}$$

Consider an index $d \leq p$. In this case $m_d \in S'_{j_d,g'_{l_d}}$, hence $m_d = \sigma'(j_d) + k g'_{l_d}$ for some $k \in \mathbb{Z}$. Note that $\sigma'(j_d) \in \{-2^{l_d-1}, \cdots - 2^{l_d-2} - 1\} \cup \{2^{l_d-2} + 1, \dots 2^{l_d-1}\}$ because $j_d \in J'_{l_d}$. Three different cases for k exist:

- If $k = 0$ then $1 + |m_d| = 1 + |\sigma'(j_d)| \geq 4^{-1} 2^{l_d}$
- If $k \geq 1$ then $1 + |m_d| = 1 + |\sigma'(j_d) + k 2^{l_d}| = 1 + \sigma'(j_d) + k 2^{l_d} \geq 4^{-1} 2^{l_d}(1 + k)$
- If $k \leq -1$ then $1 + |\sigma'(j_d) + k 2^{l_d}| = 1 - \sigma'(j_d) - k 2^{l_d} \geq 4^{-1} 2^{l_d}(1 - k)$

Hence, in all cases we have $1 + |m_d| \geq 4^{-1} 2^{l_d}(1 + |k|)$.

Consider now an index $d > p$. In this case $m_d \in S''_{j_d,g''_{l_d}}$, hence $m_d = \varepsilon j_d + 2k g''_{l_d} \geq 0$ for some $k \in \mathbb{N}_0$ and some $\varepsilon \in \{-1, 1\}$. Two different cases exist:

- If $k = 0$ then ε must be equal to 1 because $m_d \geq 0$. Then $1 + |m_d| = 1 + j_d \geq 2^{-1} 2^{l_d}$
- If $k \geq 1$ then $1 + |m_d| = 1 + \varepsilon j_d + 2(2^{l_d} + 1)k \geq 2^{-1} 2^{l_d}(1 + k)$

Hence, we get $1 + |m_d| \geq 2^{-1} 2^{l_d}(1 + |k|)$.

Recall that the cardinality of each element in the set $S_{\boldsymbol{j},\boldsymbol{g}_{\boldsymbol{l}}}$ is at most 2^{n-p}. Hence, we get the inequality

$$\begin{aligned}
\sum_{\boldsymbol{m}\in S_{\boldsymbol{j},\boldsymbol{g}_{\boldsymbol{l}}}} |w(\boldsymbol{m})|^{-2} &= \sum_{\boldsymbol{m}\in S_{\boldsymbol{j},\boldsymbol{g}_{\boldsymbol{\nu}}}} \prod_{d=1}^{n} (1+|m_d|)^{-2t} \\
&\leq 2^{n-p} \sum_{\boldsymbol{k}\in\mathbb{Z}^p\times\mathbb{N}_0^{n-p}} \prod_{d=1}^{p} (4^{-1}2^{l_d}(1+|k_d|))^{-2t} \prod_{d=p+1}^{n} (2^{-1}2^{l_d}(1+|k_d|))^{-2t} \\
&\lesssim \sum_{\boldsymbol{k}\in\mathbb{Z}^p\times\mathbb{N}_0^{n-p}} \prod_{d=1}^{n} (2^{l_d}(1+|k_d|))^{-2t} \\
&\lesssim \sum_{\boldsymbol{k}\in\mathbb{Z}^n} \prod_{d=1}^{n} (2^{l_d}(1+|k_d|))^{-2t} \\
&\lesssim 2^{-2t|\boldsymbol{l}|_1} \sum_{\boldsymbol{k}\in\mathbb{Z}^n} \prod_{d=1}^{n} (1+|k_d|)^{-2t} \\
&\lesssim 2^{-2t|\boldsymbol{l}|_1}.
\end{aligned} \tag{6.22}$$

The other weight function can be bounded much more easily:

$$\tilde{w}(\boldsymbol{\sigma}(\boldsymbol{\mu}_{\boldsymbol{v},\boldsymbol{l}}(\boldsymbol{j})))^2 = (1+|\boldsymbol{\sigma}(\boldsymbol{\mu}_{\boldsymbol{v},\boldsymbol{l}}(\boldsymbol{j}))|_\infty)^{2r} \leq (1+|2^{\boldsymbol{l}}|_\infty)^{2r} \lesssim 2^{2r|\boldsymbol{l}|_\infty}. \tag{6.23}$$

□

Now we are finally ready to move on to the proof of the mixed error bound.

Proof of Lemma 6.3. The proof of this lemma extends the proof of the analogous bound for non-mixed Fourier grids given in [GH14]. The first part of the proof is valid for arbitrary weight functions w and $\tilde{w}$ and arbitrary index set $\mathcal{I}$. The interpolation error can be bounded by

$$\begin{aligned}
||f - I_{\mathcal{I}} f||_{\tilde{w}} &= || \sum_{\boldsymbol{l}\in\mathbb{N}_0^n} \sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \check{f}_{\boldsymbol{j}} \psi_{\boldsymbol{j}} - \sum_{\boldsymbol{l}\in\mathcal{I}} \sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \check{f}_{\boldsymbol{j}} \psi_{\boldsymbol{j}} ||_{\tilde{w}} \\
&= || \sum_{\boldsymbol{l}\in\mathbb{N}_0^n\setminus\mathcal{I}} \sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \check{f}_{\boldsymbol{j}} \psi_{\boldsymbol{j}} ||_{\tilde{w}} \\
&\leq \sum_{\boldsymbol{l}\in\mathbb{N}_0^n\setminus\mathcal{I}} || \sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \check{f}_{\boldsymbol{j}} \psi_{\boldsymbol{j}} ||_{\tilde{w}}.
\end{aligned} \tag{6.24}$$

The next step is to find an upper bound for the terms in the inner sum of the last inequality:

$$\begin{aligned}
|| \sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \check{f}_{\boldsymbol{j}} \psi_{\boldsymbol{j}} ||_{\tilde{w}}^2 &= || \sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \check{f}_{\boldsymbol{j}} \psi'_{j_1} \otimes \cdots \otimes \psi'_{j_p} \otimes \psi''_{j_{p+1}} \otimes \cdots \otimes \psi''_{j_n} ||_{\tilde{w}}^2 \\
&= || \sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \check{f}_{\boldsymbol{j}} \Big(\bigotimes_{d=1}^{p} (\phi'_{\mu_{0,l_d}(j_d)} - \phi'_{\mu_{1,l_d}(j_d)}) \Big) \otimes \Big(\bigotimes_{d=p+1}^{n} (\phi''_{\mu_{0,l_d}(j_d)} - \phi''_{\mu_{1,l_d}(j_d)}) \Big) ||_{\tilde{w}}^2 \\
&= || \sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \check{f}_{\boldsymbol{j}} \sum_{\boldsymbol{v}\in\{0,1\}^n} \varepsilon_{\boldsymbol{v}} \Big(\bigotimes_{d=1}^{p} \phi'_{\mu_{v_d,l_d}(j_d)} \Big) \otimes \Big(\bigotimes_{d=p+1}^{n} \phi''_{\mu_{v_d,l_d}(j_d)} \Big) ||_{\tilde{w}}^2 \\
&= || \sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \check{f}_{\boldsymbol{j}} \sum_{\boldsymbol{v}\in\{0,1\}^n} \varepsilon_{\boldsymbol{v}} \phi_{\boldsymbol{\mu}_{\boldsymbol{v},\boldsymbol{l}}(\boldsymbol{j})} ||_{\tilde{w}}^2
\end{aligned} \tag{6.25}$$

where $\varepsilon_{\boldsymbol{v}} \in \{-1,1\}$. Next, we will discard some of the terms in the inner sum. Assume that $l_d < v_d$ for some $d \leq p$. This is only possible if $l_d = 0$ and $v_d = 1$. However, $\mu'_{1,0}(j_d) = -1$ and

$\phi_{-1} = 0$, hence $\phi_{\boldsymbol{\mu}_{\boldsymbol{v},\boldsymbol{l}}(\boldsymbol{j})} = 0$. The same argument is valid if $l_d < v_d$ for some $d > p$. Hence, all terms in the inner sum which do not satisfy $\boldsymbol{l} \geq \boldsymbol{v}$ are equal to zero. After discarding them we get

$$\begin{aligned}
||\sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \check{f}_{\boldsymbol{j}}\psi_{\boldsymbol{j}}||_{\tilde{w}}^2 &= ||\sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \sum_{\boldsymbol{v}\in\{0,1\}^n, \boldsymbol{l}\geq\boldsymbol{v}} \check{f}_{\boldsymbol{j}}\varepsilon_{\boldsymbol{v}}\phi_{\boldsymbol{\mu}_{\boldsymbol{v},\boldsymbol{l}}(\boldsymbol{j})}||_{\tilde{w}}^2 \\
&\overset{(1)}{=} ||\sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \sum_{\boldsymbol{v}\in\{0,1\}^n, \boldsymbol{l}\geq\boldsymbol{v}} \check{f}_{\boldsymbol{j}}\varepsilon_{\boldsymbol{v}}\omega_{\boldsymbol{\sigma}'(\boldsymbol{\mu}'_{\boldsymbol{v}',\boldsymbol{l}'}(\boldsymbol{j}'))} \otimes T_{\boldsymbol{\mu}''_{\boldsymbol{v}'',\boldsymbol{l}''}(\boldsymbol{j}'')}||_{\tilde{w}}^2 \\
&\overset{(2)}{=} \sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \sum_{\boldsymbol{v}\in\{0,1\}^n, \boldsymbol{l}\geq\boldsymbol{v}} |\check{f}_{\boldsymbol{j}}|^2 \tilde{w}(\boldsymbol{\sigma}(\boldsymbol{\mu}_{\boldsymbol{v},\boldsymbol{l}}(\boldsymbol{j})))^2 \\
&\overset{(3)}{=} \sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \sum_{\boldsymbol{v}\in\{0,1\}^n, \boldsymbol{l}\geq\boldsymbol{v}} |\sum_{\boldsymbol{m}\in S_{\boldsymbol{j},\boldsymbol{g}_{\boldsymbol{l}}}} c_{\boldsymbol{m}'',\boldsymbol{j}''}\hat{f}_{\boldsymbol{m}}|^2 \tilde{w}(\boldsymbol{\sigma}(\boldsymbol{\mu}_{\boldsymbol{v},\boldsymbol{l}}(\boldsymbol{j})))^2 \\
&\overset{(4)}{=} \sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \sum_{\boldsymbol{v}\in\{0,1\}^n, \boldsymbol{l}\geq\boldsymbol{v}} \Big|\sum_{\boldsymbol{m}\in S_{\boldsymbol{j},\boldsymbol{g}_{\boldsymbol{l}}}} c_{\boldsymbol{m}'',\boldsymbol{j}''}\hat{f}_{\boldsymbol{m}}\frac{w(\boldsymbol{m})}{w(\boldsymbol{m})}\Big|^2 \tilde{w}(\boldsymbol{\sigma}(\boldsymbol{\mu}_{\boldsymbol{v},\boldsymbol{l}}(\boldsymbol{j})))^2 \\
&\overset{(5)}{\leq} \sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \sum_{\boldsymbol{v}\in\{0,1\}^n, \boldsymbol{l}\geq\boldsymbol{v}} \Big|\sum_{\boldsymbol{m}\in S_{\boldsymbol{j},\boldsymbol{g}_{\boldsymbol{l}}}} \hat{f}_{\boldsymbol{m}}\frac{w(\boldsymbol{m})}{w(\boldsymbol{m})}\Big|^2 \tilde{w}(\boldsymbol{\sigma}(\boldsymbol{\mu}_{\boldsymbol{v},\boldsymbol{l}}(\boldsymbol{j})))^2 \\
&\overset{(6)}{\leq} \sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \sum_{\boldsymbol{v}\in\{0,1\}^n, \boldsymbol{l}\geq\boldsymbol{v}} \Big(\sum_{\boldsymbol{m}\in S_{\boldsymbol{j},\boldsymbol{g}_{\boldsymbol{l}}}} |\hat{f}_{\boldsymbol{m}}w(\boldsymbol{m})|^2\Big)\Big(\sum_{\boldsymbol{m}\in S_{\boldsymbol{j},\boldsymbol{g}_{\boldsymbol{l}}}} |w(\boldsymbol{m})|^{-2}\Big)\tilde{w}(\sigma(\boldsymbol{\mu}_{\boldsymbol{v},\boldsymbol{l}}(\boldsymbol{j})))^2.
\end{aligned} \tag{6.26}$$

Step (1) in the derivation above is achieved by replacing the mixed basis function $\phi_{\boldsymbol{\mu}_{\boldsymbol{v},\boldsymbol{l}}(\boldsymbol{j})}$ with its actual representation. Step (2) is achieved by applying the definition of $||.||_{\tilde{w}}$. This is possible because each basis function appears in the sum at most once. Step (3) is obtained by applying the aliasing formula from Lemma 6.4. Step (4) is the result of simply multiplying and dividing by $w(\boldsymbol{m})$. Step (5) uses the fact that $c_{\boldsymbol{m}'',\boldsymbol{j}''} \leq 1$. Finally, step (6) applies the Cauchy-Schwarz inequality.

