

Sparse Grid Methods for Higher Dimensional Approximation

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch–Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich–Wilhelms–Universität Bonn

vorgelegt von

Christian Feuersänger

aus

Düsseldorf

Bonn 2010

Angefertigt mit Genehmigung der Mathematisch–Naturwissenschaftlichen Fakultät der Rheinischen Friedrich–Wilhelms–Universität Bonn

1. Gutachter: Prof. Dr. Michael Griebel

2. Gutachter: Prof. Dr. Marc Alexander Schweitzer

Tag der Promotion: 7. September 2010

Erscheinungsjahr: 2010

Zusammenfassung

Diese Arbeit befasst sich mit Dünngitterverfahren zur Lösung von höherdimensionalen Problemen. Sie zeigt drei neue Aspekte von Dünnen Gittern auf: Erweiterungen der elementaren Werkzeuge zur Arbeit mit Dünnen Gittern, eine Analyse von sowohl inhärenten Einschränkungen als auch Vorteilen von Dünnen Gittern speziell für die Anwendung zur Dichteapproximation (Fokker–Planck–Gleichung) sowie einen neuen Ansatz zur dimensions- und ortsadaptiven Darstellung von Funktionen effektiv niedriger Dimension.

Der erste Beitrag beinhaltet die erste (dem Autor bekannte) Fehlerschranke für inhomogene Randbedingungen bei Dünngitterapproximation und eine erweiterte Operationsbibliothek zur Durchführung von Addition, Multiplikation und Hintereinanderausführung von Dünngitterdarstellungen sowie einen adaptiven Kollokationsansatz für approximative Integraltransformationen mit beliebigen Kernen. Die Analyse verwendet Konditionszahlen für den Datenfehler und verallgemeinert damit die speziellen Elementarabschätzungen aus [Gri98] und [MgF07]. Ferner wird erstmals auch der Konsistenzfehler bei derartigen Operationen berücksichtigt sowie eine adaptive Methode zur Kontrolle desselben vorgeschlagen, die insbesondere zuvor vorhandene Schwachstellen behebt und die Methode verlässlich macht.

Der zweite Beitrag ist eine Untersuchung von dimensionsabhängigen Kosten/Nutzen-Koeffizienten, wie sie bei der Lösung von Fokker–Planck–Gleichungen und der damit verbundenen Approximation von Wahrscheinlichkeitsdichten auftreten. Es werden sowohl theoretische Schranken als auch A-posteriori-Fehlermessungen anhand einer repräsentativen Fallstudie für lineare Fokker–Planck–Gleichungen und der Normalverteilung auf \mathbb{R}^d vorgestellt und die auftretenden dimensionsabhängigen Koeffizienten bei Interpolation und Bestapproximation (sowohl L_2 als auch beim Lösen der Gleichung mittels Galerkin-Verfahren) untersucht. Dabei stehen reguläre Dünne Gitter, adaptive Dünne Gitter und die speziell für die Energienorm optimierten Dünne Gitter im Vordergrund (letzteres ähnlich wie die Energieabschätzungen in [Gri06]). Insbesondere werden Schlussfolgerungen auf inhärente Einschränkungen aber auch auf Vorteile gegenüber klassischen Vollgitterverfahren diskutiert.

Der dritte Beitrag dieser Arbeit ist der erste Ansatz für dimensionsadaptive Verfeinerung, der insbesondere für Approximationsprobleme konzipiert wurde. Der Ansatz behebt bekannte Schwierigkeiten mit frühzeitiger Terminierung, wie sie bei bisherigen Ansätzen zur Verallgemeinerung der erfolgreichen Dimensionsadaptivität aus dem Bereich Dünngitterquadratur zu beobachten waren (vgl. [Gar04]). Das Verfahren erlaubt eine systematische Reduktion der Freiheitsgrade für Funktionen, die effektiv nur von wenigen (Teilmengen von) Koordinaten abhängen. Der Ansatz kombiniert die erfolgreiche ortsadaptive Dünngittertechnik aus dem Bereich der Approximation mit der ebenfalls erfolgreichen dimensionsadaptiven Verfeinerung aus dem Bereich der Dünngitterquadratur [GG03, Hol08]. Die Abhängigkeit von unterschiedlichen (Teilmengen von) Koordinaten wird mittels gewichteter Räume unter Zuhilfenahme der ANOVA-Zerlegung durchgeführt [NW08]. Die Arbeit stellt neue a priori optimierte Dünngitterräume vor, die optimale Approximation für Funktionenräume mit gewichteten gemischten zweiten Ableitungen und bekannten Gewichten erlauben. Die Konstruktion liefert die bekannten

regulären Dünnen Gitter mit gewichtsabhängigen Leveln für jede Teilmenge von Koordinaten (ANOVA Komponenten) im Unterschied zu bekannten Dünngitterkonstruktionen aus [Kna00] (der Ansatz verläuft ähnlich wie die gewichteten Quadraterräume in [Hol08]). Für unbekannte Gewichte wird eine neue a-posteriori dimensionsadaptive Methode vorgestellt, die im Unterschied zu bekannten Verfahren aus [GG03, Gar04] explizit ANOVA Komponenten ermittelt und berücksichtigt und so höhere Verlässlichkeit beim Einsatz für Approximationsanwendungen erzielt. Neben reiner dimensionsadaptiver Approximation erlaubt das Verfahren auch erstmals gekoppelte orts- und dimensionsadaptive Verfeinerung. Die Arbeit stellt die Methodik dar und verifiziert die Verlässlichkeit anhand dimensionsadaptiver Interpolation und dimensionsadaptiver Lösung partieller Differentialgleichungen.

Danksagung

An dieser Stelle gilt mein Dank Gott als meinem Schöpfer für die Fähigkeit, Stärke und auch Gelegenheit zum Studium interessanter Themen mit faszinierenden Werkzeugen – und das in exzellenter Arbeitsatmosphäre. Besonders bedanke ich mich hiermit bei Prof. Dr. Michael Griebel für die Überlassung des interessanten Themas, die hochwertige Betreuung, zahlreiche Ideen und Anregungen, den Arbeitsplatz und die exzellenten Arbeitsbedingungen, sowohl in Bezug auf Ausstattung als auch in Bezug auf das Miteinander in der Arbeitsgruppe. Des weiteren bedanke ich mich herzlich bei Prof. Dr. Marc Alexander Schweitzer für die Übernahme des Zweitgutachtens. Mein Dank gilt weiterhin meinen Kollegen für viele fruchtbringende Diskussionen, an dieser Stelle besonders Dr. Markus Holtz und Ralf Wildenhues. Mein Dank geht auch an Jens Oettershagen, Bastian Bohn und Alexander Hullmann, deren Nutzung der entstandenen Softwarebibliothek zahlreiche Qualitätsverbesserungen hervorbrachte. Bei meinen Kollegen Bastian Bohn, Jürgen Braun, Alexander Hullmann, Jutta Adelsberger und Daniel Wissel bedanke ich mich für die Hilfe bei der Korrektur des Manuskripts. Schließlich bedanke ich mich bei meiner Frau Kerstin und meinen Eltern für die Unterstützung und Ermutigung während der Zeit der Promotion.

Bonn, Sommer 2010

Christian Feuersänger

Contents

1	Introduction	1
2	A Sparse Grid Approximation Algebra	5
2.1	Sparse Grid Approximation	5
2.1.1	The Common Multiscale Grid Structure	5
2.1.2	Basic Properties of Sparse Grid Spaces	13
2.1.3	Interpolation Error Bounds for Full and Sparse Grids	20
2.1.4	Error Bounds for Non-Homogeneous Boundary Conditions	23
2.2	Commonly used Algorithms and Complexities	31
2.2.1	Adaptive Interpolation and Approximation	31
2.2.2	Fast Algorithms: the Unidirectional Principle	35
2.2.3	Data Structures for Adaptive Sparse Grids	38
2.3	Approximative Algebraic Operations	48
2.3.1	Motivation and Overview	48
2.3.2	Approximative Pointwise Operations	49
2.3.3	Approximative Function Concatenation	55
2.3.4	Approximate Integral Transformation	59
3	Sparse Grids and Moderate Dimensional Approximation – Two Case Studies	63
3.1	Overview and Motivation	63
3.2	Case Study: Density Approximation and the Normal Distribution	64
3.2.1	Interpolation Error Estimates	64
3.2.2	Error Estimation Using Numerical Experiments	72
3.2.3	Generalizations	81
3.2.4	Related Topics in the Area of Information Based Complexity	88
3.2.5	Application in Moderate Dimensions: a Fokker–Planck–Example	89
3.3	Case Study: Functions with Axis Parallel Structure	94
3.3.1	Jump along a Hyperplane	95
3.3.2	Sparse Grids, Jumps, and Overshooting	95
3.3.3	Jump along a Hypercube	97
3.3.4	Diagonal Structure and Sparse Grids	98
3.3.5	Jump along an Arc	100
3.4	Summary on Moderate Dimensional Problems	101
4	Low Effective Dimensionality – A Dimension Adaptive Method	103
4.1	From ANOVA Decompositions to Sparse Grids	104
4.1.1	ANOVA–like Decompositions	104

4.1.2	Axis Parallel Structure and Low Effective Dimension	106
4.1.3	Sparse Grids as Discretized ANOVA Decomposition	107
4.2	Weighted Spaces and A Priori Optimized Sparse Grids	114
4.2.1	Weighted Mix Spaces	115
4.2.2	Sparse Grids with Optimized Cost/Gain Ratio	116
4.2.3	Examples of Weighted Sparse Grid Spaces	126
4.2.4	Summary of A Priori Grid Optimization	129
4.3	A Posteriori Adaptive Sparse Grids in Weighted Spaces	130
4.3.1	ANOVA-based Dimension Adaptive Refinement	131
4.3.2	Coupled Space- and Dimension Adaptive Refinement	136
4.4	On Non-Linearly Weighted Axis Parallel Approaches	138
4.5	Numerical Experiments in High Dimensions	140
4.5.1	Remarks on Error Estimation and Pointwise Operations	141
4.5.2	Dimension Adaptive Interpolation	141
4.5.3	A Dimension Adaptive PDE Solver	146
4.6	Summary of the Dimension Adaptive Method	150
5	Conclusion and Outlook	151
A	Technical Reference	155
A.1	Recursive Grid Traversal Routines	155
A.1.1	Visiting Every Grid Point in Linewise Ordering	155
A.2	Basis Bestiary – Hierarchical Transformations	157
A.2.1	The Multilevel Piecewise Linear Generating System	157
A.2.2	The Linear Hierarchical Basis	161
A.2.3	The Prewavelet Bases	162
A.2.4	The Haar Wavelet	166
A.2.5	Higher Order Hierarchical Polynomial Bases	169
A.3	One-Dimensional Matrix Vector Products Algorithms	174
A.3.1	The Mass Matrix for the Hierarchical Hat Basis	174
A.3.2	The Mass Matrix for the Prewavelet Basis	174
A.3.3	More on One-Dimensional Matrix-Vector-Products	176
A.4	Adaptive Refinement: Algorithmic Details	176
A.4.1	Remarks on Prewavelet Adaptivity	177
	Bibliography	179

1 Introduction

This thesis is about sparse grids and their application to higher dimensional approximation problems. The study is motivated by numerous applications in which the dimension is beyond three, for example physical processes depending on stochastic parameters (partial differential equations with stochastic right-hand-side, domain or variables [Har10, HSS08]) which are solved in twice the space dimension, or stochastic approaches to simulate dynamics described by the Fokker–Planck–Equation. Here, d entities change with time, depending on both deterministic dynamics (like ordinary differential equations) and stochastic fluctuations. Example applications are found in rheology for the simulation of non–Newtonian fluids by means of bead–spring models (see the textbooks [Ött96, BAH87] or [DLO07] for an approach related to sparse grids), in mechanics where coupled objects are excited by environmental load [WB00], and computational finance (Black Scholes Equation and its variants, compare the textbook of [Gla04]). The arising equations are usually handled by means of sampling methods of (Quasi) Monte Carlo type: many realizations followed by averaging [Ött96, BAH87]. Alternatively, the dynamics of the underlying probability density (which solves the parabolic Fokker–Planck–Equation) can be simulated by means of high order methods. However, classical methods to determine the density suffer from the so–called “curse of dimensionality”, a term coined by [Bel61] for exponential cost increase with growing dimension d : with N_{1d} degrees of freedom in one direction, full grid methods require $N = \mathcal{O}(N_{1d}^d)$ degrees of freedom in d dimensions and achieve an accuracy of $\mathcal{O}(N_{1d}^{-r}) = \mathcal{O}(N^{-r/d})$ where r depends on smoothness and the polynomial degree of ansatz functions.

Sparse grids have been proposed by [Zen91] as a tool to reduce the curse of dimensionality: provided the solution has bounded mixed derivatives up to order r , sparse grids allow $N = \mathcal{O}(N_{1d}(\log N_{1d})^{d-1})$ cost to achieve an accuracy of $\mathcal{O}(N_{1d}^{-r}(\log N_{1d})^{d-1})$. Sparse grid methods for solving approximation problems, especially partial differential equations, have been elaborated in [Bal94, Bun98] and following works; they have been applied to problems of higher dimensionality in computational finance (see, for example [Rei04, Mer05], also [Hol08] and its references for quadrature formulations), machine learning [Gar04, PPB10] and other fields, see [BG04] for an overview. The reported dimensionality ranges from $d = 3$ to $d = 8$, i.e. it is beyond the limitations of classical methods.

This thesis contributes three new aspects of sparse grids:

1. extensions to the elementary sparse grid tools,
2. a study on inherent limitations and benefits of sparse grid approximation methods applied to density approximation,

1 Introduction

3. a new approach of space- and dimension adaptive sparse grids suitable for functions of low effective dimension.

The first aspect includes the first (known to the author) sparse grid interpolation error bound for non-homogeneous boundary conditions and generalized theory on operations like addition, multiplication, concatenation and integral transformation involving sparse grid functions. The analysis of sparse grid operations not only generalizes known data error bounds from [Gri98] and [MgF07], it also discusses – for the first time – the involved consistency error and proposes and verifies adaptive algorithms to control it. Furthermore, the thesis presents an adaptive collocation method to compute arbitrary integral transformations $\int K(x, y)f(y) dy$ similar to the approaches analyzed in [Kna00]; it is based on adaptive sparse grids.

The second new aspect, the analysis of sparse grids for density approximation and its implications for the Fokker–Planck–Equation, is realized by means of a representative case study using the normal distribution on \mathbb{R}^d , which solves linear Fokker–Planck–Equations. The emphasis is especially on d -dependent order coefficients and the log terms arising for interpolation and best approximation for isotropic and anisotropic densities with respect to L_2 - and Galerkin projection for the involved Fokker–Planck–Equations. The order coefficients are obtained by both, theoretical error bounds and a posteriori error measurements for standard sparse grids, adaptive sparse grids and energy optimal sparse grids (similar to the analysis of energy order coefficients in [Gri06]). The inherent limitations arising due to d -dependent order coefficients are quantified as well as the superiority of sparse grids over alternative full grid methods.

The third new aspect presented in this thesis is a new dimension adaptive approximation approach which employs a different type of function space to achieve higher dimensions: weighted spaces of inherently low dimension. Here, functions have nominally high dimension, but have an inherent “effective dimension” which is much smaller. Weighted spaces or the related finite order weights have been discussed as main cause for the success of sparse grid quadrature methods [GG03] and Quasi–Monte–Carlo methods [PT95] for quadrature applications in computational finance, see [SWW04, NW08] and the references therein. The notion of effective dimension is based on the decomposition of functions into their ANOVA components, i.e. an additive superposition model where each summand depends on a subset of input coordinates. Decay of these components leads to weighted spaces or finite order weights for which non-exponential cost complexities can be expected [NW08].

The contribution of this thesis is a new dimension adaptive approximation framework. Besides a way to compute approximate ANOVA decompositions more efficiently than known integral-based approaches, a new adaptive approach to find and use optimal index sets for effectively low dimensional functions is elaborated. For spaces weighted with respect to second mixed derivatives in their ANOVA decomposition and explicitly known weights, a new a priori optimized sparse grid scheme is presented. Unlike other sparse grid construction schemes as in [Kna00], it yields regular sparse grids of individual, weight-dependent levels for each subset of the input coordinates (similar to the quadrature weighted sparse grid in [Hol08]). Furthermore, a posteriori optimized

spaces for the case of unknown weights are presented, realized with a new dimension adaptive algorithm which, in comparison to known approaches for integration [GG03] or machine learning [Gar04], explicitly computes and employs the ANOVA structure. The algorithm removes the early termination restriction from [Gar04] and yields reliable dimension adaptive resolution. Besides pure dimension adaptivity, it also supports coupled space- and dimension adaptive refinement in case one or more ANOVA components have local singularities. The algorithm is verified for numerical examples of dimension adaptive interpolation and partial differential equations.

The thesis is structured into three chapters which present the main results: Chapter 2 summarizes sparse grid techniques from the literature and elaborates the new error bounds and the sparse grid function algebra. Chapter 3 is dedicated to the second contribution of this thesis, the study on sparse grids for density approximation. It also provides a separate case study on axis parallel structure which constitutes a bridge to the dimension adaptive technique. The dimension adaptive approaches, the associated ANOVA framework and its verification is subject of Chapter 4.

1 Introduction

2 A Sparse Grid Approximation Algebra

The present chapter introduces sparse grids. It defines and explains the structure and summarizes approximation properties and the underlying arguments. Besides a survey over existing results, we will also see how to deal with non-homogeneous boundary values, which has not been elaborated so far. We will also elaborate new approximation tools to combine given sparse grid functions by means of pointwise operations, function concatenation or integral transforms. The approach extends known results and handles data errors and, for the first time, consistency errors. The chapter also summarizes common algorithms and data structures.

2.1 Sparse Grid Approximation

Our first step is to define sparse grid structures and to provide an overview from elementary approximation properties up to error estimates.

2.1.1 The Common Multiscale Grid Structure

We begin with a definition of one-dimensional multiscale grid decompositions, based on grids with multiple resolutions. We continue with the aspect of function approximation and basis functions living on these grids in Section 2.1.2.

Point Set Definitions

Sparse grids are defined on bounded product domains where we assume, without loss of generality, that $\Omega = [0, 1]^d$. Thus, we deal with one-dimensional grids discretizing the unit interval $[0, 1]$. Furthermore, we restrict ourselves to dyadic meshes, $h_l := 2^{-l}$. Then, the grid on level $l \geq 0$ is defined by the set of points

$$x_{l,i} := ih_l = i2^{-l} \tag{2.1}$$

where the index $i \in \{0, \dots, 2^l\}$ is called the space index at level l . We will identify grid points by multi-indices (l, i) , even though this is not a unique representation since $x_{l+1,2i} = x_{l,i}$ (grids are nested).

Space indices include the two boundary points $i = 0$ and $i = 2^l$. However, the distinction between inner points and boundary points plays a key role since we will define sparse grids on the boundary recursively as inner sparse grid points on a boundary manifold.

For fixed level l , the set of all inner grid points on the interval $[0, 1]$ is defined by

$$V_l := \{(l, i) \mid i = 1, \dots, 2^l - 1\}. \tag{2.2}$$

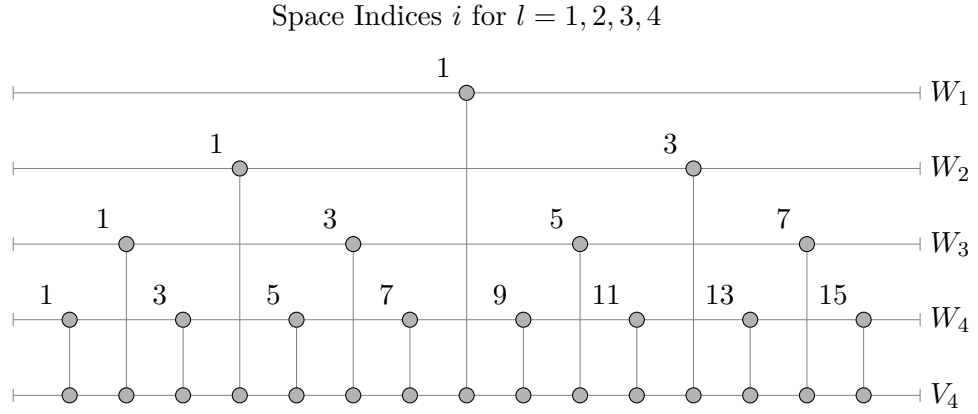


Figure 2.1: Hierarchical complements W_1, W_2, W_3, W_4 and the one scale grid V_4 together with the respective space indices.

Due to $V_0 = \emptyset$, we assume $l > 0$ when we are dealing with inner points. We write

$$x \in V_l \tag{2.3}$$

if there are indices $(l, i) \in V_l$ such that $x_{l,i} = x$. The nesting of grids in the sense of (2.3), $V_l \subset V_{l+1}$, leads to hierarchical complements

$$W_l := \{(l, i) \in V_l \mid x_{l,i} \notin V_{l-1}\}. \tag{2.4}$$

Due to the nesting rule $x_{l+1,2i} = x_{l,i}$, we have $x_{l+1,2i} \notin W_l$ whereas odd space indices are new contributions of level l :

$$W_l = \{(l, i) \mid i = 1, 3, 5, \dots, 2^l - 1, i \text{ odd}\}. \tag{2.5}$$

Applying this decomposition until $l = 1$, we can express any $x_{l,i} \in V_l$ uniquely using a multi index (\tilde{l}, \tilde{i}) in

$$G_l^1 := \bigcup_{k=1}^l W_k. \tag{2.6}$$

The grid G_l^1 describes the same points as V_l in a hierarchically structured way which we call multi scale grid decomposition. This is illustrated in Figure 2.1: the complements W_1, W_2, W_3 and W_4 are shown together with the one scale grid V_4 . In our example, the multi scale grid decomposition consists of the multi indices

$$(1, 1), (2, 1), (2, 3), (3, 1), (3, 3), (3, 5), (3, 7), (4, 1), \dots, (4, 15) \tag{2.7}$$

which make up a binary tree with root $(1, 1)$ and left child $(l + 1, 2i - 1)$ and right child $(l + 1, 2i + 1)$ for $l \geq 1$.

The tree structure implies hierarchical relations like ancestor and descendant: any point (k, j) on the direct, unique path from (l, i) to the tree's root $(1, 1)$ is called an *ancestor* whereas the subtree of (l, i) contains all *descendants* of (l, i) (see Figure 2.2).

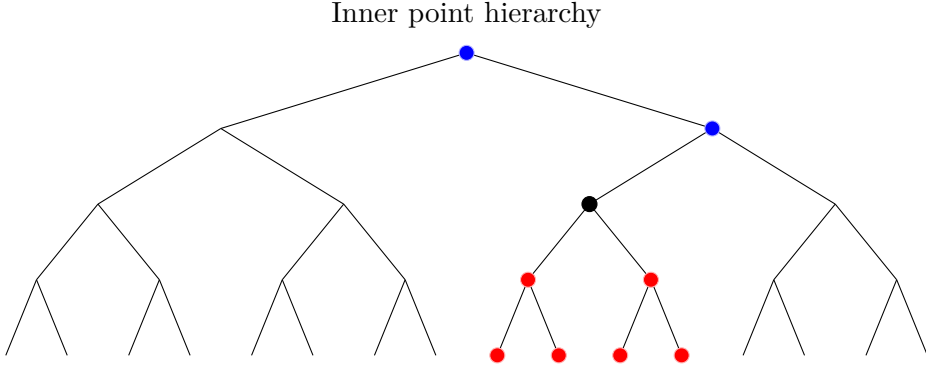


Figure 2.2: Ancestors (●) and descendants (●) of a point (●) in the grid G_4^1 .

For dimensions $d \geq 2$, the multi scale grid decomposition can be formulated componentwise, since we are given the product domain $\Omega = [0, 1]^d$. Instead of one single level, we are now given one level l_m for each component, $m = 1, \dots, d$. The definition of a scalar quantity “maximum level” will be discussed later. We define the one scale grid for a level index $\mathbf{l} = (l_1, \dots, l_d)$ component-wise using

$$V_{\mathbf{l}} := \{(\mathbf{l}, \mathbf{i}) \mid (l_m, i_m) \in V_{l_m}\} \quad (2.8)$$

where we restrict ourselves to inner points, $i_m = 1, \dots, 2^{l_m} - 1$. Coordinates are given by $x_{\mathbf{l}, \mathbf{i}} = (x_{l_1, i_1}, \dots, x_{l_d, i_d})^T$. A multi scale grid decomposition follows using the tensor product approach as component-wise formulation,

$$W_{\mathbf{l}} := \{(\mathbf{l}, \mathbf{i}) \mid x_{\mathbf{l}, \mathbf{i}} \notin V_{\mathbf{l} - \mathbf{e}_m} \text{ for } m = 1, \dots, d\}, \quad (2.9)$$

where \mathbf{e}_m is the m th unit vector. Note that due to our restriction to inner grid points, $l_m = 0$ implies $V_{\mathbf{l}} = \emptyset$. As before, $W_{\mathbf{l}}$ contains only odd space indices and is thus characterized by

$$W_{\mathbf{l}} = \{(\mathbf{l}, \mathbf{i}) \mid i \in \mathbf{I}_{\mathbf{l}}\} \quad (2.10)$$

with $\mathbf{I}_{\mathbf{l}} := \{\mathbf{l} \leq \mathbf{i} \leq 2^{\mathbf{l}-1}, i_m \text{ odd}\}$. The relation ‘ \leq ’ means componentwise comparison. For any fixed multi level \mathbf{l} , we can now describe the one scale grid $V_{\mathbf{l}}$ as composition of hierarchical complement grids. Thus, there is a one-to-one mapping from $V_{\mathbf{l}}$ to

$$G_{\mathbf{l}}^d := \bigcup_{k_1=1}^{l_1} \cdots \bigcup_{k_d=1}^{l_d} W_{k_1, \dots, k_d}. \quad (2.11)$$

This is illustrated in Figure 2.3 for two dimensions: the left picture contains all $V_{\mathbf{l}}$ for $l_1, l_2 = 1, 2, 3$ whereas the right picture shows all corresponding $W_{\mathbf{l}}$. The gray components on the right yield the same grid points as the gray one on the left, $V_{3,2}$.

Up to now, we have just treated cartesian grids with different mesh widths in different directions by introducing a hierarchical multi index description. We call splittings

2 A Sparse Grid Approximation Algebra

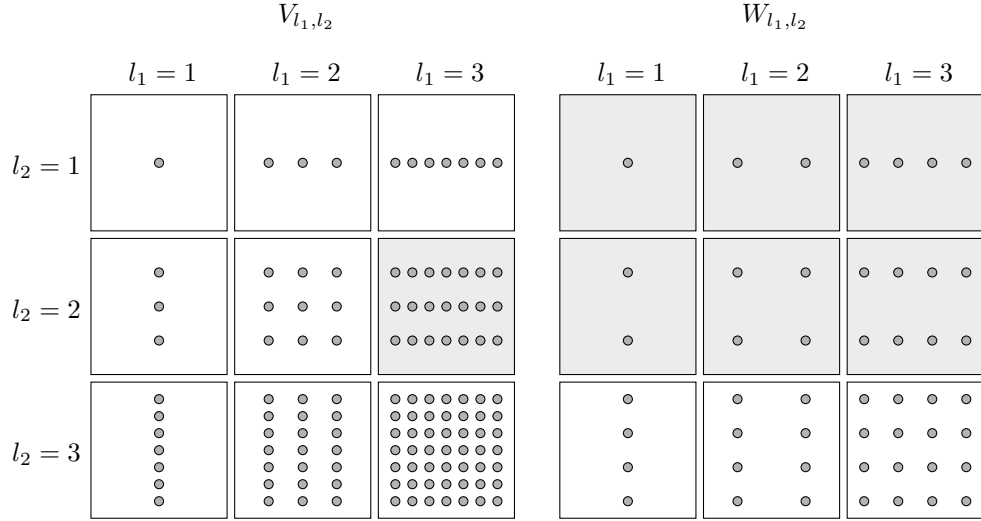


Figure 2.3: Multi scale grid decomposition in two dimensions for the anisotropic one scale grid $V_{3,2}$. The marked components on the right are the decomposition of $V_{3,2}$.

like (2.11) where all $W_{\mathbf{k}}$ for $k_m = 1, \dots, l_m$ are employed a *full* or *uniform* grid of multi-level \mathbf{l} . The full grid for $\mathbf{l} = (n, \dots, n)$ will be referred to as the full grid of level n , $n \in \mathbb{N}$,

$$\bar{G}_n^d := G_{n, \dots, n}^d = \bigcup_{1 \leq |\mathbf{l}|_\infty \leq n} W_{\mathbf{l}} =: \bigcup_{\mathbf{l} \in \mathbf{L}_\infty^n} W_{\mathbf{l}} \quad (2.12)$$

where $|\mathbf{l}|_\infty := \max\{l_1, \dots, l_d\}$.

As soon as we attach anisotropic basis functions of mesh width 2^{-l_m} to every grid point, “higher” levels will carry less information than “lower” levels. The rigorous derivation in Section 2.1.2 shows that functions with bounded second mixed derivatives are best represented by a different choice of levels, namely using $|\mathbf{l}|_1 := \sum l_m$. The associated grid

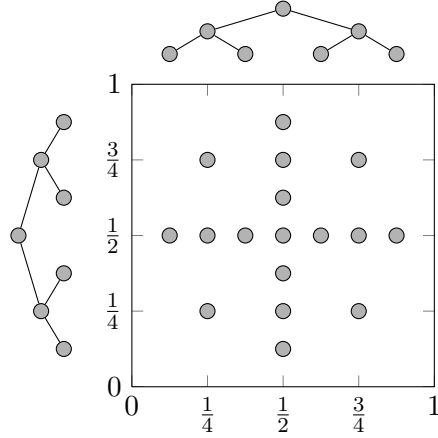
$$G_n^d := \bigcup_{1 \leq |\mathbf{l}-\mathbf{1}|_1 + 1 \leq n} W_{\mathbf{l}} =: \bigcup_{\mathbf{l} \in \mathbf{L}_1^n} W_{\mathbf{l}} \quad (2.13)$$

with $\mathbf{1} := (1, \dots, 1)$ is called the *sparse grid* or *regular sparse grid* of level n . A multi level \mathbf{l} with $l_m > 0$ for every m belongs to level

$$n(\mathbf{l}) := |\mathbf{l} - \mathbf{1}|_1 + 1 = |\mathbf{l}|_1 - d + 1, \quad (2.14)$$

normalized such that $n(1, \dots, 1) = 1$.

An example is shown in Figure 2.4 for $n = 3$ in two dimensions. The structure consists of all points with $1 \leq n(\mathbf{l}) \leq n$. Compared with Figure 2.3, only components of the upper left simplex are chosen for G_n^d . However, we can still work with the one-dimensional binary tree structure along every grid line. This is also shown in Figure 2.4: binary trees


 Figure 2.4: Sparse grid points of level $n = 3$ in two dimensions.

for x and y indicate the hierarchical relations. Taking all directions at once, we can define *ancestors* and *descendants* for the d -dimensional case component-wise: we define

$$\begin{aligned}
 (\mathbf{l}, \mathbf{i}) &\underset{H}{>} (\mathbf{k}, \mathbf{j}) : \Leftrightarrow (l_m, i_m) \text{ is ancestor of } (k_m, j_m) \text{ for all } m, \\
 (\mathbf{l}, \mathbf{i}) &\underset{H}{\leq} (\mathbf{k}, \mathbf{j}) : \Leftrightarrow \forall m \text{ holds:} \\
 &\quad (l_m, i_m) = (k_m, j_m) \text{ or } (l_m, i_m) \text{ is descendant of } (k_m, j_m), \\
 (\mathbf{l}, \mathbf{i}) &\underset{H}{\leq} (\mathbf{k}, \mathbf{j}) : \Leftrightarrow (\mathbf{l}, \mathbf{i}) \underset{H}{>} (\mathbf{k}, \mathbf{j}) \text{ or } (\mathbf{l}, \mathbf{i}) \underset{H}{\leq} (\mathbf{k}, \mathbf{j}).
 \end{aligned}$$

It should be stressed that these relations merely describe the hierarchy of points, it is not complete ordering.

Boundary Points

Up to now, our grids have no points on the boundary $\partial[0, 1]^d$. We define them recursively in this section. Let \tilde{G}^d be a grid as we used it for inner points (either the sparse grid G_n^d or the full grid \tilde{G}_n^d). For a point (\mathbf{l}, \mathbf{i}) , let $(\bar{\mathbf{l}}^m, \bar{\mathbf{i}}^m)$ be the multi index without the m th component of (\mathbf{l}, \mathbf{i}) . Then, grids with boundary points are defined recursively by

$$\begin{aligned}
 \tilde{G}^d := \tilde{G}^d \cup \bigcup_{m=1}^d \{ &(\mathbf{l}, \mathbf{i}) \mid l_m = 0, i_m = 0, (\bar{\mathbf{l}}^m, \bar{\mathbf{i}}^m) \in \tilde{G}^{d-1} \} \\
 &\cup \{(\mathbf{l}, \mathbf{i}) \mid l_m = 0, i_m = 1, (\bar{\mathbf{l}}^m, \bar{\mathbf{i}}^m) \in \tilde{G}^{d-1} \}
 \end{aligned} \tag{2.15}$$

with the initial condition

$$\tilde{G}^1 := \tilde{G}^1 \cup \{(0, 0), (0, 1)\}. \tag{2.16}$$

For one dimension, this adds only the two points $x_{0,0} = 0$ and $x_{0,1} = 1$ as we would have expected. In two dimensions, every boundary contains just a one-dimensional grid

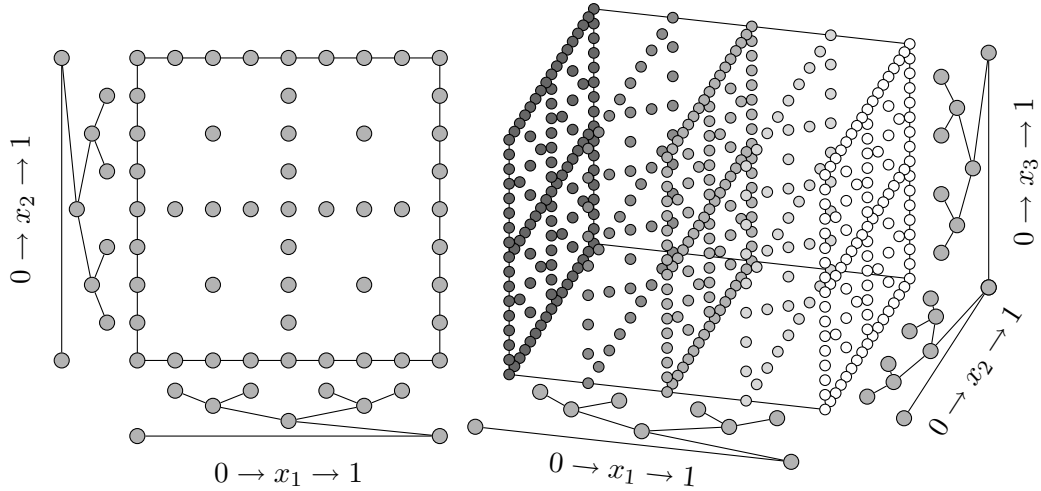


Figure 2.5: Sparse grids with boundary points in two dimensions (left, level $n = 3$) and three dimensions (right, level $n = 4$). For display reasons, three-dimensional points with $l_1 > 2$ have not been displayed and the mark colors depend on l_1 .

(which is the same for both, sparse and full grids). However, for three dimensions, the boundary structure depends strongly on the type of \tilde{G}^d ! If \tilde{G}^d is a sparse grid, \underline{G}^d will contain sparse grids of dimension $d-1$ on the boundary. If \tilde{G}^d is a full grid, the boundary will also contain full grids.

Sparse grids with boundary points in two dimensions are shown in Figure 2.5 (left): the boundary appears as one-dimensional, classical grid (with hierarchical point descriptions). The attached binary trees have been extended for level $l = 0$. However, the two points on level $l = 0$ have no clear hierarchical relationship, they are treated as special case with the artificial tree root $(0,0)$. A sparse grid with boundary points in dimension $d = 3$ is shown in Figure 2.5 (right), also together with binary trees indicating the hierarchical structure. Points with $l_1 > 2$ have been removed from the image to improve readability.

The recursive definition of boundary points yields the same boundary resolution (mesh width) as for inner points. In the full grid case, this could have been accomplished using the simple selection rule $0 \leq |\mathbf{l}|_\infty \leq n$ whereas the sparse grid selection criterion cannot be generalized that easily: something like $0 \leq |\mathbf{l}|_1 \leq n$ (or including the shifts of (2.14)) would yield different results. Instead, we define the *sparse grid level* of one particular multi level \mathbf{l} (which may now have $l_m = 0$ for some m) using

$$K(\mathbf{l}) := |\{m \mid l_m = 0\}|, \quad (2.17)$$

$$n(\mathbf{l}) := \sum_{\substack{m=1,\dots,d \\ l_m \neq 0}} (l_m - 1) + 1 = |\mathbf{l}|_1 - (d - K(\mathbf{l})) + 1, \quad (2.18)$$

with the special case

$$n(\mathbf{0}) := 0. \quad (2.19)$$

For inner points, we recover (2.14). For boundary points, we recover (2.14) on the lower dimensional boundary manifold, i.e. if we strip all directions $K(\mathbf{l})$. With this notation, the recursive definition (2.15) of a sparse grid with boundary points is equivalent to

$$\underline{G}_n^d = \bigcup_{0 \leq n(\mathbf{l}) \leq n} W_{\mathbf{l}} =: \bigcup_{\mathbf{l} \in \bar{\mathbf{L}}_1^n} W_{\mathbf{l}} \quad (2.20)$$

with $\bar{\mathbf{L}}_1^n := \{\mathbf{l} \in \mathbb{N}_0^d \mid 0 \leq n(\mathbf{l}) \leq n\}$. Occasionally, we also use an additional hierarchy to distinguish between the two boundary points $(0, 0)$ and $(0, 1)$ by introducing the artificial level -1 with $x_{-1,0} = x_{0,0}$. Thus, level -1 denotes the left boundary and level 0 the right boundary. The equivalent boundary index set for the $-1, 0, 1, 2, \dots$ hierarchy is given by

$$\bar{\mathbf{L}}_1^n := \left\{ \mathbf{l} \in (\mathbb{N} \cup \{-1, 0\})^d \mid \mathbf{l} \in \{-1, 0\}^d \text{ or } \sum_{\substack{m=1, \dots, d \\ l_m > 0}} (l_m - 1) + 1 \leq n \right\} \quad (2.21)$$

where the set $\{-1, 0\}^d$ corresponds to (2.19) and the sum to (2.18).

The Grid Complexity

The complexity of full grids is simply $|\bar{G}_n^d| = (2^n - 1)^d$ for inner grid points and $|\underline{G}_n^d| = (2^n + 1)^d$ including the boundary. Thus, the cost complexity grows exponentially in the number of unknowns required for one coordinate direction. Since approximation quality is usually of the form h^α , the cost/gain ratio degenerates exponentially with d , an observation which is usually referred to as the *curse of dimensionality*.

The complexity of sparse grids is summarized in the following lemma.

Lemma 2.1.1 (Complexity of inner sparse grids). *The complexity of a regular sparse grid without boundary points is*

$$|G_n^d| = (-1)^d + 2^n \sum_{i=0}^{d-1} \binom{n+d-1}{i} (-2)^{d-1-i} = 2^n \left(\frac{n^{d-1}}{(d-1)!} + \mathcal{O}(n^{d-2}) \right) \quad (2.22)$$

where the $\mathcal{O}(\cdot)$ notation covers terms of lower order in n and hides d -dimensional coefficients. Algorithms to compute $|G_n^d|$ should use the recurrence formula

$$|G_n^d| = \sum_{k=0}^{n-1} 2^k |G_{n-i}^{d-1}|; \quad |G_n^1| = 2^n - 1 \quad (2.23)$$

which can be evaluated iteratively in time $\mathcal{O}(dn^2)$ and helper arrays of total size $2n$.

Proof. Both proofs can be found in [BG04]. □

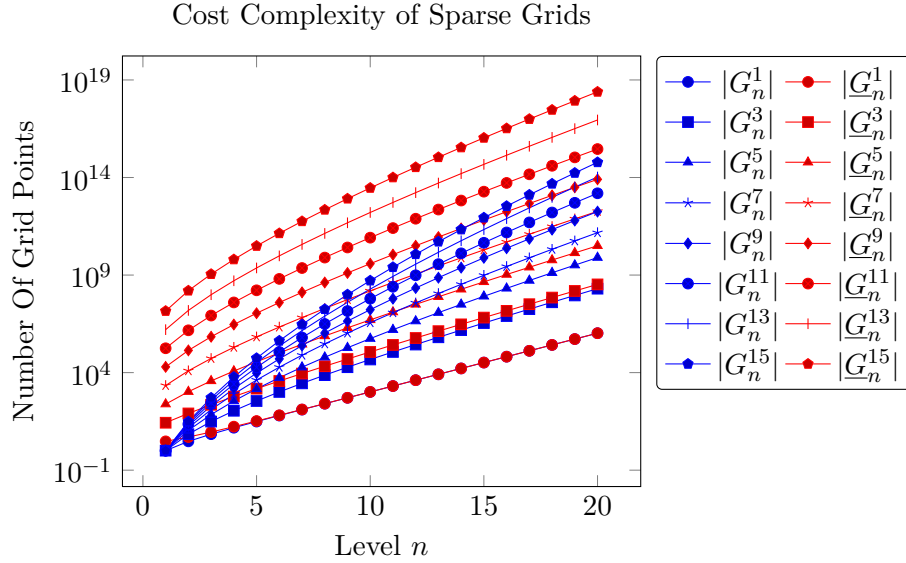


Figure 2.6: The number of grid points for inner sparse grids G_n^d and including boundary points, \underline{G}_n^d , for different dimensions, plotted against the level.

Lemma 2.1.2 (Complexity of boundary sparse grids). *The complexity of a regular sparse grid with boundary points can be bounded by*

$$|\underline{G}_n^d| \leq 3^d |G_n^d| = 3^d \left(2^n \frac{n^{d-1}}{(d-1)!} + \mathcal{O}(n^{d-2}) \right) \quad (2.24)$$

and can be evaluated iteratively using the recurrence formula

$$|\underline{G}_n^d| = 3 \cdot |\underline{G}_n^{d-1}| + \sum_{k=1}^{n-1} 2^k |\underline{G}_{n-k}^{d-1}|; \quad |G_n^1| = 2^n + 1 \quad (2.25)$$

in time $\mathcal{O}(dn^2)$ and helper arrays of total size $2n$.

Proof. It holds

$$|\underline{G}_n^d| = \sum_{j=0}^d \binom{d}{j} 2^{d-j} |G_n^j| \leq |G_n^d| \sum_{j=0}^d \binom{d}{d-j} 2^{d-j} = |G_n^d| \sum_{k=0}^d \binom{d}{k} 2^k = |G_n^d| (1+2)^d$$

since there are precisely $\binom{d}{j}$ boundary manifolds of dimensionality j , and each has $(d-j)$ directions in which it can be positioned either at $x_k = 0$ or $x_k = 1$. The last equalities follow using the symmetry $\binom{n}{k} = \binom{n}{n-k}$ and $(1+x)^d = \sum_{k=0}^d \binom{d}{k} x^k$. This shows (2.24).

The proof for recurrence formula (2.25) can be found in [BG04]. \square

A comparison of full- and sparse grid complexities reveals the potential: sparse grid points have $\mathcal{O}(2^n n^{d-1})$ degrees of freedom in fixed dimension d whereas full grids require $\mathcal{O}(2^{nd})$ degrees of freedom. Interestingly, sparse grids turn out to have almost the

same accuracy if mix smoothness is given as we will see in Section 2.1.2. Note that boundary sparse grids require a factor of about 3^d more grid points compared with inner sparse grids which is depicted in Figure 2.6 for dimensions $d \in \{1, 3, 5, 7, 9, 11, 13, 15\}$ and levels $n = 1, \dots, 20$.

2.1.2 Basic Properties of Sparse Grid Spaces

We are now in a position to attach multi scale bases to our sparse grid constructions. As already motivated, many different choices are possible, and we will encounter some of them in this thesis. Our analysis of approximation properties is focussed on one particular class of bases: the hierarchical piecewise linear spline bases. The simplest piecewise linear spline basis will serve as representative basis for which we present error bounds explicitly, other linear spline bases span (almost) the same spaces and inherit the same properties (for example the prewavelet basis studied in this thesis). We follow our tensor product approach of the last section and define d -dimensional bases as tensor product of one-dimensional ones. In this way, we define regular sparse grids, energy optimal sparse grids and adaptive sparse grids (see also [Gar04] and [Kna00]) for generalized variants.

Notation and Function Spaces

One of the most important aspects of sparse grids are smoothness assumptions: the presence of a sufficient order of *mix* smoothness allows the compression effect. We formalize mix smoothness of second order with respect to L_q by means of the spaces

$$X^{q,2} := \{f: [0, 1]^d \rightarrow \mathbb{R} \mid D^\alpha f \in L_q[0, 1]^d \text{ for } |\alpha|_\infty \leq 2\} \quad (2.26)$$

and

$$X_0^{q,2} := \{f \in X^{q,2} \mid f|_{\partial[0,1]^d} = 0\}. \quad (2.27)$$

Here,

$$D^\alpha f := \frac{\partial^{|\alpha|_1}}{\prod_{m=1}^d \partial x_m^{\alpha_m}} f \quad (2.28)$$

is the derivative taken α_m times in direction m for each $m = 1, \dots, d$. Furthermore, we introduce the seminorms

$$|f|_{\alpha,\infty} := \|D^\alpha\|_{L_\infty}, \quad |f|_{\alpha,2} := \|D^\alpha\|_{L_2}. \quad (2.29)$$

Our discussion will focus on approximation properties with respect to the L_∞ , L_2 and the energy norm

$$\|f\|_E := \left(\int_{[0,1]^d} \sum_{j=1}^d \left(\frac{\partial f(x)}{\partial x_j} \right)^2 dx \right)^{1/2} \quad (2.30)$$

which is equivalent to the H^1 norm in H_0^1 (but only a seminorm on H^1). The term energy norm is related to the finite element framework where $\|\cdot\|_E$ indeed indicates the energy norm.

The Hierarchical Hat Basis

When we refer to the *hierarchical hat basis*, we think of dilations and translations of the one-dimensional hat function

$$\phi(x) := \begin{cases} 1 - |x|, & \text{if } x \in [-1, 1] \\ 0, & \text{otherwise.} \end{cases} \quad (2.31)$$

For mesh width $h_l = 2^{-l}$, we define the hat basis function $\phi_{l,i}$ to be

$$\phi_{l,i}(x) := \phi\left(\frac{x - ih_l}{h_l}\right) \quad (2.32)$$

such that $\phi_{l,i}$ is a piecewise linear spline given by the line segments between $\phi_{l,i}(x_{l,i} - h_l) = 0$, $\phi_{l,i}(x_{l,i}) = 1$ and $\phi_{l,i}(x_{l,i} + h_l) = 0$. The one scale spline basis $\{\phi_{l,i} \mid 0 \leq i \leq 2^l\}$ can be used to approximate functions with bounded second derivative up to $\mathcal{O}(h_l^2)$ where the basis coefficients are just nodal values. For our approximation properties, we use a multi-level splitting of mesh widths h_k , $k = 0, 1, 2, \dots, l$ which spans the same space. Furthermore, it turns out to be of crucial importance to treat boundary conditions separately, just as we did for the derivation of our sparse grid point sets in the preceding section: we derive optimized approximation spaces for the case of homogeneous boundary conditions and apply these results recursively on each lower dimensional boundary manifold to realize non-homogeneous boundary conditions. The resulting point set structure has already been discussed and resembles this strategy.

With this motivation in mind, we analyze approximation properties for the hierarchical hat basis in d dimensions and the case of functions vanishing on $\partial[0, 1]^d$. Consequently, the one-scale spline basis does not need the boundary elements $\phi_{l,0}$ and $\phi_{l,2^l}$ and we get the discrete space

$$V_l := \text{span} \left\{ \phi_{l,i} \mid 1 \leq i \leq 2^l - 1 \right\} \quad (2.33)$$

where we use the *same* notation for the function space V_l and the point set (2.2) containing the underlying basis points. In addition to this nodal basis representation, we introduce the hierarchical basis for one dimension using increments W_l defined by

$$V_l = V_{l-1} \oplus W_l \quad (2.34)$$

which implies

$$W_l = \text{span} \{ \phi_{l,i} \in V_l \mid \phi_{l,i} \notin V_{l-1} \} = \text{span} \{ \phi_{l,i} \mid 1 \leq i \leq 2^l - 1, i \text{ odd} \}. \quad (2.35)$$

The resulting splitting is illustrated in Figure 2.7. Note that the supports of all basis functions $\phi_{l,i}$ for i odd are mutually disjoint. The resulting one-dimensional splitting

$$V_l = \bigoplus_{k=1}^l W_k \quad (2.36)$$

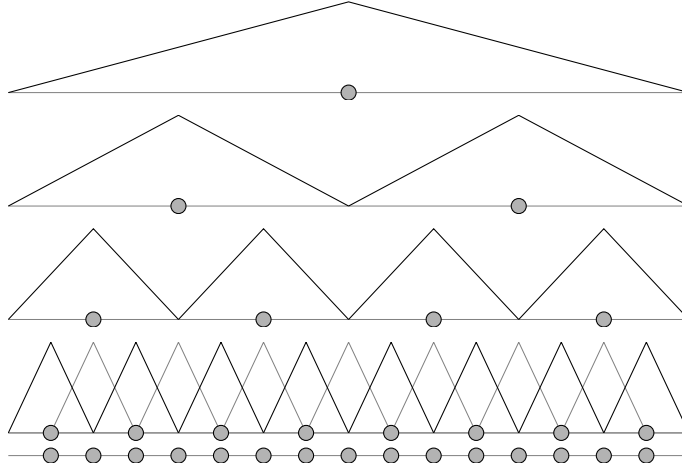


Figure 2.7: The one-dimensional multi-level hat basis (black hats) made up from levels $k = 1, 2, 3, 4$ and the associated one-scale basis (gray hats) on level $l = 4$.

is generalized to the d -dimensional case by means of a tensor product approach: we define the d -dimensional hierarchical basis by multi-levels $\mathbf{l} = (l_1, \dots, l_d)$ and multi-indices $\mathbf{i} = (i_1, \dots, i_d)$ using

$$\phi_{l,\mathbf{i}}(x) := \prod_{m=1}^d \phi_{l_m, i_m}(x_m). \quad (2.37)$$

As immediate consequence, the one-scale basis $\{\phi_{\mathbf{l},\mathbf{i}} \mid \mathbf{l} \leq \mathbf{i} \leq 2^{\mathbf{l}} - 1\}$ constitutes the nodal basis for

$$V_{\mathbf{l}} = \bigotimes_{m=1}^d V_{l_m} = \text{span} \left\{ \phi_{\mathbf{l},\mathbf{i}} \mid \mathbf{l} \leq \mathbf{i} \leq 2^{\mathbf{l}} - 1 \right\}. \quad (2.38)$$

Due to the tensor product, the multiscale splitting becomes

$$V_{\mathbf{l}} = \bigoplus_{\mathbf{1} \leq \mathbf{k} \leq \mathbf{l}} W_{\mathbf{k}} \quad (2.39)$$

where the comparison ' \leq ' is to be understood component-wise as before. As for the one-dimensional cases, the increment spaces are characterized by basis function with odd space indices,

$$W_{\mathbf{l}} = \text{span} \left\{ \phi_{\mathbf{l},\mathbf{i}} \mid \mathbf{l} \leq \mathbf{i} \leq 2^{\mathbf{l}} - 1, i_m \text{ odd} \right\}. \quad (2.40)$$

The increment spaces allow level wise refinements of the mesh width. In particular, the limit

$$V := \sum_{l_1=1}^{\infty} \cdots \sum_{l_d=1}^{\infty} W_{(l_1, \dots, l_d)} = \bigoplus_{\mathbf{l} \in \mathbb{N}^d} W_{\mathbf{l}} \quad (2.41)$$

2 A Sparse Grid Approximation Algebra

exists and yields – up to completion with respect to the H^1 norm – the underlying Sobolev space H_0^1 , i.e. $\bar{V} = H_0^1$, compare [BG04]. As our grid point sets of Section 2.1.1 already indicate, we deal with different finite dimensional subsets of V chosen by particular sets of multi-levels \mathbf{l} . Important choices are the full (or uniform) grid of level $n \in \mathbb{N}$,

$$\bar{G}_n^d := \bigoplus_{\mathbf{l} \in \mathbf{L}_\infty^n} W_{\mathbf{l}} = \bigoplus_{\mathbf{1} \leq |\mathbf{l}|_\infty \leq n} W_{\mathbf{l}} \quad (2.42)$$

and the regular sparse grid of level $n \in \mathbb{N}$,

$$G_n^d := \bigoplus_{\mathbf{l} \in \mathbf{L}_1^n} W_{\mathbf{l}} = \bigoplus_{\mathbf{1} \leq |\mathbf{l} - \mathbf{1}|_1 + 1 \leq n} W_{\mathbf{l}}. \quad (2.43)$$

The complexity of these spaces has already been discussed in Section 2.1.1, so we are interested in their approximation properties. To this end, we consider the interpolation problem of a function $f \in X_0^{q,2}$, i.e. a function whose second mixed derivatives are bounded, $D^2 f \in L_q$, and which satisfies the vanishing boundary conditions. Since $X_0^{q,2} \subset H_0^1 = \bigoplus_{\mathbf{l} \in \mathbb{N}^d} W_{\mathbf{l}}$, we can decompose f uniquely using

$$f = \sum_{\mathbf{l}} f_{\mathbf{l}}, \quad f_{\mathbf{l}} = \sum_{\mathbf{i} \in \mathbf{I}(\mathbf{l})} f_{\mathbf{l},\mathbf{i}} \phi_{\mathbf{l},\mathbf{i}}(x) \in W_{\mathbf{l}}, \quad (2.44)$$

with the index set $\mathbf{I}_{\mathbf{l}} := \{\mathbf{1} \leq \mathbf{i} \leq 2^{l-1}, i_m \text{ odd}\}$ (compare (2.10)). Assuming the finite dimensional space is defined by $\mathbf{L}^h \subset \mathbb{N}^d$ (which might be either \mathbf{L}_∞^n or \mathbf{L}_1^n), the interpolant f^h can be expanded in our finite dimensional basis as

$$f^h = \sum_{\mathbf{l} \in \mathbf{L}^h} f_{\mathbf{l}}^h, \quad f_{\mathbf{l}}^h = \sum_{\mathbf{i} \in \mathbf{I}(\mathbf{l})} f_{\mathbf{l},\mathbf{i}}^h \phi_{\mathbf{l},\mathbf{i}}(x) \in W_{\mathbf{l}}. \quad (2.45)$$

We estimate the error between f and f^h in several steps where we follow [Gri06] and [BG04]. The first step is an analysis of the procedure to obtain coefficients $f_{\mathbf{l},\mathbf{i}}^h$ from nodal values $f(x_{\mathbf{l},\mathbf{i}})$. It turns out that, in fact, $f_{\mathbf{l}}^h = f_{\mathbf{l}}$ due to the nature of the hierarchical hat basis increments $W_{\mathbf{l}}$. Thus, we focus on estimations of $f - f^h = \sum_{\mathbf{l} \notin \mathbf{L}^h} f_{\mathbf{l}}$, i.e. parts which are *not* part of the interpolant. This, in turn, involves estimates on $\|\phi_{\mathbf{l},\mathbf{i}}\|$ and $|f_{\mathbf{l},\mathbf{i}}|$ and combinatorial arguments to count the missing (\mathbf{l}, \mathbf{i}) . As soon as we have bounds for the case of homogeneous boundary conditions, we generalize them to the non-homogeneous case by applying them to each boundary manifold recursively, an approach which is elaborated for the first time in this thesis.

The Transformation From Nodal Values to the Hierarchical Hat Basis

Since basis functions of $W_{\mathbf{l}}$ have mutually disjoint supports and since the $W_{\mathbf{l}}$ are increments which do not contain coarse grid points, a hierarchical hat basis coefficient $f_{\mathbf{l},\mathbf{i}}$ can be computed as difference

$$f_{\mathbf{l},\mathbf{i}} = f(x_{\mathbf{l},\mathbf{i}}) - I_{V_{\mathbf{l}} \ominus W_{\mathbf{l}}}[f](x_{\mathbf{l},\mathbf{i}}) \quad (2.46)$$

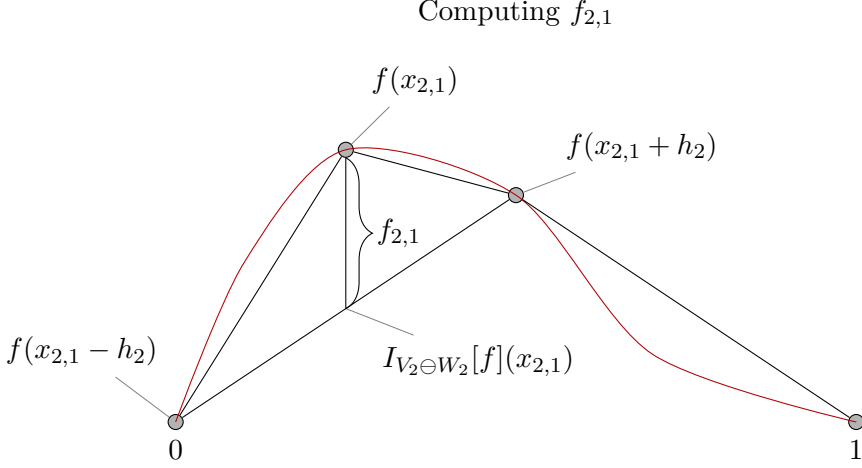


Figure 2.8: Computation of hierarchical hat basis coefficients by means of interpolation.

where $I_{V_1 \ominus W_1}[f]$ is f , interpolated on all grid points of V_1 except for the contributions of W_1 . This is illustrated in Figure 2.8: the figure shows how $f_{2,1}$ can be computed in a one-dimensional interpolation problem. The interpolated value $I_{V_2 \ominus W_2}[f](x_{1,i})$ uses only levels less than 2; it is the same as $I_{V_{l-1}}[f]$ for this one-dimensional example. Note that

$$I_{V_2 \ominus W_2}[f](x_{2,1}) = \frac{1}{2}(f(x_{2,1} - h_2) + f(x_{2,1} + h_2)) = \frac{1}{2}(f(x_{0,0}) + f(x_{1,1})). \quad (2.47)$$

Thus, we only need to compute the mean of adjacent nodal values. The mapping from nodal values to $f_{2,1}$ thus involves a stencil

$$f_{2,1} = \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix} \cdot \begin{bmatrix} f(x_{2,1} - h_2) & f(x_{2,1}) & f(x_{2,1} + h_2) \end{bmatrix}, \quad (2.48)$$

an observation which holds for any one-dimensional coefficient $f_{l,i}$ together with the boundary conditions $f(x_{l,1} - h_l) = 0$ and $f(x_{l,2^l-1} + h_l) = 0$. For $d > 1$, the same reasoning applies as well, we only need to work with a tensor product stencil of the 3^d points $f(x_{1,j})$, $j_m \in \{i_m - 1, i_m, i_m + 1\}$,

$$f_{1,i} = \left(\prod_{m=1}^d \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix}_{x_{l_m, i_m, l_m}} \right) \cdot f =: I_{x_{1,i}, 1} f. \quad (2.49)$$

We need $f_{1,i}$ for every hierarchical coefficient, i.e. points for i_j odd. A key observation is the following: the adjacent nodal values are actually grid points on *lower* levels; no information of higher levels enters at this point (this is different for other bases). This can be seen for $i = 2q + 1$ from $x_{l,i} - h_l = 2^{-l}(i - 1) = x_{l-1,q}$ and $x_{l,i} + h_l = x_{l-1,q+1}$ which are both on lower levels (including the vanishing boundary). As a consequence, higher levels can be added without changing lower order components. We conclude

$$f - f^h = \sum_1 f_l - \sum_{1 \in \mathbf{L}^h} f_1^h = \sum_1 f_l - \sum_{1 \in \mathbf{L}^h} f_1 = \sum_{1 \notin \mathbf{L}^h} f_1, \quad (2.50)$$

a property which we exploit for the following error estimates. More information about the basis transformation (including boundary and implementational aspects) can be found in Appendix A.2.2.

Preparations for Error Estimation

The representation of hierarchical basis coefficients $f_{\mathbf{l},\mathbf{i}}$ as d -dimensional stencil (2.49) leads to an integral representation of $f_{\mathbf{l},\mathbf{i}}$ as follows.

Lemma 2.1.3. *For any coefficient $f_{\mathbf{l},\mathbf{i}}$ of a basis representation for $f = \sum_{(\mathbf{l},\mathbf{i})} f_{\mathbf{l},\mathbf{i}}\phi_{\mathbf{l},\mathbf{i}}$ with vanishing boundaries, $f \in X_0^{q,2}$, the following relation holds:*

$$f_{\mathbf{l},\mathbf{i}} = \prod_{m=1}^d (-2^{-(l_m+1)}) \int_{[0,1]^d} \phi_{\mathbf{l},\mathbf{i}} \cdot D^2 f(x) \, dx. \quad (2.51)$$

Proof. Following [Gri06], we write $\psi_{l_m, i_m}(x_m) := -2^{-(l_m+1)}\phi_{l_m, i_m}(x_m)$. Furthermore, we start with the simplest case $d = 1$ from which the proof follows using tensor product arguments. Partial integration provides, together with $\psi_{l,i}(x_{l,i} \pm h_l) = 0$,

$$\begin{aligned} \int \psi_{l,i} \cdot \frac{\partial^2}{\partial x^2} f(x) \, dx &= \int_{x_{l,i}-h_l}^{x_{l,i}+h_l} \psi_{l,i}(x) \frac{\partial^2}{\partial x^2} f(x) \, dx \\ &= - \int_{x_{l,i}-h_l}^{x_{l,i}+h_l} \frac{\partial}{\partial x} \psi_{l,i}(x) \cdot \frac{\partial}{\partial x} f(x) \, dx = \int_{x_{l,i}-h_l}^{x_{l,i}} \frac{1}{2} \frac{\partial}{\partial x} f(x) \, dx - \int_{x_{l,i}}^{x_{l,i}+h_l} \frac{1}{2} \frac{\partial}{\partial x} f(x) \, dx \\ &= \frac{1}{2} (f(x_{l,i}) - f(x_{l,i} - h_l)) - \frac{1}{2} (f(x_{l,i} + h_l) - f(x_{l,i})) = I_{x_{l,i}, l} f. \end{aligned}$$

The d -dimensional case follows since $\phi_{\mathbf{l},\mathbf{i}}$ and D^2 are both of product type. \square

We have thus expressed our hierarchical coefficients by means of the second mixed derivative of the approximated function f . In the following, we summarize bounds on the hierarchical coefficients, the basis functions $\phi_{\mathbf{l},\mathbf{i}}$ and finally error bounds for functions with truncated basis expansion.

Lemma 2.1.4. *Any inner basis function $\phi_{\mathbf{l},\mathbf{i}}$ of the hierarchical hat basis yields the norm values*

$$\|\phi_{\mathbf{l},\mathbf{i}}\|_{\infty} = 1, \quad (2.52)$$

$$\|\phi_{\mathbf{l},\mathbf{i}}\|_p = \left(\frac{2}{p+1}\right)^{d/p} \cdot 2^{-|\mathbf{l}|_1/p}, \quad p \geq 1, \quad (2.53)$$

$$\|\phi_{\mathbf{l},\mathbf{i}}\|_E = \sqrt{2} \left(\frac{2}{3}\right)^{(d-1)/2} \cdot 2^{-|\mathbf{l}|_1/2} \cdot \left(\sum_{m=1}^d 2^{2l_m}\right)^{1/2}. \quad (2.54)$$

2.1 Sparse Grid Approximation

Proof. All equalities result from straightforward calculations based on the definition of $\phi_{1,i}$. We refer to [BG99] for details. \square

The integral representation (2.51) of $f_{1,i}$ combined with Lemma 2.1.4 allows the following bounds on $|f_{1,i}|$.

Lemma 2.1.5. *Let $f \in X_0^{q,2}$ be given in its hierarchical representation $f = \sum_{(1,i)} f_{1,i} \phi_{1,i}$. Then, the following estimates on $|f_{1,i}|$ hold:*

$$|f_{1,i}| \leq 2^{-d} \cdot 2^{-2|l_1|} \cdot |f|_{\mathbf{2},\infty} \quad (2.55)$$

$$|f_{1,i}| \leq 2^{-d} \cdot \left(\frac{2}{3}\right)^{d/2} 2^{-3/2 \cdot |l_1|} \cdot |f|_{\text{supp } \phi_{1,i}}|_{\mathbf{2},2} \quad (2.56)$$

where $\text{supp } \phi_{1,i}$ denotes the support of $\phi_{1,i}$.

Proof. Following [BG04], we apply the Hölder inequality to (2.51) to get

$$\begin{aligned} |f_{1,i}| &= \left| \prod_{m=1}^d (-2^{-(l_m+1)}) \right| \cdot \left| \int_{[0,1]^d} \phi_{1,i}(x) \cdot D^{\mathbf{2}} f(x) \, dx \right| \\ &= 2^{-|l_1|} \cdot 2^{-d} \cdot \left| \int_{[0,1]^d} \phi_{1,i}(x) \cdot D^{\mathbf{2}} f(x) \, dx \right| \\ &\leq 2^{-|l_1|} \cdot 2^{-d} \cdot \|\phi_{1,i}\|_1 \cdot |f|_{\text{supp } \phi_{1,i}}|_{\mathbf{2},\infty} \leq 2^{-d} \cdot 2^{-2|l_1|} \cdot |f|_{\mathbf{2},\infty}. \end{aligned}$$

The second bound follows by the Cauchy Schwartz inequality and the definition of $|f|_{\mathbf{2},2} := \|D^{\mathbf{2}} f\|_{L_2}$:

$$|f_{1,i}| \leq 2^{-|l_1|} \cdot 2^{-d} \cdot \|\phi_{1,i}\|_2 |f|_{\text{supp } \phi_{1,i}}|_{\mathbf{2},2} = 2^{-d} \left(\frac{2}{3}\right)^{d/2} 2^{-3/2 \cdot |l_1|} |f|_{\text{supp } \phi_{1,i}}|_{\mathbf{2},2}$$

where we use Lemma 2.1.4 with $p = 2$. \square

Since a function $f \in X_0^{q,2}$ can be represented by contributions of increment spaces $f = \sum_1 f_1$, we combine results of the last set of lemma to get bounds on single f_1 .

Lemma 2.1.6. *Let $f \in X_0^{q,2}$ be given in its hierarchical hat basis representation of the form $f = \sum_1 f_1$ with $f_1 \in W_1$. Then, the contribution of f_1 can be bounded as follows:*

$$\|f_1\|_{\infty} \leq 2^{-d} 2^{-2|l_1|} |f|_{\mathbf{2},\infty}, \quad (2.57)$$

$$\|f_1\|_2 \leq 3^{-d} 2^{-2|l_1|} |f|_{\mathbf{2},2}, \quad (2.58)$$

$$\|f_1\|_E \leq \frac{1}{2 \cdot 12^{(d-1)/2}} 2^{-2|l_1|} \left(\sum_{m=1}^d 2^{2l_m} \right)^{1/2} |f|_{\mathbf{2},\infty}, \quad (2.59)$$

$$\|f_1\|_E \leq \sqrt{3} \cdot 3^{-d} 2^{-2|l_1|} \left(\sum_{m=1}^d 2^{2l_m} \right)^{1/2} |f|_{\mathbf{2},2}. \quad (2.60)$$

Proof. Following the reasoning in [BG04], we find

$$\|f_1\|_\infty = \left\| \sum_{\mathbf{i}} f_{1,\mathbf{i}} \phi_{1,\mathbf{i}} \right\|_\infty = \max_{\mathbf{i}} |f_{1,\mathbf{i}}| \|\phi_{1,\mathbf{i}}\|_\infty \quad (2.61)$$

since the supports of all $\phi_{1,\mathbf{i}}$ are mutually disjoint. Thus, we find the first inequality as consequence of Lemma 2.1.4 and Lemma 2.1.5. The mutually disjoint supports of $\phi_{1,\mathbf{i}}$ allow a similar simplification for the L_2 estimate,

$$\|f_1\|_2^2 = \sum_{\mathbf{i}} |f_{1,\mathbf{i}}|^2 \|\phi_{1,\mathbf{i}}\|_2^2 \leq \frac{1}{6^d} 2^{-3|\mathbf{l}_1|} \cdot \left(\frac{2}{3}\right)^d 2^{-|\mathbf{l}_1|} \cdot \sum_{\mathbf{i}} |f|_{\text{supp } \phi_{1,\mathbf{i}}}^2_{2,2} = 9^{-d} 2^{-4|\mathbf{l}_1|} \cdot |f|_{2,2}^2,$$

again by means of Lemma (2.1.4) and Lemma (2.1.5). The same reasoning yields the two bounds on $\|f_1\|_E$, again with $\sum_{\mathbf{i}} |f|_{\text{supp } \phi_{1,\mathbf{i}}}^2_{2,2} = |f|_{2,2}^2$ and with $\sum_{\mathbf{i}} |f|_{2,\infty} \leq 2^{|\mathbf{l}_1|} 2^{-d} |f|_{2,\infty}$, respectively. \square

2.1.3 Interpolation Error Bounds for Full and Sparse Grids

Having seen how each single component f_1 contributes to the (infinite) expansion $f = \sum_{\mathbf{l} \in \mathbb{N}^d} f_{\mathbf{l}}$ for a function with vanishing boundary conditions and mix regularity, $f \in X_0^{q,2}$, we are now interested in the error made by truncating the expansion to $f^h = \sum_{\mathbf{l} \in \mathbf{L}^h} f_{\mathbf{l}}$. We are mainly interested in the finite subspaces for full grids, $\mathbf{L}_\infty^n = \{1 \leq |\mathbf{l}|_\infty \leq n\}$, and regular sparse grids, $\mathbf{L}_1^n = \{1 \leq |\mathbf{l}|_1 - d + 1 \leq n\}$. A key observation here is that truncation of a given expansion of f to a finite subset is actually the same as interpolation in this finite subspace due to the definition of hierarchical coefficients, (2.50).