Define the function $g(\boldsymbol{l}) := 2^{-2t|\boldsymbol{l}|_1+2r|\boldsymbol{l}|_\infty}$. Lemma 6.6 can now be applied to the right-hand side of the inequality:

$$\begin{aligned}
||\sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \check{f}_{\boldsymbol{j}}\psi_{\boldsymbol{j}}||_{\tilde{w}}^2 &\leq \sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \sum_{\boldsymbol{v}\in\{0,1\}^n, \boldsymbol{l}\geq\boldsymbol{v}} \Big(\sum_{\boldsymbol{m}\in S_{\boldsymbol{j},\boldsymbol{g}_{\boldsymbol{l}}}} |\hat{f}_{\boldsymbol{m}}w(\boldsymbol{m})|^2\Big) C^2 g(\boldsymbol{l})^2 \\
&\leq 2^n C^2 g(\boldsymbol{l})^2 \sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \sum_{\boldsymbol{m}\in S_{\boldsymbol{j},\boldsymbol{g}_{\boldsymbol{l}}}} |\hat{f}_{\boldsymbol{m}}|^2 |w(\boldsymbol{m})|^2 \\
&\lesssim g(\boldsymbol{l})^2 \sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} \sum_{\boldsymbol{m}\in S_{\boldsymbol{j},\boldsymbol{g}_{\boldsymbol{l}}}} |\hat{f}_{\boldsymbol{m}}|^2 |w(\boldsymbol{m})|^2 \\
&\lesssim g(\boldsymbol{l})^2 ||f||_w^2
\end{aligned} \tag{6.27}$$

The last inequality holds because the multiset $\bigcup_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} S_{\boldsymbol{j},\boldsymbol{g}_{\boldsymbol{l}}}$ contains every index $\boldsymbol{k} \in \mathbb{Z}^p \times \mathbb{N}_0^{n-p}$ at most a constant number of times, and this constant depends only on n and p.

Returning to the inequality in the beginning of this proof, we get

$$||f - I_{\mathcal{I}}f||_{\tilde{w}} \lesssim \Big(\sum_{\boldsymbol{l}\in\mathbb{N}_0^n\setminus\mathcal{I}} g(\boldsymbol{l})\Big)||f||_w. \tag{6.28}$$

Applying Lemma 6.5 finishes the proof. □

The next important question is about the number of degrees of freedom in the mixed sparse grid. It is easy to see that for every index set $\mathcal{I}_L^T$ it holds $|V'_{\mathcal{I}_L^T}| \leq |V_{\mathcal{I}_L^T}| \leq |V''_{\mathcal{I}_L^T}|$, where $|V'_{\mathcal{I}_L^T}|$ and

$|V''_{\mathcal{I}_L^T}|$ are the degrees of freedom of a Fourier and a Chebyshev grid, respectively. These values have the same asymptotic behaviour. Hence, Lemma 4.2 is also valid for mixed grids.

The same conclusion is easily reached for the computational complexity $\mathcal{T}[\mathcal{I}]$. The transform and inverse transform algorithms have complexity $O(n \log n)$ for both the Fourier and the Chebyshev transforms. The hiearchization and dehierarchization algorithms are also linear in both cases. Since we already established that the number of degrees of freedom of the grid grows with the same rate, it follows easily that Lemma 4.3 is also valid in the mixed case.

6.3 Computational complexity of dyadic Fourier-Hermite grids

Little is known about the approximation error of the interpolant on Hermite sparse grids. [LY13] explores the approximation accuracy with Hermite functions on hyperbolic cross index sets. However, the approximation there is achieved with a spectral method instead of a pseudospectral one. Thus, the results are not directly applicable to sparse grids. Error bounds for mixed sparse grids that involve the Hermite transform as well as other transforms do not exist.

Hence, the only results that are discussed in this subsection are the estimation of the degrees of freedom and of the computational complexity of dyadic Fourier-Hermite grids. The results for Chebyshev-Hermite grids are the same and therefore they are not considered separately. A dyadic Fourier-Hermite grid has the same number of grid points as the corresponding dyadic Fourier grid. Therefore, Lemma 4.2 also holds in this case. The question about the computational complexity is a bit more difficult. As already explained, the one-dimensional algorithms for the Hermite transform have quadratic complexity. The overall computational complexity is bounded in the following lemma.

Lemma 6.7. *Let $\mathcal{T}[\mathcal{I}]$ denote the computational complexity of Algorithm 9 for a dyadic Fourier-Hermite grid which uses the Fourier transform in the first p dimensions and the Hermite transform in the other $n-p$. Then the following upper bound holds:*

$$\mathcal{T}[\mathcal{I}_L^T] \lesssim (pL + (n-p)2^L) \sum_{\boldsymbol{l} \in \mathcal{I}_L^T} 2^{|l|_1} \lesssim \begin{cases} (pL + (n-p)2^L)2^L & 0 < T < 1 \\ (pL + (n-p)2^L)2^L L^{n-1} & T = 0 \\ (pL + (n-p)2^L)2^{L\frac{T-1}{T/n-1}} & T < 0 \\ (pL + (n-p)2^L)2^{Ln} & T = \infty \end{cases} \tag{6.29}$$

Proof. For a general admissible set $\mathcal{I}$ we have

$$\begin{aligned} \mathcal{T}[\mathcal{I}_L^T] &\lesssim \sum_{d=1}^{p} \sum_{\boldsymbol{l} \in \mathcal{M}_d(\mathcal{I})} 2^{l_d} l_d 2^{|l|_1 - l_d} + \sum_{d=p+1}^{n} \sum_{\boldsymbol{l} \in \mathcal{M}_d(\mathcal{I})} 2^{2l_d} 2^{|l|_1 - l_d} \\ &= \sum_{d=1}^{p} \sum_{\boldsymbol{l} \in \mathcal{M}_d(\mathcal{I})} l_d 2^{|l|_1} + \sum_{d=p+1}^{n} \sum_{\boldsymbol{l} \in \mathcal{M}_d(\mathcal{I})} 2^{l_d} 2^{|l|_1} \\ &\leq l_{max} \sum_{d=1}^{p} \sum_{\boldsymbol{l} \in \mathcal{M}_d(\mathcal{I})} 2^{|l|_1} + 2^{l_{max}} \sum_{d=p+1}^{n} \sum_{\boldsymbol{l} \in \mathcal{M}_d(\mathcal{I})} 2^{|l|_1} \\ &\leq p l_{max} \sum_{\boldsymbol{l} \in \mathcal{I}} 2^{|l|_1} + (n-p) 2^{l_{max}} \sum_{\boldsymbol{l} \in \mathcal{I}} 2^{|l|_1} \\ &= \left(p l_{max} + (n-p) 2^{l_{max}} \right) \sum_{\boldsymbol{l} \in \mathcal{I}} 2^{|l|_1} \end{aligned} \tag{6.30}$$

where $l_{max} := \max_{\boldsymbol{l} \in \mathcal{I}} |\boldsymbol{l}|_\infty$. For the set $\mathcal{I}_L^T$, we have $l_{max} = L$. Applying the inequality from Lemma 4.2 finishes the proof.

□

7 Adaptive Sparse Grids

Sections 4 and 6 contained error estimates for sparse grids with an index set in the form $\mathcal{I}_L^T$. The reason for this choice is that if the interpolated function belongs to a certain smoothness class then the constructed sparse grid is optimal([BG04; GK09]). Therefore, the decision for using such a sparse grid for the problem at hand is based on the a priori knowledge of the function's properties. Often, however, we need to work with functions which do not possess the required smoothness or whose properties are not known. In high-dimensional problems it is common to have the different dimensions contributing to the function in a different degree. The interactions between the dimensions can also vary strongly. The importance of these interactions should be captured by the sparse grid by adding more refinement levels in dimensions with higher contribution. In the general case, index sets in the form $\mathcal{I}_T^L$ do not suffice. One straightforward generalization is to consider level indices which are weighted with a weight vector $\boldsymbol{a} \in \mathbb{R}_+^n$([GG02]). This results in anisotropic grids and allows the grid to focus in dimensions which are more important. However, the a priori choice of the weight vector $\boldsymbol{a}$ is still a difficult task.

A common method to handle such problems is to execute a dimension-adaptive algorithm which constructs a sparse grid depending on the approximated function([Heg03]). The goal of the adaptive algorithm is to construct the grid levels by evaluating their contribution to the approximation properties of the interpolant. The resulting sparse grid is tailored to the specific function instead of to an entire function class. Moreover, the difficult task of choosing an appropriate index set is delegated to the adaptive algorithm.

7.1 Basic adaptive algorithm

There are many variations of the adaptive algorithm which arise from the various applications of sparse grids([GG03; Gar07; Mat14]). This subsection introduces the most simple one of them. There are two main reasons for this choice. First, working with adaptive sparse grids can be difficult and can produce unexpected results. There are certain pitfalls that should be avoided. With the simple algorithm it is easy to demonstrate them. The second reason is that this algorithm was used by default in the previous version of HCFFT.

The algorithm is provided in pseudocode in Algorithm 10. During the adaptive refinement the set of grid indices is split into two disjoint subsets – the set of *old* indices, $\mathcal{O}$, and the set of *active* indices, $\mathcal{A}$. At any given point, the grid index set $\mathcal{I}$ is equal to the union of these two sets. The algorithm starts by setting $\mathcal{O}$ to the empty set and inserting the index $\mathbf{0}$ into $\mathcal{A}$. Each index in $\mathcal{A}$ has an associated weight which denotes its importance, i.e. its contribution to the final approximation. Each step starts by pulling the index with the highest weight from $\mathcal{A}$ and inserting it into $\mathcal{O}$. The algorithm then inspects all forward neighbours of the newly inserted index. All of the admissible ones are inserted in the active set. An index is considered to be admissible if all of its backward neighbours are in $\mathcal{O}$. The algorithm continues as long as there are active indices whose weight is larger than some ε. The rest of the active indices are simply popped from the active set and inserted in the old set. The final result is the old set since at the end the active set is empty.

Algorithm 10 Basic adaptive algorithm

$\mathcal{O} = \{\}$
$\mathcal{A} = \{\mathbf{0}\}$
while $\mathcal{A}$ is not empty **do**
  $\boldsymbol{i} = \arg\max_{\boldsymbol{k}\in\mathcal{A}}\{weight(\boldsymbol{k})\}$
  $\mathcal{O} = \mathcal{O} \cup \{\boldsymbol{i}\}$
  $\mathcal{A} = \mathcal{A} \setminus \{\boldsymbol{i}\}$
  if $weight(\boldsymbol{i}) > \varepsilon$ **then**
    for $d = 1, \dots n$ **do**
      $\boldsymbol{j} = \boldsymbol{i} + \boldsymbol{e}_d$
      if $\boldsymbol{j} \notin \mathcal{O}$ **and** $admissible(\mathcal{O} \cup \{\boldsymbol{j}\})$ **then**
        $\mathcal{A} = \mathcal{A} \cup \{\boldsymbol{j}\}$
      end if
    end for
  end if
end while
return $\mathcal{O}$

The described algorithm is a greedy one. On every step it chooses the best index according to some heuristically computed importance indicator. Neighbours of important indices are also inspected, hoping that importance is propagated locally. As soon as an unimportant index is encountered, the algorithm is no longer interested in continuing the expansion in this direction. An example application of this algorithm can be seen in Figure 16 and Figure 17.

It is important to note that at every step of the algorithm the index set $\mathcal{I}$ is equal to $\mathcal{O} \cup \mathcal{A}$ instead of just $\mathcal{O}$. The reason is that the weight function $weight(\boldsymbol{l})$ typically uses the coefficients at the grid points $S_{\boldsymbol{l}}^h$. Since this weight function is computed for all indices in the active set, it follows that the active set should be contained in the grid index set. Of course, it is possible to terminate the algorithm once the condition $weight(\boldsymbol{i}) > \varepsilon$ is violated without adding the rest of the active indices to the final index set. However, this would not make much sense because we have already computed the target function at all grid points $\mathcal{O} \cup \mathcal{A}$. Terminating the algorithm prematurely would artificially shrink the index set. Any subsequent numerical $benefit/cost$ estimates would be inaccurate because they would not account for the function computations in the rest of the indices in $\mathcal{A}$. Typically, the most time is spent on function computations because the target function is very complicated. Therefore, this inaccuracy would be significant. Moreover, adding $\mathcal{A}$ to the final index set should not increase the computational time of the interpolant significantly. As a consequence, the final grid contains some indices on the boundary with a weight less than the given threshold ε.