Lemma 2.1.7 (Interpolation error of full grids). *For any function with bounded second mixed derivatives and homogeneous boundary conditions, $f \in X_0^{q,2}$, the following estimates for the interpolation error $f - f^{n,\infty}$, $f^{n,\infty} \in \bar{G}_n^d$, hold:*

$$\|f - f^{n,\infty}\|_\infty \leq \frac{d}{6^d} 2^{-2n} |f|_{2,\infty} = \mathcal{O}(h_n^2), \quad (2.62)$$

$$\|f - f^{n,\infty}\|_2 \leq \frac{d}{9^d} 2^{-2n} |f|_{2,2} = \mathcal{O}(h_n^2), \quad (2.63)$$

$$\|f - f^{n,\infty}\|_E \leq \frac{d^{3/2}}{2 \cdot 3^{(d-1)/2} \cdot 6^{d-1}} \cdot 2^{-n} |f|_{2,\infty} = \mathcal{O}(h_n), \quad (2.64)$$

$$\|f - f^{n,\infty}\|_E \leq \frac{d^{3/2}}{\sqrt{3} \cdot 9^{d-1}} \cdot 2^{-n} |f|_{2,2} = \mathcal{O}(h_n). \quad (2.65)$$

Proof. We have the decompositions $f = \sum_{\mathbf{l} \in \mathbb{N}^d} f_{\mathbf{l}}$, $f_{\mathbf{l}} \in W_{\mathbf{l}}$ and $f^{n,\infty} = \sum_{1 \leq |\mathbf{l}|_\infty \leq n} f_{\mathbf{l}}^n$, $f_{\mathbf{l}}^n \in W_{\mathbf{l}}$, respectively. Furthermore, we know $f_{\mathbf{l}}^n = f_{\mathbf{l}}$ for the hierarchical hat basis. Thus, we have for any norm $\|\cdot\|$

$$\|f - f^{n,\infty}\| = \left\| \sum_{\mathbf{l} \notin \mathbf{L}_\infty^n} f_{\mathbf{l}} \right\| = \left\| \sum_{|\mathbf{l}|_\infty > n} f_{\mathbf{l}} \right\| \leq \sum_{|\mathbf{l}|_\infty > n} \|f_{\mathbf{l}}\|. \quad (2.66)$$

We conclude

$$\|f - f^{n,\infty}\|_\infty \leq 2^{-d} |f|_{\mathbf{2},\infty} \cdot \sum_{|\mathbf{l}|_\infty > n} 2^{-2|\mathbf{l}|_1}, \quad (2.67)$$

$$\|f - f^{n,\infty}\|_2 \leq 3^{-d} |f|_{\mathbf{2},2} \cdot \sum_{|\mathbf{l}|_\infty > n} 2^{-2|\mathbf{l}|_1}, \quad (2.68)$$

$$\|f - f^{n,\infty}\|_E \leq \frac{1}{2 \cdot 12^{(d-1)/2}} |f|_{\mathbf{2},\infty} \sum_{|\mathbf{l}|_\infty > n} 2^{-2|\mathbf{l}|_1} \left(\sum_{j=1}^d 2^{2l_j} \right)^{1/2}, \quad (2.69)$$

$$\|f - f^{n,\infty}\|_E \leq \sqrt{3} \cdot 3^{-d} |f|_{\mathbf{2},2} \sum_{|\mathbf{l}|_\infty > n} 2^{-2|\mathbf{l}|_1} \left(\sum_{j=1}^d 2^{2l_j} \right)^{1/2}. \quad (2.70)$$

We focus on the level dependent sums and get using geometric series arguments [BG04]

$$\begin{aligned} \sum_{|\mathbf{l}|_\infty > n} 2^{-2|\mathbf{l}|_1} &= \sum_1 4^{-|\mathbf{l}|_1} - \sum_{|\mathbf{l}|_\infty \leq n} 4^{-|\mathbf{l}|_1} = 3^{-d} - \left(\sum_{l_1 \leq n} 4^{-l_1} \right)^d \\ &= 3^{-d} - (3^{-1}(1 - 4^{-n}))^d = 3^{-d}(1 - (1 - 4^{-n})^d) \\ &\leq 3^{-d}(1 - (1 - d4^{-n})) = 3^{-d} \cdot d \cdot 4^{-n} \end{aligned} \quad (2.71)$$

and

$$\begin{aligned} \sum_{|\mathbf{l}|_\infty > n} 2^{-2|\mathbf{l}|_1} \left(\sum_{j=1}^d 2^{2l_j} \right)^{1/2} &\leq \sqrt{d} \sum_{|\mathbf{l}|_\infty > n} 2^{-2|\mathbf{l}|_1} \max_j 2^{l_j} \leq d^{3/2} \cdot \sum_{|\mathbf{l}|_\infty = l_1 > n} 2^{-2|\mathbf{l}|_1} 2^{l_1} \\ &= d^{3/2} \cdot \sum_{l_1 > n} 2^{-l_1} \left(\sum_{l_j=1}^{l_1} 4^{-l_j} \right)^{d-1} \leq d^{3/2} \frac{1}{3^{d-1}} 2^{-n}. \end{aligned} \quad (2.72)$$

Plugging (2.71) into (2.67) and (2.68) and the second result (2.72) into the two energy estimate (2.69) and (2.70) completes the proof. \square

The interpolation error estimate for sparse grid spaces follows with similar arguments, but the reminder term associated with $|\mathbf{l}|_1 > n + d - 1$ involves complicated binomial coefficients. We summarize a combinatorial intermediate result from [BG04] which bounds the associated coefficients. To this end, let

$$A(d, n) := \sum_{k=0}^{d-1} \binom{n+d-1}{k} = \frac{n^{d-1}}{(d-1)!} + \mathcal{O}(n^{d-2}) \quad (2.73)$$

where the $\mathcal{O}(\cdot)$ notation hides d dependent coefficients of the lower order components of n .

Lemma 2.1.8. *It holds for arbitrary $s \in \mathbb{N}$*

$$\sum_{|\mathbf{l}|_1 > n+d-1} 2^{-s|\mathbf{l}|_1} = 2^{-sn} \cdot 2^{-s \cdot d} \sum_{i=0}^{\infty} 2^{-si} \binom{n+i+d-1}{d-1} \leq 2^{-sn} \cdot 2^{-sd} \cdot 2 \cdot A(d, n).$$

2 A Sparse Grid Approximation Algebra

Proof. The proof involves elementary combinatorial and analytical arguments, we refer the interested reader to [BG04, Lemma 3.7]. \square

Lemma 2.1.9 (Interpolation error bounds for regular sparse grids). *The sparse grid interpolant $f^{n,1}$ of a function $f \in X_0^{q,2}$ exhibits the following errors:*

$$\|f - f^{n,1}\|_\infty \leq \frac{2}{8^d} \cdot |f|_{\mathbf{2},\infty} \cdot 2^{-2n} A(d, n) = \mathcal{O}(h_n^2 n^{d-1}), \quad (2.74)$$

$$\|f - f^{n,1}\|_2 \leq \frac{2}{12^d} \cdot |f|_{\mathbf{2},2} \cdot 2^{-2n} A(d, n) = \mathcal{O}(h_n^2 n^{d-1}), \quad (2.75)$$

$$\|f - f^{n,1}\|_E \leq \frac{d|f|_{\mathbf{2},\infty}}{2 \cdot 3^{(d-1)/2} \cdot 4^{d-1}} 2^{-n} = \mathcal{O}(h_n), \quad (2.76)$$

$$\|f - f^{n,1}\|_E \leq \frac{d|f|_{\mathbf{2},2}}{\sqrt{3} \cdot 6^{d-1}} 2^{-n} = \mathcal{O}(h_n). \quad (2.77)$$

Proof. It holds

$$\|f - f^{n,1}\|_\infty \leq \sum_{|\mathbf{l}| > n+d-1} \|f_{\mathbf{l}}\|_\infty \leq \frac{|f|_{\mathbf{2},\infty}}{2^d} \cdot \sum_{|\mathbf{l}| > n+d-1} 2^{-2|\mathbf{l}|} \leq \frac{2|f|_{\mathbf{2},\infty}}{8^d} 2^{-2n} A(d, n)$$

where we used Lemma 2.1.8 with $s = 2$. The L_2 norm estimate can be obtained analogously. The energy norm error can be obtained as for the full grid case with the special term

$$\sum_{|\mathbf{l}| > n+d-1} 2^{-2|\mathbf{l}|} \left(\sum_{j=1}^d 2^{2l_j} \right)^{1/2} = \sum_{i=n+d}^{\infty} 4^{-i} \cdot \sum_{|\mathbf{l}|=i} \left(\sum_{j=1}^d 4^{l_j} \right)^{1/2} \leq \sum_{i=n+d}^{\infty} d 2^{-i} \quad (2.78)$$

since $\sum_{|\mathbf{l}|=i} \left(\sum_{j=1}^d 4^{l_j} \right)^{1/2} \leq d 2^i$ which can be proved by complete induction with respect to d (compare [BG04]). \square

A direct comparison of full- and sparse grids reveals the qualitative improvement with respect to the cost/gain ratio: full grids requires $\mathcal{O}(2^{nd})$ points and yield an approximation order of $\mathcal{O}(2^{-2n})$ with respect to L_2 and L_∞ . Thus, the cost grows exponentially with d while its benefit stays of the same order – a severe form of the “curse of dimensionality”. Sparse grids require $\mathcal{O}(2^n n^{d-1})$ degrees of freedom and provide almost the same approximation order, namely $\mathcal{O}(2^{-2n} n^{d-1})$ for L_2 and L_∞ , so the exponential dependency of d is reduced significantly. Note that sparse grids have been derived to be optimal with respect to this cost/benefit ratio approach when the approximant is in $X_0^{q,2}$, see [BG04] for details. Improvements are only possible for approximation in other norms. A variant of sparse grids which is optimal with respect to the energy norm and which exhibits even better cost/benefit ratios with respect to d has been proposed in [Bun98]. We summarize its definition and properties here.

Definition 2.1.1 (Energy Sparse Grids). The finite dimensional subspace splitting

$$G_n^{d,E} := \bigoplus_{\mathbf{l} \in \mathbf{L}_E^n} W_{\mathbf{l}} \quad (2.79)$$

with $n \in \mathbb{N}$ and

$$\mathbf{L}_E^n := \{\mathbf{l} \in \mathbb{N}^d \mid \|\mathbf{l}\|_1 - \frac{1}{5} \text{ld}(\sum_j 4^{l_j}) \leq n + d - 1 - \frac{1}{5} \text{ld}(4^n + 4d - 4)\} \quad (2.80)$$

is called the energy sparse grid of level n in dimension d .

Lemma 2.1.10. *The grid for $G_n^{d,E}$ has*

$$|G_n^{d,E}| \leq 2^n \frac{d}{2} (1 - 2^{-2/3})^{-d} \leq 2^n \frac{d}{2} e^d = \mathcal{O}(2^n) \quad (2.81)$$

degrees of freedom.

Proof. See [Gri06]. □

Lemma 2.1.11. *The energy norm of the interpolation error for $f \in X_0^{q,2}$ in the energy sparse grid $G_n^{d,E}$ is bounded by*

$$\|f - f^{n,E}\|_E \leq \frac{d|f|_{\mathbf{2},\infty}}{3^{(d-1)/2} \cdot 4^{d-1}} \cdot \left(\frac{1}{2} + \left(\frac{5}{2}\right)^{d-1}\right) 2^{-n} = \mathcal{O}(h_n) \quad (2.82)$$

$$\|f - f^{n,E}\|_E \leq \frac{2d|f|_{\mathbf{2},2}}{\sqrt{3} \cdot 6^{d-1}} \cdot \left(\frac{1}{2} + \left(\frac{5}{2}\right)^{d-1}\right) 2^{-n} = \mathcal{O}(h_n). \quad (2.83)$$

Furthermore, it exhibits the optimal cost/gain ratio with respect to the energy norm and approximation of functions $f \in X_0^{q,2}$.

The error bound for energy optimal sparse grids is thus the same as the one for regular sparse grids, up to a factor $2^{(1/2 + (5/2)^{d-1})}$ (compare (2.82) with (2.76) and (2.83) with (2.77)).

Proof. The proof can be found in [BG04] or [Gri06]; the optimality follows from the construction method of $G_n^{d,E}$, compare [BG04]. □

2.1.4 Error Bounds for Non-Homogeneous Boundary Conditions

We finally derive error bounds for the case of non-homogeneous boundary conditions. These bounds are obtained using a dimension-recursive approach: a d -dimensional function is decomposed into separate components defined on left and right boundaries and inner parts of $[0, 1]^d$. On a boundary, function components are effectively of lower dimension. Furthermore, the decomposition is built such that each term of the decomposition has vanishing boundary conditions in all directions in which it varies. In two dimensions, we find a component which varies in (x_1, x_2) , so it vanishes for $x_1 \in \{0, 1\}$ or for $x_2 \in \{0, 1\}$. A two-dimensional function also has one-dimensional components: one placed on $x_2 = 0$ which varies only in x_1 (and vanishes for $x_1 \in \{0, 1\}$), one for $x_2 = 1$ which also varies only in x_1 and, analogously, components with x_1 fixed to either $x_1 = 0$ or $x_1 = 1$. The details of this decomposition, the required smoothness assumptions and the resulting error estimate have – to the knowledge of the author – not been published before and follow ideas motivated in [Gri06].

The Hierarchical Basis on the Boundary

As a first step, we define the hierarchical basis on the boundary. Two choices are possible, namely to extend the inner hierarchical basis by the two one-dimensional nodal basis functions $\phi_{0,0}(x) = 1 - x$ and $\phi_{0,1}(x) = x$ on level $l = 0$ or using the hierarchical approach with $\phi_{0,0}(x) = 1$ and $\phi_{0,1}(x) = x$. The nodal boundary basis is useful for partial differential equations where one likes to discard boundary values whenever possible. The hierarchical boundary approach is useful to estimate errors and to perform dimension adaptive refinement as in Chapter 4. Our following discussion is based on the hierarchical approach, so we defer the modifications arising for the nodal approach to Appendix A.2.2.

To define the hierarchical boundary basis, we introduce a further, artificial level $l = -1$ to emphasise the hierarchical structure. The only basis function on level $l = -1$ is the constant, $\phi_{-1,0}(x) := 1$ which we attach to the left boundary point $x = 0$. Note that $x_{-1,0} = 0 \cdot 2^1 = 0$ and also $x_{0,0} = 0 \cdot 2^0 = 0$ provide valid multi-level descriptions of this coordinate. The next level in our hierarchy is defined by $\phi_{0,1}(x) := x$ which is attached to the right boundary; it is the only basis function on level 0. All basis functions on level $l \geq 1$ remain unchanged. The hierarchical coefficients of an expansion $f = \sum_{l=-1}^{\infty} \sum_{i \in I(l)} f_{l,i} \phi_{l,i}$ (if it exists) are determined by $f_{-1,0} = f(0)$, and $f_{l,i} = f(x_{l,i}) - I_{V_i \in W_l}[f](x_{l,i})$ for $l \geq 0$ as before. Now, the interpolation operator $I_{V_i \in W_l}[f] = I_{V_{i-1}}[f]$ incorporates the two boundary levels -1 and 0 as well. For example, we find $f_{0,1} = f(x_{0,1}) - I_{V_{-1}}[f](x_{0,1}) = f(1) - f_{-1,0}$. As before, the basis coefficients depend only on lower levels and we have with $W_{-1} := \text{span}\{\phi_{-1,0}\}$ and $W_0 := \text{span}\{\phi_{0,1}\}$ the splitting

$$V_l = \text{span}\{\phi_{l,i} \mid 0 \leq i \leq 2^l\} = \bigoplus_{k=-1}^l W_k. \quad (2.84)$$

The d -dimensional case follows using the tensor product construction as before and we find

$$V := \bigoplus_{\mathbf{l} \in (\mathbb{N} \cup \{-1,0\})^d} W_{\mathbf{l}} = H^1 \quad (2.85)$$

up to completion. Thus, any function in V can be uniquely represented by $f = \sum_{\mathbf{l} \in (\mathbb{N} \cup \{-1,0\})^d} f_{\mathbf{l}}, f_{\mathbf{l}} \in W_{\mathbf{l}}$ where $f_{-1,\dots,-1}$ is just the constant $f(0, \dots, 0)$.

A Dimension-Wise Three-Term-Boundary Decomposition

The key idea for the treatment of boundary conditions is now to look at the three cases $l = -1$, $l = 0$ and $l > 0$ in each component. From our multi-level representation, we conclude the following statement.

Lemma 2.1.12. *Let $\widetilde{W}^{(1)} := \bigoplus_{l>0} W_l$ with $W_l = \text{span}\{\phi_{l,i} \mid i \in I(l)\}$ be the one-dimensional subspaces spanned only by inner basis functions. Then, $H^1[0, 1] = \widetilde{V}^{(1)}$ with*

$$V^{(1)} = \mathbf{1} \oplus \text{lin} \oplus \widetilde{W}^{(1)} \quad (2.86)$$

where $\mathbf{1} = \text{span}\{1\} = \text{span}\{\phi_{-1,0}\}$ and $\text{lin} = \text{span}\{x\} = \text{span}\{\phi_{0,1}\}$.

2.1 Sparse Grid Approximation

Furthermore, $H^1[0, 1]^d = \bar{V}$ where V can be written as

$$V = \bigotimes_{m=1}^d (\mathbf{1}_m \oplus \text{lin}_m \oplus \widetilde{W}_m) = \bigoplus_{\mathbf{l} \in \{-1, 0, *\}^d} \bar{W}_{\mathbf{l}} \quad (2.87)$$

with decomposition terms

$$\bar{W}_{\mathbf{l}} := \bigotimes_{\mathbf{l}^{(-1)} := \{j | l_j = -1\}} \mathbf{1}_j \otimes \bigotimes_{\mathbf{l}^{(0)} := \{j | l_j = 0\}} \text{lin}_j \otimes \bigotimes_{\mathbf{l}^{(*)} := \{j | l_j = *\}} \widetilde{W}_j. \quad (2.88)$$

Here, the subscript j in $\mathbf{1}_j$ and lin_j merely indicates the respective coordinate direction.

Proof. The one-dimensional splitting follows immediately from the basis representation and the d -dimensional variant is its tensor product version consisting of all 3^d possible combinations. \square

Definition 2.1.2. Let $f \in V_h^{(d)}$ be given in hierarchical basis representation

$$f^h = \sum_{\bar{\mathbf{l}} \in \mathbf{L}} \sum_{\mathbf{i} \in \mathbf{I}(\bar{\mathbf{l}})} f_{\bar{\mathbf{l}}, \mathbf{i}} \phi_{\bar{\mathbf{l}}, \mathbf{i}} = \sum_{\bar{\mathbf{l}} \in \mathbf{L}} f_{\bar{\mathbf{l}}}. \quad (2.89)$$

Furthermore, let $\mathbf{l} \in \{-1, 0, *\}^d$ be a description as we used it before (-1 means constant part, 0 means linear part and $*$ means varying part).

Then, we define the subset of hierarchical coefficients based on the split description $\mathbf{l} \in \{-1, 0, *\}^d$ by associating $*$ with $l > 0$, -1 with $l = -1$ and 0 with $l = 0$. We define

$$\bar{f}_{\mathbf{l}}(x) := \sum_{\substack{(\tilde{l}_1, \dots, \tilde{l}_d) \in \mathbf{L} \\ l_j = -1 \Rightarrow \tilde{l}_j = -1 \\ l_j = 0 \Rightarrow \tilde{l}_j = 0 \\ l_j = * \Rightarrow \tilde{l}_j > 0}} \sum_{\mathbf{i} \in \mathbf{I}(\tilde{\mathbf{l}})} f_{\tilde{\mathbf{l}}, \mathbf{i}} \phi_{\tilde{\mathbf{l}}, \mathbf{i}}(x) \quad (2.90)$$

$$= \prod_{j \in \mathbf{l}^{(0)}} x_j \cdot \sum_{\substack{(\tilde{l}_1, \dots, \tilde{l}_d) \in \mathbf{L} \\ l_j = -1 \Rightarrow \tilde{l}_j = -1 \\ l_j = 0 \Rightarrow \tilde{l}_j = 0 \\ l_j = * \Rightarrow \tilde{l}_j > 0}} \sum_{\mathbf{i} \in \mathbf{I}(\tilde{\mathbf{l}})} f_{\tilde{\mathbf{l}}, \mathbf{i}} \prod_{j \in \mathbf{l}^{(*)}} \phi_{\tilde{l}_j, i_j}(x_j) \quad (2.91)$$

$$=: \prod_{j \in \mathbf{l}^{(0)}} x_j \cdot f_{\mathbf{l}, *}(x_{\mathbf{l}^{(*)}}). \quad (2.92)$$

Thus, one $\bar{f}_{\mathbf{l}}$ can contain many $f_{\tilde{\mathbf{l}}}$ of (2.89) since $l_j = *$ includes all $\tilde{l}_j > 0$. The $\bar{f}_{\mathbf{l}}$ is a coarse-graining; there are only 3^d possible choices of the $\{-1, 0, *\}^d$.

Lemma 2.1.13. Any function $f \in V \subseteq H^1[0, 1]^d$ can be split uniquely into

$$f(x) = \sum_{\mathbf{l} \in \{-1, 0, *\}^d} \bar{f}_{\mathbf{l}}(x), \quad \bar{f}_{\mathbf{l}} \in \bar{W}_{\mathbf{l}} \quad (2.93)$$

2 A Sparse Grid Approximation Algebra

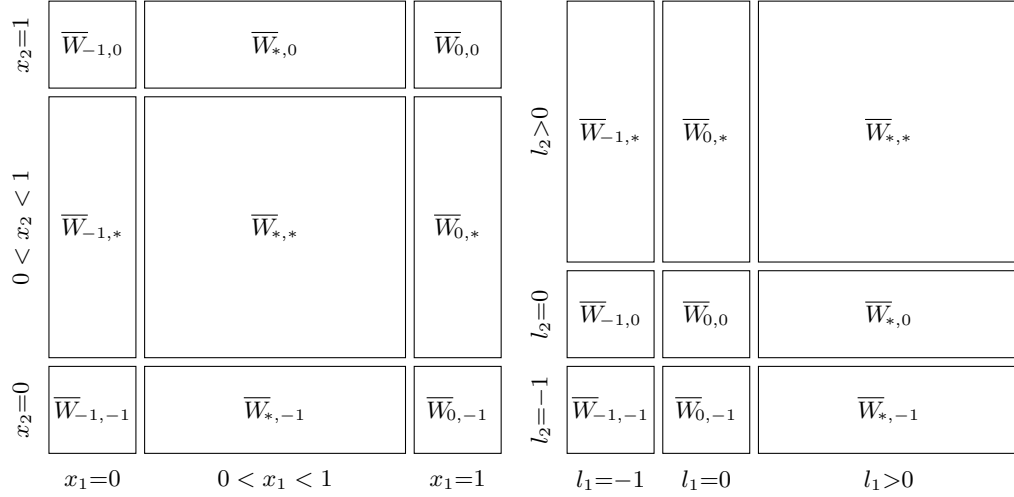


Figure 2.9: Illustration of splitting (2.88) into boundary and inner components W_{l_1, l_2} for $d = 2$ in nodal positioning, i.e. sorted by x_1 / x_2 , (left) and in hierarchical positioning, i.e sorted by l_1 / l_2 (right).

such that each element consists of inner and linear contributions

$$\bar{f}_{\mathbf{1}}(x) = f_{\mathbf{1},*}(x_{\mathbf{1}^{(*)}}) \cdot \prod_{j \in \mathbf{1}^{(0)}} x_j. \quad (2.94)$$

Here, the uniquely determined $f_{\mathbf{1},*}$ is a $|\mathbf{1}^{(*)}|$ dimensional function satisfying the homogeneous boundary condition $x_j \in \{0, 1\} \Rightarrow f_{\mathbf{1},*}(x_{\mathbf{1}^{(*)}}) = 0$ for any $j \in \mathbf{1}^{(*)}$. Thus, $\bar{f}_{\mathbf{1}}$ does not depend on x_j for $j \in \mathbf{1}^{(-1)}$ and it is linear in directions $j \in \mathbf{1}^{(0)}$ whereas the dynamics in the remaining directions is governed by $f_{\mathbf{1},*}: [0, 1]^{|\mathbf{1}^{(*)}|} \rightarrow \mathbb{R}$. Note that $|\mathbf{1}^{(*)}| = 0$ implies that $f_{\mathbf{1},*}$ is a constant, compare (2.91).

Proof. The splitting follows from Lemma 2.1.12 and the boundary conditions follow since

$$\bigotimes_{j \in \mathbf{1}^{(*)}} \widetilde{W}_j = \text{span} \left\{ \prod_{j \in \mathbf{1}^{(*)}} \phi_{l_j, i_j} \mid l_j > 0, i_j \in I_j(l_j) \right\} \quad (2.95)$$

where every single basis function satisfies $x_j \in \{0, 1\} \Rightarrow \phi_{l_j, i_j}(x_j) = 0$ for any $j \in \mathbf{1}^{(*)}$. \square

The splitting is illustrated in Figure 2.9 for the case $d = 2$: a total of $3^d = 9$ different components $\bar{W}_{\mathbf{1}}$ are shown, once sorted by their coordinates x_1 and x_2 in Figure 2.9 (left) and once sorted by their levels l_i according to $-1 < 0 < *$ in Figure 2.9 (right). The component $\bar{W}_{*,*}$ contains only two-dimensional inner contributions while $\bar{W}_{*,-1}$ and $\bar{W}_{*,0}$ contain one-dimensional contributions with varying x_1 direction and fixed x_2 direction (fixed in the sense that associated functions are either constant or linear with respect to x_2). Similarly, $\bar{W}_{-1,*}$ and $\bar{W}_{0,*}$ are constant and linear with respect to

x_1 , respectively, and vary in direction x_2 . Finally, there are four corner contributions consisting only of constant and linear contributions in every direction.

Excursion Our three-term boundary splitting is closely related to the ANOVA decomposition used in statistics (for statistical applications, see [ES81]). The ANOVA decomposition and its relation to multi-level basis systems, and, in particular, the hierarchical hat basis is investigated in Section 4.1.3 where we consider dimension adaptive approximation approaches, but we will briefly discuss its relevance at this point.

The ANOVA decomposition is based on a one-dimensional two-term splitting $V^{(1)} = \mathbf{1} \oplus \widetilde{W}$ into constant and rest, defined by a projector $Pf := \int f(x) d\mu(x)$ and its complementary projector $(I - P)f$. Similarly to our approach, the d -dimensional ANOVA decomposition follows using a tensor product construction which is shown explicitly in Section 4.1.1. The special projector $P^0 f := \int f(x) \delta(x) dx = f(0)$ yields the anchor ANOVA decomposition and is closely related to our hierarchical basis: it holds $P^0 \phi_{l,i} = \phi_{l,i}(0) = 0$ for any $l \neq -1$. Consequently, $P^0 f = f_{-1,0} \phi_{-1,0}$ and $(I - P^0)f = \sum_{l \geq 0} \sum_{i \in I(l)} f_{l,i} \phi_{l,i}$. In other words, the hierarchical basis is the anchor ANOVA decomposition. However, the rest term \widetilde{W} does not care about the right boundary.

There is also an equivalent formulation of our *three*-term splitting in terms of ANOVA projectors: Let $V^{(1)}$ be our one-dimensional function space and define

$$P: V \rightarrow \mathbf{1}, \quad Pf := f(0) \quad (2.96)$$

and

$$\tilde{P}: V \rightarrow \text{lin}, \quad \tilde{P}f := f(1) \cdot x. \quad (2.97)$$

Then, $V^{(1)} = \mathbf{1} \oplus (\text{lin} \oplus \widetilde{W})$ given by $\mathbf{1} = P(V^{(1)})$, $\text{lin} \oplus \widetilde{W} = (I - P)(V^{(1)})$ yields a splitting into left boundary (the constant) and rest $\text{lin} \oplus \widetilde{W}$. Furthermore, the identity $I = \tilde{P} + (I - \tilde{P})$ applied to $\text{lin} \oplus \widetilde{W}$ yields the single terms $\text{lin} = \tilde{P}(\text{lin} \oplus \widetilde{W})$ and $\widetilde{W} = (I - \tilde{P})(\text{lin} \oplus \widetilde{W})$. Formulating both steps together, we obtain the identity

$$I = P + (\tilde{P} + (I - \tilde{P}))(I - P) \quad (2.98)$$

which induces the one-dimensional splitting

$$V^{(1)} = \mathbf{1} \oplus \text{lin} \oplus \widetilde{W}, \quad (2.99)$$

or, equivalently, a unique splitting

$$f = \underbrace{f_{-1}}_{\in \mathbf{1}} + \underbrace{\bar{f}}_{=(I-P)f \in \text{lin} \oplus \widetilde{W}} \quad \text{and} \quad \bar{f} = c_f x + \underbrace{\tilde{f}}_{=(I-\tilde{P})\bar{f} \in \widetilde{W}}. \quad (2.100)$$

2 A Sparse Grid Approximation Algebra

Note that \tilde{f} satisfies homogeneous boundary conditions $\tilde{f}(0) = \tilde{f}(1) = 0$. This can be seen using the projector properties $P^2 = P$ and $\tilde{P}^2 = \tilde{P}$ by

$$\tilde{f}(0) = P\tilde{f} = P[(I - \tilde{P})\tilde{f}] = P(I - P)f - P\tilde{P}(I - P)f \quad (2.101)$$

$$= 0 - (P[xf(1)] - P[xf(0)]) = 0 \text{ and} \quad (2.102)$$

$$x\tilde{f}(1) = \tilde{P}\tilde{f} = \tilde{P}[(I - \tilde{P})\tilde{f}] = \tilde{P}\tilde{f} - \tilde{P}^2\tilde{f} \equiv 0 \Rightarrow \tilde{f}(1) = 0. \quad (2.103)$$

Considering $P\phi_{l,i} = \phi_{l,i}(0) = 0$ for any $l > -1$ and $\tilde{P}\phi_{l,i} = \phi_{l,i}(1) \cdot x \equiv 0$ for $l > 0$, we can compare the splitting with the one induced by the hierarchical hat basis. It holds with the ANOVA identity (2.98) and with $f = \sum_{l=-1}^{\infty} \sum_{i \in I(l)} f_{l,i} \phi_{l,i}$

$$f = Pf + \tilde{P}[(I - P)f] + (I - \tilde{P})[(I - P)f] \quad (2.104)$$

$$= f(0) + (f(1) - f(0)) \cdot x + (f(x) - f(0) - [(f(1) - f(0)) \cdot x]) \quad (2.105)$$

$$= f_{-1,0} \phi_{-1,0} + f_{0,1} \phi_{0,1} + \sum_{l>0} \sum_{i \in I(l)} f_{l,i} \phi_{l,i} \quad (2.106)$$

by definition of the $f_{l,i}$. So, the two splitting are, indeed, the same for $f \in V^{(1)}$. A tensor product approach will produce an ANOVA-like three term splitting identical to (2.88), adapted to the ANOVA notation which is usually based on index sets $u \subseteq \{1, \dots, d\}$. Again, the tensor-product equivalent for \tilde{W} will exhibit homogeneous boundary conditions.

In other words: we have a three-term unique decomposition of ANOVA type with 3^d terms, once formulated in terms of ANOVA projectors (constant, linear and rest) and once in terms of multi-level basis functions (by levels $l = -1, l = 0$ and $l = *$).

Let us note the following prerequisite in order to estimate interpolation errors in lower-dimensional boundary manifolds.

Lemma 2.1.14. *Let $p, q \subset \{1, \dots, d\}$ be disjoint sets of directions which cover $\{1, \dots, d\}$, i.e. $p \cup q = \{1, \dots, d\}$, $p \cap q = \emptyset$. Let $g(x) = g_p(x_p) \cdot g_q(x_q)$ be a continuous (block) tensor product function defined on $[0, 1]^d$. Then, it holds*

$$\|g\|_{L_2[0,1]^d} = \|g_p\|_{L_2[0,1]^{|p|}} \cdot \|g_q\|_{L_2[0,1]^{|q|}} \quad (2.107)$$

and

$$\|g\|_{L_\infty[0,1]^d} = \|g_p\|_{L_\infty[0,1]^{|p|}} \cdot \|g_q\|_{L_\infty[0,1]^{|q|}}. \quad (2.108)$$

Proof. We find immediately

$$\begin{aligned} \|g_p g_q\|_{L_2}^2 &= \int g_p^2(x_p) \cdot g_q^2(x_q) \, dx \\ &= \int g_p^2(x_p) \, dx_p \int g_q^2(x_q) \, dx_q \\ &= \|g_p\|_{L_2[0,1]^{|p|}}^2 \cdot \|g_q\|_{L_2[0,1]^{|q|}}^2. \end{aligned}$$

Let us assume, the equation for L_∞ does not hold and $\|g\|_{L_\infty[0,1]^d} > 0$.

Case ‘>’ Since $[0, 1]^d$ is compact and g continuous, we could find \bar{x}_p and \bar{x}_q such that

$$\begin{aligned} \|g\|_{L_\infty[0,1]^d} &= |g_p(\bar{x}_p)g_q(\bar{x}_q)| > \sup\{|g_p(x_p)|\} \cdot \sup\{|g_q(x_q)|\} > 0 \\ &\Rightarrow 1 > \underbrace{\frac{\sup\{|g_p(x_p)|\}}{|g_p(\bar{x}_p)|}}_{\geq 1} \cdot \underbrace{\frac{\sup\{|g_q(x_q)|\}}{|g_q(\bar{x}_q)|}}_{\geq 1} \geq 1 \\ &\Rightarrow 1 > 1, \end{aligned}$$

so we have to conclude the opposite.

Case ‘<’ We would have for *every* \bar{x}_p and \bar{x}_q that

$$\begin{aligned} |g_p(\bar{x}_p)g_q(\bar{x}_q)| &< \sup\{|g_p(x_p)|\} \cdot \sup\{|g_q(x_q)|\} \\ \Rightarrow \frac{|g_p(\bar{x}_p)|}{\sup\{|g_p(x_p)|\}} \cdot \frac{|g_q(\bar{x}_q)|}{\sup\{|g_q(x_q)|\}} &< 1 \\ &\underbrace{\hspace{1.5cm}}_{\leq 1} \end{aligned}$$

which is not true for $\bar{x}_p := \operatorname{argmax}\{|g_p(x)|\}$ and $\bar{x}_q := \operatorname{argmax}\{|g_q(x)|\}$ although $(\bar{x}_p, \bar{x}_q) \in [0, 1]^d$ due to the continuity.

Thus, we have to conclude the validity of our lemma. \square

Note that such a statement does not hold for the energy norm for which one finds the different relation

$$\|g\|_E^2 = \|g_p\|_{L_2[0,1]^{|p|}}^2 \cdot \|g_q\|_{E[0,1]^{|q|}}^2 + \|g_q\|_{L_2[0,1]^{|q|}}^2 \cdot \|g_p\|_{E[0,1]^{|p|}}^2. \quad (2.109)$$

We employ Lemma 2.1.14 to reduce the dimension for each term in our boundary splitting as follows.

Lemma 2.1.15. *Let $f \in V^{(d)}$ with boundary splitting $f = \sum_{\mathbf{l} \in \{-1, 0, *\}^d} \bar{f}_{\mathbf{l}}$, $\bar{f}_{\mathbf{l}} \in \bar{W}_{\mathbf{l}}$ according to (2.93). Let $f^h \in V_h^{(d)}$ be the interpolant on some finite dimensional subset. Then, we have for $p \in \{2, \infty\}$*

$$\|\bar{f}_{\mathbf{l}} - \bar{f}_{\mathbf{l}}^h\|_{L_p[0,1]^d} = c_p^{|\mathbf{l}^{(0)}|} \cdot \|f_{\mathbf{l},*} - f_{\mathbf{l},*}^h\|_{L_p[0,1]^{|\mathbf{l}^{(*)}|}} \quad (2.110)$$

with constants $c_p := \|x_1\|_{L_p[0,1]}$, i.e. $c_\infty := 1$ and $c_2 := 3^{-1/2}$.