Some typical choices of weighting function are:

$$weight_1(\boldsymbol{l}) := \sum_{\boldsymbol{i}\in\mathcal{J}_{\boldsymbol{l}}} |\hat{f}_{\boldsymbol{i}}^{\boldsymbol{l}}|^2, \quad weight_2(\boldsymbol{l}) := \sum_{\boldsymbol{i}\in\mathcal{J}_{\boldsymbol{l}}} |\hat{f}_{\boldsymbol{i}}^{\boldsymbol{l}}|^2/|\mathcal{J}_{\boldsymbol{l}}|^2, \tag{7.1}$$

$$weight_3(\boldsymbol{l}) := \sum_{\boldsymbol{i}\in\mathcal{J}_{\boldsymbol{l}}} |\check{f}_{\boldsymbol{i}}|^2, \quad weight_4(\boldsymbol{l}) := \sum_{\boldsymbol{i}\in\mathcal{J}_{\boldsymbol{l}}} |\check{f}_{\boldsymbol{i}}|^2/|\mathcal{J}_{\boldsymbol{l}}|^2. \tag{7.2}$$

The functions $weight_3$ and $weight_4$ measure the importance of an index by summing its corresponding hierarchical coefficients. An important property of the hierarchical coefficients is that

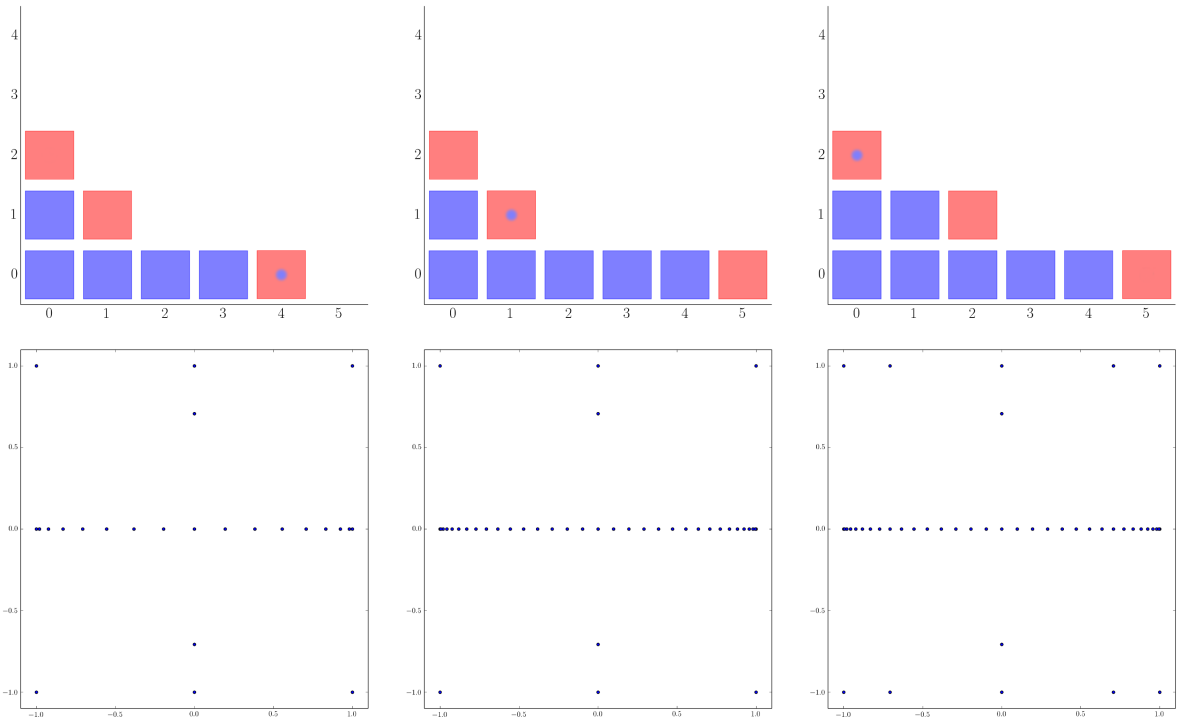


Figure 16: Several steps of the algorithm executed on the function $f(x_1, x_2) = e^{-x_1^2} + e^{-x_2^2/10} + e^{-x_1^2/10 - x_2^2/10}$. The grid uses Chebyshev basis functions in both directions. The weight function is $weight_2()$. The top row depicts the index set $\mathcal{I}$. Blue squares indicate indices in the old set, while red squares are used for the active set. The selected active index for expansion is marked with a blue dot. The bottom row shows the corresponding grid.

they do not change when the grid evolves. This means that the coefficients corresponding to index $\boldsymbol{l}$ stay the same through the entire algorithm. The case is different with the regular coefficients $\hat{f}_{\boldsymbol{i}}^{\boldsymbol{l}}$ and the weight functions $weight_1$ and $weight_2$. The regular coefficients that correspond to index $\boldsymbol{l}$ can change when the grid index set $\mathcal{I}$ grows. This means that if an index $\boldsymbol{l}$ had a weight greater than ε at the time of its insertion, this weight might actually become less than ε at the end of the algorithm. This behaviour is not particularly important for the efficiency of the method. However, one should keep that in mind when inspecting the final generated index set since the results might seem confusing.

An important feature of this algorithm is that its termination criteria is local. This means that the termination decision is taken only based on the local properties of the indices. There is no way to stop the index set generation by inspecting the entire grid. A partial solution to this problem is to execute the algorithm in a loop with a decreasing threshold ε. The resulting grid is inspected after every iteration. If the grid has the desired properties the loop is terminated. If it does not, then ε is decreased and the grid is refined further. When ε reaches some minimal value ε_{min} the loop is terminated, even if the grid still does not have the desired properties. The algorithm is written in pseudocode in Algorithm 11. The function $testGrid()$ contains the global termination criteria.

Another consequence of the local termination criteria is the fact that the algorithm can be forced to terminate early. This may happen when the weight function does not decrease when the index increases. If an index has a backward neighbour whose weight is less than ε, then

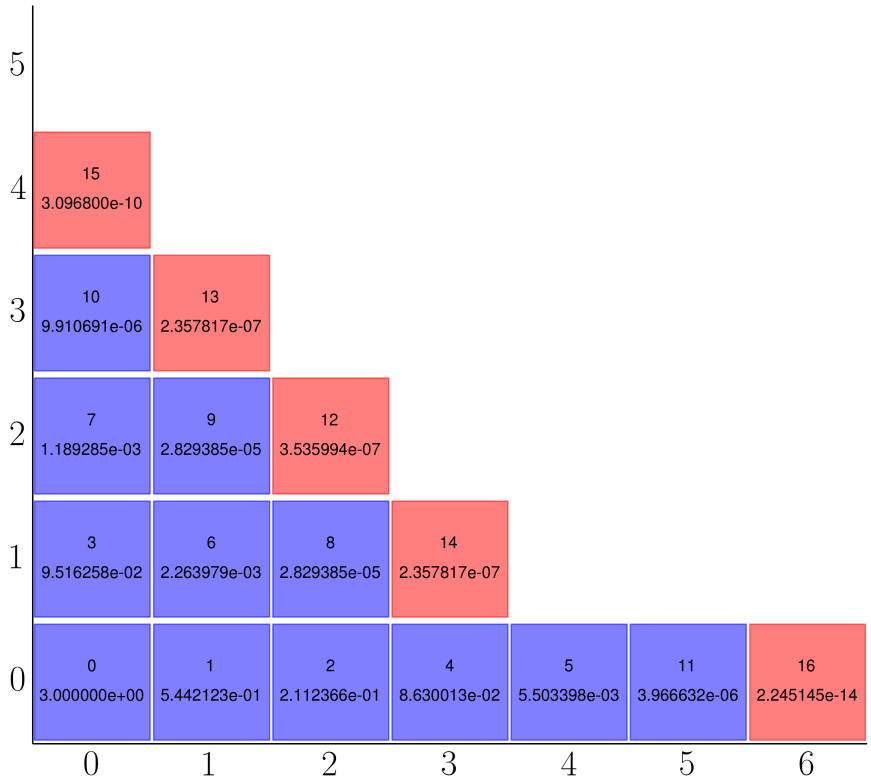


Figure 17: The final result of the adaptive algorithm executed on the function $f(x_1, x_2) = e^{-x_1^2} + e^{-x_2^2/10} + e^{-x_1^2/10 - x_2^2/10}$. The chosen ε is 10^{-6}. Red indices indicate the state of the active set after the last index was expanded. Each square has two numbers in it. The first line shows the moment at which the index was added to the old set. Thus, the indices were added in the order $(0,0), (1,0), (2,0), (0,1), (3,0) \ldots$ The second line contains the weight of the index, calculated with the $weight_2()$ function.

there is no way to add this index to the grid, even if its weight is greater than ε. This situation is particularly easy to happen with the so-called PLUS1 grids([Mat14]). The reason is that every index $\boldsymbol{l}$ corresponds to a single grid point. Thus, the algorithm is extremely sensitive to fluctuations in the coefficients. Unfortunately, there is no easy way to solve this problem. If the weight function does not decrease when $|\boldsymbol{l}|_1$ increases, then the application of the algorithm might not be possible.

7.2 Generalized adaptive algorithm

This subsection aims to transform the basic adaptive algorithm into an extensible and customizable procedure. The goal is to define a general algorithm that can be applied to a large set of problems by simply adjusting its parameters. The algorithm that is currently implemented in HCFFT is presented in pseudocode in Algorithm 12.

Algorithm 11 Iterative adaptive algorithm

$\varepsilon = \varepsilon_{init}$
while $\varepsilon \geq \varepsilon_{min}$ **do**
    $\mathcal{I} :=$ { result of Algorithm 10 with the current ε }
    if $testGrid(\mathcal{I})$ **then**
        break
    end if
    $\varepsilon = \varepsilon/2$
end while
return $\mathcal{I}$

Algorithm 12 Generalized adaptive algorithm

$\mathcal{O} = \{\}$
$\mathcal{A} = \mathcal{I}_0$
while $\mathcal{A}$ is not empty **do**
    $\boldsymbol{i} = \arg\max_{\boldsymbol{k} \in \mathcal{A}} \{weight(\boldsymbol{k})\}$
    $\mathcal{O} = \mathcal{O} \cup \{\boldsymbol{i}\}$
    $\mathcal{A} = \mathcal{A} \setminus \{\boldsymbol{i}\}$
    if $expand(\boldsymbol{i}, \mathcal{O} \cup \mathcal{A})$ **then**
        for $d = 1, \ldots n$ **do**
            $\boldsymbol{j} = \boldsymbol{i} + \boldsymbol{e}_d$
            if $\boldsymbol{j} \notin \mathcal{O}$ **and** $admissible(\mathcal{O} \cup \{\boldsymbol{j}\})$ **and** $insert(\boldsymbol{j}, \mathcal{O} \cup \mathcal{A})$ **then**
                $\mathcal{A} = \mathcal{A} \cup \{\boldsymbol{j}\}$
            end if
        end for
    end if
end while
return $\mathcal{O}$

The goal of Algorithm 12 is to allow more control over the grid indices that go into the index set. There are two user-defined functions that are used for this purpose – *expand* and *insert*. The former decides when the neighbours of an index should be inspected and eventually inserted into the active set. The latter performs a finer selection by allowing to exclude specific neighbours. Note that both functions have two arguments – the local index $\boldsymbol{i}$ and the current index set $\mathcal{I} = \mathcal{O} \cup \mathcal{A}$. Therefore, they can be used to make a non-local decision. Note also that this algorithm allows the active index set to acquire some initial value, $\mathcal{I}_0$. This way the algorithm can start refining some already existing grid. It is common to set $\mathcal{I}_0 = \mathcal{I}_L^T$ for some small value L.

The combination of the functions *expand*, *insert* and *weight* allows a great deal of flexibility. Several common choices are discussed below.

Basic adaptive algorithm

The basic algorithm easily fits in the described framework. The concrete functions are described in Algorithm 13.

Algorithm 13 Functions for the basic adaptive algorithm

function EXPAND($\boldsymbol{l}, \mathcal{O} \cup \mathcal{A}$)
    return $Weight(\boldsymbol{l}) > \varepsilon$
end function

function INSERT($\boldsymbol{l}, \mathcal{O} \cup \mathcal{A}$)
    return true
end function

function WEIGHT($\boldsymbol{l}, \mathcal{O} \cup \mathcal{A}$)
    return $\sum_{\boldsymbol{j} \in \mathcal{J}_{\boldsymbol{l}}} |\check{f}_{\boldsymbol{j}}|^2$
end function

Regular grid generation

The adaptive algorithm can actually be used to generate a regular grid with index set $\mathcal{I}_L^T$. There is no reason to use this in practice but it is still interesting to see how it works since it provides a deeper understanding about the general algorithm. This is demonstrated in Algorithm 14. To make the example more interesting, we can impose an additional constraint on the index set. Define the order of an index as $order(\boldsymbol{i}) := \#\{i_k > 0\}$. Only indices with orders less than or equal to *maxOrder* will be allowed to enter the grid. Thus the grid will neglect function terms that involve the interaction of more than *maxOrder* variables.

Algorithm 14 Functions for the generation of a regular sparse grid

function EXPAND($\boldsymbol{l}, \mathcal{O} \cup \mathcal{A}$)
    return true
end function

function INSERT($\boldsymbol{l}, \mathcal{O} \cup \mathcal{A}$)
    return ($order(\boldsymbol{l}) \leq maxOrder$) **and** ($|\boldsymbol{l}|_1 - T|\boldsymbol{l}|_\infty \leq (1-T)L$)
end function

function WEIGHT($\boldsymbol{l}, \mathcal{O} \cup \mathcal{A}$)
    return -$|\boldsymbol{l}|_1$
end function

L_2-error driven refinement

A more interesting application is an adaptive algorithm which terminates when the approximate L_2 error drops under a certain threshold. This involves a time-consuming approximate error computation for every inserted grid index. Therefore, the algorithm is generally slow. However, it has the advantage that it avoids overrefinement by terminating as soon as the error threshold is passed. Algorithm 11 can be used in a similar way. However, it usually inserts more indices in the grid because the global L_2 error is not computed for every added index.

Algorithm 15 Functions for the L_2-driven refinement

function EXPAND($\boldsymbol{l}, \mathcal{O} \cup \mathcal{A}$)
    return true
end function

function INSERT($\boldsymbol{l}, \mathcal{O} \cup \mathcal{A}$)
    return $||f - I_{\mathcal{O}\cup\mathcal{A}} f||_2 > \varepsilon$
end function

function WEIGHT($\boldsymbol{l}, \mathcal{O} \cup \mathcal{A}$)
    return $\sum_{\boldsymbol{j}\in\mathcal{J}_{\boldsymbol{l}}} |\check{f}_{\boldsymbol{j}}|^2$
end function

7.3 Dimension-adaptive algorithm

The adaptive method from the previous section inspects each level index individually and decides whether to insert it in the grid or not. The adaptive behaviour of the algorithm can be enhanced even more by allowing it to also adapt to the dimension of the problem. Assume that we have an n-dimensional function f with decreasing importance of its dimensions. This means that its values depend less on the variables from the higher dimensions compared to the variables from the lower dimensions. Such a function should be approximated with an anisotropic grid. If L_d denotes the maximum refinement level in direction d, then L_d should be a decreasing function of d. If the dependence of the function of the last dimensions is very low, then it might be best to choose $L_d = 1$ for $d > n_0$ for some $n_0 < n$. With this choice, the resulting interpolant is in practice a function of only n_0 variables.

Consider now what would happen if the adaptive algorithm is executed on such a function. Assume that the initial index set is $\mathcal{I}_0 = \{\mathbf{0}\}$. On the first step, the index $\mathbf{0}$ will be inspected and all of its neighbours will be added to the active set. The neighbours have indices $\boldsymbol{e}_d$, $d \in \{1, \dots n\}$. On each step the algorithm will choose the most important active index. Since the variation in dimensions $d > n_0$ is insignificant, the weight of the indices $\boldsymbol{e}_d$ will be close to zero. Thus, the algorithm will never insert an index $\boldsymbol{e}_d$ with $d > n_0$ to the old set. The resulting index set will contain no indices $\boldsymbol{\nu}$ with $\nu_d > 0$ for some $d > n_0$(apart from the initially inserted $\boldsymbol{e}_d$). The constructed interpolant will effectively be an n_0-dimensional function. The index set can be analyzed after the algorithm has finished and the non-important dimensions can be discarded.

This approach is perfectly valid. However, it suffers from the disadvantage that it inspects an n-dimensional sparse grid for the construction of an n_0-dimensional interpolant. Working with n-dimensional grids is inherently slower than working with n_0-dimensional grids even if the index sets are the same(ignoring the last $n - n_0$ elements of the larger indices, which are equal to 0). The extra computational time comes mostly from the slower operations on the data structures that represent the sparse grid. Some overhead also comes from the target function computation for the indices $\boldsymbol{e}_d$ for $d > n_0$. The number of extra evaluations is $(g_1 - g_0)(n - n_0)$.

This problem can be solved by allowing the algorithm to adapt also to the dimension of

the function. The modified algorithm starts by constructing a grid with a small number of dimensions. On each step it decides which index to insert in the old set. The modification is that it also decides whether the current grid dimension should be increased. If this is the case then two things should be done. First, all indices(old and active) should be widened by appending a 0 to their end. Second, the index $\boldsymbol{e}_d$ should be added to the active set, where d is the newly added dimension. The algorithm only adds the dimensions which are needed. The data structure maintenance cost remains low and the resulting index set is the same(except for the indices e_d for $d > n_0$, but they do not contribute to the approximation). This improvement can have a significant impact on the computational time of the adaptive algorithm, especially when $n_0 << n$. The pseudocode of the algorithm is included in Algorithm 16.

Algorithm 16 Dimension adaptive algorithm

$n_0 = n_{init}$
$\mathcal{O} = \{\}$
$\mathcal{A} = \{\mathbf{0}\}$
while $\mathcal{A}$ is not empty **do**
    $\boldsymbol{i} = \arg\max_{\boldsymbol{k} \in \mathcal{A}} \{weight(\boldsymbol{k})\}$
    $\mathcal{O} = \mathcal{O} \cup \{\boldsymbol{i}\}$
    $\mathcal{A} = \mathcal{A} \setminus \{\boldsymbol{i}\}$
    if $expand(\boldsymbol{i}, \mathcal{O} \cup \mathcal{A})$ **then**
        for $d = 1, \ldots n$ **do**
            $\boldsymbol{j} = \boldsymbol{i} + \boldsymbol{e}_d$
            if $\boldsymbol{j} \notin \mathcal{O}$ **and** $admissible(\mathcal{O} \cup \{\boldsymbol{j}\})$ **and** $insert(\boldsymbol{j}, \mathcal{O} \cup \mathcal{A})$ **then**
                $\mathcal{A} = \mathcal{A} \cup \{\boldsymbol{j}\}$
            end if
        end for
        if $\boldsymbol{i} = \boldsymbol{e}_{n_0}$ **and** $n_0 < n_{max}$ **then**
            $n_0 = n_0 + 1$
            $updateDataStructures()$
        end if
    end if
end while
return $\mathcal{O}$

8 Practical Remarks

One of the main goals of this work is to implement the theoretical results from the previous sections in an efficient and flexible C library. Implementing numerical algorithms can be a challenging task. Often the gap between theory and implementation is significant and the practical application of the methods is difficult. Section 4 contained a discussion about efficient algorithms for the computation of sparse grid operators. Section 7 also contained algorithms for adaptive grid refinement. However apart from these few examples, there are many issues that arise in practice and that have not been discussed yet. The goal of this section is to address several of them and to present ways to tackle them efficiently.

8.1 Computation of Leja sequences

The recursive definition of classic and weighted Leja sequences is simple from a mathematical point of view but presents some computational difficulties. The main problem arises from the limited floating point precision of most programming languages. The term $\prod_{k=0}^{n} |z - z_k|$ in the optimization problem is difficult to handle because it can easily cause a precision overflow or underflow. In fact, the computation becomes unstable much before overflow is reached because floating point precision degrades quickly when the values increase. Another problem is that the optimized function has many local maxima and finding the global one requires some additional observations.

Let us first consider the problem about finding the global maximum. The Leja sequence is generated by finding the maximizer of the function $|p_n(z)|$, where $p_n(z) = \prod_{k=0}^{n}(z - z_k)$. $p(z)$ is simply a polynomial with $n+1$ different real roots. Hence it has a unique local extremum between every two consecutive roots. Therefore, the function $|p_n(z)|$ has exactly one local maximum in every interval (z_k, z_{k+1}) for $k = 0, \dots n-1$.

In the case of weighted Leja sequences, the optimized function is $|v(z)p_n(z)|$, where $v(z) := \sqrt{w(z)} = e^{-z^2/2}$. The goal now is to prove that the function $v(z)p_n(z)$ also has a single extremum between every two consecutive points. Each extremum corresponds to a root of the derivative $(vp_n)'$:

$$(vp_n)'(z) = v'(z)p_n(z) + v(z)p_n'(z) = -zv(z)p_n(z) + v(z)p_n'(z) = v(z)[p_n'(z) - zp_n(z)] \quad (8.1)$$

Since v is positive for every $z \in \mathbb{R}$, it follows that the roots of $(vp_n)'$ are the same as the roots of $h(z) := p_n'(z) - zp_n(z)$. h is a polynomial of degree $n+2$ and has at most $n+2$ real roots. Therefore vp_n has at most $n+2$ extrema. On the other hand, vp_n has at least one extremum in every interval (z_k, z_{k+1}) for $k = 0, \dots n-1$. We also know that $\lim_{z\to\infty} v(z)p_n(z) = 0$ because p_n is a polynomial and $v(z)$ has an exponential decay. Since $v(z_n)p_n(z_n) = 0$, it follows that vp_n has at least on other extremum in the interval (z_n, ∞). The same holds for $(-\infty, z_0)$. Since the maximum number of extrema of vp_n is $n+2$, it follows that it has exactly one extremum in (z_n, ∞), $(-\infty, z_0)$ and in every interval (z_k, z_{k+1}) for $k = 0, \dots n-1$. Thus, $|vp_n|$ has exactly one local maximum in each of these intervals.

The goal of this discussion was to justify the usage of a simple local optimizer for each interval (z_k, z_{k+1}). Using Newton's method is sufficient. To find the global maximizer we need to compute all local maximizers and then take the one that yields the biggest function value.

This means that there might be up to several thousand Newton's method executions just to find a single Leja point. Unfortunately, this is unavoidable. A partial solution to this problem is to run the optimization problems in parallel, since they are independent of each other.

Let us now consider the more subtle problem of avoiding the floating point limitations. To unify the notation, define $v(z) := 1$ for the case of classic Leja sequences. The way to avoid the numerical overflows and underflows is to transform the problematic product to a summation. The usual way to do this is by taking a logarithm of the product. The resulting local optimization problem is

$$z_{n+1,k} \in \underset{z \in (z_k, z_{k+1})}{\arg\max} \left\{ \log(v(z)) + \sum_{i=0}^{n} \log |z - z_i| \right\} \tag{8.2}$$

for every $k = 0, \dots n-1$. In the weighted case we can set $z_{-1} := -\infty$ and $z_{n+1} := \infty$ and define $z_{n+1,-1}$ and $z_{n+1,n+1}$ in the same way. This optimization problem does not suffer from the numerical instability as the product version.

As a conclusion, let us discuss the necessity of this method. In practice, it is rare to need more than the first several hundred terms of a Leja sequence. In some cases these terms can be precomputed and simply stored on the disk for future usage. The initial computation can be done with a more advanced software package that has infinite floating point precision as well as local and global optimizers. Therefore, it is tempting to think that the discussion in this subsection is unnecessary.

This observation is valid for the case of classic Leja sequences since its basis functions are not parametrized. It is actually also true for the generalized Hermite transform because the parametrized weighted Leja sequences can be generated from the base weighted Leja sequence by simply scaling and translating it with α and β. Thus, it is enough to precompute and store the base sequence. However, this is not always the case. The weighted Leja sequence can also be defined for the Jacobi transform by simply changing the weight function in the optimization problem. The Jacobi basis functions also depend on two parameters α and β but in this case their interpretation is not as straightforward as in the Hermite case. The parametrized Leja sequences that are generated for different pairs (α, β) are not related to each other. There is no way to precompute all possible sequences because the space of parameters is infinite. Therefore, the only possible option is to compute each Leja sequence on the fly. The method described in this section can easily be extended to work with generalized Jacobi weight functions.

8.2 Fast inversion of matrix sequences

The general algorithms for forward and inverse transforms, hierarchization and dehierarchization, are performed with the help of the change of the matrix A_{g_n-1}(as defined in equation (3.6)) and its inverse. In a mixed sparse grid these matrices must be computed for all types of transforms and for all possible levels g_n. If this computation is done inefficiently, it can substantially increase the computational time of the algorithm. In fact, if it is done naively, most of the computational time is spent on matrix inversion.

One possible solution to this problem is to store the precomputed matrices on the disk. The matrices can later be read directly from the disk instead of computed anew. This approach is only a partial solution because it introduces a new problem - the amount of stored data is

enormous. One matrix must be stored for each type of transform and for each possible level. For some transforms the supported levels must contain up to several thousand interpolation points. The required disk space can easily become many gigabytes. This is clearly impractical. Thus, the only viable option is to compute the matrices on the fly in a very efficient manner.

To optimize the process, we should consider the following more general problem. Assume that $n \in \mathbb{N}$ and $M \in \mathbb{C}^{n\times n}$ is a matrix whose inverse M^{-1} is known. Let M and M^{-1} have the following representation

$$M := \begin{pmatrix} A & B \\ C & D \end{pmatrix}, \qquad M^{-1} := \begin{pmatrix} E & F \\ G & H \end{pmatrix}, \tag{8.3}$$

where $A, E \in \mathbb{C}^{m\times m}$, $B, F \in \mathbb{C}^{m\times(n-m)}$, $C, G \in \mathbb{C}^{(n-m)\times m}$ and $D, H \in \mathbb{C}^{(n-m)\times(n-m)}$ for some $m < n$. Assume also that H is invertible. The question is how to find the inverse of A by using the fact that M^{-1} is already computed. We know that

$$\begin{pmatrix} E & F \\ G & H \end{pmatrix}\begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} I_m & 0 \\ 0 & I_{n-m} \end{pmatrix}. \tag{8.4}$$

Multiplying this equation on the left with the matrix

$$\begin{pmatrix} I_m & -FH^{-1} \\ 0 & I_{n-m} \end{pmatrix}. \tag{8.5}$$

gives

$$\begin{pmatrix} E - FH^{-1}G & 0 \\ G & H \end{pmatrix}\begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} I_m & -FH^{-1} \\ 0 & I_{n-m} \end{pmatrix}. \tag{8.6}$$

This equation implies that $(E - FH^{-1}G)A = I_m$ and thus

$$A^{-1} = E - FH^{-1}G. \tag{8.7}$$

The next questions is what is the computational complexity of equation (8.7). It involves one matrix inversion, two matrix multiplications and one matrix subtraction. Let us examine these operations separately:

- Computing the inverse H^{-1} has complexity $O((n-m)^3)$.
- Computing the product $H^{-1}G$ has complexity $O((n-m)^2 m)$.
- Computing the product $F(H^{-1}G)$ has complexity $O((n-m)m^2)$.
- Computing the subtraction $E - F(H^{-1}G)$ has complexity $O(m^2)$.