Proof. It holds per definition of $\bar{f}_{\mathbf{l}}$ and with the help of Lemma 2.1.14

$$\begin{aligned} \|\bar{f}_{\mathbf{l}} - \bar{f}_{\mathbf{l}}^h\|_{L_p[0,1]^d} &= \left\| \prod_{j \in \mathbf{l}^{(-1)}} 1 \cdot \prod_{j \in \mathbf{l}^{(0)}} x_j \cdot (f_{\mathbf{l},*} - f_{\mathbf{l},*}^h) \right\|_{L_p[0,1]^d} \\ &= \left\| \prod_{j \in \mathbf{l}^{(-1)}} 1 \right\|_{L_p[0,1]^{|\mathbf{l}^{(-1)}|}} \cdot \left\| \prod_{j \in \mathbf{l}^{(0)}} x_j \right\|_{L_p[0,1]^{|\mathbf{l}^{(0)}|}} \cdot \|f_{\mathbf{l},*} - f_{\mathbf{l},*}^h\|_{L_p[0,1]^{|\mathbf{l}^{(*)}|}} \\ &= c_p^{|\mathbf{l}^{(0)}|} \cdot \|f_{\mathbf{l},*} - f_{\mathbf{l},*}^h\|_{L_p[0,1]^{|\mathbf{l}^{(*)}|}}. \end{aligned}$$

\square

2 A Sparse Grid Approximation Algebra

Our aim is now to estimate the error in the varying terms only. Thus, we have to make sure that each component of our boundary splitting is represented by a at least the left and right boundary point. The conditions on grid points are satisfied for our regular sparse grid point set \underline{G}_n^d . To allow for more general decompositions, we summarize the precise conditions in the following

Definition 2.1.3 (Hanging node conditions). A finite dimensional subspace $\mathbf{L} \subset (\mathbb{N} \cup \{-1, 0\})^d$ discretizing the three-term boundary decomposition (2.93) is said to fulfill the *hanging node condition* if the following conditions are met:

1. $\mathbf{l} \in \mathbf{L} \Rightarrow \{\mathbf{k} \mid -\mathbf{1} \leq \mathbf{k} \leq \mathbf{l}\} \subset \mathbf{L}$, i.e. ancestors of any existing point are included in all directions, up to the boundaries,
2. $\bar{f}_1 \neq 0$ implies that the single level \mathbf{k} with $k_j = -1$ for $j \in \mathbf{l}^{(-1)}$, $k_j = 0$ for $j \in \mathbf{l}^{(0)}$ and $k_j = 1$ for $j \in \mathbf{l}^{(*)}$ exists in \mathbf{L} , $\mathbf{k} \in \mathbf{L}$.

In other words: the hanging node condition is met if every non-vanishing boundary component is represented by at least level 1 and hierarchical ancestors exist for every point. This implies that components with $|\mathbf{l}^{(*)}| = 0$ have no error.

Combining the previous results, we finally arrive at

Lemma 2.1.16 (Sparse grid errors for boundary case). *Let $f \in X^{q,2}$ be given such that*

$$\|f\|_{2,p} := \max_{\substack{\mathbf{l} \in \{-1,0,*\}^d \\ |\mathbf{l}^{(*)}| > 0}} \{ |f_{\mathbf{l},*}|_{2,p} \} < \infty \quad (2.111)$$

for $p \in \{2, \infty\}$ and $f = \sum_{\mathbf{l} \in \{-1,0,*\}^d} \bar{f}_1$ its unique boundary splitting according to (2.93). Let f^h be its hierarchical basis interpolant on a regular sparse grid (including boundaries),

$$f^h = \sum_{\mathbf{l} \in \bar{\mathbf{L}}_1^n} \sum_{\mathbf{i} \in \mathbf{I}(\mathbf{l})} f_{\mathbf{l},\mathbf{i}} \phi_{\mathbf{l},\mathbf{i}} = \sum_{\mathbf{l} \in \{-1,0,*\}^d} \bar{f}_1^h \quad (2.112)$$

with regular sparse grid index set $\bar{\mathbf{L}}_1^n$, formulated in $-1, 0, 1, \dots$ hierarchy notation according to (2.21).

Then, the interpolation error can be estimated for $p \in \{2, \infty\}$ by

$$\|f - f^h\| \leq \sum_{\substack{\mathbf{l} \in \{-1,0,*\}^d \\ |\mathbf{l}^{(*)}| > 0}} c_p^{|\mathbf{l}^{(0)}|} \cdot C_{p,|\mathbf{l}^{(*)}|} \cdot 2^{-2n} \cdot |f_{\mathbf{l},*}|_{2,p} \cdot A(|\mathbf{l}^{(*)}|, n) \quad (2.113)$$

$$< C_p \cdot 2^{-2n} \cdot \|f\|_{2,p} \cdot A(d, n) \quad (2.114)$$

$$= \mathcal{O}(2^{-2n} n^{d-1}) \quad (2.115)$$

where $C_p := (3^d - 2^d) \cdot \max_{j=1,\dots,d} C_{p,j}$. The single constant c_p from Lemma 2.1.15 is $c_\infty = 1$ and $c_2 = 3^{-1/2}$; The constant $C_{p,j}$ from inner interpolation error bounds are given by $C_{\infty,j} := 2 \cdot 8^{-d}$ and $C_{2,j} := 2 \cdot 12^{-d}$ (see Lemma 2.1.9). Thus, $C_\infty = (3^d - 2^d)/4$ and $C_2 = (3^d - 2^d)/6$.

The approach to measure smoothness as in (2.111) and errors on sub-components is related to the variation of Hardy and Krause [Owe03]: it computes derivatives along boundary slices and sums them together.

Proof. Due to the hanging node conditions 2.1.3, we find the boundary special case $|\mathbf{l}^{(*)}| = 0 \Rightarrow (f_{\mathbf{l},*} - f_{\mathbf{l},*}^h) = 0$. Together with the previous set of lemma, we find

$$\begin{aligned}
 \|f - f^h\|_{L_p[0,1]^d} &= \left\| \sum_{\mathbf{l} \in \{-1,0,*\}^d} (\bar{f}_{\mathbf{l}} - \bar{f}_{\mathbf{l}}^h) \right\|_{L_p[0,1]^d} \leq \sum_{\mathbf{l} \in \{-1,0,*\}^d} \|\bar{f}_{\mathbf{l}} - \bar{f}_{\mathbf{l}}^h\|_{L_p[0,1]^d} \\
 &= \sum_{\mathbf{l} \in \{-1,0,*\}^d} c_p^{|\mathbf{l}^{(0)}|} \cdot \|f_{\mathbf{l},*} - f_{\mathbf{l},*}^h\|_{L_p[0,1]^{|\mathbf{l}^*|}} \\
 &\leq \sum_{\substack{\mathbf{l} \in \{-1,0,*\}^d \\ |\mathbf{l}^{(*)}| > 0}} c_p^{|\mathbf{l}^{(0)}|} \cdot C_{p,|\mathbf{l}^{(*)}|} \cdot 2^{-2n} \cdot |f_{\mathbf{l},*}|_{2,p} \cdot A(|\mathbf{l}^{(*)}|, n) \\
 &< \underbrace{\max_{j=0,\dots,d-1} c_p^j}_{=c_p^0=1} \cdot \max_{j=1,\dots,d} C_{p,j} \cdot 2^{-2n} \cdot \max\{|f_{\mathbf{l},*}|_{2,p}\} \cdot A(d, n) \cdot \underbrace{\sum_{\substack{\mathbf{l} \in \{-1,0,*\}^d \\ |\mathbf{l}^{(*)}| > 0}} 1}_{=3^d - 2^d} \\
 &= C_p \cdot 2^{-2n} \|f\|_{2,p} A(d, n).
 \end{aligned}$$

□

2.2 Commonly used Algorithms and Complexities

In this section, we summarize algorithms which are used in different contexts in this thesis, along with their cost complexities.

2.2.1 Adaptive Interpolation and Approximation

We consider an adaptive refinement algorithm which is suitable to resolve local irregularities in the context of function approximation (interpolation or partial differential equations or the-like). Space adaptive procedures for sparse grids are known since the first works on sparse grids, see [Bun92, Gri98] and the references therein. The idea is to employ the hierarchical coefficients as indicators of high local variation. Since a (locally) smooth function exhibits exponentially decaying hierarchical coefficients according to Lemma 2.1.5, we can simply drop small coefficients in smooth regions. The remaining leaf node governs the introduced error due to a geometric series argument. Coefficients with large value should be refined by inserting their sons in all directions, resulting in $2 \cdot d$ new points. Since hanging nodes are not admissible, we need to insert every new node along with all its hierarchical ancestors into the grid.

The error indicator identifies large local variation by means of the weighted coefficient $|f_{\mathbf{l},\mathbf{i}}| \cdot \|\phi_{\mathbf{l},\mathbf{i}}\|$, where $|f_{\mathbf{l},\mathbf{i}}|$ measures the second mixed derivative of f according to (2.51) and $\|\phi_{\mathbf{l},\mathbf{i}}\|$ weights the indicator for the norm of interest, compare Lemma 2.1.4.

Logically, an adaptive sparse grid is then defined via the grid compression

$$f^\epsilon = \sum_{(\mathbf{l}, \mathbf{i}) \in G_f^\epsilon} f_{\mathbf{l}, \mathbf{i}} \phi_{\mathbf{l}, \mathbf{i}} \text{ with } G_f^\epsilon := \{(\mathbf{l}, \mathbf{i}) \mid |f_{\mathbf{l}, \mathbf{i}}| \cdot \|\phi_{\mathbf{l}, \mathbf{i}}\| \geq \epsilon\} \quad (2.116)$$

combined with the set of all required ancestors to avoid hanging nodes. In practice, we resort usually to additive grid refinement, i.e. we start on a coarse level, compute the local error indicators and refine all points with large indicator. Provided the error indicator works correctly, the approach will yield comparable results at less cost. Note that the hat basis coefficients of an interpolation problem do not change by inserting new nodes whereas other bases (for example the prewavelets studied later) have coefficients depending on all nodal values and need to be recomputed after every refinement.

Since hierarchical coefficients are related to second derivatives, every coefficient whose basis point is a reflection point of the function will vanish. This early termination problem is a known weakness of the indicator; it can be circumvented by investigating the node's children as well. Look-ahead strategies of this kind require to compute the approximant at child nodes even if the node as such is "irrelevant" according to the error indicator. One will typically discard such information if it turns out to be of minor importance, so look-ahead strategies are of the form

- refine all points,
- compute all values,
- compress the result,
- iterate until convergence.

Note that all discussed methods apply to inner- and boundary grid points. A boundary grid point belongs to a lower dimensional part of the function as elaborated in Lemma 2.1.13, but it works as expected (see also [PPB10] for a special space adaptive refinement which eliminates boundary points for data mining applications).

The parameter ϵ can be either an absolute number $\epsilon^{(\text{abs})}$ or a relative threshold of the form $\epsilon = \|f\| \cdot \epsilon^{(\text{rel})}$. If the value $\|f\|$ is not known analytically, one might use $\|f^{(i)}\|$, the norm of the actual approximant, instead.

The overall space adaptive algorithm is summarized in Algorithm 1. It is parameterized with a "set values" interface which should compute the (missing) values $f_{\mathbf{l}, \mathbf{i}}$ somehow. It supports a look-ahead parameter q and both, additive refinement and compression (using a boolean `bCompress`). A look-ahead of $q > 0$ implies `bCompress=true`. If the set values routine works incrementally, with $\mathcal{O}(1)$ operations for every newly inserted point, the overall runtime complexity is bounded by the final grid size, $\mathcal{O}(|G|)$. For interpolation, hierarchical basis coefficients can be computed either incrementally in time $\mathcal{O}(3^d)$ per point using the stencil (2.49) or, together with all basis coefficients, in time $\mathcal{O}(d|G|)$ using the fast transformation. Computing all coefficients once involves mean cost per unknown of $\mathcal{O}(d)$ for the non-incremental routine in comparison to $\mathcal{O}(3^d)$ for the incremental version. In the general case, the set values routine will need to (re)compute

Algorithm 1 Space adaptive refinement

Input: Initial Grid $G^{(0)}$

Input: Target threshold $\epsilon = \epsilon^{(\text{abs})}$ or $\epsilon = \epsilon^{(\text{rel})}$

Input: A “set values” algorithm (for example, incremental interpolation, complete interpolation or a PDE)

Input: refinement parameters: a look-ahead-depth $q \geq 0$ and a boolean **bCompress** such that $q > 0 \Rightarrow \text{bCompress} = \text{true}$.

Output: a grid G and the approximated function f on G

```

1:  $i := 0$ 
2: repeat
3:    $G^{(i,0)} := G^{(i)}$ 
4:   for  $m = 1$  to  $q$  do
5:      $G^{(i,m)} := \text{Insert all } 2 \cdot d \text{ child nodes of existing nodes into } G^{(i,m-1)}$ 
6:   end for
7:    $f^{(i)} := \text{set values on } G^{(i,q)}$ 
8:   if bCompress then
9:      $G^{(i+1)} := \text{compress } G^{(i,q)} \text{ based on } f^{(i)}$  // Details in Algorithm 26
10:  else
11:     $G^{(i+1)} := \text{refine } G^{(i,q)} \text{ based on } f^{(i)}$  // Details in Algorithm 25
12:  end if
13:   $i := i + 1$ 
14: until  $G^{(i)} \setminus G^{(i-1)} = \emptyset$ 
15: return  $G := G^{(i)}$  and  $f := f^{(i)}$ 
16: See Appendix A.4 for algorithmic details.
```

$f^{(i)}$ for every point in $G^{(i)}$. The overall runtime can thus become larger than $|G|$ which is analyzed in

Lemma 2.2.1 (Runtime Complexity of Algorithm 1). *Algorithm 1, used with look-ahead parameter $q = 0$ and a set values routine which recomputes all values in linear time $\mathcal{O}(d|G|)$, has the following runtime complexity with respect to the final grid size $|G|$ and k iterations.*

1. If **bCompress** = true, the runtime complexity is $\mathcal{O}(G^{(0)})$.
2. If **bCompress** = false, the runtime depends crucially on the number of points inserted in every iteration. We provide it for the following cases:
 - a) If $|G^{(i)}| \leq \mathcal{O}(2^{i_0+i}(i_0+i)^{d-1})$ (a regular sparse grid of level i_0+i), the overall runtime is in $\mathcal{O}(|G|)$.
 - b) If $|G^{(i)}| \leq \gamma \cdot |G^{(i-1)}|$ with $\gamma > 1$, the complexity is also bounded by $\mathcal{O}(|G|)$.
 - c) If $|G^{(i)}| \leq |G^{(i-1)}| + \gamma$, $\gamma \in \mathbb{N}$, the complexity is bounded by $\mathcal{O}(k|G| + (k-k^2))$.

Proof. The compression case is clear, we show the additive refinement cases. The first two employ geometric series arguments:

2 A Sparse Grid Approximation Algebra

a)

$$\begin{aligned} \text{Runtime} &= \mathcal{O}\left(\sum_{i=0}^k |G^{(i)}|\right) = \mathcal{O}\left(\sum_{i=0}^k 2^{i_0+i}(i_0+i)^{d-1}\right) \\ &= \mathcal{O}\left((i_0+k)^{d-1}2^{i_0}(2^{k+1}-1)\right) = \mathcal{O}\left(2 \cdot 2^{i_0+k}(i_0+k)^{d-1}\right) = \mathcal{O}(|G^{(k)}|)\checkmark \end{aligned}$$

b) We find $|G^{(i)}| = \gamma^i |G^{(0)}|$ and thus

$$\text{Runtime} = \mathcal{O}(|G^{(0)}| \sum_{i=0}^k \gamma^i) = \mathcal{O}\left(\frac{\gamma|G| - |G^{(0)}|}{\gamma - 1}\right) = \mathcal{O}(|G|)\checkmark \quad (2.117)$$

c) We find $|G^{(i)}| = |G^{(0)}| + i\gamma$ and thus

$$\begin{aligned} \text{Runtime} &= \mathcal{O}(k|G^{(0)}| + \sum_{i=0}^k i \cdot \gamma) = \mathcal{O}\left(k(|G^{(0)}| + \gamma k + \frac{\gamma}{2} - \frac{\gamma k}{2})\right) \\ &= \mathcal{O}(k|G^{(k)}| + \gamma/2(k - k^2))\checkmark \end{aligned}$$

□

Since we can expect k to grow like the sparse grid level, the case c) can be interpreted to need a further logarithmic runtime factor for the grid computation.

We are now in a position to create adaptive representations of functions and to compute the hierarchical coefficients by means of the linear-time transformations. Furthermore, we can compute all nodal values at basis points simultaneously in linear time by means of the inverse transformation¹.

We turn to a discussion of the task to evaluate a sparse grid function at an arbitrary point $x \in [0, 1]^d$. Its runtime complexity is subject of

Lemma 2.2.2 (Cost of Arbitrary Point Evaluation). *Let $f^h \in V^h$ be a d -dimensional sparse grid function on a grid with sparse grid level n (n may denote the largest used level for an adaptive grid). Let f^h be represented in the hierarchical hat basis.*

Then, the point evaluation at $x \in [0, 1]^d$, $f^h(x)$, can be computed in time $\mathcal{O}(n^d)$. Furthermore, a first order extrapolation at $x \notin [0, 1]^d$, defined by

$$f^h(x) = f^h(x') + \sum_{j=1}^d \frac{\partial}{\partial x_j} f^h(x') \cdot (x_j - x'_j), \quad (2.118)$$

$$x'_j = \begin{cases} x_j, & x_j \in [0, 1], \\ 0, & x_j < 0, \\ 1, & x_j > 1, \end{cases} \quad (2.119)$$

is possible in time $\mathcal{O}(n^d)$ as well. Here, $\frac{\partial}{\partial x_j} f^h(x')$ is either a forward or a backward difference stencil (depending on the position of x' relative to $[0, 1]^d$).

¹See Appendix A.2 for implementational details.

Proof. First, we deal with the case of homogeneous boundary conditions. In this case,

$$f^h(x) = \sum_{|\mathbf{l}|_1 \leq n+d-1} \sum_{i \in W_{\mathbf{l}}} f_{\mathbf{l},i} \phi_{\mathbf{l},i}(x) \quad (2.120)$$

where $\{\phi_{\mathbf{l},i}\}$ is the hierarchical hat basis. Since the supports of all $\phi_{\mathbf{l},i}$ for fixed \mathbf{l} are mutually disjoint, there is at most one index (\mathbf{l}, i) such that $\phi_{\mathbf{l},i}(x) \neq 0$ for every level. In general, we have to assume that there is also at least one such index. Thus, we count $\sum_{|\mathbf{l}|_1 \leq n+d-1} 1$ many evaluations $\phi_{\mathbf{l},i}(x)$. Furthermore, the operation $\phi_{\mathbf{l},i}(x)$ involves $\mathcal{O}(d)$ cost (including cost to determine i for given \mathbf{l}). The total cost $|W_n^d|$ is thus

$$\begin{aligned} |W_n^d| &= d \cdot \sum_{d \leq |\mathbf{l}|_1 \leq n+d-1} 1 = d \cdot \sum_{j=d}^{n+d-1} \sum_{|\mathbf{l}|_1=j} 1 = d \cdot \sum_{j=d}^{n+d-1} \binom{j-1}{d-1} = d \cdot \sum_{j=0}^{n-1} \binom{j+d-1}{d-1} \\ &= d \cdot \binom{d+n-1}{d} = d \cdot \frac{1}{d!} \underbrace{(d+n-1) \cdots (d+n-d)}_{d+n-1-(d+n-d-1)=d \text{ factors}} = \frac{1}{(d-1)!} n^d + \mathcal{O}(n^{d-1}), \end{aligned}$$

where we employed the same reasoning as for the cost of sparse grids, compare [BG04].

The forward or backward derivative of $\phi_{\mathbf{l},i}$ can be computed as well using an additional factor d . Summing each intermediate result yields $f^h(x)$.

The case of inhomogeneous boundary conditions follows using a similar argumentation as for Lemma 2.1.2: by recursion into lower dimensional boundary manifolds which have the same level n . As for Lemma 2.1.2, we find the loose upper bound

$$|W_n^d| = \sum_{j=0}^d \binom{d}{j} 2^{d-j} |W_n^j| \leq |W_n^d| \cdot \sum_{j=0}^d \binom{d}{j} 2^{d-j} = 3^d \cdot |W_n^d| = \mathcal{O}(n^d) \quad (2.121)$$

for the total cost $|W_n^d|$. □

Algorithmically, there are two ways to compute $f^h(x)$, depending on the type of grid. If every subspace $W_{\mathbf{l}}$ contains all possible basis points, we can simply iterate through all level indices \mathbf{l} and determine the single, uniquely identified space index $i(\mathbf{l})$ for which $x \in \text{supp } \phi_{\mathbf{l},i}$ by table lookup in time $\mathcal{O}(d)$. If we are working with a space adaptive grid, we can start in the tree's root, visit all points hierarchically as described in Appendix A.1.1 and prune subtrees as soon as $x \notin \text{supp } \phi_{\mathbf{l},i}$. Since grids have no hanging nodes and the hierarchical hat basis is nested, this approach yields the correct result.

2.2.2 Fast Algorithms: the Unidirectional Principle

In this thesis, we employ different linear operations on sparse grid functions u like hierarchical transformations, the application of mass matrices or stiffness matrices in the context of partial differential equations. An important property of these linear operations is a tensor product representation $A = A_1 \cdots A_d$.

This section summarizes briefly how tensor product type linear operations can be realized in linear time even though a sparse grid is no full tensor product. The idea is to rely on one-dimensional matrix decompositions and a recursive combination rule for the d -dimensional case which is compatible with the sparse grid tree structure. The method goes back to [Bal94] and has been elaborated in [Bun98] and [Ach03], with modifications for special matrices in [Feu05]. We summarize the key properties and their runtime requirements.

A linear operator with tensor product structure can be characterized by its matrix $A = A_1 \cdots A_d \in \mathbb{R}^{N \times N}$ where $N = |u|$ is the grid size and $A_i \in \mathbb{R}^{N \times N}$ operates on all grid lines in direction i (but has no interaction between different grid lines). An example is the mass matrix with separable entries $M_{(\mathbf{l},i),(\mathbf{k},j)} = \int \phi_{\mathbf{l},i}(x) \phi_{\mathbf{k},j}(x) dx = \prod \int \phi_{l_m,i_m}(x_m) \phi_{k_m,j_m}(x_m) dx_m$. On a full grid, the application of A to a grid function u can be realized using successive matrix-vector products $A_i v$ from outer to inner, i.e. by $Au = (A_1 \cdots (A_{d-2}(A_{d-1}(A_d u))) \cdots)$. Intermediate results between successive steps can be stored on full grid unknowns, making the matrix decomposition compatible with the grid structure. The operation A_i as such can be formulated in a line-wise fashion by means of one-dimensional operations (standard tensor product approach).

For sparse grids, the storage of intermediate results is more involved. Balder proposed a splitting of the one-directional matrices A_i and a particular recombination formula to compute Au which allows to store intermediate results on sparse grids in [Bal94]. The idea is to employ the hierarchy in every direction, together with the condition that every node has all its ancestors in the grid: first, consider a one-dimensional level-wise ordering for the degrees of freedom (i.e. lexicographically in (l, i)). Thus, a matrix entry $a_{(l,i),(l+1,2i+1)}$ is *above* the diagonal since (l, i) has smaller index than its son $(l+1, 2i+1)$. Then, split the one-dimensional matrix A^{1d} into an upper triangular part B^{1d} and a lower triangular part T^{1d} such that $A^{1d} = T^{1d} + B^{1d}$. Since $(B^{1d}v)_{k,j} = \sum_{(l,i) > (k,j)} B_{(k,j),(l,i)}^{1d} v_{l,i}$ contains only interactions which are deeper than (k, j) in the tree, B^{1d} is called “bottom-up” algorithm ([Bun98]). Similarly, $(T^{1d}v)_{k,j} = \sum_{(l,i) \leq (k,j)} T_{(k,j),(l,i)}^{1d} v_{l,i}$ handles only interactions from the top of the tree and is thus called “top-down” algorithm. Now apply the same idea to every grid line in direction i , yielding a splitting $A_i = T_i + B_i$.

The algorithm presented in [Bal94] decomposes A into

$$A = \prod_{m=1}^d A_m = \prod_{m=1}^{d-1} A_m \cdot B_d + T_d \cdot \prod_{m=1}^{d-1} A_m. \quad (2.122)$$

Since $\prod_{m=1}^{d-1} A_m$ has the same structure, the decomposition can be applied recursively – and the particular sequence of applications (first recursion, then top-down T_d , and first bottom-up B_d , then recursion) ensures compatibility with the d -dimensional sparse grid structure for storage of intermediate results. The overall algorithm is called *Unidirectional Principle* [Bun98] and has been described for the general case in [Ach03] and [Feu05], see also its derivation in [Bal94].

The Unidirectional Principle can be applied to linear tensor product operators on sparse grids, it relies on commutation of the sequence in which the A_m are applied and fast top-down and bottom-up splittings. This covers Galerkin discretizations with

either constant coefficients or variable coefficients of product type, compare [Ach03]. Furthermore, the hierarchical transformations can be seen as special case of (2.122): the transformation from nodal values to the hierarchical hat basis, for example, has $B^{1d} = 0$ and is thus given by $\prod_{m=1}^d T_m$ (compare Appendix A.2.2).

The Unidirectional Principle is a matrix–vector–product algorithm: it allows to compute Au in linear time $\mathcal{O}(c(d) \cdot N)$ even though A has much more non–vanishing elements than $\mathcal{O}(N)$ due to finger structure. In the general case $c(d) = 2^{d+1}$ many one–directional matrix–vector–products are necessary, i.e. time $\mathcal{O}(2^{d+1}N)$, and each recursion step needs one copy, resulting in $\mathcal{O}(dN)$ memory (see [Feu05]).

For special cases, the runtime- and memory requirements can be reduced drastically. Such a case is given by the hierarchical transformations which usually require *either* a top–down *or* a bottom–up step (compare Appendix A.2), thereby avoiding the recursion of (2.122). They need $\mathcal{O}(dN)$ time and $\mathcal{O}(N)$ memory. Another important case is posed by partial differential equations and orthogonal ansatz spaces: a product differential operator $D^{(\alpha)}u$ which operates only on $\mathcal{O}(1)$ directions has $\mathcal{O}(d)$ directions for the identity. In terms of a Galerkin discretization, it can be discretized by $\mathcal{O}(d)$ one–directional mass matrices and $\mathcal{O}(1)$ other one–directional operations. Using an L_2 (semi–) orthogonal basis yields $M_i = T_i$ with $B_i = 0$ and thus also considerable savings in (2.122). For example, the Laplace operator with diffusion terms A_m and mass matrix terms M_m yields the matrix–vector–product

$$Ru := \sum_{m=1}^d \left(\prod_{k=1}^{m-1} M_k A_m \prod_{k=m+1}^d M_k \right) u \quad (2.123)$$

$$= \sum_{m=1}^d A_m \prod_{k \neq m} M_k u \quad (2.124)$$

$$= \sum_{m=1}^d A_m M_m^{-1} v \quad (2.125)$$

with $v := \prod_{m=1}^d M_m u$. Here, (2.123) requires time $\mathcal{O}(d^2N)$ and memory $\mathcal{O}(N)$ by means of (2.122), where the largest number of matrix vector multiplications for a single summand is $2 \cdot d$. The variant (2.124) has the minimum number of multiplications since it avoids recursions by using A_m as last operator: it needs just $(d+1)$ multiplications for each summand. Finally, (2.125) needs just $d + 2 \cdot d = 3d$ multiplications for the complete product Ru if a fast inverse mass matrix product is available, resulting in optimal time $\mathcal{O}(dN)$ for one matrix–vector–product. This is possible for the prewavelet basis (M_m^{-1} requires to solve a pentadiagonal linear system on each level, which is possible in time $\mathcal{O}(N)$). See [Feu05] for details about these cases. Thus, mass–dominated operators result in optimal time and memory for matrix–vector–products by means of the Unidirectional Principle if (semi–) orthogonal basis functions are used.

The only ingredients besides the combination formula (2.122) is a fast $\mathcal{O}(N)$ implementation for each of the one–dimensional top–down and bottom–up algorithms. A few of these algorithms are summarized in Appendix A.3, see also [Ach03] for a generic derivation of these operations.

2.2.3 Data Structures for Adaptive Sparse Grids

One can say that sparse grids trade beneficial cost–gain ratio versus complicated algorithms and advanced data structures. In this section, we summarize essential data structures used to realize sparse grid approximation. The proposed structures rely on *unique hashing*, i.e. unique mappings from high dimensional points (\mathbf{l}, \mathbf{i}) to scalar, unique integer numbers. Instead of storing d -dimensional coordinates, we store only integers. For adaptive sparse grids, these unique integers are stored in hashmaps whereas regular sparse grids can be stored in arrays. We will see that such an approach is fast, requires few memory, and belongs to the most flexible data structures (supporting all required algorithms).

We also provide a survey over other data structures and show how to trade memory usage versus access speed or algorithmic flexibility, including approaches taken in the literature.

Algorithmic Requirements on Data Structures

The key to perform linear time sparse grid algorithms like interpolation (i.e. hierarchical transformations), partial differential equations (which boil down to matrix–vector–products) and evaluation of L_2 - or energy norms is to apply one–dimensional algorithms on each grid line in a particular direction, see Section 2.2.2 for details. Thus, any data structure needs to support line traversals in any direction, where the one–dimensional operations during line traversals simply require access to the two sons of every given point, and the father of a point (compare the algorithms listed in Appendix A.1).

Besides line traversals, we also need adaptive refinement, i.e. the possibility to insert new nodes (and their ancestors).

Furthermore, to allow operations involving different grids (like interpolation or restriction from one grid on another), or to realize the arbitrary point evaluation discussed in Lemma 2.2.2, we also need to access arbitrary points $(\mathbf{l}, \mathbf{i}) \in G$, i.e. without relation to line traversals and occasionally even without reference to a father of (\mathbf{l}, \mathbf{i}) .

Fulfilling all of these requirements is a demanding task which is probably only possible using fast associative container structures which map an arbitrary (\mathbf{l}, \mathbf{i}) to the respective coefficient $u_{\mathbf{l}, \mathbf{i}}$ (or, in practice, to an index into the respective coefficient vector). There are, however, exceptions where a specialized data structure is used even though it performs poorly at one or more of the mentioned requirements, in favor of other advantages. These specific advantages and disadvantages are subject of the following section before we present a fast associative container structure.

A Survey over Specialized Structures

Sparse grids are kind of d -dimensional binary trees, with one–dimensional binary tree structure in every single direction (but unlike k - d -trees known in computer science, they are fully d -dimensional at each point). Consequently, one can store pointers for every sparse grid point: $2 \cdot d$ pointers to access child nodes in every direction and another d pointers to the fathers. Boundary points need special handling: they can be stored by

using $(l_m, i_m) = (0, 0)$ as point without father, with left (!) son $(l_m, i_m) = (0, 1)$ and right son $(l_m, i_m) = (1, 1)$. Then, $(l_m, i_m) = (0, 1)$ has $(l_m, i_m) = (1, 1)$ as left son, but no right son and $(l_m, i_m) = (1, 1)$ has $(l_m, i_m) = (0, 1)$ as its father². Once such a data structure is built, it supports fast line traversals and fast arbitrary point evaluations, even in the fully adaptive case. In fact, the task to perform a huge sequence of arbitrary point evaluations $f(\xi_i)$ is probably fastest using such a pointer structure³. However, constructing or changing a pointer structure is quite involved: it can be realized by means of a second data structure which supports more flexible, direct point access to determine the required pointer targets (see also [Ach03] and the references therein).

A possible application for these d -trees is function compression with fast decompression: once the compression (and construction) step is ready, one can perform fast point evaluations at arbitrary positions $x \in [0, 1]^d$ (local decompression). The storage cost per node is $\mathcal{O}(d)$ (more precisely: $3 \cdot d$ pointers and the stored value) and any grid traversal is possible in time $\mathcal{O}(N)$, i.e. $\mathcal{O}(1)$ per node.

A different approach to implement the d -dimensional binary tree structure is used in [Ach03, Bal94, Bun98] and [Nie98]: the idea is to use a conventional (one-dimensional) binary tree for direction x_1 , which stores a $(d - 1)$ dimensional sparse grid structure recursively. Only the last direction actually contains scalar values. Such a structure has compact storage $\mathcal{O}(1)$ per node, and each point can be visited in $\mathcal{O}(1)$ operations per node – but only in the specific storage sequence; line traversals are not directly possible in all directions. Thus, algorithms which work on lines need to be redesigned in order to work with slices, compare [Ach03]. For wavelets of larger support than one mesh width (like our prewavelets), this involves even more than just two slices per point. Thus, the recursive structure considerably complicates the algorithms, not talking about the additional slice memory.

A further structure of different type has been used in [Kos01] and (among others) in [Feu05]: one can store every grid line in each coordinate direction explicitly. Thus, the complete structure is stored d times, once for every direction (in fact, coefficient vectors are stored exactly once, only indices into coefficient arrays are replicated). Since each single line container needs the associated multi indices as well, the memory requirements here is $\mathcal{O}(d^2N)$, or, if the grid has regular sparse grid structure, $\mathcal{O}(dN)$ by means of memory sharing. On the other side, each line traversal is possible in optimal time $\mathcal{O}(N)$, independent of d . Building or changing such a structure involves copy operations which are possible in time $\mathcal{O}(d^2N)$ if one uses hash based structures for the direct access to arbitrary (\mathbf{l}, \mathbf{i}) , see the discussion in [Feu05].

Finally there are data structures which rely on associative containers to map (\mathbf{l}, \mathbf{i}) directly to its associated coefficient $u_{\mathbf{l}, \mathbf{i}}$ (or an index into the respective vector). The idea is to formulate all algorithms in terms of multi indices (\mathbf{l}, \mathbf{i}) and to query required coefficient values via the associative container. Thus, such a structure supports all operations which can be formulated by means of multi indices. An example for such a

²This particular structure has been implemented and works with the algorithms of Appendix A.1.

³Note that cache alignment is best if coefficients $f_{\mathbf{l}, \mathbf{i}}$ are stored in a similar sequence as the point evaluation routine visits them.

container are hash maps, which employ a hash function $q(\mathbf{l}, \mathbf{i}) \mapsto \{0, \dots, m\}$ to map multi indices to one of m so-called buckets. Such structures are used in [Bun96] and [Sch99]. Hash maps are not necessarily injective, so collisions need to be resolved. One possibility is that every multi index with the same hash value is stored into a list and access to (\mathbf{l}, \mathbf{i}) involves to compare (\mathbf{l}, \mathbf{i}) with all candidates of same hash value. With a reasonable hash function and resize strategies, hash maps yield a mean access time of $\mathcal{O}(d)$: the hash function $q(\mathbf{l}, \mathbf{i})$ typically involves $2 \cdot d$ arithmetic steps and so does each comparison to resolve conflicts. The mean number of conflicts is $\mathcal{O}(1)$ due to resize strategies, resulting in $\mathcal{O}(d)$ for one lookup. In order to resolve conflicts, each multi index needs to be stored explicitly, resulting in memory usage of $\mathcal{O}(dN)$ for the complete structure. Thus, such a hash based approach directly yields both, $\mathcal{O}(dN)$ memory and $\mathcal{O}(dN)$ time for any grid traversal.

Note that dimension adaptive grids as discussed in Chapter 4 have potential to reduce memory usage: we will see that highly dimension adaptive grids yield indices where most multi index components have the same value $(l_m, i_m) = (0, 0)$ (Chapter 4 actually uses the special notation $(-1, 0)$ for the same point). If only components (l_m, i_m) with $m \in u(\mathbf{l}) := \{m \mid (l_m, i_m) \neq (0, 0)\}$ are actually stored, memory usage becomes $\mathcal{O}(\max_{\mathbf{l}} |u(\mathbf{l})| \cdot N) \leq \mathcal{O}(dN)$. Such a technique is used for dimension adaptive integration in [Nah05]. We will come back to dimension adaptive data structures at the end of this section.

Hash based data structures and similar associative containers allow the greatest flexibility⁴. The next section shows that it can be realized with optimal memory complexity $\mathcal{O}(N)$ as well – with the help of unique hashing techniques.

Data Structures based on Unique Hashing

We turn to a versatile data structure which supports all operations on sparse grids with optimal memory usage and access time $\mathcal{O}(d)$. The idea is to employ hash functions which provide a *unique* hash for every (\mathbf{l}, \mathbf{i}) . More precisely, we search a map $r(\mathbf{l}, \mathbf{i}) \rightarrow \mathbb{N}_0$ such that (\mathbf{l}, \mathbf{i}) can be identified uniquely with its hash $r(\mathbf{l}, \mathbf{i})$ and the hash can be computed in time $\mathcal{O}(d)$, without reference to any particular grid. Consequently, we do not need to store (\mathbf{l}, \mathbf{i}) in a hash map anymore – it is sufficient to store its unique hash $r(\mathbf{l}, \mathbf{i})$, which reduces the task of resolving conflicts to a comparison of integer numbers. The memory usage is thus $\mathcal{O}(N)$, independent of d and we have a good hash function for every dimension⁵. Furthermore, we can refine such a structure, since the hashes are independent of the old grid. A refinement means to insert further integer hashes into the associative container.

The unique hashing procedure is summarized from [Feu05]. Its realization provides the following features:

⁴They allow to visit every grid line by means of abstract tree operations which are listed in Appendix A.1.

⁵It should be stressed that storing the values $r(\mathbf{l}, \mathbf{i})$ in a hash map still results in conflicts. This is because a hash map has a finite bucket size m (number of storage places) and it will place $r(\mathbf{l}, \mathbf{i})$ into the bucket $r(\mathbf{l}, \mathbf{i}) \bmod m$. The use of unique hashes reduces the memory usage and simplifies collision detection and memory allocation.

1. It can be computed in time $\mathcal{O}(d)$.
2. It provides a unique mapping from (\mathbf{l}, \mathbf{i}) to a scalar number.
3. It is independent of any grid by assigning scalar numbers in a levelwise fashion,

$$n(\mathbf{l}) < n(\mathbf{k}) \Rightarrow r(\mathbf{l}, \cdot) < r(\mathbf{k}, \cdot) \quad (2.126)$$

with $n(\mathbf{l}) = \sum l_m - d + 1$.

4. For regular sparse grids, the mapping is in fact bijective between

$$r(\mathbf{l}, \mathbf{i}): G_n^d \leftrightarrow \{0, 1, 2, \dots, |G_n^d|\}. \quad (2.127)$$

5. The mapping has been split into a separate map for inner points and one for boundary points.
6. The runtime requirements for such a data structure is $\mathcal{O}(dN)$ to visit all points with memory usage $\mathcal{O}(N)$.

Feature 4) yields a particularly efficient associative container for regular sparse grids: instead of a hash map, a simple array A is sufficient. Then, $A[r(\mathbf{l}, \mathbf{i})] = u_{\mathbf{l}, \mathbf{i}}$ due to the one-to-one mapping and the memory usage is exactly one real coefficient per grid point (without overhead). Property 5 is technically unnecessary, but it allows to discard boundary nodes completely which is important for our experiments in Chapter 3.

We start with an index map $r_I(\mathbf{l}, \mathbf{i})$ which is only defined for *inner* nodes of a sparse grid,

$$G_n^d = \bigcup_{1 \leq n(\mathbf{l}) \leq n} W_{\mathbf{l}}, \quad W_{\mathbf{l}} := \{\mathbf{i} = (i_1, \dots, i_d) \mid i_m \in \{1, 3, 5, \dots, 2^{l_m} - 1\}\}. \quad (2.128)$$

According to the derivation in [Feu05], we find the unique hash

$$r_I(\mathbf{l}, \mathbf{i}) := P^{<n(\mathbf{l})} + 2^{n(\mathbf{l})-1} \cdot r_{n(\mathbf{l})}(\mathbf{l}) + r_1(\mathbf{i}) \quad (2.129)$$

where $P^{<n} := |G_{n-1}^d|$ counts all points of a regular sparse grid of level $(n-1)$, $2^{n(\mathbf{l})-1} = |W_{\mathbf{l}}|$ is the number of different space indices,

$$r_{n(\mathbf{l})}(\mathbf{l}) := \sum_{m=2}^d P^{n_m(\mathbf{l})-1, d-(m-1)}, \quad n_m(\mathbf{l}) := n(\mathbf{l}) - \sum_{k < m} (l_k - 1) \quad (2.130)$$

defines an index of \mathbf{l} inside of $\{\mathbf{k} \mid n(\mathbf{k}) = n(\mathbf{l})\}$ (by means of parameterized simplex enumeration, compare [Feu05]) and

$$r_1(\mathbf{i}) := \sum_{j=1}^d \lfloor \frac{i_j}{2} \rfloor \prod_{k < j} 2^{l_k - 1} \quad (2.131)$$

2 A Sparse Grid Approximation Algebra

defines an index of \mathbf{i} inside of W_1 . The value

$$P^{n,d} := |\{\mathbf{l} \in \mathbb{N}_{>0}^d \mid n(\mathbf{l}) \leq n\}| \quad (2.132)$$

is the number of integer vectors in a simplex defined by $n(\mathbf{l}) \leq n$. It can be computed once by means of the recurrence formula

$$P^{n,d} = \sum_{m=1}^n P^{m,d-1}, \quad P^{n,1} = n \quad (2.133)$$

which can be shown by complete induction with respect to d . The precomputation step includes the computation of all values $|G_k^m|$, $P^{k,m}$ for $k = 0, \dots, n$ and $m = 1, \dots, d$ in time $\mathcal{O}(dn^2)$ and memory $\mathcal{O}(dn)$ where n is the highest required level⁶. The complete algorithm to compute $r_I(\mathbf{l}, \mathbf{i})$ is summarized in Algorithm 2, together with the setup phase in Algorithm 3.

Algorithm 2 Computation of the unique hash $r_I(\mathbf{l}, \mathbf{i})$ for inner sparse grid points according to (2.129).

Input: (\mathbf{l}, \mathbf{i}) belonging to an inner grid point (i.e. $l_m \geq 1$, i_m odd)

Output: $r_I(\mathbf{l}, \mathbf{i}) \in \mathbb{N}_0$ in time $\mathcal{O}(d)$

```

1:  $r_{n(\mathbf{l})}(\mathbf{l}) := 0$ 
2:  $r_1(\mathbf{i}) := 0$ 
3:  $n := 1$ 
4:  $\hat{d} := 0$ 
5: for  $m = d; m \geq 2; m = m - 1$  do
6:    $\hat{d} := \hat{d} + 1$ 
7:    $n := n + l_m - 1$ 
8:    $r_{n(\mathbf{l})}(\mathbf{l}) := r_{n(\mathbf{l})}(\mathbf{l}) + P^{n-1, \hat{d}}$ 
9:    $r_1(\mathbf{i}) := i_m \operatorname{div} 2 + 2^{l_m-1} \cdot r_1(\mathbf{i})$  // Telescope sum
10: end for
11:  $r_1(\mathbf{i}) := i_1 \operatorname{div} 2 + 2^{l_1-1} \cdot r_1(\mathbf{i})$ 
12:  $n := n + l_1 - 1$ 
13:  $r_I(\mathbf{l}, \mathbf{i}) := |G_{n-1}^d| + 2^{n-1} \cdot r_{n(\mathbf{l})}(\mathbf{l}) + r_1(\mathbf{i})$ 

```

The unique hashes $r_I(\mathbf{l}, \mathbf{i})$ are shown in Figure 2.10 (left) for a regular sparse grid of level $n = 4$ in dimension $d = 2$. We see that it works in a level wise ordering: the first index is associated to the point $(1, \dots, 1|1, \dots, 1)$, the following indices make up $n(\mathbf{l}) = 2$ and so on.

Note that $r_{n(\mathbf{l})}(\mathbf{l})$ can be used as well to create a unique hash for level indices only: simply eliminate everything which depends on the space index \mathbf{i} and replace the final $|G_{n-1}^d|$ by $P^{n-1,d}$. A variant of such a unique hashing is used at the end of this section to create a dimension adaptive data structure. We summarize the unique hash for level indices of the form $\mathbf{l} = (l_1, \dots, l_d)$ and $l_m = l_{\min}, l_{\min} + 1, l_{\min} + 2, \dots \in l_{\min} + \mathbb{N}_0$ for smallest level $l_{\min} \in \mathbb{Z}$ in Algorithm 4. The unique level index hashes $\tilde{r}(\mathbf{l})$ are illustrated

⁶See (2.23) for how to compute $|G_k^m|$.

Algorithm 3 Setup phase to precompute values for Algorithm 2

Input: n, d
Output: one array for $P^{k,m}$ and one for $|G_k^m|$, $m = 1, \dots, d$, $k = 0, \dots, n$.

```

1: for  $k = 0, \dots, n$  do
2:   set  $P^{k,1} := k$ , and  $|G_k^1| := 2^k - 1$  //  $G_0^d$  is never used
3: end for
4: for  $m = 2, \dots, d$  and  $k = 0, \dots, n$  do
5:   compute  $P^{k,m} := \sum_{q=1}^k P^{q,m-1}$  and  $|G_k^m| = \sum_{q=0}^{k-1} 2^q |G_{k-q}^{m-1}|$  // compare (2.23)
6: end for
    
```

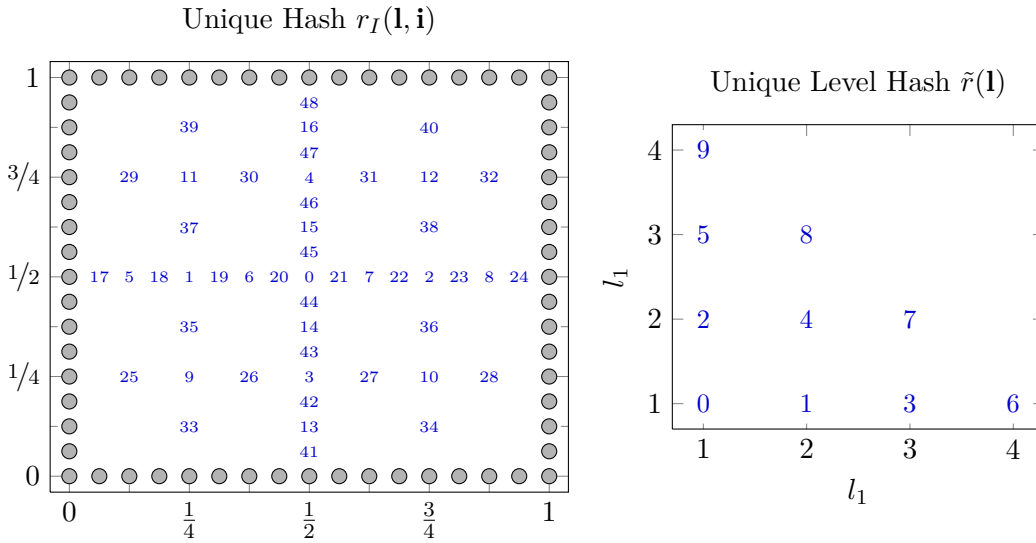


Figure 2.10: Unique hashes $r_I(\mathbf{l}, \mathbf{i})$ for points of a regular sparse grid (left) and level only unique hashes for $1 \leq |\mathbf{l}|_1 - d + 1 \leq 4$ (right).

Algorithm 4 Unique hashing for level indices only

Input: lowest allowed level $l_{\min} \in \mathbb{Z}$
Input: Level index $\mathbf{l} = (l_1, \dots, l_d)$, $l_m \in l_{\min} + \mathbb{N}_0$
Output: unique hash $\tilde{r}(\mathbf{l})$ such that $\tilde{r}(l_{\min}, \dots, l_{\min}) = 0$

```

1:  $\tilde{r}(\mathbf{l}) := 0$ 
2:  $n := 1$ 
3:  $\hat{d} := 0$ 
4: for  $m = d; m \geq 1; m = m - 1$  do
5:    $\hat{d} := \hat{d} + 1$ 
6:    $n := n + l_m - l_{\min}$ 
7:    $\tilde{r}(\mathbf{l}) := \tilde{r}(\mathbf{l}) + P^{n-1, \hat{d}}$ 
8: end for
9: return  $\tilde{r}(\mathbf{l})$  //  $P^{n-1, d}$  = number of points up to  $n - 1$  plays role of  $|G_{n-1}^d|$ 
    
```

2 A Sparse Grid Approximation Algebra

in Figure 2.10 (right) for $1 \leq |\mathbf{l}|_1 - d + 1 \leq 4$, determined with Algorithm 4 and $l_{\min} = 1$.

The unique hash $r_B(\mathbf{l}, \mathbf{i})$ for boundary nodes (\mathbf{l}, \mathbf{i}) , i.e. points for which $l_m = 0$ for at least one direction, is considerably more involved: it should work in a level wise fashion, and it should be independent of any given grid. Consequently, it relies on the recursive definition of sparse grids on the boundary, enumerates each boundary segment separately (and separately for each level) and uses the already determined inner hash $r_I(\mathbf{l}, \mathbf{i})$ inside of boundary manifolds. Such a unique hash has been derived thoroughly in [Feu05]; we simply summarize how to use it with a brief description how each occurring term can be interpreted. The key here is that most quantities can be precomputed and the remaining values are computable in an $\mathcal{O}(d)$ ‘for’ loop.

Let $K(\mathbf{l}) := |\{j \mid l_j = 0\}|$ denote the number of zeros in \mathbf{l} . Remind that $n(\mathbf{l}) = |\mathbf{l}|_1 - (d - K(\mathbf{l})) + 1$ for a boundary level (compare (2.18)), with the special case $n(\mathbf{0}) = 0$. Furthermore, let $P^{<n} := |\underline{G}_{n-1}^d| - |G_{n-1}^d|$ denote the number of regular sparse grid *boundary* points of lesser level than n and $I_n^d := |G_n^d| - |G_{n-1}^d|$ the number of regular sparse grid *inner* points on level n (only). Let $\text{BIN}(\mathbf{l}) := (b_1, \dots, b_d) \in \{0, 1\}^d$ denote the zero/non-zero bit pattern of \mathbf{l} , $b_m := \delta_{0, l_m}$. Let $P(d, k, n)$ be the number of regular sparse grid boundary points of fixed level $n(\mathbf{l}) = n$ and number of zero components $K(\mathbf{l}) = k$ (we will find its value later). Let

$$R_{\mathbf{l}}(\mathbf{i}) := \sum_{\substack{m=1, \dots, d \\ l_m=0}} i_m \prod_{\substack{k=1, \dots, m \\ l_k=0}} 2 \quad (2.134)$$

denote a unique hash of the space index \mathbf{i} for directions $l_m = 0$. In order to define a sequence for the set of same zero count K , $\{\mathbf{k} \in \mathbb{N}_0^d \mid K(\mathbf{k}) = K\}$, we use a map $r_K: \{0, 1\}^d \rightarrow \mathbb{N}_0$ which maps $\text{BIN}(\mathbf{l})$ to a unique hash. This map will be defined below. Finally, let $(\bar{\mathbf{l}}, \bar{\mathbf{i}})$ denote only those components of (\mathbf{l}, \mathbf{i}) for which $l_m \neq 0$.

Theorem 2.2.1 (Unique hash $r_B(\mathbf{l}, \mathbf{i})$). *Let (\mathbf{l}, \mathbf{i}) be a regular sparse grid point on the boundary, i.e. $l_m = 0$ for at least one $m \in \{1, \dots, d\}$ and $l_m = 0 \Rightarrow i_m \in \{0, 1\}$, $l_m > 0 \Rightarrow i_m$ odd. Then, $r_B(\mathbf{l}, \mathbf{i})$ defined by*

$$r_B(\mathbf{l}, \mathbf{i}) := P^{<n(\mathbf{l})} + \sum_{k=K(\mathbf{l})+1}^{d-1} P(d, k, n(\mathbf{l})) + (2^{K(\mathbf{l})} \cdot r_K(\text{BIN}(\mathbf{l})) + R_{\mathbf{l}}(\mathbf{i})) \cdot I_n^{d-K(\mathbf{l})} + r_I(\bar{\mathbf{l}}, \bar{\mathbf{i}}), \quad (2.135)$$

together with the special case

$$r_B((0, \dots, 0), \mathbf{i}) := \sum_{m=1}^d i_m \prod_{k < m} 2, \quad (2.136)$$

defines a unique hash among all such boundary points. The unique hash is actually an indexing of all boundary points of a regular sparse grid in a level wise ordering.

Proof. The proof follows constructively from the derivation in [Feu05]. \square

To actually compute (2.135), we need formulas for $P(d, k, n)$ and $r_K(\text{BIN}(\mathbf{1}))$. Fortunately, the quantities $P^{<n(\mathbf{1})} + \sum_{k=K(\mathbf{1})+1}^{d-1} P(d, k, n(\mathbf{1}))$, $I_n^{d-K(\mathbf{1})}$ and r_K can be precomputed for fast access such that only the simple loops for $R_{\mathbf{1}}(\mathbf{i})$ and $r_I(\bar{\mathbf{1}}, \bar{\mathbf{i}})$ need to be computed to evaluate $r_B(\mathbf{1}, \mathbf{i})$.

Lemma 2.2.3. *It holds*

$$P(d, k, n) = I_n^{d-k} \cdot 2^k \cdot \left(\max_{\substack{z \in \{0,1\}^d \\ |z|_1=k}} r_K(z) + 1 \right). \quad (2.137)$$

Proof. See [Feu05]. □

Algorithm 5 Pre-compute $r_K(z)$ and $\max_{|z|_1=k} r_K(z) + 1$ for all $z \in \{0, 1\}^d$ in an array $A[\cdot]$ of size 2^d . Then, $r_K(\text{BIN}(\mathbf{1})) = A[z_l]$ with $z_l = \sum_{i=0}^{d-1} b_{i+1} 2^i$ and $b_i = 0$ for $l_i = 0$ and $b_i = 1$ for $l_i \neq 0$.

Input: $d \leq$ word length of the computer (typically 32 or 64)

Output: array $A[\cdot]$ of size 2^d such that $A[z_l] = r_K(\text{BIN}(\mathbf{1}))$

Output: array $C[\cdot]$ of size $d + 1$ such that $\max_{z, |z|_1=k} r_K(z) + 1 = C[d - k]$.

1: $\text{resize}(A, 2^d)$

2: $\text{resize}(C, d + 1)$ // $C[k]$ = the so far largest index for numbers z with k zeros

3: **for all** $k = 0, \dots, d$ **do**

4: $C[k] := 0$

5: **end for**

6: **for** $j := 0; j < 2^d; j := j + 1$ **do**

7: $n :=$ count zeros in binary representation of the integer j

8: $A[j] := C[n]$

9: $C[n] := C[n] + 1$

10: **end for**

Algorithm 5 yields a possible implementation to pre-compute $r_K(\text{BIN}(\mathbf{1}))$, compare [Feu05]. Once the setup step is complete, it holds $r_K(\text{BIN}(\mathbf{1})) = A[z_l]$ where $z_l = \sum_{i=0}^{d-1} \delta_{0, l_{i+1}} 2^i$ is an integer number determined from $\text{BIN}(\mathbf{1})$; it is used as index into the precomputed array $A[\cdot]$. Note that Algorithm 5 also yields the required quantity $\max_{|z|_1=k} r_K(z) + 1$ as a side product: the maximum is stored in the output array at $C[d - k]$. Consequently, we can pre-compute the quantities

$$B[\tilde{n}][K] := P^{<\tilde{n}} + \sum_{k=K}^{d-1} P(d, k, \tilde{n}), \quad \tilde{n} = 1, \dots, n; \quad K = 1, \dots, d, \quad (2.138)$$

and

$$I[m][\tilde{n}] := I_n^m = |G_n^m| - |G_{n-1}^m|, \quad m = 1, \dots, d; \quad \tilde{n} = 1, \dots, n \quad (2.139)$$

in arrays of size $d \cdot n$ using Lemma 2.2.3 and again the recurrence formulas (2.23) and (2.25) to evaluate $P^{<\tilde{n}} = |G_{\tilde{n}-1}^d| - |G_{\tilde{n}-1}^d|$ and $I[m][\tilde{n}]$. The complete unique hash can

Algorithm 6 Computation of the unique hash $r_B(\mathbf{l}, \mathbf{i})$ for boundary points

Input: a boundary sparse grid point (\mathbf{l}, \mathbf{i}) , i.e. $l_m = 0$ for at least one direction

Input: precomputed values of Algorithm 5, $B[\cdot][\cdot]$ and $I[\cdot][\cdot]$

Output: the unique hash $r_B(\mathbf{l}, \mathbf{i}) \in \mathbb{N}_0$

```

1: if  $\mathbf{l} = (0, \dots, 0)$  then
2:    $r_B(\mathbf{l}, \mathbf{i}) := 0$ 
3:   for  $m = d, m \geq 1, m := m - 1$  do
4:      $r_B(\mathbf{l}, \mathbf{i}) := i_m + 2 \cdot r_B(\mathbf{l}, \mathbf{i})$ 
5:   end for
6:   return  $r_B(\mathbf{l}, \mathbf{i})$ 
7: end if
8:  $R_1(\mathbf{i}) := 0; z_l := 0; r_1(\mathbf{i}) := 0; r_{n(\mathbf{l})}(\mathbf{l}) := 0; K := 0; \hat{d} := 0$ 
9:  $n := 1$ 
10: for  $m := d; m \geq 1; m := m - 1$  do
11:   if  $l_m = 0$  then
12:      $R_1(\mathbf{i}) := i_m + 2 \cdot R_1(\mathbf{i})$ 
13:      $z_l := z_l + 2^{d-i}$  // in bit shifts:  $z_l |= (1 \ll (d-i))$ 
14:      $K := K + 1$ 
15:   else
16:      $\hat{d} := \hat{d} + 1$ 
17:      $n := n + l_m - 1$ 
18:      $r_{n(\mathbf{l})}(\mathbf{l}) := r_{n(\mathbf{l})}(\mathbf{l}) + P^{n-1, \hat{d}}$ 
19:      $r_1(\mathbf{i}) := i_m \text{ div } 2 + 2^{l_m-1} \cdot r_1(\mathbf{i})$ 
20:   end if
21: end for
22:  $r_{n(\mathbf{l})}(\mathbf{l}) := r_{n(\mathbf{l})}(\mathbf{l}) - P^{n-1, \hat{d}}$  // we added one too much, compare Algorithm 2
23:  $r_I(\bar{\mathbf{l}}, \bar{\mathbf{i}}) := 2^{n-1} \cdot r_{n(\mathbf{l})}(\mathbf{l}) + r_1(\mathbf{i})$ 
24:  $r_B(\mathbf{l}, \mathbf{i}) := B[n][K] + (2^K A[z_l] + R_1(\mathbf{i})) \cdot I[d-K][n] + r_I(\bar{\mathbf{l}}, \bar{\mathbf{i}})$ 
25: return  $r_B(\mathbf{l}, \mathbf{i})$ 

```

then be computed by means of Algorithm 6. The resulting unique hashes are shown in Figure 2.11 for boundary points of a regular sparse grid of level $n = 4$ in dimension two. The first four points are the corners, followed by level 1 near $x = 0.5$ and $y = 0.5$, then level 2 and so on.

The pre computation step of Algorithm 5 requires time $\mathcal{O}(\max(d2^d, d \cdot n^2))$ and memory $\mathcal{O}(\max(2^d, d \cdot n))$ for one fixed dimension d . For dimensions of about $d \approx 15$ and realistic levels, this makes just a few kilobytes (say, 100kb). Note that the 2^d occurring in the memory pre-allocation poses no serious restriction as long as the data structure is used for d -dimensional (adaptive) approximation since there are 2^d corner points anyway. Consequently, the pre-allocated memory *per grid point* is negligible.

Taking the unique hashes for inner- and boundary points together, we thus have a complete memory representation for sparse grids which is optimal for regular sparse

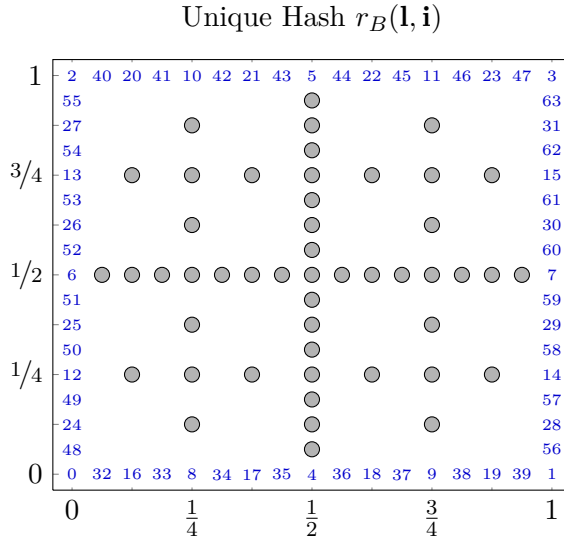


Figure 2.11: Unique hashes generated by the index map $r_B(\mathbf{l}, \mathbf{i})$, applied to points of a regular sparse grid in dimension $d = 2$ and level $n = 4$.

grids with respect to memory usage. It is highly flexible and has good access times.

Dimension Adaptive Data Structures

So far, the proposed unique hashing data structures yield highly flexible, compactly stored tools with good $\mathcal{O}(d)$ access times for fully d -dimensional approximation problems. However, the dimension adaptive setting developed in Chapter 4 has a further requirement: dimension adaptive methods need even more sparsity than a standard (space adaptive) sparse grid. We will see in Chapter 4 that dimension adaptivity relies on multi index representations (\mathbf{l}, \mathbf{i}) where most directions are fixed to $(l_m, i_m) = (0, 0)$. This particular point is associated with the constant basis function. If a function has low effective dimension (compare Section 4.1.2), the constant directions play a key role⁷. A dimension adaptive grid can have a huge nominal dimension, but most index components will have $l_m = -1$.

In such a scenario, the unique hashing approach as we used it so far is inappropriate: the memory usage of 2^d required for $r_B(\mathbf{l}, \mathbf{i})$ is the most obvious reason, but a more subtle problem is the approach to index all points of a regular sparse grid: the required range of numbers will exceed the available integer range of standard computers if the nominal dimension is huge, even though most of these indices will never be used.

A possible alternative which keeps the advantages of unique hashing is to introduce *separate* hashes for level- and space indices and to work with a tuple $(\tilde{r}(\mathbf{l}), r_1(\mathbf{i})) \in \mathbb{N}_0^2$. An access to a grid point (\mathbf{l}, \mathbf{i}) requires two lookups: one to find data associated with

⁷Note that Chapter 4 identifies the constant using the artificial level index $l_m = -1$ (which is equivalent to $(l_m, i_m) = (0, 0)$).

level $\mathbf{1}$ (using the first unique hash $\tilde{r}(\mathbf{1})$) and then, inside of the identified structure, the coefficient $r_1(\mathbf{i})$ which is a trivial generalization of (2.131)⁸. A possible unique hash $\tilde{r}(\mathbf{1})$ for level indices has already been discussed in Algorithm 4: it is unique among all integer multi indices $(l_1, \dots, l_d) \in l_{\min} + \mathbb{N}_0^d$ where $l_{\min} = -1$ in our case. It relies on a simplex enumeration. Now, the unique hashing procedure $(\tilde{r}(\mathbf{1}), r_1(\mathbf{i}))$ is suitable for larger dimensions since the simplex enumeration grows less fast. Nevertheless, there are still $\mathcal{O}(n^d)$ integer points in the simplex defined by $1 \leq |\mathbf{1}|_1 - d + 1 \leq n$, compare the proof of Lemma 2.2.2. If we have at most 2^q integers available, we find $n^d < 2^q \Leftrightarrow d < q / \log_2 n$. For example $q = 64$ and $n = 10$ allows $d < 20$. Thus, the unweighted unique hash has its limitations when it comes to dimension adaptivity, but it allows simple memory allocation and fast hashing.

Possible alternatives are to use non-unique hashing. Then, hash map collisions are resolved with the help of *sparse vector representations* of $\mathbf{1}$, i.e. without explicitly storing the $l_m = -1$ components, compare [Nah05]. Here, $\tilde{r}(\mathbf{1}) \bmod m$ can serve as a possible non-unique hash function. The implementation used so far relies on unique hashing.

Thus, dimension adaptive grids can be managed either by unique hashing with access time $\mathcal{O}(d)$ per node and total memory usage $\mathcal{O}(N)$ or by means of non-unique hashing which has access time $\mathcal{O}(d)$ per node and total memory usage $\mathcal{O}(d_{\text{eff}}N)$ where $d_{\text{eff}} = \max_{\mathbf{1}} \{m \mid l_m \neq -1\}$ is the highest used effective dimension.

2.3 Approximative Algebraic Operations

2.3.1 Motivation and Overview

We turn to a study of common operations like ‘+’, ‘−’, ‘×’, ‘/’ and the functional operations $f_1(f_2(x))$ and $\int K(x, y) f(y) dy$ applied to discretized functions f_1, f_2, K , and f . Our aim here is to provide a toolbox to be used in linear and nonlinear sparse grid approximation applications. For example, consider integro partial differential equations of the form

$$u_t - Lu + \int K(x, y; u) f(y; u) dy = r \quad (2.140)$$

which contain an (elliptic) partial differential operator Lu and a u -dependent integral transformation. One possible solution is an explicit approach for the integro term which requires to deal with discretized kernels $K(x, y; u)$ and integrands $f(y; u)$. The differential parts can then be solved by means of sparse grid methods, and the integro term needs to be computed by means of an approximate integral transformation (which will also handle the pointwise multiplication $K \cdot f$). A different field of potential interest is to solve nonlinear approximation problems of the form

$$(\bar{f}_1, \bar{f}_2) := \operatorname{argmin}_{f_1 \in V_1, f_2 \in V_2} \|F(x) - f_1(f_2(x))\| \quad (2.141)$$

⁸Now, the special level index $l_m = -1$ allows one possible point $i_m = 0$ whereas the level index $l_m = 0$ allows the only possible point $i_m = 1$.

for given spaces V_1 and V_2 and given F . Here, finite dimensional spaces V_i lead to approximative function concatenation $f_1(f_2(x))$ and the nonlinear optimization can work by means of coefficient vectors.

We discuss how to perform pointwise operations $f_1(x) \square f_2(x)$, $\square \in \{+, -, \times, /\}$, $f_1(f_2(x))$ and $\int K(x, y) f(y) dy$ if the involved functions are living on sparse grid spaces. The focus of our consideration is how such operations can be applied with a beneficial cost–gain ratio where the cost measure is the time and memory needed to compute and represent the result. The gain measure indicates how inaccuracies of operands or the inexact methods influence the result.

Our analysis complements and refines results presented in [Gri98] for the elementary operations $\{+, -, \times, /\}$ and the concatenation presented in [MgF07] and contains new insight especially into the consistency error and methods to compute the integral transformation.

2.3.2 Approximative Pointwise Operations

In the following, let $f_1, f_2 \in V$ be given functions defined on a rectangular domain $\Omega = \otimes [a_i, b_i]$. Let \square be an elementary pointwise operation of two arguments, and

$$f_3(x) = f_1(x) \square f_2(x) \quad (2.142)$$

the result.

Furthermore, let f_1^h and f_2^h be (adaptive) approximations in sparse grid subspaces $V_1^h \subset V$, $V_2^h \subset V$, respectively. Since sparse grids are always defined on the unit cube $[0, 1]^d$, f_1^h and f_2^h are defined by means of a linear coordinate transformation $\kappa: [0, 1]^d \rightarrow \Omega$ using $f_i^h = \tilde{f}_i^h \circ \kappa^{-1}$ where \tilde{f}_i^h is defined on $[0, 1]^d$.

Before we actually study approximations of f_3 based on f_1^h and f_2^h , we introduce preliminary conventions and present common results.

Definition 2.3.1 (Pointwise Data Error). Given functions $f_1 \in V_1$, $f_2 \in V_2$, a binary map $M = M(\cdot, \cdot): V_1 \times V_2 \rightarrow V_3$ and approximations $f_1^h \approx f_1$, $f_2^h \approx f_2$, the entity

$$M(f_1, f_2)(x) - M(f_1^h, f_2^h)(x) \quad (2.143)$$

or its norm is called the (pointwise) *data error* at an (arbitrary) point x .

Definition 2.3.2 (Consistency Error). Let $f_1 \in V_1$ and $f_2 \in V_2$ be given functions and $M: V_1 \times V_2 \rightarrow V_3$ a binary map with an approximation $\tilde{M} \approx M$.

Then, the entity

$$M(f_1, f_2) - \tilde{M}(f_1, f_2) \quad (2.144)$$

is called the *consistency error* of \tilde{M} .

We are now interested in obtaining an approximation for $f_3 = f_1 \square f_2$ by means of f_1^h and f_2^h . A simple approach is feasible for every operation on discrete functions: use the exact operation ‘ \square ’, applied to the basis expansion of f_1^h and f_2^h . Its properties are summarized in

Lemma 2.3.1. *The abstract operation $f_3^{h,*}(x) := f_1^h(x) \square f_2^h(x)$ which simply employs the arbitrary point evaluation routine $f_i^h(x)$ requires $\mathcal{O}(n_1^d + n_2^d)$ arithmetic operations for every evaluation of $f_3^{h,*}$. Here n_1 and n_2 are the sparse grid levels of f_1 and f_2 , respectively.*

The pointwise error depends only on the data error

$$|f_1(x) \square f_2(x) - f_1^h(x) \square f_2^h(x)|, \quad (2.145)$$

provided the point evaluation algorithm is numerically stable.

Proof. The cost complexities follow immediately from Lemma 2.2.2 and since there is no consistency error, we only have to deal with input (data) errors. \square

Note that $f_3^{h,*}(x) = f_1^h(x) \square f_2^h(x)$ as such is an abstract construction which is not directly related to a (sparse) grid, it may not even be possible to find a finite basis expansion for $f_3^{h,*}$ at all. Nevertheless, it might be beneficial to use $f_3^{h,*}$, especially if f_1^h and f_2^h both require few degrees of freedom and allow fast point evaluations.

The alternative is to discretize the result $f_3^{h,*}$ on a properly chosen grid. Since ‘ \square ’ operates pointwise on function values, a discretization of $f_3^{h,*}$ can rely on adaptive re-interpolation

$$f_3^h := f_1^h \square_h f_2^h := I[f_3^{h,*}] = I[f_1^h \square f_2^h] \quad (2.146)$$

where $I[\cdot]$ denotes the sparse grid interpolation operator on a properly chosen grid. Such a grid G_3 will depend on the grids of f_1^h and f_2^h . A good starting point is the grid union of both. Before we come back to the choice of G_3 , we state Lemma 2.3.2:

Algorithm 7 Re-interpolation of a pointwise operation \square

Input: Pointwise binary map \square

Input: Grids G_1, G_2, G_3 discretizing V

Input: Functions f_1^h on G_1 and f_2^h on G_2

Output: $f_3^h = I_{G_3}[f_1^h \square f_2^h]$

- | | |
|--|---|
| 1: $\tilde{f}_1 := \text{InterpolateOrRestrict}(f_1^h \text{ on } G_3)$ | // $\mathcal{O}(G_1 + G_3)$ |
| 2: $\tilde{f}_2 := \text{InterpolateOrRestrict}(f_2^h \text{ on } G_3)$ | // $\mathcal{O}(G_2 + G_3)$ |
| 3: transform \tilde{f}_1 to nodal values | // $\mathcal{O}(G_3)$ |
| 4: transform \tilde{f}_2 to nodal values | // $\mathcal{O}(G_3)$ |
| 5: compute $f_3^h(x) = \tilde{f}_1(x) \square \tilde{f}_2(x)$ for each $x \in G_3$ | // $\mathcal{O}(G_3 \cdot \square)$ |
| 6: transform f_3^h to the hierarchical basis | // $\mathcal{O}(G_3)$ |
| 7: return f_3^h | |
-

Lemma 2.3.2 (Cost And Error of a General Interpolated Pointwise Operation). *Let $f_1, f_2 \in V$ be given functions and $f_1^h \in V_1^h \subset V$, $f_2^h \in V_2^h \subset V$ be approximations of f_1 and f_2 , respectively. The underlying grids G_1 and G_2 may be chosen adaptively and can be different. Furthermore, let ‘ \square ’ be a binary pointwise operation and f_3^h given as interpolant $f_3^h := I[f_1^h \square f_2^h] = f_1^h \square_h f_2^h$ on a (fixed) grid G_3 .*

2.3 Approximative Algebraic Operations

Then, f_3^h can be computed in time $\mathcal{O}(|G_3|(1 + |\square|) + |G_1| + |G_2|)$ and its pointwise error is characterized by both, a data error and a consistency error

$$\begin{aligned} & |f_1(x) \square f_2(x) - f_1^h(x) \square_h f_2^h(x)| \\ & \leq |f_1(x) \square f_2(x) - f_1^h(x) \square f_2^h(x)| + |f_1^h(x) \square f_2^h(x) - f_1^h(x) \square_h f_2^h(x)|. \end{aligned} \quad (2.147)$$

Here, $|\square|$ denotes the cost for one evaluation.