Therefore, the total complexity of equation (8.7) is $O((n-m)\max((n-m)^2, (n-m)m, m^2))$. The question now becomes: for what values of m this method is faster than direct matrix inversion, whose complexity is $O(m^3)$. For example, if $m = n-1$ the complexity of equation (8.7) becomes $O(n^2)$ which is a major improvement over the direct $O(n^3)$. On the other hand, if m is a small constant, then equation (8.7) has complexity $O(n^3)$ while the direct inversion method takes constant time. Clearly, when m gets smaller the benefit of using the new formula decreases. The equilibrium point is $m = n/2$. In this case, both formulas have complexity

$O(n^3/8)$. Therefore, for all values of $m > n/2$ equation (8.7) is the preferred method. If $m \leq n/2$ or if H is not invertible, one should use direct inversion of the matrix A.

This approach can be applied to sparse grids in the following way. For each type of transform the matrices $A_{g_0-1}, A_{g_1-1}, \dots A_{g_n-1}$ and their inverses $A^{-1}_{g_0-1}, A^{-1}_{g_1-1}, \dots A^{-1}_{g_n-1}$ must be computed. The first step is to compute $A^{-1}_{g_n-1}$ with a direct matrix inversion(e.g. with LU decomposition). Next, the algorithm proceeds down the grid levels and computes $A^{-1}_{g_k-1}$ for $k = n-1, \dots 0$. If $g_k > g_{k+1}/2$ then equation (8.7) is applied, with $A = A_{g_k-1}$ and $M = A_{g_{k+1}-1}$. Otherwise $A^{-1}_{g_k-1}$ is computed directly. This procedure is summarized in Algorithm 17.

Algorithm 17 Computes the inverse matrices $A^{-1}_{A_0-1}, A^{-1}_{g_1-1}, \dots A^{-1}_{g_n-1}$.

```
1: procedure MATRIX_SEQUENCE_INVERSION
2:     A^{-1}_{g_n-1} = INVERSE(A_{g_n-1})
3:     for k = n − 1, downto 0 do
4:         if g_k > g_{k+1}/2 and IS_INVERTIBLE(H_{g_{k+1}-1}) then
5:             A^{-1}_{g_k-1} = E_{g_{k+1}-1} − F_{g_{k+1}-1} H^{-1}_{g_{k+1}-1} G_{g_{k+1}-1}
6:         else
7:             A^{-1}_{g_k-1} = INVERSE(A_{g_k-1})
8:         end if
9:     end for
10: end procedure
```

To demonstrate the efficiency of the method, consider the PLUS1 grids introduced in [Mat14]. The sparse grid levels in PLUS1 grids grow as slowly as possible, i.e. $g_n = n + 1$. A direct computation of all inverses leads to a complexity of $O(1^3) + O(2^3) + \dots O((n+1)^3) = O(n^4)$. On the other hand, the method described here can be applied to every level because the corresponding matrix H_{g_k-1}(which in this case is simply a number) is always invertible. The total computational complexity is $O(1^2) + O(2^2) + \dots O(n^2) + O((n+1)^3) = O(n^3) + O((n+1)^3) = O(n^3)$. This is a significant improvement that allows for the computation of much bigger problems.

9 Numerical Results

9.1 Mixed sparse grids

This subsection demonstrates the convergence behaviour of mixed sparse grids. The tests combine typical functions for the Fourier, Chebyshev and Hermite transforms. The functions are combined in a tensor product manner. The constructed grids are regular dyadic grids and have up to 12 dimensions. The tests measure the relative L_2-norm and the L_∞-norm, which are computed numerically. The number of grid nodes is limited to 10^7 due to limitations of the testing machine. In a few of the graphics it is evident that the number of points is not enough to demonstrate the limit convergence behaviour for dimensions $n = 10$ and $n = 12$. Finally, the Hermite transform in this subsection is used with $(\alpha, \beta) = (1, 0)$.

The testing functions are:

- For the Fourier transform([GH14]):

$$F_p(x) := (2 + \operatorname{sign}(x - \pi)) \sin(x)^p \tag{9.1}$$

  for $p = 1, 2, 3, 4$ and on the interval $\mathbb{T}$.

- For the Chebyshev transform([BNR00]):

$$C_1(x) := \frac{1}{4 + (x - 0.5)^2} \tag{9.2}$$

  on the interval $[0, 1]$.

- For the Hermite transform:

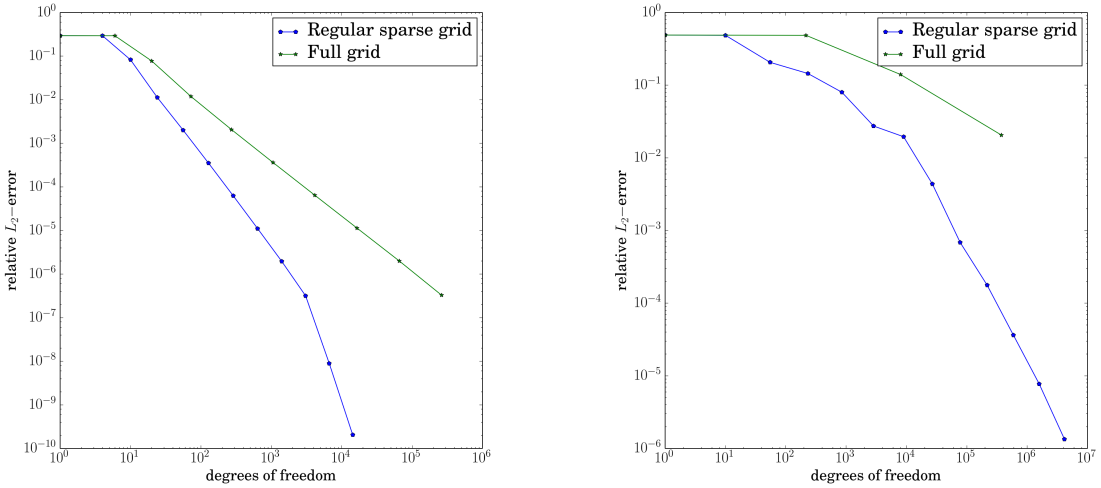
$$H_1(x) := (1 + x)e^{-x^2} \tag{9.3}$$

  on $\mathbb{R}$.

The multidimensional tensor product functions are denoted with a superscript, i.e.

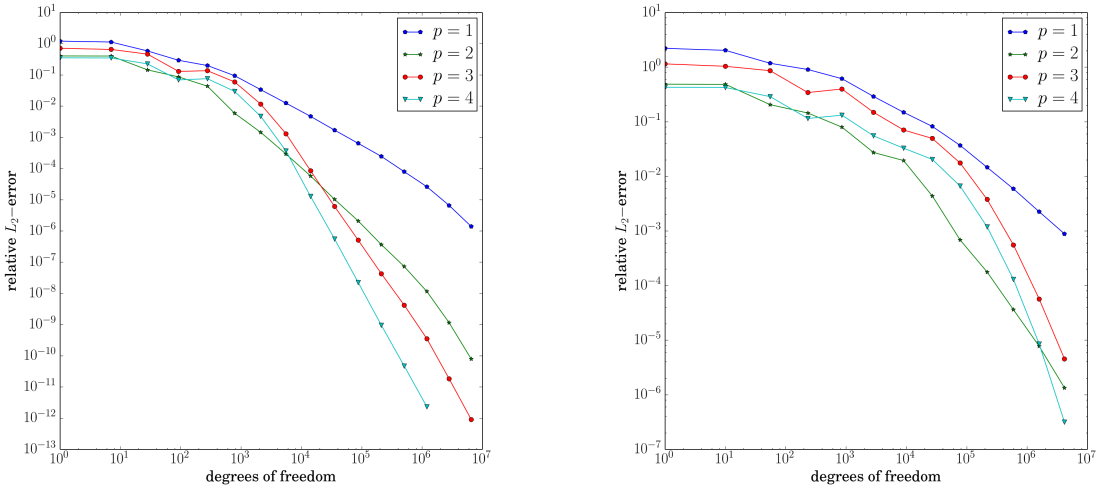
$$F_p^{(n)} = \bigotimes_{d=1}^{n} F_p. \tag{9.4}$$

Mixed test functions are written explicitly, e.g. $(F_p^{(3)} \otimes C_1^{(3)} \otimes H_1^{(3)})$.



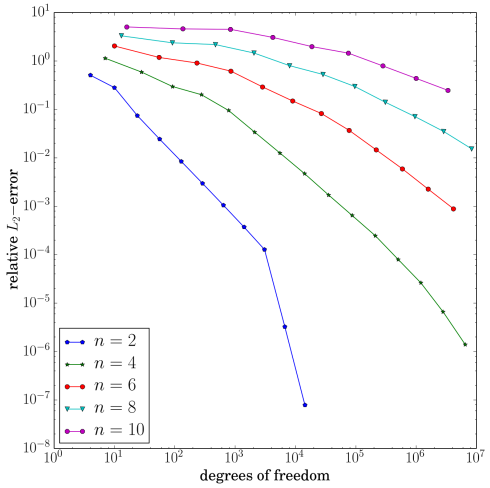
(a) $n = 2$ (b) $n = 6$

Figure 18: Convergence behaviour of full and regular sparse Fourier-Chebyshev grids for different dimensions. The test function is $F_2^{(n/2)} \otimes C_1^{(n/2)}$.

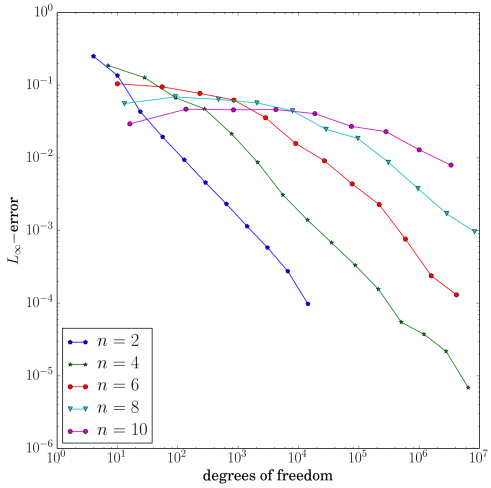


(a) $n = 4$ (b) $n = 6$

Figure 19: Convergence behaviour of regular sparse Fourier-Chebyshev grids for different dimensions and different values of the parameter p. The test function is $F_p^{(n/2)} \otimes C_1^{(n/2)}$.

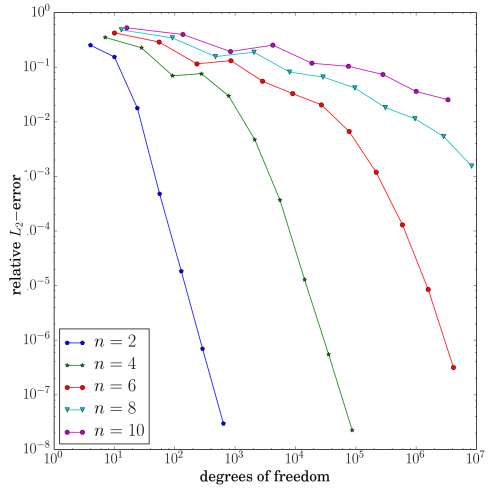


(a) L_2-error

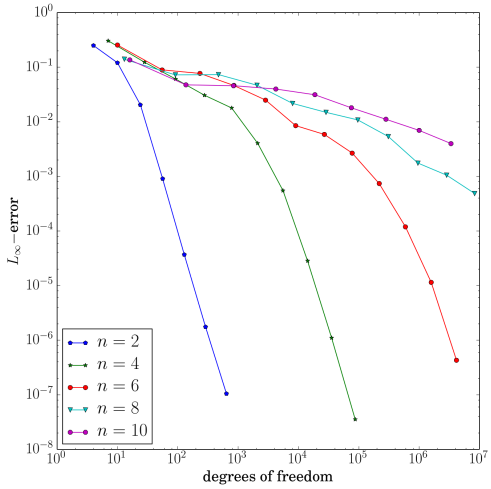


(b) L_∞-error

Figure 20: Convergence behaviour of regular sparse Fourier-Chebyshev grids for different dimensions. The test function is $F_1^{(n/2)} \otimes C_1^{(n/2)}$.

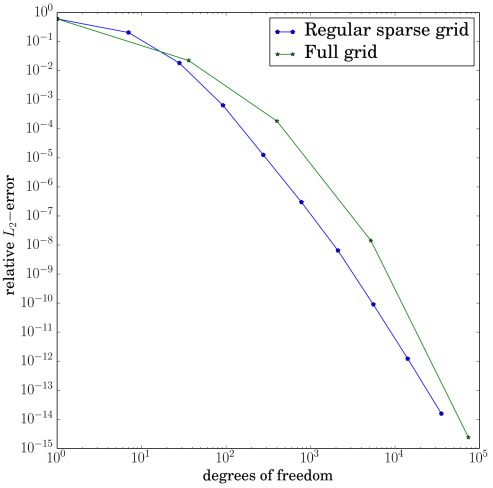


(a) L_2-error

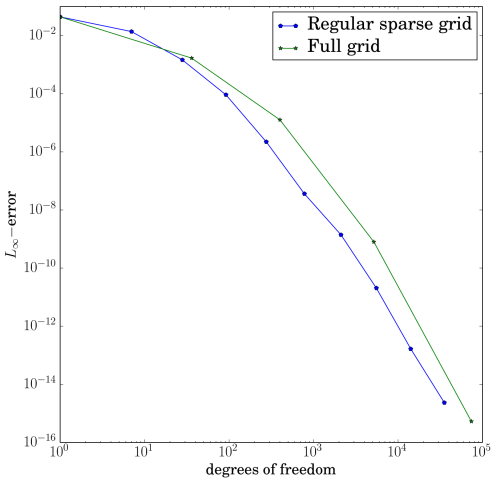


(b) L_∞-error

Figure 21: Convergence behaviour of regular sparse Fourier-Chebyshev grids for different dimensions. The test function is $F_4^{(n/2)} \otimes C_1^{(n/2)}$.