Proof. The cost complexity is proven constructively using Algorithm 7. The error remark is straight-forward. The InterpolateOrRestrict step of Algorithm 7 simply assigns hierarchical values of 0 to any new grid point (interpolation) or discards coefficients which exists in G_1 but not in G_3 (restrict). \square

The result grid G_3 should be chosen such that the consistency error is of the same order as the data error (which cannot be avoided anyway). The approach in [Gri98] is presented for $\square \in \{+, -, \times, /\}$ and suggests using the grid union $G_1 \cup G_2$ followed by a final grid compression. Unfortunately, the grid union without further refinement might yield a consistency error which is several orders of magnitude *larger* than the data error. An example of such an effect is $f_1(x, y) = x^2$, $f_2(x, y) = y^2$ and $\square = \times$. Figure 2.12 (top left) and Figure 2.12 (top right) show f_1^h and f_2^h , respectively. The data errors are both $9.5 \cdot 10^{-7}$, computed using the relative L_∞ norm and an adaptive threshold $\epsilon^{(\text{rel})} = 1 \cdot 10^{-6}$. Figure 2.12 (bottom left) shows the result f_3^h approximated on the grid union. Since f_1 and f_2 are inherently one-dimensional, the grid union is just a superposition of one-dimensional grids. However, it should be a two-dimension grid built by a (sparse) tensor product. Consequently, the consistency error is $4 \cdot 10^{-3}$. Figure 2.12 (bottom right) shows the corrected result obtained by the approach described in the following (Algorithm 8), its accuracy is of order $1 \cdot 10^{-6}$.

Algorithm 8 Adaptive Resolution of \square

Input: Pointwise binary map \square

Input: Functions f_1^h, f_2^h on grids G_1, G_2 , respectively

Input: Refinement target value $\epsilon = \epsilon^{(\text{abs})}$ or $\epsilon = \epsilon^{(\text{rel})}$

Output: Grid G_3 and $f_3^h = I_{G_3}[f_1^h \square f_2^h]$

- 1: $G_3^{(0)} := G_1 \cup G_2$ // $\mathcal{O}(|G_3^{(0)}|)$
 - 2: $(f_3^h, G_3) := \text{adaptTo}(f_1^h \square f_2^h)$ using the standard refine Algorithm 1 with the special “set values” routine of Algorithm 7 and $G_3^{(0)}$ as initial grid.
// $\mathcal{O}(|G_3^{(i)}| \cdot (1 + |\square|))$ for loop iteration i in Lemma 2.2.1
 - 3: **return** G_3 and f_3^h
-

In order to balance consistency and data error, we propose to use the following approach: logically, we compute f_3^h (and its grid G_3) by resolving $f_1^h \square f_2^h$ adaptively. This adaptive interpolation procedure is exactly the same as the one which might have been applied to compute f_1^h and f_2^h in the first place – but since $f_1^h \square f_2^h$ already exhibits a data error, the refinement should stop as soon as this accuracy is achieved. In order

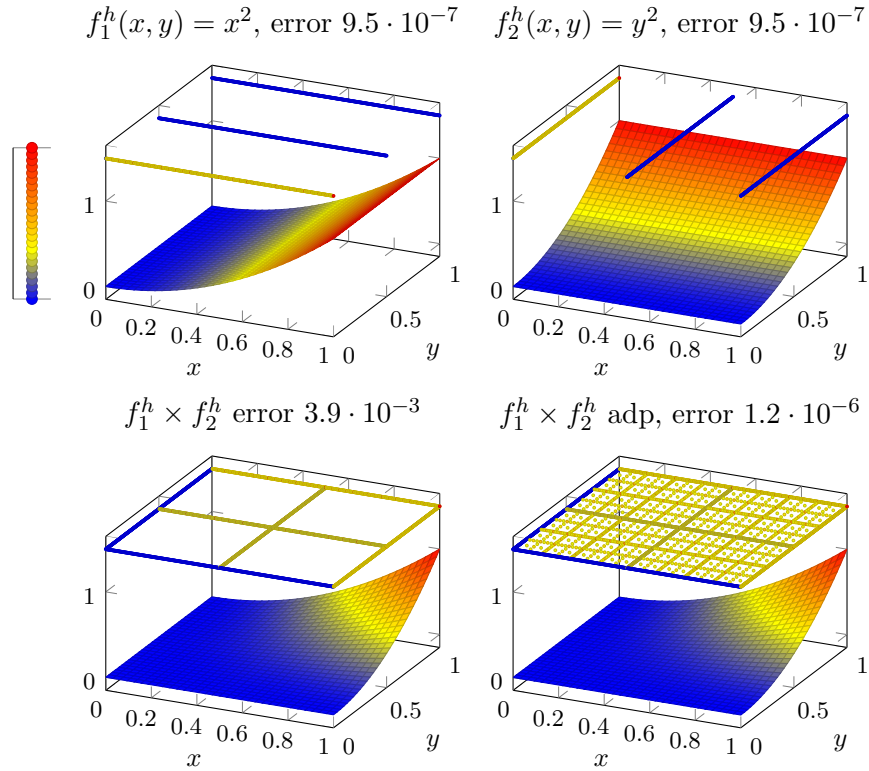


Figure 2.12: Computation of $f_1^h(x, y) \times f_2^h(x, y)$ once by means of the simple grid union (bottom left) and once by means of Algorithm 8 (bottom right); all experiments use $\epsilon^{(\text{rel})} = 1 \cdot 10^{-6}$. Grid points are colored according to their absolute value.

to avoid the costly point evaluation routine, the fast re-interpolation of \square according to Algorithm 7 should be a component of the adaptive refinement. The overall procedure is summarized in Algorithm 8, which expects the target accuracy as input argument and employs the re-interpolation in each step. Note that Algorithm 8 exhibits the same cost complexity as the generic adaption algorithm analyzed in Lemma 2.2.1. For sufficiently smooth $f_1 \square f_2$, it is linear in the final grid size $|G_3|$ using a geometric series argument.

The input value ϵ is yet to be determined. A reliable choice for ϵ is to make it small enough such that the consistency error is less than the data error. An efficient choice is one where both errors are balanced. We summarize results about the data error in the following.

The Data Error For Standard Operations

We start with a well-known definition to characterize error amplification.

Definition 2.3.3 (Condition of a function). Let $\mathcal{F}: \mathbb{R}^d \rightarrow \mathbb{R}$ be sufficiently smooth.

2.3 Approximative Algebraic Operations

Then, the vector

$$\kappa_{\text{abs}}(x) := \left(\left| \frac{\partial \mathcal{F}}{\partial x_i}(x) \right| \right)_{i=1, \dots, d} \quad (2.148)$$

is called the *absolute condition* of \mathcal{F} , whereas for $\mathcal{F}(x) \neq 0$ and $x \neq 0$,

$$\kappa_{\text{rel}}(x) := \left(\left| \frac{\partial \mathcal{F}}{\partial x_i}(x) \cdot \frac{x_i}{\mathcal{F}(x)} \right| \right)_{i=1, \dots, d} \quad (2.149)$$

denotes the *relative condition* of \mathcal{F} at x .

The condition expresses the amplification of either an absolute input error by means of

$$\mathcal{F}(x + \delta x) = \mathcal{F}(x) + \sum_{i=1}^d \frac{\mathcal{F}}{\partial x_i}(x) \cdot \delta x_i + \underbrace{R(x)}_{\|\cdot\| = \mathcal{O}(\|\delta x\|^2)} \quad (2.150)$$

or a relative input error $\delta x/x_i$ for $x_i \neq 0$ and $\mathcal{F}(x) \neq 0$,

$$\frac{\mathcal{F}(x + \delta x) - \mathcal{F}(x)}{\mathcal{F}(x)} = \sum_{i=1}^d \left(\frac{\mathcal{F}}{\partial x_i}(x) \cdot \frac{x_i}{\mathcal{F}(x)} \right) \cdot \frac{\delta x_i}{x_i} + \underbrace{\tilde{R}(x)}_{\|\cdot\| = \mathcal{O}(\|\delta x_i\|^2)}. \quad (2.151)$$

Condition numbers of the order $\mathcal{O}(1)$ are generally well-behaved while large numbers indicate huge error amplification. We find immediately the condition numbers for elementary operations.

Lemma 2.3.3 (Condition of elementary operations). *The following condition numbers characterize the pointwise data error for elementary operations:*

$$\mathcal{F}(a, b) = a + b, \quad \kappa_{\text{abs}} \equiv (1, 1)^T, \quad \kappa_{\text{rel}} = \left(\frac{|a|}{|a + b|}, \frac{|b|}{|a + b|} \right)^T; \quad (2.152)$$

$$\mathcal{F}(a, b) = a \cdot b, \quad \kappa_{\text{abs}} = (|b|, |a|)^T, \quad \kappa_{\text{rel}} = (1, 1)^T; \quad (2.153)$$

$$\mathcal{F}(a, b) = a/b, \quad \kappa_{\text{abs}} = \left(\frac{1}{|b|}, \frac{|a|}{|b^2|} \right)^T, \quad \kappa_{\text{rel}} = (1, 1)^T. \quad (2.154)$$

Usually, the amplification of relative errors is more important since a method should work on all scales. Here the case $a + b$ for $a \approx -b$ is the worst case of cancellation with huge relative errors whereas the multiplication and division is generally well-behaved. Note that the absolute error for multiplication can become very large for large arguments and the division is absolutely bad for either large $|a|$ or small $|b|$. The absolute error for a sum of two numbers as such is well-behaved (cancellation becomes a problem if the sum is just an intermediate result or if relative errors are required).

Lemma 2.3.3 allows a characterization for pointwise function operations $\{+, -, \times, /\}$ immediately. For the sake of completeness, we write the associated data errors explicitly.

Lemma 2.3.4 (The Data Error Of Elementary Pointwise Function Operations). *Let $f_1, f_2 \in V$ be given functions and $f_1^h \in V_1^h \subset V$, $f_2^h \in V_2^h \subset V$ be approximations on f_1, f_2 , respectively. Let $\epsilon_1(x)$ and $\epsilon_2(x)$ denote the absolute pointwise errors*

$$\epsilon_1(x) := |f_1(x) - f_1^h(x)| \leq \tilde{\epsilon}_1, \quad (2.155)$$

$$\epsilon_2(x) := |f_2(x) - f_2^h(x)| \leq \tilde{\epsilon}_2 \quad (2.156)$$

and $\epsilon_i^{(rel)}(x) := \epsilon_i(x)/|f_i(x)| \leq \tilde{\epsilon}_i^{(rel)}$ for $i = 1, 2$ the associated pointwise relative errors. Let $\square \in \{+, -, \times, /\}$.

Then, we find the data errors

$$|(f_1(x) + f_2(x)) - (f_1^h(x) + f_2^h(x))| \leq \epsilon_1(x) + \epsilon_2(x) \quad (2.157)$$

$$\leq \tilde{\epsilon}_1 + \tilde{\epsilon}_2, \quad (2.158)$$

$$\frac{|(f_1(x) + f_2(x)) - (f_1^h(x) + f_2^h(x))|}{|f_1(x) + f_2(x)|} \leq \frac{|f_1(x)|}{|f_1(x) + f_2(x)|} \tilde{\epsilon}_1^{(rel)} + \frac{|f_2(x)|}{|f_1(x) + f_2(x)|} \tilde{\epsilon}_2^{(rel)}, \quad (2.159)$$

$$|f_1(x) \cdot f_2(x) - f_1^h(x) \cdot f_2^h(x)| \leq |f_2(x)| \cdot \epsilon_1(x) + |f_1(x)| \cdot \epsilon_2(x) + \mathcal{O}(\tilde{\epsilon}_1^2 + \tilde{\epsilon}_2^2) \quad (2.160)$$

$$\leq |f_2(x)| \cdot \tilde{\epsilon}_1 + |f_1(x)| \cdot \tilde{\epsilon}_2 + \mathcal{O}(\tilde{\epsilon}_1^2 + \tilde{\epsilon}_2^2), \quad (2.161)$$

$$\frac{|f_1(x) \cdot f_2(x) - f_1^h(x) \cdot f_2^h(x)|}{|f_1(x) \cdot f_2(x)|} \leq \tilde{\epsilon}_1^{(rel)} + \tilde{\epsilon}_2^{(rel)} + \mathcal{O}(\tilde{\epsilon}_1^2 + \tilde{\epsilon}_2^2), \quad (2.162)$$

$$|f_1(x)/f_2(x) - f_1^h(x)/f_2^h(x)| \leq \frac{1}{|f_2(x)|} \tilde{\epsilon}_1 + \frac{|f_1(x)|}{|f_2(x)|^2} \tilde{\epsilon}_2 + \mathcal{O}(\tilde{\epsilon}_1^2 + \tilde{\epsilon}_2^2), \quad (2.163)$$

$$\frac{|f_1(x)/f_2(x) - f_1^h(x)/f_2^h(x)|}{|f_1(x)/f_2(x)|} \leq \tilde{\epsilon}_1^{(rel)} + \tilde{\epsilon}_2^{(rel)} + \mathcal{O}(\tilde{\epsilon}_1^2 + \tilde{\epsilon}_2^2). \quad (2.164)$$

Proof. The proof follows immediately by Lemma 2.3.3, together with (2.150) and (2.151) applied to $f_1(x)$ and $f_2(x)$. Note that here, x is considered to be exact, so we have $a \equiv f_1(x)$ and $b \equiv f_2(x)$.

Note that addition and subtraction do not depend on higher order error terms since the derivatives vanish. All other data errors are precise up to first order in the input errors. \square

Note that we recover the results for absolute data errors as they have been shown in [Gri98], with slight modifications for the multiplication: the bound in [Gri98] uses $\max(|f_i(x)|, |f_i^h(x)|)$ where we use $|f_i(x)|$. Our results extend those of [Gri98] in the sense that the operation's condition is a simple yet general formulation of the data error, which can be applied to other types of pointwise operations as well.

Choosing the Refinement Parameter For The Consistency Error

Given the information how input errors of f_1^h and f_2^h are amplified by \square , we can now configure the adaptive refinement parameter ϵ of our adaptive re-interpolation Algorithm 8. Generally, a well-conditioned operation \square allows to refine up to the input error

$\max\{\tilde{\epsilon}_1, \tilde{\epsilon}_2\}$. Thus, the adaptive multiplication routine can employ the same relative thresholds $\epsilon^{(\text{rel})}$ as have been used for the approximation of the f_i^h . The situation for sums or differences is even simpler:

Lemma 2.3.5. *Let $\square \in \{+, -\}$. Then, the consistency error is zero if the grid $G_3 = G_1 \cup G_2$ is chosen for the re-interpolation of $f_1^h(x) \square f_2^h(x)$.*

Proof. Since $f_i^h(x) = \sum_{(\mathbf{l}, \mathbf{i}) \in G_i} f_{\mathbf{l}, \mathbf{i}}^{(i)} \phi_{\mathbf{l}, \mathbf{i}}(x)$, we find

$$f_1^h(x) \square f_2^h(x) = \sum_{(\mathbf{l}, \mathbf{i}) \in G_1 \cup G_2} (f_{\mathbf{l}, \mathbf{i}}^{(1)} \square f_{\mathbf{l}, \mathbf{i}}^{(2)}) \phi_{\mathbf{l}, \mathbf{i}}(x) = I_{G_1 \cup G_2}[f_1^h \square f_2^h](x) \quad (2.165)$$

due to the linearity of the interpolation operator. Here, we assume that missing values are zero. \square

Thus, for sums and differences, we only need to perform the grid union, no adaptive threshold is necessary (we do not even need to switch basis representations). It may be beneficial to compress the result if single coefficients become small afterwards.

2.3.3 Approximative Function Concatenation

We will now study the function concatenation operation ‘ \circ ’. If we regard it as functional, it maps $\circ: V_{\text{outer}} \times \vec{V}_{\text{inner}} \rightarrow V$ using $f_{\text{outer}} \circ \vec{f}_{\text{inner}} \mapsto f_{\text{result}} = f_{\text{outer}}(\vec{f}_{\text{inner}}(\cdot))$ where the inner function space \vec{V}_{inner} can be vector valued, $\vec{f}_{\text{inner}} \in \vec{V}_{\text{inner}}$, $\vec{f}_{\text{inner}} = (f_{\text{inner},1}, \dots, f_{\text{inner},m}): \Omega \rightarrow \mathbb{R}^m$, and V_{outer} contains functions $V_{\text{outer}} \ni f_{\text{outer}}: \mathbb{R}^m \rightarrow \mathbb{R}$ or $f_{\text{outer}}: \Omega_{\text{outer}} \rightarrow \mathbb{R}$. Finally, V is a function space with functions $V \ni f_{\text{result}}: \Omega \rightarrow \mathbb{R}$.

We assume a rectangular domain for the inner functions, $\Omega = [a_1, b_1] \times \dots \times [a_d, b_d]$.

Given sparse grid approximations for all involved functions, f_{outer}^h and \vec{f}_{inner}^h , we are interested in a representation for $f_{\text{outer}}^h \circ \vec{f}_{\text{inner}}^h(x)$. The domains must be compatible. More precisely, we assume that \vec{f}_{inner}^h is continuous, so it takes its maximum and minimum values on its domain Ω . Furthermore, we assume that f_{outer}^h has been discretized on a (sub-) domain $\Omega_{\text{outer}}^h \subset \Omega_{\text{outer}}$ such that $\text{img } \vec{f}_{\text{inner}}^h \subseteq \Omega_{\text{outer}}^h$ in order to avoid extrapolation for the outer discrete function. In particular, Ω_{outer}^h will be a rectangular tensor product domain since sparse grids are always defined on cubes. It should be chosen as axis-parallel bounding rectangle for $\text{img } \vec{f}_{\text{inner}}^h$, i.e.

$$f_{\text{outer}}^h: \bigotimes_{j=1}^m [A_j, B_j] \rightarrow \mathbb{R} \quad (2.166)$$

where $A_j = \min(\text{img } f_{\text{inner},j}^h)$ and $B_j = \max(\text{img } f_{\text{inner},j}^h)$ for $j = 1, \dots, m$.

As for the case of pointwise operations, we have to deal with consistency errors and data errors. A consistency error occurs if we represent the result on a grid, $f_{\text{result}}^h = I_{G_3}[f_{\text{outer}} \circ \vec{f}_{\text{inner}}] =: f_{\text{outer}} \circ_h \vec{f}_{\text{inner}}$, and the data error is caused by the amplification of input errors.

Note that an abstract representation of the form $f_{\text{result}}^{h,*} := f_{\text{outer}}^h(\vec{f}_{\text{inner}}^h(x))$ which employs the point evaluation routine of Lemma 2.2.2 for every access might save a lot

of time compared to a grid representation due to its non-linearity. This is possible if outer and inner functions have different dimensionality and can be represented with few degrees of freedom. An example is the Gaussian, represented as concatenation of $f_{\text{outer}}(x) = \exp(-x)$ and $\vec{f}_{\text{inner}}(\vec{x}) = \vec{x}^T \vec{x}$. The outer function is one-dimensional and all inner components are superpositions of one-dimensional functions (and are thus effectively one-dimensional). However, the composition $\exp(-x^T x)$ is effectively high dimensional and requires considerably more degrees of freedom if represented on a grid (which is analyzed in more detail in Chapter 3.2.1). We will see the example of a Gaussian later in this section.

An abstract representation $f_{\text{result}}^{h,*}$ has no consistency error. Its data error is subject of

Lemma 2.3.6 (Data Error For Concatenation). *Let $\mathcal{F}: V \times \mathbb{R}^d \rightarrow \mathbb{R}$, $\mathcal{F}(f, x) := f(x)$ be the point evaluation functional and $V \subset C^2$. Let $f^h \in V^h$ be an approximation on f with (pointwise) error $f^h = f - \epsilon_f$ which is evaluated at the inexact point $x^h = x - \epsilon_x$.*

Then, the absolute input error $(\epsilon_f(\cdot), \epsilon_x)$ propagates according to

$$\mathcal{F}(f, x) - \mathcal{F}(f^h, x^h) \approx 1 \cdot \epsilon_f(x^h) + f'(x)\epsilon_x, \quad (2.167)$$

and the relative input errors $\epsilon_f^{(rel)}(\cdot) = \epsilon_f(\cdot)/|f(\cdot)|$, $\epsilon_x^{(rel)} := \epsilon_x/\|x\|$ for $f(x) \neq 0$, $x \neq 0$ are amplified according to

$$\frac{\mathcal{F}(f, x) - \mathcal{F}(f^h, x^h)}{|\mathcal{F}(f, x)|} \approx 1 \cdot \epsilon_f^{(rel)}(x^h) + \frac{\|x\| \cdot f'(x)}{|f(x)|} \cdot \epsilon_x^{(rel)}. \quad (2.168)$$

Both equations are precise up to first order in $\|\epsilon_x\|$.

Proof. We find

$$\begin{aligned} \mathcal{F}(f, x) - \mathcal{F}(f^h, x^h) &= \mathcal{F}(f, x) - \mathcal{F}(f, x^h) + \mathcal{F}(f, x^h) - \mathcal{F}(f^h, x^h) \\ &= f(x) - f(x^h) + f(x^h) - f^h(x^h) \\ &\approx f'(x)\epsilon_x + 1 \cdot \epsilon_f(x^h) \end{aligned}$$

using $f(x^h) = f(x) - f'(x)\epsilon_x + R(x)$ with rest term $R(x)$ of order $\mathcal{O}(\|\epsilon_x\|^2)$ due to the smoothness assumptions. The relative error follows using division by $|f(x)|$. \square

Thus, we find $\tilde{\kappa}_{\text{abs}}(f, x) := (\|f'(x)\|, 1)$ and $\tilde{\kappa}_{\text{rel}} = (\|\frac{\|x\|}{f(x)} f'(x)\|, 1)$ as generalized condition values for the evaluation⁹. The result yields the data error of $f_{\text{outer}} \circ \vec{f}_{\text{inner}}(x)$ using $\mathcal{F}(f_{\text{outer}}^h, \vec{f}_{\text{inner}}^h(x))$. The parameter x of the inner function is exact and is not of interest for the data error.

Our result complements the data error presented in [MgF07] in which higher order error terms are handled as well. The upper bound derived in [MgF07] appears to involve the *mixed* first derivative of the outer function, $\frac{\partial^d}{\partial x_1 \dots \partial x_d} f$, instead of f' , compare [MgF07].

⁹Note that the Gâteaux derivative of \mathcal{F} in direction of f is, indeed, the identity.

In summary, the data error is characterized by the amplification of the inner error which is of magnitude $\|f'_{\text{outer}}\|$. The error of the outer function as such enters only with factor 1; it will not be amplified.

In case we need a (finite) expansion of $f_{\text{outer}}^h \circ \vec{f}_{\text{inner}}^h$, we introduce a consistency error. We propose to approximate $f_{\text{outer}}^h \circ \vec{f}_{\text{inner}}^h$ using a re-interpolation similar to the pointwise function maps discussed in Section 2.3.2. Thus, we determine a grid G_{result} and an interpolant $f_{\text{result}}^h := I_{G_{\text{result}}}[f_{\text{outer}}^h \circ \vec{f}_{\text{inner}}^h]$. A naive approach to compute G_{result} and f_{result}^h is to form the abstract composition $f_{\text{result}}^{h,*}$ and apply the adaptive interpolation procedure of Algorithm 1 using the point evaluation routine of Lemma 2.2.2. But since each evaluation $f_{\text{result}}^h(x)$ requires a nodal value $\vec{f}_{\text{inner}}^h(x)$, it is more cost-effective to compute all these nodal values by means of the fast basis transformation (compare [MgF07]). The complete procedure is shown in Algorithm 9. It is logically the same as our generic refinement Algorithm 1; it only employs an optimized point evaluation. Thus, the algorithm makes use of the fast transformation for the *inner* function(s), but it requires the arbitrary point evaluation routine for the outer function. Note that we assumed compatible domains, so no extrapolation for the outer function is necessary.

Algorithm 9 Adaptive Resolution of $f_{\text{outer}}^h \circ \vec{f}_{\text{inner}}^h$

Input: $\vec{f}_{\text{inner}}^h = (f_{\text{inner},j})_{j=1,\dots,m}$ on grids $G_{\text{inner},j}$

Input: f_{outer}^h

Input: Refinement target value $\epsilon = \epsilon^{(\text{abs})}$ or $\epsilon = \epsilon^{(\text{rel})}$

Output: Grid G_{result} and $f_{\text{result}}^h = I_{G_{\text{result}}}[f_{\text{outer}}^h \circ \vec{f}_{\text{inner}}^h]$

1: $G_{\text{result}}^{(0)} := \bigcup_{j=1}^m G_{\text{inner},j}$ // $\mathcal{O}(|\vec{G}_{\text{inner}}|)$

2: $i = 0$

3: **repeat**

4: $f_{\text{result}}^h := \text{Interpolate } f_{\text{result}}^{h,*} \text{ on } G_{\text{result}}^{(i)} \text{ using Algorithm 10}$

// $\mathcal{O}(m|G_{\text{result}}^{(i)}|n_{\text{outer}}^m)$

5: $G_{\text{result}}^{(i+1)} := \text{refine } G_{\text{result}}^{(i)} \text{ based on } f_{\text{result}}^h \text{ up to } \epsilon$

// $\mathcal{O}(|G_{\text{result}}^{(i+1)}|)$

6: $i := i + 1$

7: **until** $G_{\text{result}}^{(i)} \setminus G_{\text{result}}^{(i-1)} = \emptyset$

8: $G_{\text{result}} := G_{\text{result}}^{(i)}$

// lines 2 – 8 \equiv refinement Algorithm 1 with special “set values”

9: **return** G_{result} and f_{result}^h

The consistency error depends on the smoothness of $f_{\text{outer}} \circ \vec{f}_{\text{inner}}$, we can essentially apply sparse grid interpolation error bounds provided it exhibits bounded second mixed derivatives.

Let us see the approximate function concatenation in action. Our first example is the already mentioned Gaussian: we use $f_{\text{outer}}(x) := \exp(-x)$ and $\vec{f}_{\text{inner}}(\vec{x}) = \vec{x}^T \vec{x}$. The adaptive representations of f_{outer} and \vec{f}_{inner} are shown in Figure 2.13 (top left and top right) together with their optimal grids. Since \vec{f}_{inner} is inherently one-dimensional, its grid uses only coefficients of level $l = -1$ for the hierarchical hat basis with constant

Algorithm 10 Subroutine of Algorithm 9: Interpolate f_{result}^h on given grid G_{result}

Input: G_{result} assuming $G_{\text{inner},j} \subset G_{\text{result}}$

Output: $f_{\text{result}}^h = I_{G_{\text{result}}}[f_{\text{outer}}^h \circ \vec{f}_{\text{inner}}^h]$

- 1: Interpolate all $f_{\text{inner},j}$ onto G_{result} // $\mathcal{O}(m|G_{\text{result}}|)$
- 2: change results to nodal values $n_{\mathbf{l},i,j} := f_{\text{inner},j}(x_{\mathbf{l},i})$ // $\mathcal{O}(m|G_{\text{result}}|)$
- 3: compute

$$(f_{\text{result}}^h)_{\mathbf{l},i} := f_{\text{outer}}^h(n_{\mathbf{l},i,1}, \dots, n_{\mathbf{l},i,m}) \quad (2.169)$$

for all $(\mathbf{l}, i) \in G_{\text{result}}$ // $\mathcal{O}(m|G_{\text{result}}| \cdot n_{\text{outer}}^m)$

- 4: change f_{result}^h to hierarchical basis // $\mathcal{O}(|G_{\text{result}}|)$
 - 5: **return** f_{result}^h
-

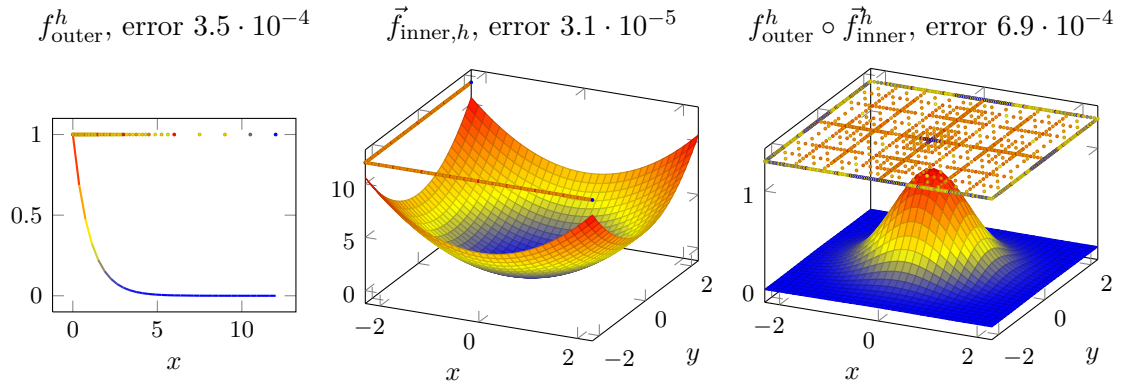


Figure 2.13: An example of $f_{\text{outer}}^h \circ \vec{f}_{\text{inner}}^h$ with cheap input functions and expensive output, in particular $f_{\text{outer}}(x) = \exp(-x)$ and $\vec{f}_{\text{inner}}(x, y) = x^2 + y^2$.

on the left boundary; the non-vanishing coefficients are also shown¹⁰. The result of Algorithm 9 is shown in Figure 2.13 (right); it requires more degrees of freedom than the low dimensional input grids.

A second example (with vector valued inner function) is shown in Figure 2.14: we use the outer function $f_{\text{outer}}(x, y) = x^2 + y^2$ and the inner function $\vec{f}_{\text{inner}}(x, y) = (x^2, \exp(-4x) \sin y)^T$ defined on $[0, 1] \times [0, 2\pi]$. We enabled grid sharing for components of \vec{f}_{inner} which explains why the respective grids displayed in Figure 2.14 are the same. Again, the result requires more degrees of freedom.

¹⁰Actually, we determined the result using the space- and dimension adaptive routine of Section 4.3, Algorithm 12.

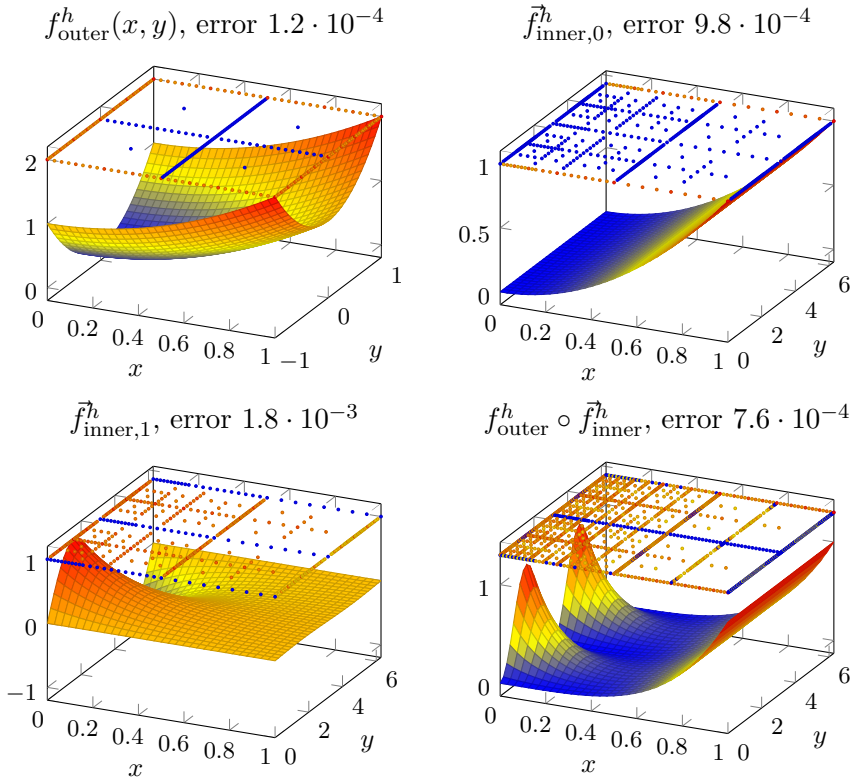


Figure 2.14: An example of $f_{\text{outer}}^h \circ \vec{f}_{\text{inner}}^h$ for vector valued inner function and the components $f_{\text{outer}}(x, y) = x^2 + y^2$, $\vec{f}_{\text{inner},0}(x, y) = x^2$ and $\vec{f}_{\text{inner},1}(x, y) = \exp(-4x) \sin y$, computed on shared grids for the inner function.

2.3.4 Approximate Integral Transformation

We study a further operation on approximate function representations, namely the integral transformation map

$$g(x) = \mathcal{F}(K, f)(x) = \int_{\Omega_y} K(x, y) \cdot f(y) \, dy \quad (2.170)$$

for a kernel function $K: \Omega_x \times \Omega_y \rightarrow \mathbb{R}$ defined on a rectangular product domain and a function $f: \Omega_y \rightarrow \mathbb{R}$. The main idea is again to employ adaptive sparse grids to get a compressed representation of both, $K(\cdot, \cdot)$ and $f(\cdot)$ and to combine these inputs to compute the integral transformation. For theory on optimized sparse grid spaces for integral transformations, we refer to [Kna00]. Our emphasize here is the algorithmic realization of an adaptive collocation scheme based on sparse grid error indicators.

The motivation to compute an approximate integral transformation is to employ the beneficial sparse grid complexity. Let us assume for the moment that the kernel can be approximated with cost complexity $|G_K| = \mathcal{O}(2^{n_K} n_K^{2d-1})$ in dimension $2d = |x| +$

$|y|$. This would be the case for smooth kernels, discretized on a regular sparse grid in dimension $2d$ with level n_K . Furthermore, let us assume that f can be discretized on a regular sparse grid of the same level, but on dimension d . Thus, f has cost complexity $N := |G_f| = \mathcal{O}(2^{n_K} n_K^{d-1})$. Since the approximate integral transformation has a runtime complexity of $\mathcal{O}(|G_K|)$, we can then expect to apply the integral transformation for a quite general kernel in time $\mathcal{O}(|G_K|) = \mathcal{O}(2^{n_K} n_K^{2d-1}) = \mathcal{O}(N \cdot (\log N)^{2d-1})$ if we neglect the small $\log n_K$ term. In particular, the special case $d = 1$ allows an approximate general integral transformation in time $\mathcal{O}(N \log N)$. This is to be compared with a full grid approach for which we find a complexity of $\mathcal{O}(N^2)$ for any dimension using the same assumptions. Note that adaptive sparse grids have potential to allow similar log-complexities even for irregular, perhaps singular kernels.

We propose to use an approach as described in Algorithm 11: the first step is to represent $f^h(y)$ as $|x| + |y|$ dimensional function. In the hierarchical hat basis with constant on level -1 , this does not introduce further grid points. Then, we employ the pointwise approximate multiplication method of Algorithm 8 and integrate the result exactly on slices along the y direction. Note that the formula for $G(x)$ in line 4 of

Algorithm 11 Approximate Integral Transformation

Input: $f^h : \Omega_y \rightarrow \mathbb{R}$, $\Omega_y \subset \mathbb{R}^{|y|}$ a tensor product interval,

Input: $K^h : \Omega_x \times \Omega_y \rightarrow \mathbb{R}$, $\Omega_x \subset \mathbb{R}^{|x|}$ a tensor product interval,

Input: Refinement target value $\epsilon = \epsilon^{(\text{abs})}$ or $\epsilon = \epsilon^{(\text{rel})}$

Output: Approximation $g^h(\cdot)$ on $g(x) = \int_{\Omega_y} K^h(x, y) \cdot f^h(y) dy$

- 1: compute $F(x, y) := f^h(y)$ using a trivial basis extension to dimension $|x| + |y|$.
- 2: compute grid G_R and $R(x, y) := I_{G_R}^\epsilon[K \cdot F] = \sum_{(\mathbf{l}, \mathbf{i}) \in G_R} r_{\mathbf{l}, \mathbf{i}} \phi_{\mathbf{l}, \mathbf{i}}(x, y)$ using the adaptive pointwise multiplication Algorithm 8
(with obvious transformations to the unit cube for $\phi_{\mathbf{l}, \mathbf{i}}$)
- 3: compute

$$g_{\mathbf{l}_x, \mathbf{i}_x} := \sum_{\{(\mathbf{l}_y, \mathbf{i}_y) \mid (\mathbf{l}_x \mathbf{l}_y, \mathbf{i}_x \mathbf{i}_y) \in G_R\}} r_{\mathbf{l}, \mathbf{i}} \int_{\Omega_y} \phi_{\mathbf{l}_y, \mathbf{i}_y}(y) dy \quad (2.171)$$

for every possible value $(\mathbf{l}_x, \mathbf{i}_x) \in G_R^{(x)}$ with the x slice

$$G_R^{(x)} := \{(\mathbf{l}_x, \mathbf{i}_x) \mid \exists (\mathbf{l}_y, \mathbf{i}_y) : (\mathbf{l}_x \mathbf{l}_y, \mathbf{i}_x \mathbf{i}_y) \in G_R\}. \quad (2.172)$$

The operation can be performed during *one* traversal through G_R (visit every point, integrate and multiply, then add the result to the correct output point).

- 4: define $G(\tilde{x}) := \sum_{(\mathbf{l}_x, \mathbf{i}_x) \in G_R^{(x)}} g_{\mathbf{l}_x, \mathbf{i}_x} \phi_{\mathbf{l}_x, \mathbf{i}_x}(\tilde{x})$
- 5: compress $G(\cdot)$ with ϵ and return the result

Runtime: The runtime requirements are dominated by the computation of $I_{G_R}^\epsilon[K \cdot F]$.

All other operations are linear in G_R . One can expect $|G_R| = \mathcal{O}(|G_K|)$ for non-trivial kernels, see in-text argumentation.

Algorithm 11 can be found by reordering summands of $\int_{\Omega_y} R(x, y) dy$; the representation

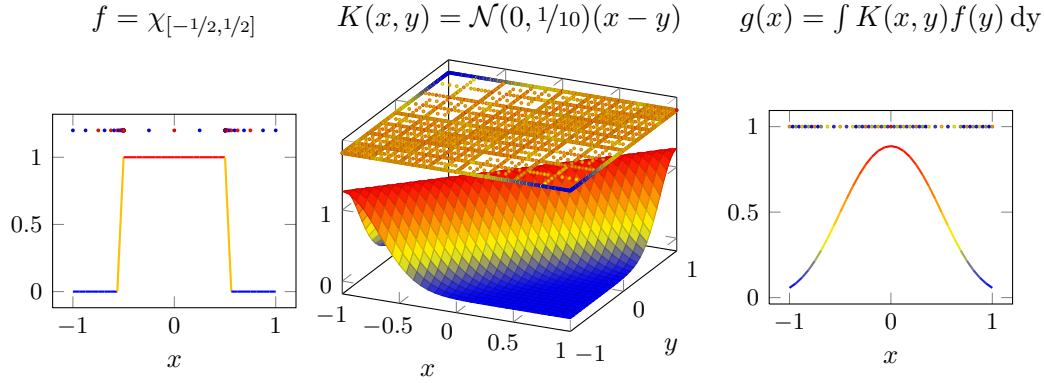


Figure 2.15: An example of $\int K(x, y) f(y) dy$ for a non-smooth input function, $f = \chi_{[-1/2, 1/2]}$ and a Gaussian convolution kernel.

is exact. Finally $G(x)$ is represented in the same basis as $R(\cdot, \cdot)$. The method introduces a consistency error for the re-interpolated multiplication. The data error depends on the multiplication whereas the final integration as linear operation is well conditioned (its condition number depends mainly on the domain size $|\Omega_y|$).

The approximate integral transformation relies on compressed representations of the input functions $K(\cdot, \cdot)$ and $f(\cdot)$. Since any relevant kernel will depend on both, x and y , we can expect its grid to be a relative good starting point for the interpolated product $R(x, y) = K(x, y) \cdot F(x, y)$. In other words, we can expect that $|G_R| \approx \mathcal{O}(|G_K|)$. The runtime complexity is dominated by the product grid $|G_R|$: multiplication, integration and the final compression are all of complexity $\mathcal{O}(|G_R|)$ and thus of complexity $\mathcal{O}(|G_K|)$.

We apply the algorithm to some examples. Our first example uses the singular function $f(x) = \chi_{[-1/2, 1/2]}(x)$ and the smooth convolution kernel $K(x, y) = k(x - y)$ with $k(x) = \mathcal{N}(0, 1/10)(x)$ where $\mathcal{N}(\mu, \sigma)(x) = \frac{1}{\sqrt{2\pi^d} \sqrt{|\det(\sigma)|}} \cdot \exp(-\frac{1}{2}(x - \mu)^T \sigma^{-1}(x - \mu))$ in one dimension. The approximations f^h and K^h together with the result are shown in Figure 2.15. The argument f is resolved up to a maximum level of 20 and $\epsilon = 10^{-3}$, the same for $K(\cdot, \cdot)$. The input grid for f is highly adapted; the grid for K is a sparse grid with few points away from the diagonal. The result is again a smooth function.

Another example uses the same function f , together with $k = \chi_{[-1/2, 1/2]}$. The result is the hat function up to ϵ . It is displayed in Figure 2.16. Here, the diagonal singularity of $K(x, y)$ requires a lot of points; it has been computed up to a prescribed maximum level. The resulting hat function is precise up to ϵ and requires a lot of points despite the compression.

A further example uses the (arbitrary) singular kernel

$$K(x, y) = \cos x \cos y \cdot \begin{cases} -\log(x - y) & y < x \\ \sqrt{y - x} & y \geq x, \end{cases} \quad (2.173)$$

evaluated at $[0, \pi]$ and the function f as before, this time evaluated at $[0, \pi]$ as well. It is

2 A Sparse Grid Approximation Algebra

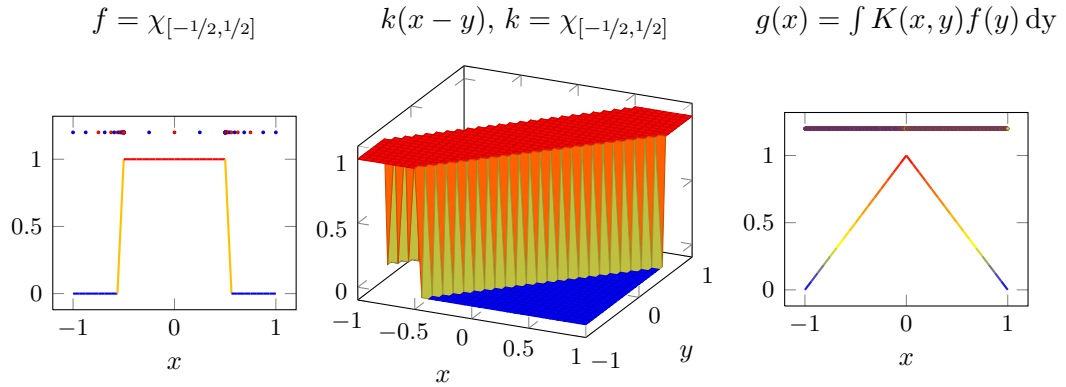


Figure 2.16: An example of $\int K(x, y)f(y) dy$ with box function f and box convolution kernel k .

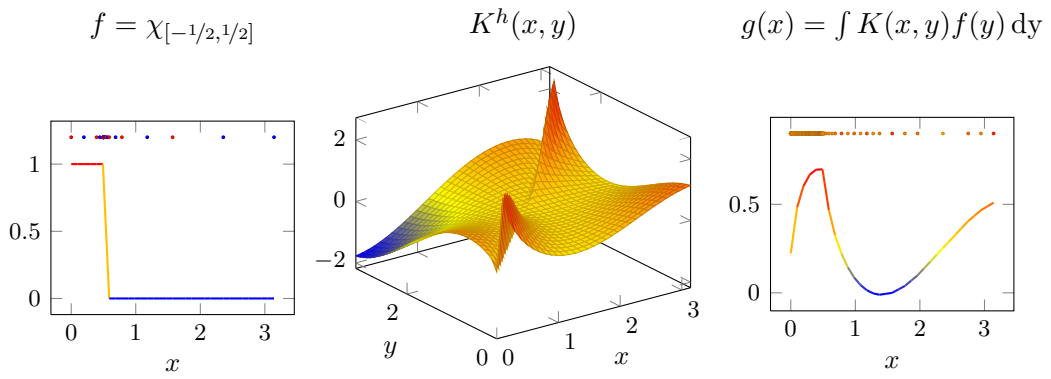


Figure 2.17: An example of $\int K(x, y)f(y) dy$ with the box function $f = \chi_{[-1/2, 1/2]}$ and

$$K(x, y) = \cos x \cos y \cdot \begin{cases} -\log(x - y) & y < x \\ \sqrt{y - x} & y \geq x \end{cases}.$$

shown in Figure 2.17. The kernel exhibits a jump and a square root singularity near the diagonal and is resolved up to a prescribed maximum level. The transformed function is continuous, however.

This completes our chapter about sparse grid basics. We will now apply sparse grids to high dimensional problems, first on density approximation and in Chapter 4 on generalized dimension adaptive grids.

3 Sparse Grids and Moderate Dimensional Approximation – Two Case Studies

3.1 Overview and Motivation

In this chapter, we study the performance of sparse grid methods using two case studies. Our first case study is to analyze the approximation of unimodal probability density functions in high dimensions, a task motivated by the Fokker–Planck–Equation. This second order diffusion equation describes the time–dependent behavior of a probability density function for stochastic processes, i.e. it is a transport equation for (coupled) particles subjected to fluctuating forces on a very small time scale. This fluctuation is modelled as “random noise”. Applications range from kinetic models for non–Newtonian fluids like polymeric solutions [DLO07, Kne08] over engineering systems subjected to environmental load (wind, water etc.) [WB00] up to models for financial engineering, see [Hol08] and the references therein. These stochastic methods can be simulated by means of sampling methods of Monte–Carlo type or by approximation of the probability density function (see the textbook of Öttinger, [Ött96]). Our case study analyzes sparse grid approximation properties of the normal distribution which is probably the most important probability density in this area: white noise is defined to be Gaussian, so the normal distribution solves the Fokker–Planck–Equation for the case of linear transport coefficients and white noise forces when the particle’s initial position and velocity is known (Ornstein–Uhlenbeck process, we come back to this specific Fokker–Planck–Equation in Section 3.2.5). If the transport term vanishes, this resembles the well–known diffusion equation with Dirac delta as initial value. A linear approximation method which can solve these prototypic equations can probably represent other unimodal densities as well. Conversely, an efficient approximation of the Gaussian is a necessary condition: if its approximation requires exponential cost with respect to d , other densities will be even more involved. Furthermore, the Gaussian constitutes a common initial value for Fokker–Planck–Equations which needs to be resolved properly. Our aim for the case study is to analyze the approximation properties with respect to the convergence rate *and* the dimension–dependent factors. Since we know from Lemma 2.1.9 that the convergence rate is independent of the dimension up to log terms, it remains to analyze dimension–dependent factors. These, however, will depend exponentially on d due to the tensor product structure (in domain, approximant, log terms and ansatz space), so we can only hope that these coefficients do not grow exponentially. The result of our analysis in both theory and practice is that the factors grow like about 10^d : we need effectively 10 times more degrees of freedom to get the same relative accuracy in one dimension larger. While this is still substantially better than full grid methods, it clearly

restricts the feasible dimension; even $d = 5$ might already be quite involved.

Results for the first case study are related to the area of Information Based Complexity: this relatively young branch studies the minimum number of operations required for any algorithm to compute an approximate solution up to prescribed error ϵ . Information Based Complexity always states results for problem *classes*, not particular problem instances (the worst element of a complete function space), involving in particular dimension-dependent coefficients and convergence rates. We survey related topics of IBC and their relation to our case study in Section 3.2.4.

Our second case study is to approximate functions with more structure: we analyze functions whose dynamics is essentially oriented along the axes. Sparse grids are specifically strong for such kind of structure and we present results of adaptive sparse grids applied to functions with jumps along axis parallel manifolds. As long as the dynamics are essentially parallel to the axes, the complexity of our anisotropic ansatz spaces allows considerable savings compared to isotropic (even adaptive) methods.

3.2 Case Study: Density Approximation and the Normal Distribution

Our case study on approximation of the density of the normal distribution consists of two parts: the first is to collect all dimension-dependent factors (domain transformation, method coefficients, relative errors and regularity norms) and instantiate the interpolation error bounds. The second step is to compute interpolation errors a posteriori using numerical experiments, allowing us to formulate both, limitations and gains of sparse grids for density approximation.

To simplify the notation in this section, we use the term “normal distribution” to refer to the probability density function of the normal distribution.

3.2.1 Interpolation Error Estimates

Our goal in this section is to get qualitative results about the dimension dependent coefficients for the special case of parameter dependent normal distributions which are defined on \mathbb{R}^d . To this end, we use a domain truncation followed by a linear coordinate transformation to the unit cube and track the resulting dimension dependent coefficients. Due to the bounded integral, every probability density on \mathbb{R}^d has some decay for $\|x\| \rightarrow \infty$ which justifies to use a bounding rectangle of proper size (measured in the norm of interest). So, choosing a rectangular domain $\Omega \subset \mathbb{R}^d$ and setting up a linear transformation allows to use sparse grids. Since the density function decays rapidly towards the boundary, we impose vanishing boundary conditions on the boundary and thereby reduce the grid complexity considerably (a sparse grid with boundary nodes has about a factor of 3^d more points than a sparse grid without boundary points, compare Lemma 2.1.2). Of course, one might also think about nonlinear mappings from \mathbb{R}^d to the unit cube $[0, 1]^d$, perhaps chosen to match the function which should be approximated. We will discuss such efforts in Section 3.2.3.

3.2 Case Study: Density Approximation and the Normal Distribution

We restrict ourselves to normal distributions with diagonal covariance matrix, i.e. to a tensor product of one-dimensional normal distributions $f(x) := \mathcal{N}(\mu, \Sigma)(x)$ with $\Sigma = \text{diag}(\sigma_1, \dots, \sigma_d)$ and

$$\mathcal{N}(\mu, \Sigma)(x) = \frac{1}{\sqrt{2\pi}^d \sqrt{|\det(\Sigma)|}} \cdot \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu)\right) \quad (3.1)$$

$$= \prod_{i=1}^d \mathcal{N}(\mu_i, \sigma_i)(x_i) =: \prod_{i=1}^d f_i(x_i) \quad (3.2)$$

with $f_i := \mathcal{N}(\mu_i, \sigma_i) = \frac{1}{\sqrt{2\pi\sigma_i}} \exp(-1/2(x - \mu_i)^2/\sigma_i)$. Since the variance σ is essentially the width of $\mathcal{N}(\mu, \sigma)$, the truncated domain size will depend on σ whereas its position will depend on μ . We express the truncated domain using multiples of the standard deviations $\sqrt{\sigma_i}$,

$$\Omega := \bigotimes_{i=1}^d [\mu_i - k_i \sqrt{\sigma_i}, \mu_i + k_i \sqrt{\sigma_i}] \quad (3.3)$$

and introduce truncation parameters $k_i > 0$. These parameters control the truncation error. It turns out that a single parameter $k_i \equiv k$ is enough to get a balanced truncation error.

Since the shift μ will be eliminated by the transformation to the unit cube, we assume $\mu = 0$ without loss of generality. With the same argument, we use the symmetrically shifted unit cube $[-1/2, 1/2]^d$ instead of $[0, 1]^d$ to simplify the notation. This leads to the linear transformation

$$\phi: [-1/2, 1/2]^d \rightarrow \Omega = \bigotimes_{i=1}^d [-k\sqrt{\sigma_i}, k\sqrt{\sigma_i}], \quad (3.4)$$

$$\phi_i(x_i) = 2k\sqrt{\sigma_i}x_i \quad (3.5)$$

and we get the transformed tensor product factors

$$f_i \circ \phi_i(x_i) = \frac{1}{\sqrt{2\pi\sigma_i}} \exp(-2k^2 x_i^2). \quad (3.6)$$

We are now in a position to insert $f \circ \phi$ into interpolation error bounds known for functions on the unit cube, keeping in mind that we seek for approximations on \mathbb{R}^d . We start by introducing the following notation for intermediate steps:

Our goal is to approximate $f: \mathbb{R}^d \rightarrow \mathbb{R}$.

1. We truncate \mathbb{R}^d to Ω and introduce

$$f_{\Omega}^{\text{cut}}(x) := f(x) \cdot \chi_{\Omega}(x) = \begin{cases} f(x) & x \in \Omega, \\ 0 & \text{otherwise.} \end{cases} \quad (3.7)$$

2. We transform to the unit cube using $\phi: [-1/2, 1/2]^d \rightarrow \Omega$ and denote the transformed function by

$$f_{[-1/2, 1/2]^d} := f \circ \phi: [-1/2, 1/2]^d \rightarrow \mathbb{R}. \quad (3.8)$$

3 Sparse Grids and Moderate Dimensional Approximation – Two Case Studies

3. Our approximation is performed on the unit cube, using the superscript ‘ h ’,

$$f_{[-1/2, 1/2]^d}^h := I[f_{[-1/2, 1/2]^d}], \quad (3.9)$$

where $I[\cdot]$ denotes the (sparse grid) interpolation operator. As already motivated, we use vanishing boundary conditions on $\partial[-1/2, 1/2]^d$ to reduce the grid complexity. This step yields a jump of one mesh width near the boundary.

4. We transform back and get

$$f_{\Omega}^h := f_{[-1/2, 1/2]^d}^h \circ \phi^{-1} : \Omega \rightarrow \mathbb{R} \quad (3.10)$$

and complete the definition to \mathbb{R}^d using zeros,

$$f^h(x) := \begin{cases} f_{\Omega}^h(x) & x \in \Omega, \\ 0 & \text{otherwise.} \end{cases} \quad (3.11)$$

5. Finally, we estimate the error

$$\|f - f^h\| \quad (3.12)$$

for the L_2 norm $\|\cdot\|_{L_2[\mathbb{R}^d]}$ and the L_{∞} norm $\|\cdot\|_{L_{\infty}[\mathbb{R}^d]}$. Furthermore, we provide estimates on the energy norm restricted to the truncated domain,

$$\|g\|_{A[\Omega]} := \left(\sum_{i=1}^d \left\| \frac{\partial g}{\partial x_i} \right\|_{L_2[\Omega]}^2 \right)^{1/2}. \quad (3.13)$$

Measuring the energy norm requires attention due to the jump of one mesh width near the boundary: in the limit of vanishing mesh width, the error becomes unbounded. Consequently, convergence rates only hold up to a certain h , then the jump dominates.

The norm evaluation on Ω involves the inverse transformation ϕ^{-1} and yields dimension dependent coefficients

$$\|g \circ \phi^{-1}\|_{L_2[\Omega]}^2 = \prod_{i=1}^d (2k\sqrt{\sigma_i}) \|g\|_{L_2[-1/2, 1/2]^d}^2, \quad (3.14)$$

$$\|g \circ \phi^{-1}\|_{L_{\infty}[\Omega]} = \|g\|_{L_{\infty}[-1/2, 1/2]^d}, \quad (3.15)$$

$$\|g \circ \phi^{-1}\|_{A[\Omega]}^2 = \sum_{i=1}^d \left\| \frac{\partial}{\partial x_i} (g \circ \phi^{-1}) \right\|_{L_2[\Omega]}^2 \quad (3.16)$$

$$= \sum_{i=1}^d \left(\frac{\prod_{j \neq i} 2k\sqrt{\sigma_j}}{2k\sqrt{\sigma_i}} \right) \left\| \frac{\partial}{\partial x_i} g \right\|_{L_2[-1/2, 1/2]^d}^2 \quad (3.17)$$

$$\leq (2k\sqrt{\bar{\sigma}})^{d-1} \cdot (2k\sqrt{\underline{\sigma}})^{-1} \|g\|_{A[-1/2, 1/2]^d}^2 \quad (3.18)$$

where we define $\underline{\sigma} := \min\{\sigma_i\}$ and $\bar{\sigma} := \max\{\sigma_i\}$.

3.2 Case Study: Density Approximation and the Normal Distribution

The norm evaluation on \mathbb{R}^d finally requires to measure the domain truncation error. It holds with the notation $f^h(x) = \chi_\Omega(x)f_\Omega^h(x)$ for the L_2 norm

$$\|f - f^h\|_{L_2[\mathbb{R}^d]}^2 = \int_{\mathbb{R}^d} (f^2 - 2ff^h + (f^h)^2) dx \quad (3.19)$$

$$= \int_{\mathbb{R}^d} f^2 dx - \int_{\mathbb{R}^d} 2ff^h dx + \int_{\mathbb{R}^d} (f^h)^2 dx \quad (3.20)$$

$$= \int_{\mathbb{R}^d} f^2 dx - \int_{\Omega} f^2 dx + \int_{\Omega} f^2 dx - \int_{\Omega} 2ff^h dx + \int_{\Omega} (f^h)^2 dx \quad (3.21)$$

$$= \int_{\mathbb{R}^d} f^2 dx - \int_{\Omega} f^2 dx + \int_{\Omega} (f - f_\Omega^h)^2 dx. \quad (3.22)$$

Note that all involved integrals exist. The case $f^h = f_\Omega^{\text{cut}}$ yields the squared L_2 truncation error which simplifies to

$$\|f - f_\Omega^{\text{cut}}\|_{L_2[\mathbb{R}^d]}^2 = \int_{\mathbb{R}^d} f^2 dx - \int_{\Omega} f^2 dx = \prod_{i=1}^d (2\sqrt{\pi\sigma_i})^{-1} (1 - \text{erf}(k)^d) \quad (3.23)$$

where $\text{erf}(k) = 2/\pi \int_0^k e^{-t^2} dt$ denotes the error function [AS65, Chapter 7]. Division by $\|f\|_{L_2[\mathbb{R}^d]}$ yields the relative L_2 truncation error

$$\frac{\|f - f_\Omega^{\text{cut}}\|_{L_2[\mathbb{R}^d]}}{\|f\|_{L_2[\mathbb{R}^d]}} = \sqrt{1 - \text{erf}(k)^d}, \quad (3.24)$$

using

$$\|f\|_{L_2[\mathbb{R}^d]} = \prod_{i=1}^d (2^{-1/2}(\pi\sigma_i)^{-1/4}). \quad (3.25)$$

We also note the function norms on \mathbb{R}^d for

$$\|f\|_{L_\infty[\mathbb{R}^d]} = \prod_{i=1}^d (2\pi\sigma_i)^{-1/2}, \quad (3.26)$$

$$\left\| \frac{\partial}{\partial x_i} f \right\|_{L_2[\mathbb{R}^d]}^2 = (4\sqrt{\pi}\sigma_i^{3/2})^{-1} \prod_{j \neq i} (2\sqrt{\pi}\sigma_j)^{-1} \quad (3.27)$$

and thus

$$\|f\|_{A[\mathbb{R}^d]} = \left(\sum_{i=1}^d (4\sqrt{\pi}\sigma_i^{3/2})^{-1} \prod_{j \neq i} (2\sqrt{\pi}\sigma_j)^{-1} \right)^{1/2} \quad (3.28)$$

which can be bounded by

$$\sqrt{d}2^{-1}\bar{\sigma}^{-1/2}(2\sqrt{\pi\bar{\sigma}})^{-(d-1)/2} \leq \|f\|_{A[\mathbb{R}^d]} \leq \sqrt{d}2^{-1}\underline{\sigma}^{-1/2}(2\sqrt{\pi\underline{\sigma}})^{-(d-1)/2}. \quad (3.29)$$

The domain truncation error measured in the energy norm can be obtained similarly¹

¹Note that the undefined gradient poses no problems since it occurs only on a set of measure 0.

3 Sparse Grids and Moderate Dimensional Approximation – Two Case Studies

to (3.19) to (3.22) as long as no numerical error is involved, i.e. as long as $f^h = f_\Omega^{\text{cut}}$. A numerical error leads to a jump of one mesh width which is unbounded in the limit of vanishing mesh width. The domain truncation error can be expressed by

$$\|f - f_\Omega^{\text{cut}}\|_{A[\mathbb{R}^d]}^2 = \|f\|_{A[\mathbb{R}^d]}^2 - \|f\|_{A[\Omega]}^2; \quad (3.30)$$

its dependence on the cut radius k is shown in Figure 3.1 (right) and is discussed at the end of this section.

For the L_∞ error on \mathbb{R}^d , we get

$$\|f - f^h\|_{L_\infty[\mathbb{R}^d]} = \max\{\|f - f_\Omega^{\text{cut}}\|_{L_\infty[\mathbb{R}^d]}, \|f - f_\Omega^h\|_{L_\infty[\Omega]}\}. \quad (3.31)$$

We will now plug the regularity norm of $f \circ \phi$ into the interpolation error bounds of Lemma 2.1.9. To compute $|f \circ \phi|_{2,2}$, we need to integrate $\exp(-x^2)$ over the compact domain Ω – which cannot be simplified analytically. For reasons of simplicity, we compute the L_2 regularity norm on \mathbb{R}^d , neglecting the domain truncation (but using the transformation ϕ). This yields for $f_i \circ \phi_i(x_i) = (2\pi\sigma_i)^{-1/2} \exp(-2k^2x_i^2)$ the regularity norms

$$|f \circ \phi|_{2,2} = \left\| \frac{\partial^{2d}}{\partial_{x_1}^2 \dots \partial_{x_d}^2} (f \circ \phi) \right\|_{L_2[-1/2, 1/2]^d} \leq \left\| \frac{\partial^{2d}}{\partial_{x_1}^2 \dots \partial_{x_d}^2} (f \circ \phi) \right\|_{L_2[\mathbb{R}^d]} \quad (3.32)$$

$$= \prod_{i=1}^d \frac{\sqrt{3}k^{3/2}}{\sqrt{\sigma_i\pi^{1/4}}} \leq \left(\frac{\sqrt{3}k^{3/2}}{\sqrt{\underline{\sigma}\pi^{1/4}}} \right)^d, \quad (3.33)$$

$$|f \circ \phi|_{2,\infty} = \left\| \frac{\partial^{2d}}{\partial_{x_1}^2 \dots \partial_{x_d}^2} (f \circ \phi) \right\|_{L_\infty[-1/2, 1/2]^d} \quad (3.34)$$

$$= \prod_{i=1}^d \frac{4k^2}{\sqrt{2\pi\sigma_i}} \leq \left(\frac{4k^2}{\sqrt{2\pi\underline{\sigma}}} \right)^d. \quad (3.35)$$

Interpolation error bounds on regular sparse grids

The interpolation error of our normal distribution f on regular sparse grids, measured using the transformation rules (3.14) – (3.18) and the transformed regularity semi norms $|f \circ \phi|_{2,2}$ and/or $|f \circ \phi|_{2,\infty}$ can be simplified to

$$\|f - f_\Omega^h\|_{A[\Omega]} \leq \sqrt{(2k\sqrt{\bar{\sigma}})^{d-1}/(2k\sqrt{\underline{\sigma}})} \|f \circ \phi - f_{[-1/2, 1/2]^d}^h\|_{A[-1/2, 1/2]^d} \quad (3.36)$$

$$\leq \sqrt{(2k\sqrt{\bar{\sigma}})^{d-1}/(2k\sqrt{\underline{\sigma}})} \frac{d|f \circ \phi|_{2,\infty}}{2 \cdot 3^{(d-1)/2} \cdot 4^{d-1}} 2^{-n} \quad (3.37)$$

$$= 2^{-n} (k^{5/2}\bar{\sigma}^{1/4}\underline{\sigma}^{-1/2}(3\pi)^{-1/2})^d k^{-1}(\bar{\sigma}\underline{\sigma})^{-1/4}\sqrt{3} \cdot d \quad (3.38)$$

$$\Rightarrow \frac{\|f - f_\Omega^h\|_{A[\Omega]}}{\|f\|_{A[\mathbb{R}^d]}} \leq 2^{-n} (\sqrt{2/3} \cdot k^{5/2}(\bar{\sigma}/\underline{\sigma})^{1/2}\pi^{-1/4})^d \cdot \sqrt{6d} \cdot k^{-1}\underline{\sigma}^{-1/4}\pi^{-1/4}. \quad (3.39)$$

The quotient of largest and smallest variance, $\bar{\sigma}/\underline{\sigma} \geq 1$, occurs since we estimate the transformation coefficient and the regularity norm from above but $\|f\|_{A[\mathbb{R}^d]}$ from below.

3.2 Case Study: Density Approximation and the Normal Distribution

Experiments suggest that the *relative* error is actually independent of σ_i at all, as we will also see for L_2 and L_∞ below. Let us stress again that we do not really have vanishing boundary conditions – instead, the jump over one mesh width near the boundary will yield an unbounded energy error for $h \rightarrow 0$. We expect these estimates to hold as long as the mesh width is not too fine.

In a similar way, we get error bounds for the regular sparse grid interpolant using L_2 and L_∞ norms, although we will have the log term $A(d, n)$ shown in (2.73). It holds

$$\|f - f_\Omega^h\|_{L_\infty[\Omega]} = \|f \circ \phi - f_{[-1/2, 1/2]^d}^h\|_{L_\infty[-1/2, 1/2]^d} \quad (3.40)$$

$$\leq 2^{-2n} 2A(d, n) \left(\frac{k^2}{2}\right)^d \prod_{i=1}^d (2\pi\sigma_i)^{-1/2} \quad (3.41)$$

for the L_∞ norm; its relative error is thus

$$\frac{\|f - f_\Omega^h\|_{L_\infty[\Omega]}}{\|f\|_{L_\infty[\mathbb{R}^d]}} \leq 2^{-2n} A(d, n) \cdot 2 \cdot \left(\frac{k^2}{2}\right)^d. \quad (3.42)$$

Similarly, we find the L_2 error bound

$$\|f - f_\Omega^h\|_{L_2[\Omega]} = \sqrt{\prod_{i=1}^d (2k\sqrt{\sigma_i})} \|f \circ \phi - f_{[-1/2, 1/2]^d}^h\|_{L_2[-1/2, 1/2]^d} \quad (3.43)$$

$$\leq 2^{-2n} \cdot 2 \cdot A(d, n) \cdot (2^{-1} 6^{-1/2} \pi^{-1/4} k^2)^d \prod_{i=1}^d \sigma_i^{-1/4} \quad (3.44)$$

and its relative counterpart

$$\frac{\|f - f_\Omega^h\|_{L_2[\Omega]}}{\|f\|_{L_2[\mathbb{R}^d]}} \leq 2^{-2n} \cdot 2 \cdot A(d, n) (2^{-1} 3^{-1/2} k^2)^d. \quad (3.45)$$

We see that L_2 and L_∞ relative discretization *and* truncation errors are both independent of μ_i and σ_i , although they depend on the cut radius parameter k . We have effectively eliminated σ_i by coupling it to the domain size.

However, it becomes clear that each involved upper bound *grows* exponentially with d unless the only remaining parameter k is very small. But since k denotes multiples of the standard deviation $\sqrt{\sigma_i}$ in each direction, there is an intuitive measure on its magnitude: for dimension $d = 1$, $k = 1$ discards 31.7% of the probability mass (i.e. truncation error in L_1 norm), $k = 2$ discards 4.5% of its mass. For larger dimensions, the losses in probability mass become more serious: $d = 7$ and $k = 1$ loses already 93%, $d = 7$ and $k = 2$ loses 28%. Values of $k \in [3, 5]$ yield truncation errors roughly between 10^{-3} and 10^{-6} in both L_1 and relative L_2 norm – but these values already lead to exponential growth in the interpolation bounds. Fortunately, the cut error does not depend too strongly on the dimension as can be seen in Figure 3.1 (left) for the L_1 cut error (probability mass), Figure 3.1 (middle) for the relative L_2 cut error and Figure 3.1 (right) for the relative energy cut error if we silently ignore the jump on the boundary.

3.2 Case Study: Density Approximation and the Normal Distribution

the error ϵ in terms of the cost N in form of the ϵ complexity, which yields

$$\begin{aligned} \epsilon(N) \leq N^{-1} (2^{-1/2} (4 - 2 \cdot 2^{1/3})^{-1} k^2 \underline{\sigma}^{-1/2} \bar{\sigma}^{1/4} \pi^{-1/4})^d (5 \cdot 2^d + 4 \cdot 5^d) \\ \cdot 10^{-1} \cdot \sqrt{3} \cdot d^2 \cdot k^{-1} \cdot (\underline{\sigma} \bar{\sigma})^{-1/4}. \end{aligned} \quad (3.51)$$

The relative ϵ complexity is thus

$$\begin{aligned} \frac{\epsilon(N)}{\|f\|_{A[\mathbb{R}^d]}} \leq N^{-1} ((4 - 2 \cdot 2^{1/3})^{-1} k^2 (\bar{\sigma}/\underline{\sigma})^{1/2})^d (5 \cdot 2^d + 4 \cdot 5^d) \\ \cdot \sqrt{3/2} \cdot 5^{-1} d^{3/2} k^{-1} \underline{\sigma}^{-1/4} \pi^{-1/4} \\ \approx N^{-1} (0.68 k^2 (\bar{\sigma}/\underline{\sigma})^{1/2})^d (5 \cdot 2^d + 4 \cdot 5^d) \cdot \text{Poly}(d, k, \bar{\sigma}, \underline{\sigma}). \end{aligned} \quad (3.52)$$

The alternative energy error bound involving $|f \circ \phi|_{\mathbf{2}, \infty}$ (compare (2.82) and (2.83)) yields the slightly different bounds

$$\begin{aligned} \epsilon(N) \leq N^{-1} (3^{-1/2} (2 - 2^{1/3})^{-1} k^{5/2} \underline{\sigma}^{-1/2} \bar{\sigma}^{1/4} \pi^{1/2})^d (5 \cdot 2^d + 4 \cdot 5^d) \\ \cdot 10^{-1} \cdot \sqrt{3} \cdot d^2 \cdot k^{-1} \cdot (\underline{\sigma} \bar{\sigma})^{-1/4} \end{aligned} \quad (3.53)$$

and

$$\begin{aligned} \frac{\epsilon(N)}{\|f\|_{A[\mathbb{R}^d]}} \leq N^{-1} \left(\frac{3}{2} (2 - 2^{1/3})^{-1} k^{5/2} \pi^{-1/4} (\bar{\sigma}/\underline{\sigma})^{1/2} \right)^d (5 \cdot 2^d + 4 \cdot 5^d) \\ \cdot 5^{-1} \cdot (3/2)^{1/2} d^{3/2} k^{-1} \underline{\sigma}^{-1/4} \pi^{-1/4} \\ \approx N^{-1} (0.68 k^{5/2} (\bar{\sigma}/\underline{\sigma})^{1/2})^d (5 \cdot 2^d + 4 \cdot 5^d) \text{Poly}(d, k, \bar{\sigma}, \underline{\sigma}) \end{aligned} \quad (3.54)$$

where $\text{Poly}(d, k, \bar{\sigma}, \underline{\sigma})$ depends only polynomially on its arguments. Thus, the overall ϵ complexity *grows* when applied to the normal distribution on Ω , even though the method coefficients as such (i.e. without transformation and not including the regularity norm) decrease exponentially (compare [Gri06]).