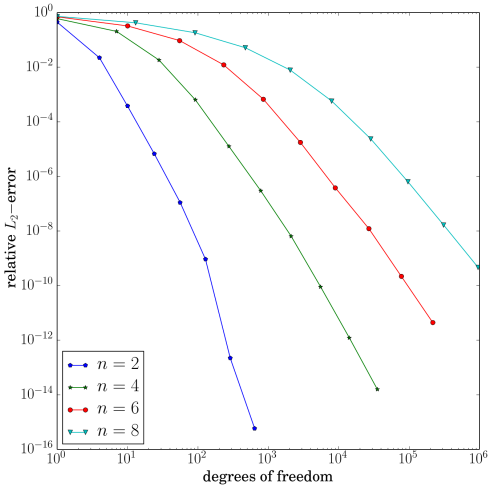


(a) L_2-error

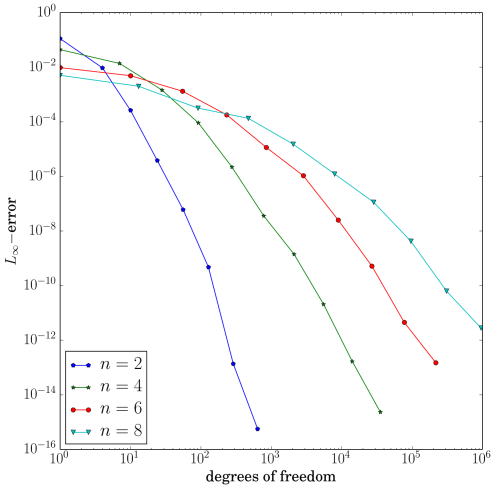


(b) L_∞-error

Figure 22: Convergence behaviour of full and regular sparse Chebyshev-Hermite grids. The test function is $C_1^{(2)} \otimes H_1^{(2)}$.

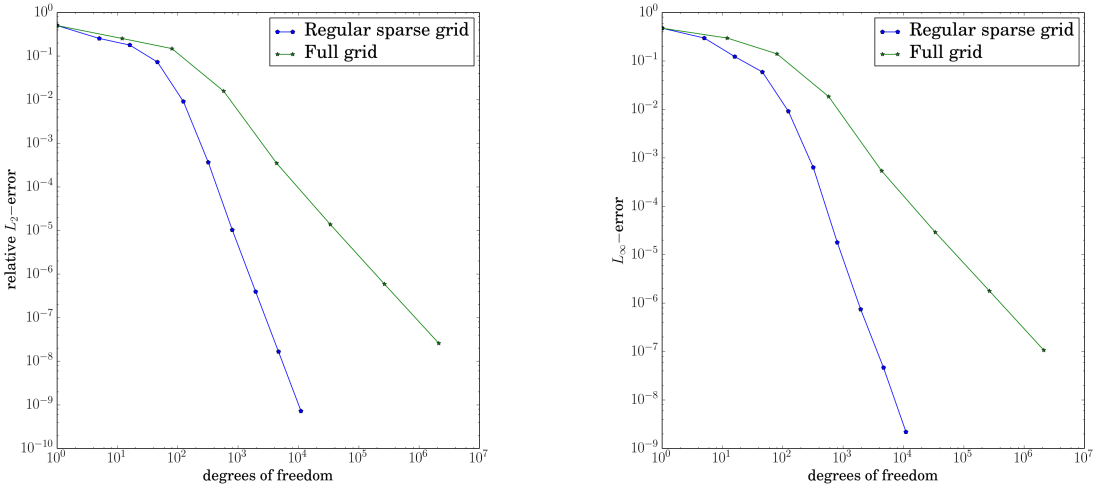


(a) L_2-error



(b) L_∞-error

Figure 23: Convergence behaviour of regular sparse Chebyshev-Hermite grids for different dimensions. The test function is $C_1^{(n/2)} \otimes H_1^{(n/2)}$.



(a) L_2-error (b) L_∞-error

Figure 24: Convergence behaviour of full and regular sparse Fourier-Chebyshev-Hermite grids. The test function is $F_4^{(1)} \otimes C_1^{(1)} \otimes H_1^{(1)}$.

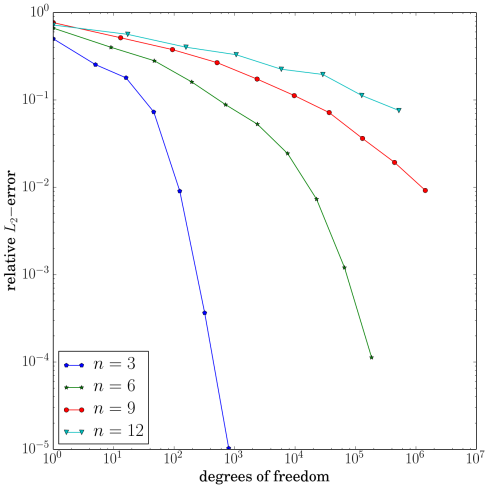


Figure 25: Convergence behaviour of regular sparse Fourier-Chebyshev-Hermite grids for different dimensions. The test function is $F_4^{(n/3)} \otimes C_1^{(n/3)} \otimes H_1^{(n/3)}$.

9.2 General grids

This subsection discusses the convergence properties of general grids. First, we show a comparison between dyadic Chebysev grids using the classic interpolation nodes and Leja points. The examples confirm that Leja sequences perform well but still incur an accuracy penalty. This is not visible in the one-dimensional cases but is obvious in higher dimensions. Several functions are compared because they have different properties and this reflects in the convergence comparison.

Next we demonstrate the convergence of Hermite grids with weighted Leja sequences. Unfortunatelly, a comparison between Leja points and the classic nodes is not possible because a sparse grid cannot be constructed with the latter. The examples are limitted to a convergence comparison between full and sparse grids.

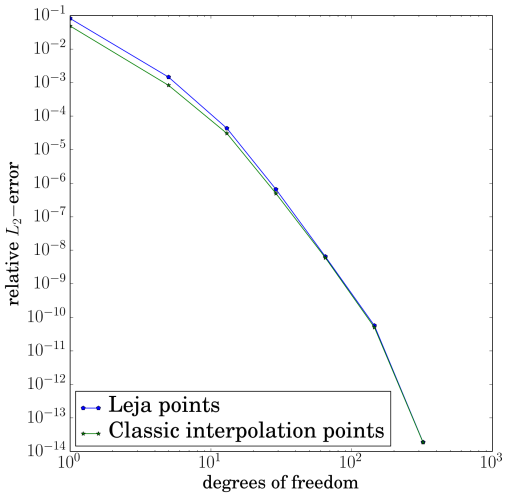
Finally, we show a possible use case of slowly growing sparse grids. The convergence behaviour between full, sparse and PLUS1 grids is compared. PLUS1 one grids grow as slowly as possible, i.e. $g_n = n + 1$ in every direction. They are studied in detail in the context of the Fourier transform in [Mat14]. Here, we demonstrate that the results are also applicable to Chebyshev and mixed Fourier-Chebyshev grids.

For testing functions we use C_1 and H_1 from the previous subsection, as well as the following:

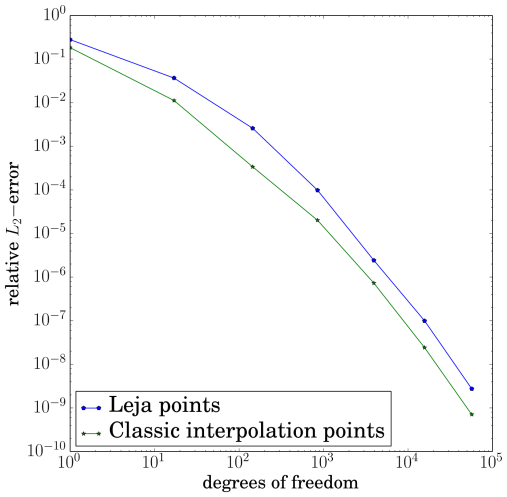
- For the Fourier transform:
$$F_2(x) := e^{\cos(x)} \tag{9.5}$$
on the domain $\mathbb{T}$.

- For the Chebyshev transform:
$$C_2^{(n)}(\boldsymbol{x}) := \frac{1}{1 + 0.5 \sum_{i=1}^{n} x_i} \tag{9.6}$$
on the domain $[0, 1]^n$.

- For the Hermite transform:
$$H_2(x) := (1 + |x|)e^{-x^2} \tag{9.7}$$
on $\mathbb{R}$.

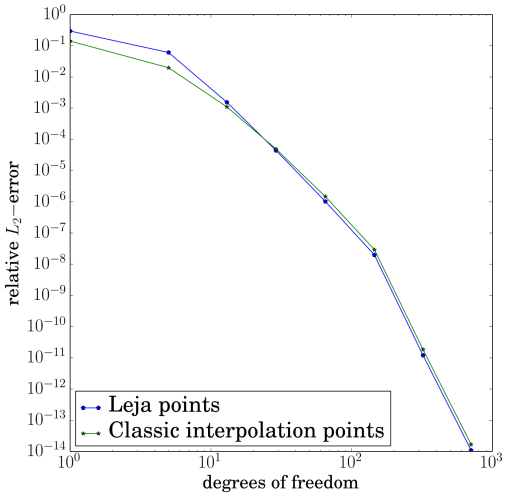


(a) $n = 2$

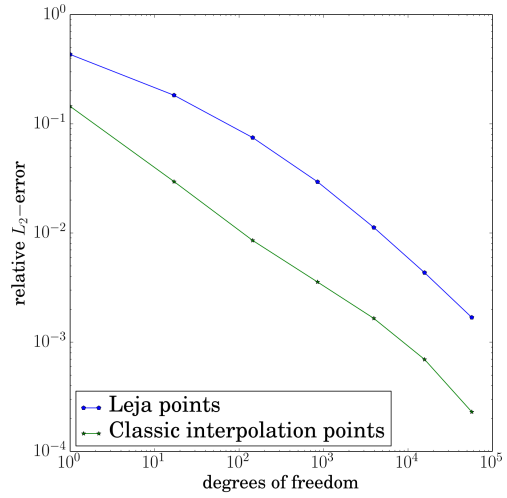


(b) $n = 8$

Figure 26: Convergence behaviour on dyadic Chebyshev grids with classic interpolation points and with Leja points in different dimensions. The test function is $C_1^{(n)}$.

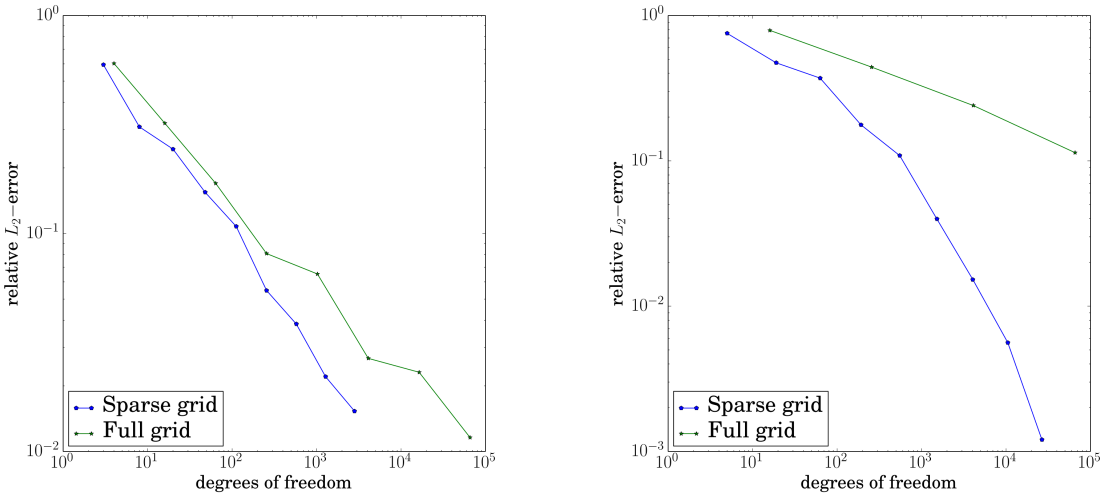


(a) $n = 2$



(b) $n = 8$

Figure 27: Convergence behaviour on dyadic Chebyshev grids with classic interpolation points and with Leja points in different dimensions. The test function is $C_2^{(n)}$.



(a) n = 2 (b) n = 4

Figure 28: Comparison between Hermite full and dyadic grids in different dimensions. The test function is $H_2^{(n)}$.

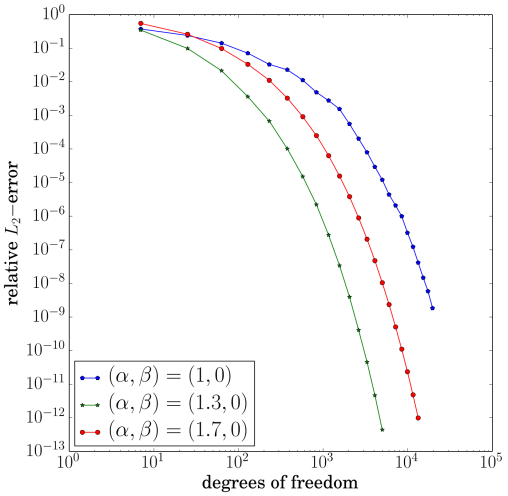
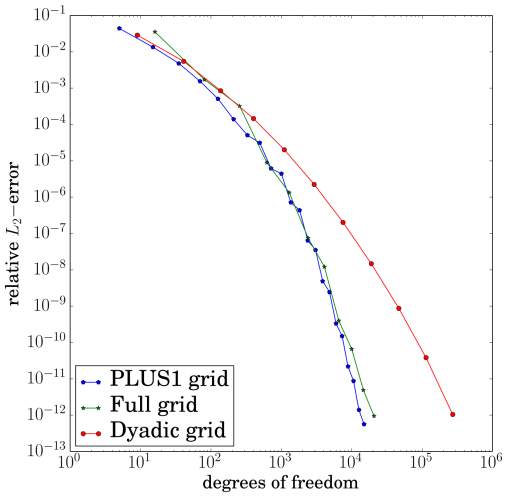
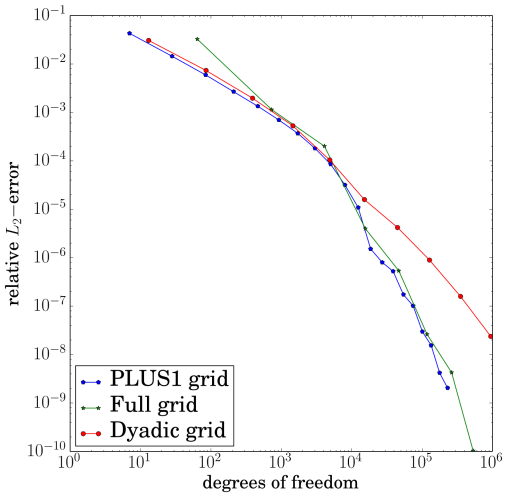


Figure 29: Comparison of convergence on general Hermite grids with $g_n = 2n + 1$ and with different parameters α and β. The test function is $H_1^{(3)}$.

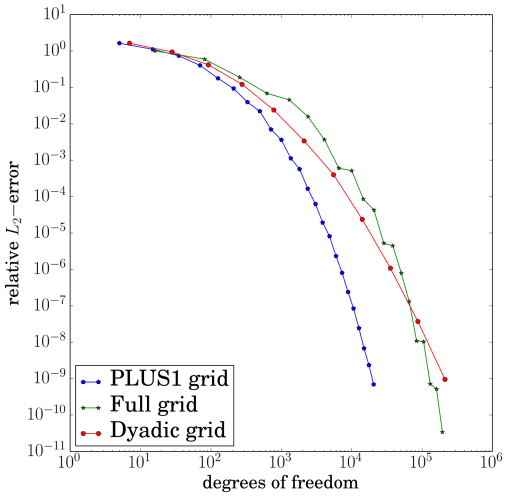


(a) $n = 4$

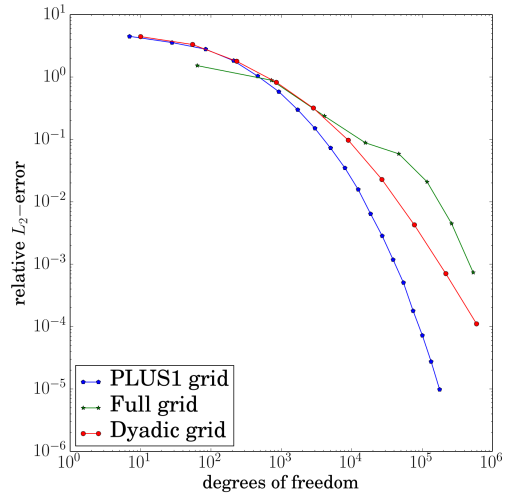


(b) $n = 6$

Figure 30: Comparison of the convergence behaviour on full, dyadic and PLUS1 Chebyshev grids in different dimensions. The test function is $C_2^{(n)}$.



(a) $n = 4$



(b) $n = 6$

Figure 31: Comparison of the convergence behaviour on full, dyadic and PLUS1 Fourier-Chebyshev grids in different dimensions. The test function is $F_2^{(n/2)} \otimes C_2^{(n/2)}$.

9.3 Adaptive grids

One typical application of the adaptive algorithm is on functions which do not depend on interactions between many variables. More precisely, consider the ANOVA decomposition([ES81]) of a function f:

$$f(\boldsymbol{x}) = \sum_{u \subset \{1,\dots n\}} f_u(\boldsymbol{x}_u), \tag{9.8}$$

where $\boldsymbol{x}_u$ is a subvector of $\boldsymbol{x}$ formed by the elements corresponding to the indices in u. The dependence of f on interactions between s variables is captured by the terms with $|u| = s$. This subsection provides some examples for the effectiveness of the adaptive algorithm when $f_u = 0$ for $|u| > C$ for some small constant $C << n$. In this case we have

$$f(\boldsymbol{x}) = \sum_{u \subset \{1,\dots n\}, |u| \leq C} f_u(\boldsymbol{x}_u). \tag{9.9}$$

If we know a priori that the function f has this form, we can limit the index set of the grid to contain only indices of order not greater than C, i.e. $order(\boldsymbol{i}) := \#\{i_k > 0\} \leq C$. This is true because every level index $\boldsymbol{i}$ typically corresponds to basis functions which involve $order(\boldsymbol{i})$ variables. Basis functions with more than C variables will not contribute to the function f. Therefore, all coefficients from $\mathcal{J}_{\boldsymbol{i}}$ with $order(\boldsymbol{i}) > C$ will be equal to zero. Note that hyperbolic cross index sets include level indices with arbitrary high order and thus are not directly appropriate for approximation of functions of this type. If we know the order C, we can eventually use hyperbolic cross index sets with excluded high order indices.

If we do not know the maximum order C, we can execute an adaptive algorithm to detect it. The adaptive algorithm is guaranteed to excluded all indices with order higher than C because the weight of each such index will be zero. This leads to a big improvement compared to hyperbolic cross grids. This is demonstrated in Figure 32a with a Fourier-Chebyshev grid and test function

$$F_C^{(n)}(\boldsymbol{x}) := \sum_{d=1}^{n} e^{\frac{1}{n} \sum_{k=0}^{C-1} \cos(x_{(d+k) \bmod C})} \tag{9.10}$$

on $\mathbb{T}^{n/2} \times \mathbb{I}^{n/2}$. Note that F_C is a sum of n functions with C variables.

In fact, this reasoning is only valid under certain assumptions. It was assumed that an index with order C includes basis functions which depend only on C variables. This is indeed true for Fourier and Chebyshev transforms because the first basis function in both cases is simply equal to 1. For example, in the Chebyshev case we have

$$T_{\boldsymbol{k}}(\boldsymbol{x}) = \prod_{i=1}^{n} T_{k_i}(x_i) = \prod_{i=1}^{C} T_{k_i}(x_i) \tag{9.11}$$

if $k_i = 0$ for $i > C$.

Things are not that straightforward with the Hermite transform and the reason is that $\mathcal{H}_0^{1,0}(x) = ce^{-x^2/2}$. Therefore, every basis function depends on n variables and this holds even for $\mathcal{H}_{\boldsymbol{0}}^{1,0}$. Thus, the application of adaptive grids involving the Hermite transform (both mixed and non-mixed) to functions with such ANOVA expansion is impossible. Assume now that the function has the modified form

$$f(\boldsymbol{x}) = \prod_{i \in S} \mathcal{H}_0^{\alpha_i, \beta_i} \sum_{u \subset \{1,\dots n\}, |u| \leq C} f_u(\boldsymbol{x}_u), \tag{9.12}$$

where $S \subset \{1, \dots n\}$ is the set of indices that correspond to the directions in the grid that use the Hermite transform and (α_i, β_i) are the parameters in these directions. In this case, the adaptive algorithm can be applied as in the Fourier-Chebyshev case. The order of all inserted level indices will be smaller or equal to C. An example of this can be seen in Figure 32b on a Fourier-Chebyshev-Hermite grid with the modified test function

$$\tilde{F}_C^{(n)}(\boldsymbol{x}) := \prod_{i=\frac{2}{3}n+1}^{n} \mathcal{H}_0^{1,0} \sum_{d=1}^{n} e^{\frac{1}{n} \sum_{i=0}^{C-1} \cos(x_{(d+k) \bmod C})}. \tag{9.13}$$

The Hermite transform is used with parameters $(1, 0)$ for dimensions $\{2n/3 + 1, \dots n\}$. The Fourier and the Chebyshev transforms are used for dimensions $\{0, \dots n/3\}$ and $\{n/3{+}1, \dots 2n/3\}$, respectively.

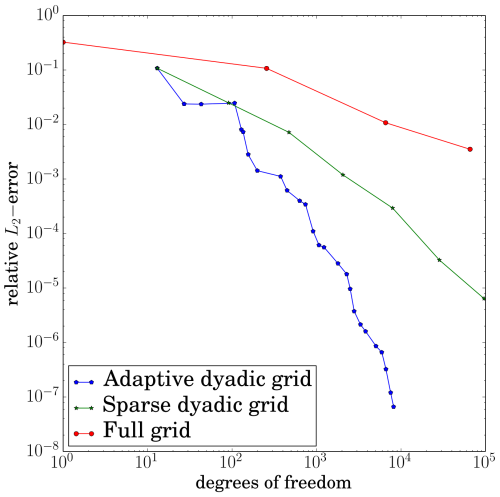
Apart from the finite order convergence results, we also examine the generated degrees of freedom when different weights are used. In most cases the average square sum of hierarchical coefficients provides slightly better results (Figure 33).

Then we compare the number of degrees of freedom in grids generated with Algorithm 11 and Algorithm 15 with the same global termination criteria. In Algorithm 15 this criteria is checked on every index insertion and thus the number of nodes in this case is slightly smaller. The iterative ε reduction algorithm typically adds a few more indices before it stops.

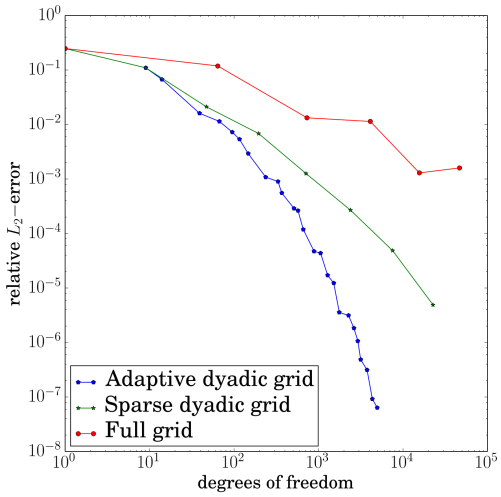
Finally, we demonstrate the execution time of the dimension-adaptive approach for the function

$$G(\boldsymbol{x}) :== \left(1 + \sum_{d=1}^{n} 2^{-2i+1} x_i\right)^{-1} \tag{9.14}$$

on a dyadic Chebyshev grid with domain $[0, 1]^n$. The importance of higher dimensions in this function decreases quickly. The dimension adaptive version of the basic algorithm with $\varepsilon = 10^{-7}$ and weighting function $weight_4$ adds just 12 dimensions, no matter how high we set the maximum allowed dimension. The execution time does not depend on the maximum allowed dimension at all. We present a comparison to an adaptive algorithm that always works with a maximum dimension grid. Even though this algorithm never refines the grid in direction higher than 12, the overhead of the internal grid representation is visible (Figure 35).

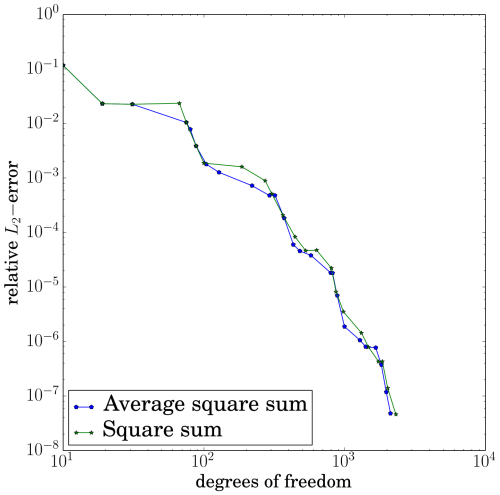


(a) $F_4^{(8)}$

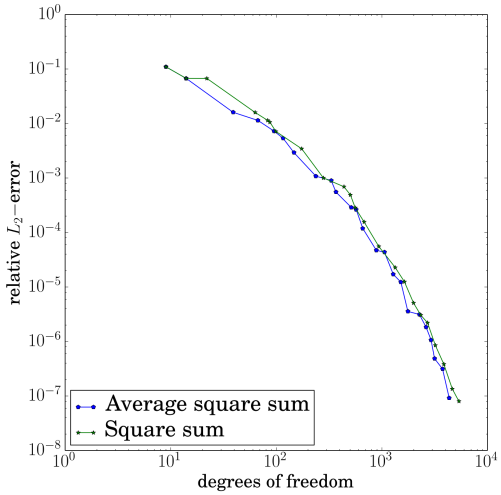


(b) $\tilde{F}_3^{(6)}$

Figure 32: Comparison between the convergence on adaptive, sparse and full grids for different functions.



(a) $F_3^{(6)}$ on a Fourier-Chebyshev grid



(b) $\tilde{F}_3^{(6)}$ on a Fourier-Chebyshev-Hermite grid

Figure 33: Convergence comparison between the functions $weight_3$ and $weight_4$ from section 7. In most cases averaged sum of squared hierarchical coefficient is a better importance indicator than the non-averaged one.

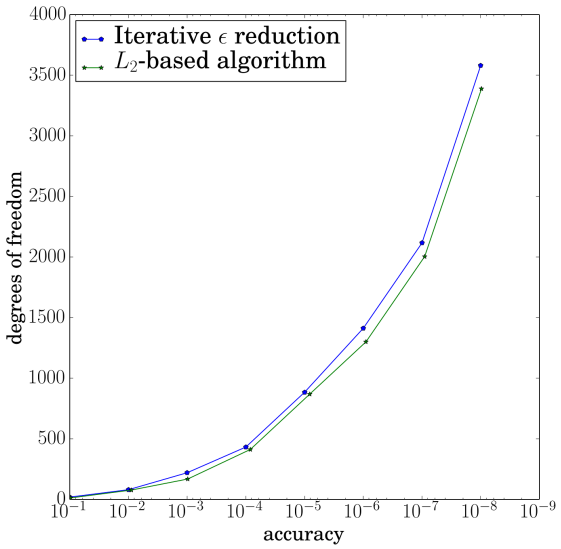


Figure 34: Comparison between the iterative Algorithm 11 and the L_2-driven Algorithm 15. The figure shows the number of nodes in the generated grid for a desired error accuracy. The L_2-driven approach always adds less points and consequently performs less function computations.

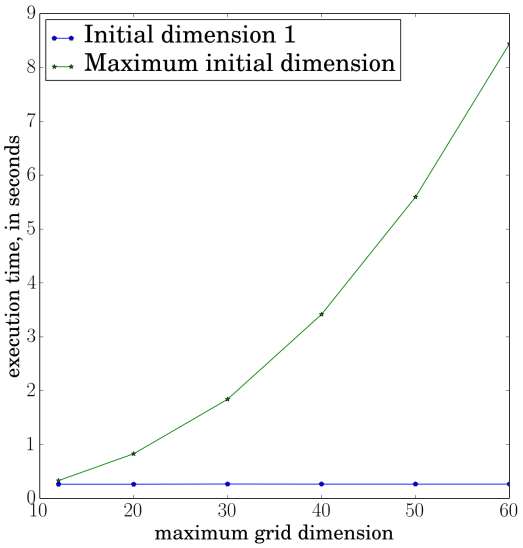


Figure 35: Execution time of a dimension-adaptive algorithm with a different maximum dimension. The blue line shows the execution time when the initial dimension is set to 1. The green line shows the time when the initial dimension is equal to the maximum one.

10 Conclusion

In this work we discussed several extensions to the regular sparse grids setting. The most focus was given on mixed sparse grids, i.e. grids in which the different directions make use of different transforms. The definition of this kind of grids is straighforward but their error analysis is not. Mixed Fourier-Chebyshev grids were presented in more detail. Error bounds were derived for the dyadic case on parametrized hyperbolic cross index sets. A key element in these derivations was the mixed aliasing lemma, which was also proven separately.

Another important part of this work was the discussion about general sparse grids. Dyadic sparse grids suffer from a lack of granularity, i.e. the control over the number of degrees of freedom is limited. Defining non-dyadic grids requires an appropriate set of alternative interpolation points since the classic ones are not nested anymore. The natural choice for polynomial grids are the Leja sequences because of their presumably good approximation properties. For the case of Hermite grids the interpolation points are given by a weighted Leja sequence. This sequence allows the very existence of nested Hermite grids because the classic Hermite interpolation nodes do not allow any nesting.

We also considered several important algorithms as well as optimization techniques. The adaptive algorithm was presented in several forms, with increasing level of complexity and flexibility. We discussed the advantages and disadvantages of the simple version of the algorithm and showed some cases in which it may produce unexpected results. Furthermore, an extension of the algorithm was considered that allows the application of non-local termination criteria. Apart from this algorithm, we also presented ways to decrease the complexity in key parts of the interpolation operator computation. The first one is the efficient computation of Leja sequences. The second and more crucial one is the efficient computation of the change of basis matrices and their inverses. The approach proposed here allows the computation of much larger general sparse grids.

Finally, we demonstrated how all of these methods can work together. The numerical results were based on classic functions used for the Fourier, Chebyshev and Hermite transforms. All computations were performed with the HCFFT library. A non-trivial task was to implement them efficiently. A lot of time was spent to add the ability to combine all of them transparently to the user. Apart from simply implementing the algorithms we also put a lof of effort on designing, optimizing and documenting the library.

The results in this work are by no means extensive or conclusive. The approximation properties of general polynomial sparse grids with Leja points are currently unproven. Even less is known in the case of Hermite sparse grids and no estimates exist for mixed grids involving the Hermite transform. An interesting application of Hermite sparse grid would be to test their efficiency for the solution of PDE. Hermite grids are already used in this context but they are non-nested because they are based on the classic Hermite interpolation nodes. Using Hermite grids with weighted Leja sequences will be less accurate but the reduced number of grid nodes might compensate for the worse approximation properties.

References

[Ada75] R. Adams. *Sobolev Spaces*. Academic Press, 1975.

[Bel61] Richard E. Bellman. *Adaptive control processes - A guided tour*. Princeton, New Jersey, U.S.A.: Princeton University Press, 1961, p. 255.

[BG04] Hans-Joachim Bungartz and Michael Griebel. "Sparse grids". In: *Acta Numerica* 13 (2004), pp. 1–123.

[BNR00] V. Barthelmann, E. Novak, and K. Ritter. "High dimensional polynomial interpolation on sparse grids". In: *Advances in Computational Mathematics* 12.4 (2000), pp. 273–288.

[Boy01] John P. Boyd. *Chebyshev and Fourier Spectral Methods*. Second. Dover Books on Mathematics. Mineola, NY: Dover Publications, 2001. ISBN: 0486411834 9780486411835.

[CC15] Albert Cohen and Abdellah Chkifa. "On the stability of polynomial interpolation using hierarchical sampling." In: *Sampling theory, a renaissance. Compressive sensing and other developments*. Cham: Birkhäuser/Springer, 2015, pp. 437–458. ISBN: 978-3-319-19748-7. DOI: 10.1007/978-3-319-19749-4_12.

[Chk13] Moulay Abdellah Chkifa. "Full Length Article: On the Lebesgue Constant of Leja Sequences for the Complex Unit Disk and of Their Real Projection". In: *J. Approx. Theory* 166 (Feb. 2013), pp. 176–200. ISSN: 0021-9045. DOI: 10.1016/j.jat.2012.11.005. URL: http://dx.doi.org/10.1016/j.jat.2012.11.005.

[CM11] J.-P. Calvi and P. V. Mahn. "Lagrange interpolation at real projections of Leja sequences for the unit disk". In: *Journal of Approximation Theory* 163(5) (2011), pp. 608–622.

[DM04] Stefano De Marchi. "On Leja Sequences: Some Results and Applications". In: *Appl. Math. Comput.* 152.3 (May 2004), pp. 621–647. ISSN: 0096-3003. DOI: 10.1016/S0096-3003(03)00580-0. URL: http://dx.doi.org/10.1016/S0096-3003(03)00580-0.

[ES81] B. Efron and C. Stein. "The Jackknife Estimate of Variance". In: *Ann. Statist.* 9 (1981), pp. 586–596.

[Fek23] M. Fekete. "Über die Verteilung der Wurzeln bei gewissen algebraischen Gleichungen mit ganzzahligen Koeffizienten". In: *Mathematische Zeitschrift* 17.1 (Dec. 1923), pp. 228–249. ISSN: 0025-5874. DOI: 10.1007/bf01504345. URL: http://dx.doi.org/10.1007/bf01504345.

[Gar07] Jochen Garcke. "A dimension adaptive sparse grid combination technique for machine learning". In: *Proceedings of the 13th Biennial Computational Techniques and Applications Conference, CTAC-2006*. Ed. by Wayne Read, Jay W. Larson, and A. J. Roberts. Vol. 48. ANZIAM J. http://anziamj.austms.org.au/ojs/index.php/ANZIAMJ/article/view/70 [December 27, 2007]. Dec. 2007, pp. C725–C740.

[Gaw87] Wolfgang Gawronski. "On the asymptotic distribution of the zeros of Hermite, Laguerre, and Jonquière polynomials". In: *Journal of Approximation Theory* 50(3) (1987), pp. 214–231.

[GG02] Jochen Garcke and Michael Griebel. “Classification with sparse grids using simplicial basis functions”. In: *Intell. Data Anal.* 6.6 (2002), pp. 483–502. URL: http://content.iospress.com/articles/intelligent-data-analysis/ida00107.

[GG03] Thomas Gerstner and Michael Griebel. “Dimension-Adaptive Tensor-Product Quadrature”. In: *Computing* 71.1 (2003), pp. 65–87. DOI: 10.1007/s00607-003-0015-5. URL: http://dx.doi.org/10.1007/s00607-003-0015-5.

[GH14] M. Griebel and J. Hamaekers. “Fast Discrete Fourier Transform on Generalized Sparse Grids”. In: *Sparse grids and Applications*. Vol. 97. Lecture Notes in Computational Science and Engineering 1305. INS Preprint No. 1305. Springer, 2014, pp. 75–108.

[GK00] M. Griebel and S. Knapek. “Optimized tensor-product approximation spaces”. In: *Constructive Approximation* 16.4 (2000), pp. 525–540.

[GK09] M. Griebel and S. Knapek. “Optimized general sparse grid approximation spaces for operator equations”. In: *Mathematics of Computation* 78 (Dec. 2009), pp. 2223–2257. DOI: 10.1090/s0025-5718-09-02248-0. URL: http://dx.doi.org/10.1090/s0025-5718-09-02248-0.

[Hal92] K. Hallatschek. “Fouriertransformation auf dünnen Gittern mit hierarchischen Basen”. In: *Numerische Mathematik* 63 (1992), pp. 83–97.

[Heg03] M. Hegland. “Adaptive sparse grids”. In: *Proc. of 10th Computational Techniques and Applications Conference CTAC-2001*. Ed. by K. Burrage and Roger B. Sidje. Vol. 44. [Online] http://anziamj.austms.org.au/V44/CTAC2001/Hegl [April 1, 2003]. Apr. 2003, pp. C335–C353.

[Kna00] S. Knapek. “Approximation und Kompression mit Tensorprodukt-Multiskalenräumen”. Dissertation. Universität Bonn, 2000.

[Lej57] Franciszek Leja. “Sur certaines suites liées aux ensembles plans et leur application à la représentation conforme”. fre. In: *Annales Polonici Mathematici* 4.1 (1957), pp. 8–13. URL: http://eudml.org/doc/208291.

[LY13] X. Luo and S. Yau. “Hermite Spectral Method with Hyperbolic Cross Approximations to High-Dimensional Parabolic PDEs”. In: *SIAM Journal on Numerical Analysis* 51(6) (2013), pp. 3186–3212.

[Mat14] K. Matuschke. “Trigonometrische Interpolation auf verallgemeinerten dünnen Gittern mit beliebiger Levelstruktur”. Diplomarbeit. Institut für Numerische Simulation, Universität Bonn, 2014.

[NJ14] Akil Narayan and John D. Jakeman. “Adaptive Leja Sparse Grid Constructions for Stochastic Collocation and High-Dimensional Approximation”. In: *SIAM J. Scientific Computing* 36.6 (2014). DOI: 10.1137/140966368. URL: http://dx.doi.org/10.1137/140966368.

[Pey02] R. Peyret. *Spectral Methods for Incompressible Viscous Flow*. Vol. 148. Applied Mathematical Sciences. 2002.

[Smo63] S. A. Smolyak. “Quadrature and interpolation formulas for tensor products of certain class of functions”. In: *Dokl. Akad. Nauk SSSR* 148.5 (1963). Transl.: Soviet Math. Dokl. 4:240-243, 1963, pp. 1042–1053.

[ST87] H.J. Schmeisser and H. Triebel. *Topics in Fourier Analysis and Function Spaces*. Mathematik und ihre Anwendungen in Physik und Technik. Akademische Verlagsgesellschaft Geest & Portig K.-G., 1987. ISBN: 9783321000010. URL: https://books.google.lu/books?id=bg35AQAACAAJ.

[Tan93] Tao Tang. "The Hermite Spectral Method for Gaussian-Type Functions". In: *SIAM J. Scientific Computing* 14.3 (1993), pp. 594–606. DOI: 10.1137/0914038. URL: http://dx.doi.org/10.1137/0914038.

[TT10] R. Taylor and V. Totik. "Lebesgue constants for Leja points". In: *Ima Journal of Numerical Analysis* 30 (2 2010), pp. 462–486. DOI: 10.1093/imanum/drn082.

[Zen91] C. Zenger. "Sparse grids". In: *Parallel Algorithms for Partial Differential Equations* 31 (1991), pp. 241–251.