We summarize the interpolation error bounds as follows. The first thing to note is that relative L_∞ and L_2 errors can be bounded independently of the position *and* the variance. The reason for this parameter invariance is the domain size: a large variance requires a large domain whereas a small variance needs a small domain to get the same truncation error. The results for the energy norm are the same for the isotropic Gaussian (the same variance for every direction). Our upper bounds allow an exponential increase in the energy norm error which contains the coefficient $(\frac{\bar{\sigma}}{\underline{\sigma}})^d \geq 1$. Moreover, we see exponential growth in each of the analyzed coefficients, there are no cut radii of interest for which exponential growth can be avoided.

In short: theory suggests that the relative error approximation of Gaussians with sparse grids still requires exponential effort (although “only” in the order coefficients, i.e. of the form C^d with $C > 1$). This holds also for the energy norm ϵ complexity, so we do not expect considerable improvements for the ϵ complexity of L_2 optimal sparse grids (whose cost complexity is more difficult to analyze due to the log terms).

3.2.2 Error Estimation Using Numerical Experiments

We will now measure interpolation errors to complement the upper bounds from theory, especially for the cost/error relation which is hard to analyze for L_2 optimal sparse grids. We can restrict the study to the standard normal distribution with variance 1 and mean 0 since relative errors are parameter independent as we have seen above.

We show error measurements for the error on \mathbb{R}^d , especially for the relative L_2 norm

$$\frac{\|f - f_h\|_{L_2[\mathbb{R}^d]}}{\|f\|_{L_2[\mathbb{R}^d]}} = \left(\frac{\|f - f_\Omega^{\text{cut}}\|_{L_2[\mathbb{R}^d]}^2}{\|f\|_{L_2[\mathbb{R}^d]}^2} + \frac{\|f - f_\Omega^h\|_{L_2[\Omega]}^2}{\|f\|_{L_2[\mathbb{R}^d]}^2} \right)^{1/2}. \quad (3.55)$$

Note that

$$\frac{\|f - f_h\|_{L_\infty[\mathbb{R}^d]}}{\|f\|_{L_\infty[\mathbb{R}^d]}} = \max \left\{ \frac{\|f - f_\Omega^{\text{cut}}\|_{L_\infty[\mathbb{R}^d]}}{\|f\|_{L_\infty[\mathbb{R}^d]}}, \frac{\|f - f_\Omega^h\|_{L_\infty[\Omega]}}{\|f\|_{L_\infty[\mathbb{R}^d]}} \right\} \quad (3.56)$$

requires too much computational effort for reliable measurements. We will also consider the relative energy norm discretization error, i.e. the error on Ω only,

$$\frac{\|f - f_\Omega^h\|_{A[\Omega]}}{\|f\|_{A[\mathbb{R}^d]}}, \quad (3.57)$$

in regimes where the jump on $\partial\Omega$ will not be seen.

Since the relative truncation error depends only on k if we hold d fixed, and the relative discretization error depends on the discretization level n and k , an optimal setting will balance discretization- and truncation errors using $n = n(k(\epsilon, d), \epsilon, d)$ and $k = k(\epsilon, d)$ for a fixed, prescribed relative accuracy ϵ . This can be realized using several possibilities:

1. An a priori functional relation for n and k , relying on theoretical error bounds,
2. A sub-optimal choice which chooses a fixed, upper bound for k which is large enough such that the truncation error drops below ϵ , combined with a choice $n = n(d, \epsilon)$ which does not really consider k ,
3. A posteriori grid optimization which determines n (or, more generally, the degrees of freedom) in dependence of the target error ϵ and $k(d, \epsilon)$,
4. A posteriori adaptive optimization which chooses both, the degrees of freedom and k automatically.

The choice 1.) is the most specialized one as it requires the most knowledge about the function f (at least upper bounds for discretization and truncation error); its generalization to other functions is thus involved. Choice 2.) requires insight about the cut error, i.e. insight into the decay properties of f , and yields sub-optimal (but maybe still acceptable) results since n is chosen using some rule-of-thumb. Choice 3.) also requires some insight into the truncation error but optimizes the discretization error (at the cost of more programming effort a refinement loop like Algorithm 1, compare page 33) whereas

3.2 Case Study: Density Approximation and the Normal Distribution

choice 4.) requires almost no knowledge at the cost of (probably) one further refinement loop to optimize k (and thus, the domain Ω).

We believe that choices 2.) and 3.) have the highest practical value: our initial motivation was to approximate solutions of the Fokker Planck Equation which provides knowledge about the initial value's decay properties and (hopefully) a good guess for an upper bound of the domain size Ω . Then, choices 2.) and 3.) can be used to distribute the degrees of freedom (or the level n) "properly" on Ω to reduce unnecessary work. Note that adaptive refinement in 3.) has the purpose to balance discretization and cut errors, i.e. to improve the pre-asymptotics only. Its asymptotic convergence rate will be the same. Here, the Fokker Planck Equation also introduces the parameter time, i.e. the degrees of freedom should be chosen differently for different times in such a context.

We are going to analyze choices 1.), 2.) and 3.) for our model density where we believe that choices 2.) and 3.) have model character and allow qualitative generalizations to other unimodal types of densities as well.

Description of the Methods

Clearly, our choice 2.) is the most simple to realize: we fix the domain of interest somehow (by choosing k) and compute the sparse grid interpolant on a regular sparse grid of level n . The choice of k yields fixed coefficients in the error bounds, so the expected convergence rate is clear from theory (although the choice may be sub-optimal with respect to the dimension dependent coefficients).

The evaluation of error norms requires some effort: one possible solution is to interpolate the reference solution on a "very fine" grid and compute the exact error against this approximation – but such an attempt might affect the order coefficients which we are going to analyze, especially for higher dimensions where it might not be feasible to compute the fine resolution. For this reason, we employ a different method: for the L_2 discretization error, we use the identity

$$\begin{aligned} \|f \circ \phi - f_{[-1/2, 1/2]^d}^h\|_{L_2[-1/2, 1/2]^d}^2 &= \|f \circ \phi\|_{L_2[-1/2, 1/2]^d}^2 \\ &\quad - 2(f \circ \phi, f_{[-1/2, 1/2]^d}^h) + \|f_{[-1/2, 1/2]^d}^h\|_{L_2[-1/2, 1/2]^d}^2 \end{aligned} \quad (3.58)$$

and compute each single summand separately: the first summand is known analytically and the last one is $\bar{f}^T M \bar{f}$ where \bar{f} is the coefficient vector of $f_{[-1/2, 1/2]^d}^h$ and M the mass matrix. This matrix vector product can be computed in linear time, without additional error, see Section 2.2.2. The L_2 inner product $(f \circ \phi, f_{[-1/2, 1/2]^d}^h)$ can be computed to very high precision as sums of *one-dimensional* integrals since both parts are (sums of) tensor product functions. Thus, we only need to compute one-dimensional integrals of $f_i \circ \phi_i$ against piecewise linear spline functions which can even be computed up to machine precision. However, every numerical realization of formula (3.58) *will* suffer from cancellation because we subtract numbers of similar magnitude. In other words: we will lose relative precision, especially as the error $f \circ \phi - f_{[-1/2, 1/2]^d}^h$ becomes small. We argue that we only need about three significant digits: the difference between $1.23 \cdot 10^{-6}$ and $1.234 \cdot 10^{-6}$ is negligible for post-processing. However, the *absolute* precision (the

exponent) is important. This is exactly what (3.58) allows – until the error drops below, say 10^{-7} . This is perfectly acceptable for low order splines.

Note that the same idea can be applied to energy norm computations as well, with the same features and the same restrictions.

However, the computations of $\|\cdot\|_{L_\infty}$ cannot be realized without spending a lot of point evaluations; either on a (sparse) grid or using random samples. In either case, only huge numbers of point evaluations yield precise error measurements. For this reason, we restrict ourselves to the L_2 and energy norms.

The balancing approach 1.) can be realized using our interpolation– and truncation error bounds: we fix the desired accuracy ϵ , compute the cut radius k using the truncation formula (3.24),

$$\epsilon = \frac{\|f - f_h^{\text{cut}}\|_{L_2[\mathbb{R}^d]}}{\|f\|_{L_2[\mathbb{R}^d]}} = \sqrt{1 - \text{erf}(k)^d} \quad (3.59)$$

and insert the resulting $k = k(\epsilon, d)$ into the interpolation error bounds. For L_2 , this can be simplified to

$$\epsilon := \frac{\|f - f_\Omega^h\|_{L_2[\Omega]}}{\|f\|_{L_2[\mathbb{R}^d]}} \lesssim 2^{-2n} (k^2/2)^d \quad (3.60)$$

where the ‘ \lesssim ’ indicates that we neglected the log term n^{d-1} to simplify the approach. Then, we compute $n = n(k(\epsilon, d), \epsilon, d)$ which yields

$$n(k(\epsilon, d), \epsilon, d) \approx \left\lceil -\frac{1}{2} \log_2 \left(\frac{\epsilon}{(k(\epsilon, d)^2/2)^2} \right) \right\rceil \quad (3.61)$$

where we have chosen the (optimistic) choice of the next lower level number.

The adaptive balancing method 3.) has been realized by means of prewavelet adaptivity: every wavelet coefficient, weighted with its relative L_2 norm of the basis function is used as error indicator. Coefficients which exceed a prescribed limit are refined. This yields a subset of regular sparse grids. The approach can be carried out as refinement loop of the form refine, recompute wavelet coefficients, estimate again and iterate until no new points have been inserted, compare Algorithm 1 on page 33. Note that a posteriori wavelet *compression* instead of additive refinement yields more information as the interpolant created on the resulting grid, since information of fine mesh widths is still contained in remaining basis coefficients. A fair comparison of interpolation errors requires to use only the interpolant on an (probably optimally chosen) grid, not more.

Numerical Estimation of Dimension Dependence

Figure 3.2 shows the resulting relation between degrees of freedom and the associated interpolation error of our model problem for dimensions $d = 1$ (red), $d = 2$ (blue), $d = 3$ (black) up to $d = 7$. For each dimension, we see the results for fixed domain parameter $k = 4.9$ with the circle marker, results for the a priori coupling of k and d according to (3.61) with the square markers and a posteriori grid optimization combined with fixed $k = 4.9$ drawn with triangle markers. As expected, the unbalanced case is sub-optimal

3.2 Case Study: Density Approximation and the Normal Distribution

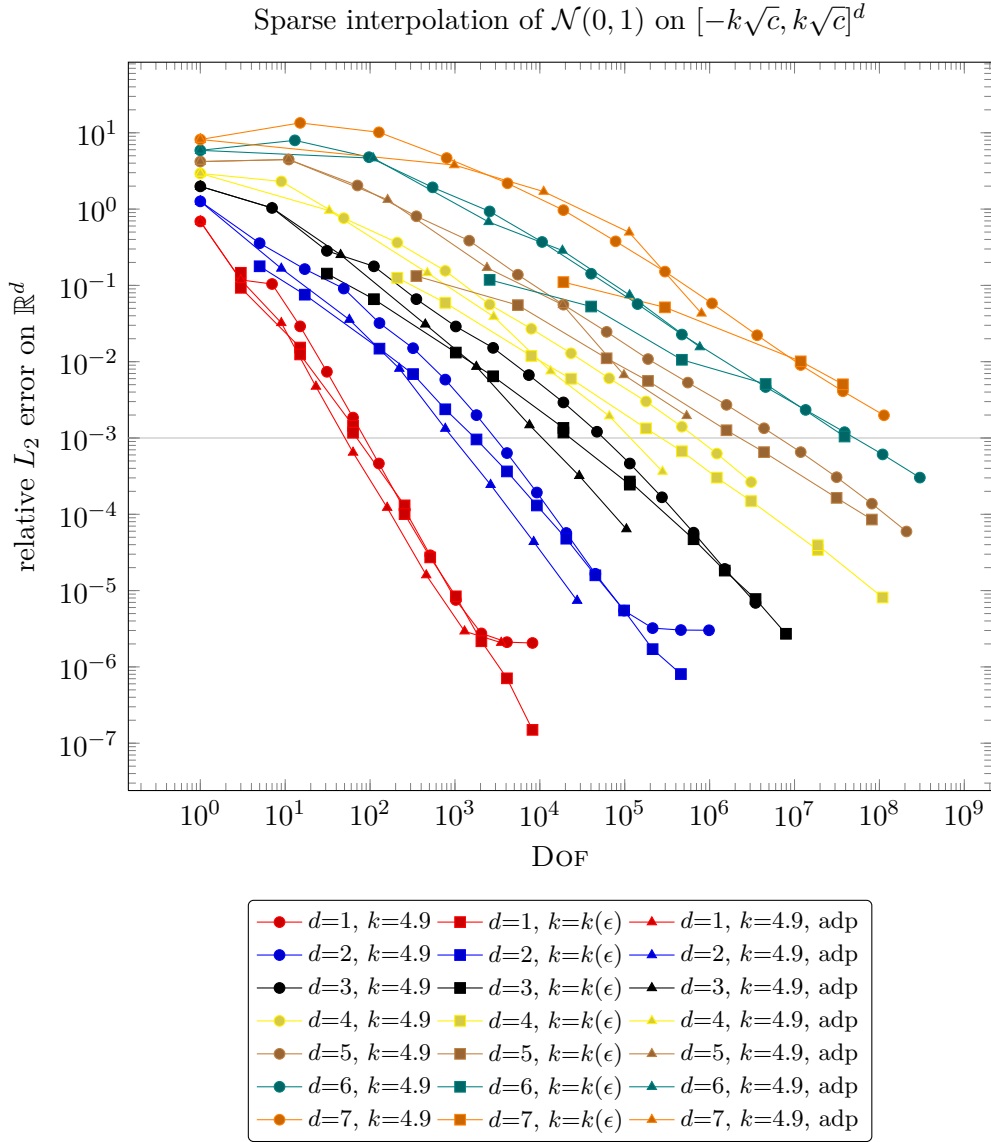


Figure 3.2: Measurement of relative L_2 errors on regular sparse grids with fixed k , regular sparse grids with coupled k and n according to (3.61) and adaptive sparse grids starting from fixed k . Same dimension yields the same color. Same method yields the same plot marker. The line for $\epsilon = 1 \cdot 10^{-3}$ is analyzed in Figure 3.3.

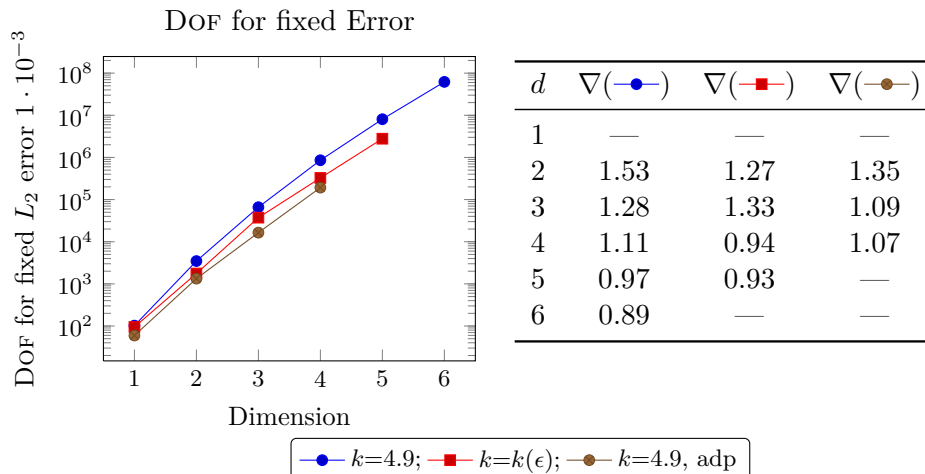


Figure 3.3: Degrees of freedom to achieve a fixed relative precision of $\epsilon = 1 \cdot 10^{-3}$ computed from Figure 3.2 (left) and their semilogarithmic slopes for increasing d (right).

and yields inferior results compared to the other methods. Nevertheless, the slopes² have the expected order for fixed k . We also see that a posteriori grid optimization yields the predicted slopes since the function is smooth and we only attempt to compensate for the domain truncation. The a priori coupling of n and k finally yields higher accuracy for less degrees of freedom compared to the unbalanced case, and its cost/gain ratio is somewhat between the adaptive and the unbalanced case. Note that it becomes worse for larger dimensions which has to be accounted for the log term. Further post processing reveals that for $d \leq 5$, the quotient between discretization and truncation error is between 1 and 2 whereas for $d = 7$, the truncation error is a factor of seven larger than the discretization error. This indicates that we spent about one or two levels too much here.

But despite the improved balancing, the results in Figure 3.2 exhibit the same qualitative behavior as indicated by our theoretical error bounds: the cost/gain ratio becomes worse for increasing dimension. If we hold the error fixed at a prescribed ϵ and collect the degrees of freedom required for such an error as intersections of our plot lines with a line parallel to the x axis passing through ϵ , we get Figure 3.3. Here we choose the fixed relative L_2 error $\epsilon = 10^{-3}$.

The lines in Figure 3.3 are almost parallel with slope $0.9 \leq m \leq 1$ in the semilogarithmic plot. The figure uses a logarithmic base of 10, which we emphasize in the following slope computations using the “ x base 10” notation. Thus, to get a relative L_2 interpolation error of $\epsilon = 10^{-3}$ for our model problem of an isotropic Gaussian, we need $N(d; \epsilon) = 10^{m \cdot d} \cdot C$ degrees of freedom. From Figure 3.3 (right), we find experimental slopes of approximately $m \approx 1$ base 10, with tendency to become slightly smaller for increasing d for all three methods. Thus, we need 10 times more degrees of freedom for

²The electronic version of this document supports to drag-and-drop into the figure to compute slopes.

3.2 Case Study: Density Approximation and the Normal Distribution

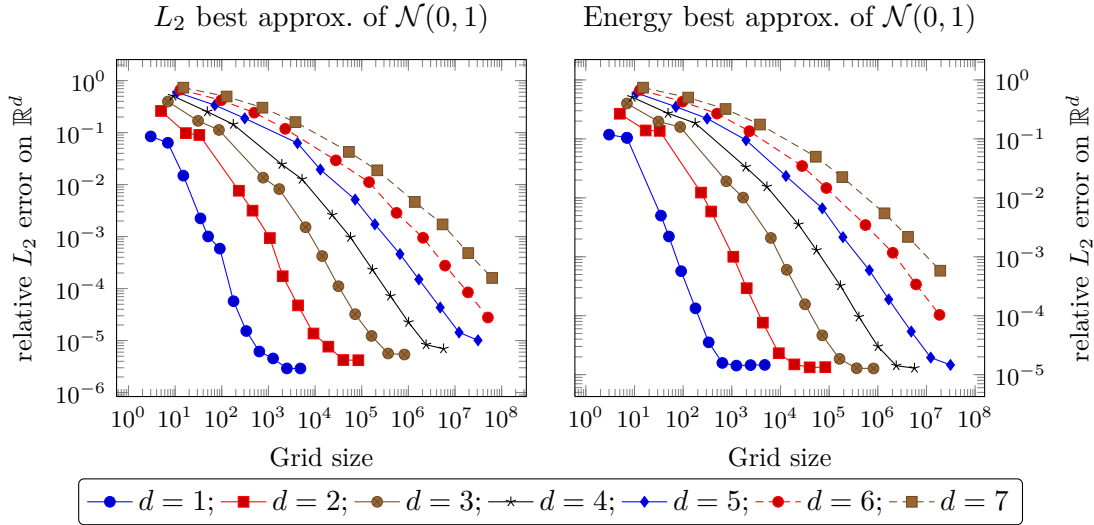


Figure 3.4: Best approximation of a normal distribution $\mathcal{N}(0,1)$ obtained with respect to L_2 inner products (left) and energy norm (right). Both measure the L_2 error, rely on adaptive refinement, and use the domain $[-4.9, 4.9]^d$.

the same problem posed in one dimension larger for fixed $\epsilon = 10^{-3}$.

Speaking in terms of sparse grid levels n instead of degrees of freedom N , the experiments always use sparse grid level $n = 1$ as first data point (i.e. 10^0 degrees of freedom), followed by level step sizes of 1. Thus, the fixed error line requires about $n = 6$ for $d = 1$, $n = 8$ for $d = 2$, and $n = 10$ for $d = 3$ which can be found by counting data points in Figure 3.2.

Note that the rule of thumb “cost factor 10 for increasing d ” does not degenerate exponentially for smaller ϵ since only the logarithmic terms in our error estimates depend exponentially on d . If we omit the logarithmic terms to a theoretical justification for our rule of thumb, we have $N = 2^{n+c_1 \cdot d}$ degrees of freedom with a cost constant c_1 (including logs, it would have been $\mathcal{O}(2^n n^{d-1} 2^{C \cdot d})$ with a different constant C) and an error $\epsilon = 2^{-2n+c_2 \cdot d}$ with error constant c_2 (again, this would be of order $\mathcal{O}(2^{-2n} n^{d-1} 2^{C \cdot d})$ with logs). Thus, we can eliminate n to get the rule of thumb $N = \epsilon^{-1/2} 2^{(c_1+c_2/2) \cdot d}$ where we got experimentally $2^{(c_1+c_2/2) \cdot d} \approx 10^d$. Thus, the dependence of ϵ does not degenerate exponentially with d and we find that about 10 times more degrees of freedom are necessary to increase the dimension by 1 and maintain the same relative L_2 error.

For the sake of completeness, we document the effect also for best approximation: Figure 3.4 (left) shows similar results obtained by adaptive L_2 best approximation. They exhibit the same fan-like structure indicating exponentially growing order coefficients. Figure 3.4 (right) shows the results obtained by means of energy norm best approximation (a Galerkin method using techniques of [Feu05]) which is related to solvers for PDEs. It exhibits similar exponential growth like interpolation and L_2 best approximation with respect to its coefficients.

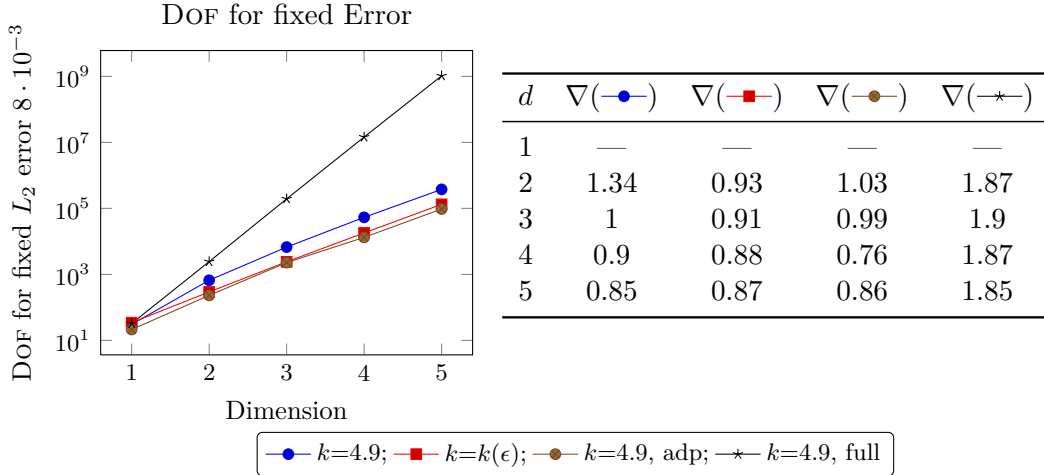


Figure 3.5: Degrees of freedom to achieve a fixed relative precision of $\epsilon = 8 \cdot 10^{-3}$ computed from Figure 3.2 and for a full grid method (left) and the associated semilogarithmic slopes (right).

We summarize that sparse grid approximation of the normal distribution is still subdued by the “curse of dimension” which manifests itself in the complexity coefficients.

This raises the question in how far sparse grids are superior to classical full grids. We complement our study with a short experiment with full grids, carried out with brute force up to 10^9 degrees of freedom in dimension $d = 5$: we compute the degrees of freedom of a full grid method required to get a relative L_2 error of ϵ and compare the resulting curve with the one for sparse grids (and the same ϵ). Before we discuss the experiment, we summarize well known results about their cost/gain relation. For our dyadic mesh width $h = 2^{-n}$, we have $N = 2^{n \cdot d}$ degrees of freedom which achieve an accuracy of the form $\epsilon = 2^{-2n + c_2 \cdot d}$, again with a generic error constant c_2 , measured in the relative L_2 norm as before (neglecting non-exponential d dependent factors and collecting any occurring exponential coefficient in c_2). Elimination of n yields the full grid ϵ complexity

$$N = \epsilon^{-d/2} 2^{1/2 c_2 \cdot d^2}. \quad (3.62)$$

Note that we do not have the mix regularity norm here, only the H^2 norm enters the regularity, so c_2 can be expected to be less than the one of sparse grids. The $\epsilon^{-d/2}$ term is a severe exponential degeneration of ϵ with d as opposed to sparse grids. Let us turn to the experimental results when we set a fixed relative error $\epsilon = 8 \cdot 10^{-3}$: we compute the required degrees of freedom to achieve this relative error for our three sparse grid variants and classical full grids (without truncation error balancing). The result is shown in Figure 3.5. The slope of the sparse grid curves is again in the range $0.8 \leq m \leq 1$ base 10 for all our variants whereas the full grid complexity grows linearly with slope 2 base 10. The linear growth indicates $c_2 = 0$ (or at least very small) in (3.62). Thus, we need about 100 times more degrees of freedom for one more dimension (100 is the number of points to achieve accuracy $8 \cdot 10^{-3}$ in one dimension). This is to be compared with

3.2 Case Study: Density Approximation and the Normal Distribution

10 times more degrees of freedom for any of the sparse grid variants. Due to the $\epsilon^{-d/2}$ term, this relation becomes worse with smaller ϵ . In this respect, sparse grids reduce the effects of the curse of dimension and increase the number of dimensions for which computer experiments become feasible when we consider functions like the Gaussian.

So far, we have only considered L_2 (or L_∞) optimal sparse grids. We will now show that energy norm optimized sparse grids exhibit almost the same factor $m \approx 1$ base 10 of exponential growth as their L_2 optimal relatives. As already mentioned, the energy error on \mathbb{R}^d for our simple truncation method is unbounded due to the jump on $\partial\Omega$. Since this could be fixed by a simple smoothing, we neglect this effect and measure errors in regimes which are barely affected: we compute the energy norm error on Ω ,

$$\epsilon = \frac{\|f - f_\Omega^h\|_{A[\Omega]}}{\|f\|_{A[\mathbb{R}^d]}}, \quad (3.63)$$

and neglect the truncation error. This is not too serious since our previous experiments revealed that a priori balancing of truncation and discretization error does not reduce the slopes which we are going to measure. Of course, the energy norm will become unbounded for $n \rightarrow \infty$ as well: we have a jump of width $h = 2^{-n}$ near the boundary since we assume $f = 0$ on $\partial\Omega$. This will dominate the error if n becomes large enough. Our experiments will be useful only up to a certain resolution.

We compare energy optimized grids without balance of truncation- and discretization error and adaptively refined energy optimal grids (i.e. choice 2.) and 3.) of our balancing methods). Since the adaptive approach will attempt to resolve the jump near the boundary for high target accuracies, we limit the highest allowed resolution for the adaptive approach. As soon as the adaptivity starts resolving the jump, we abort the experiment.

Results of this approach are shown in Figure 3.6: we see the degrees of freedom plotted against the relative energy error on Ω for both methods. The first thing to note is the jump which becomes visible at a certain resolution. Furthermore, we find that adaptive and regular methods for fixed dimension yield almost parallel lines, where the adaptive approach requires less degrees of freedom. The convergence rate with respect to n is, indeed, 1 for dimension $d = 1$, but asymptotics start considerably later for higher dimensions: for $d = 2$, we observe a convergence rate of about 0.9 instead of 1 and the case $d = 3$ has about 0.8. The theoretical rate 1 is valid for very large levels n only.

Despite the lesser rate of convergence, the picture has the same quality as Figure 3.2 for L_2 optimal sparse grids: we still observe the fan structure indicating exponentially d -dependent coefficients. This becomes more evident if we compute the degrees of freedom required to achieve a fixed error, which is depicted in Figure 3.7: both methods yield parallel lines when computing the degrees of freedom required for a fixed relative energy error $\epsilon = 5 \cdot 10^{-2}$. The effect is slightly lower for the adaptive approach whereas the slope is $m \approx 0.9$ base 10, i.e. we need $10^{0.9}$ times more degrees of freedom to get the same accuracy for one dimension larger.

We conclude so far that the standard normal distribution leads to order coefficients which grow exponentially. The effect occurs for interpolation and bestapproximation and for L_2 - and energy norms. It also affects sparse grid spaces which are optimized for the energy norm.

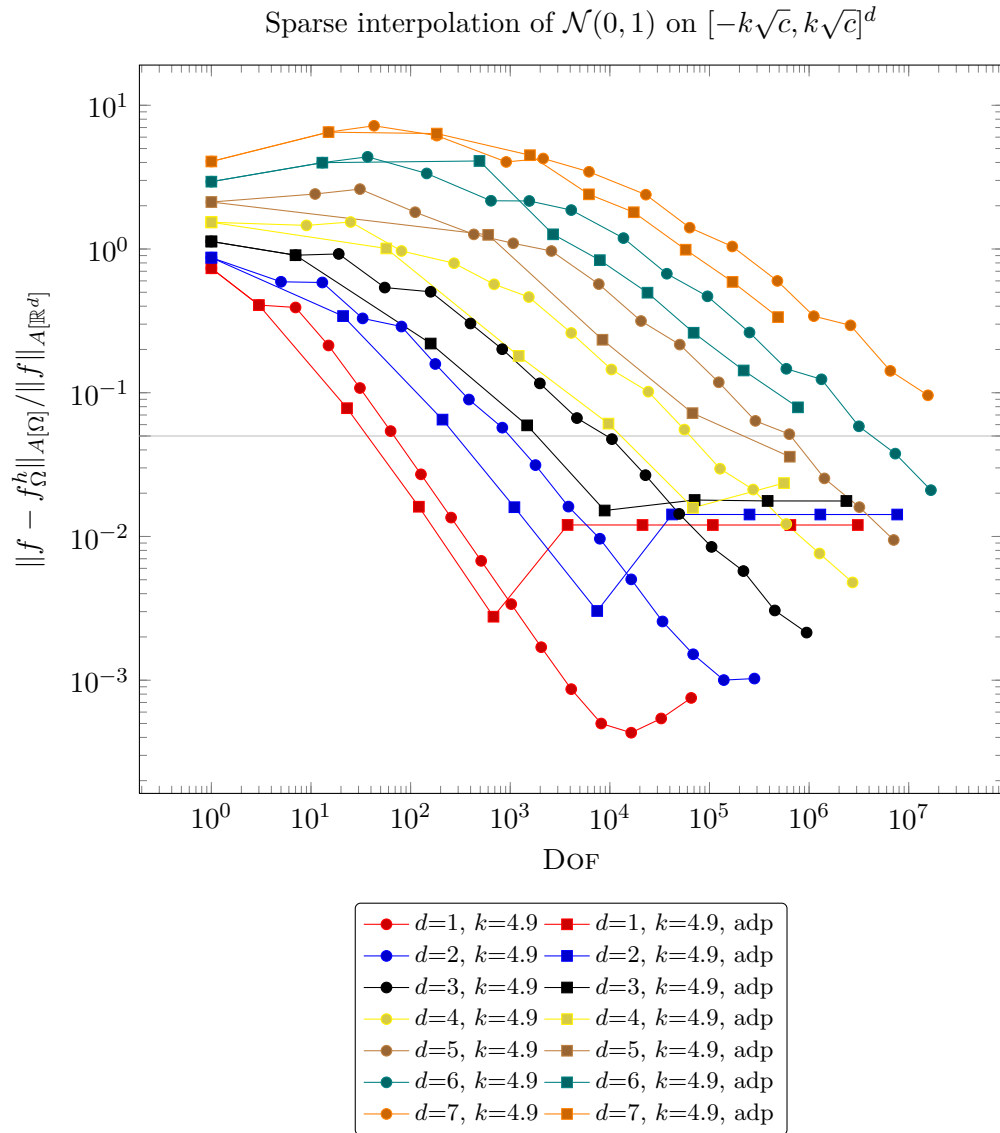


Figure 3.6: Measurement of relative energy errors with energy optimal sparse grids. The jump near the boundary has been avoided by providing a maximum level.

Confirming the Parameter Independence – Decaying Variances

We complement our experiments on the standard normal distribution with experiments on normal distributions with decaying variances of the form $\sigma_i = 2^{-i}$. Figure 3.8 displays the results of three experiments: the first (red) replicates results of the last section which uses a fixed variance $\sigma = 1$ and $\Omega = [-4.9, 4.9]^d$. The different dimensions are identified by different markers. The second experiment is displayed in blue and uses decaying variances $\sigma_i = 2^{-i}$ and, at the same time, decreasing domain size with

3.2 Case Study: Density Approximation and the Normal Distribution

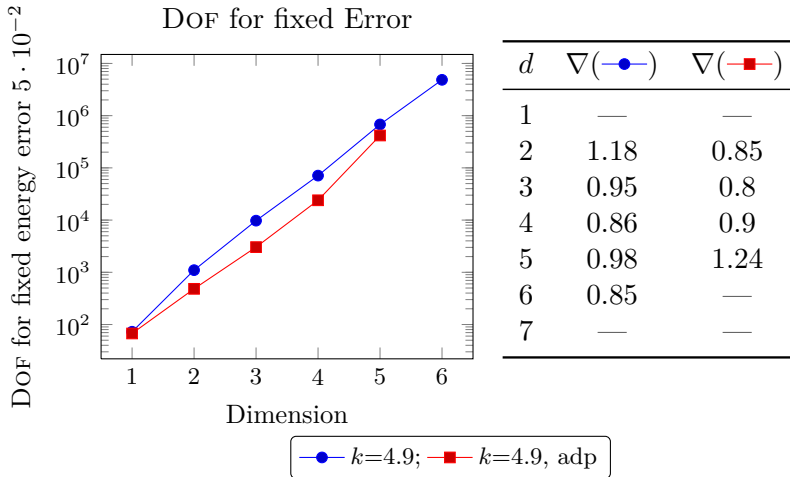


Figure 3.7: Degrees of freedom to achieve a fixed relative energy error of ϵ computed from Figure 3.6. The error is evaluated on Ω , more precisely using $\|f - f_{\Omega}^h\|_{A[\Omega]} / \|f\|_{A[\mathbb{R}^d]}$.

$\Omega = \otimes[-4.9\sqrt{\sigma_i}, 4.9\sqrt{\sigma_i}]$. The plot marks identify the dimension and are the same as for the first (red) experiment. The third experiment is displayed in green and uses decaying variances $\sigma_i = 2^{-i}$ but fixed domain $\Omega = [-4.9, 4.9]^d$. All three experiments use adaptive grid refinement to optimize the degrees of freedom (thus, they have non-linearly graded grids). On the x axis, we show the (resulting) grid size as degrees of freedom, plotted against the relative L_2 error measured on \mathbb{R}^d . We see that the first two experiments yield exactly the same cost/gain ratio as predicted by theory; the anisotropy is compensated by the domain size (the “window” onto the function). Furthermore, the third experiment is considerable more expensive with increasing dimension d . This is due to the fact that we are resolving a narrow peak relative to the fixed domain size. The additional degrees of freedom are required to cover the domain until finally the peak is resolved. Note, however, that the convergence plots exhibit the same slopes as for the first experiments.

In summary, we confirm the independence of the variance σ_i for the case where σ is coupled to the domain size and relative errors are of interest. Furthermore, we see that adaptive grid refinement for a poor choice of the simulation domain yields the same slopes with respect to the mesh width h , but it cannot compensate for the increase of dimension dependent coefficients induced by the large domain.

3.2.3 Generalizations

Our approach in the previous sections was to cover the relevant parts of \mathbb{R}^d by coupling the width (variance) of the probability distribution to the truncated domain size, followed by a linear coordinate transformation. The combination yields parameter independence for relative error interpolation and results in order coefficients which grow exponentially with the dimension d . Furthermore, our adaptive approach provides non-linearly graded

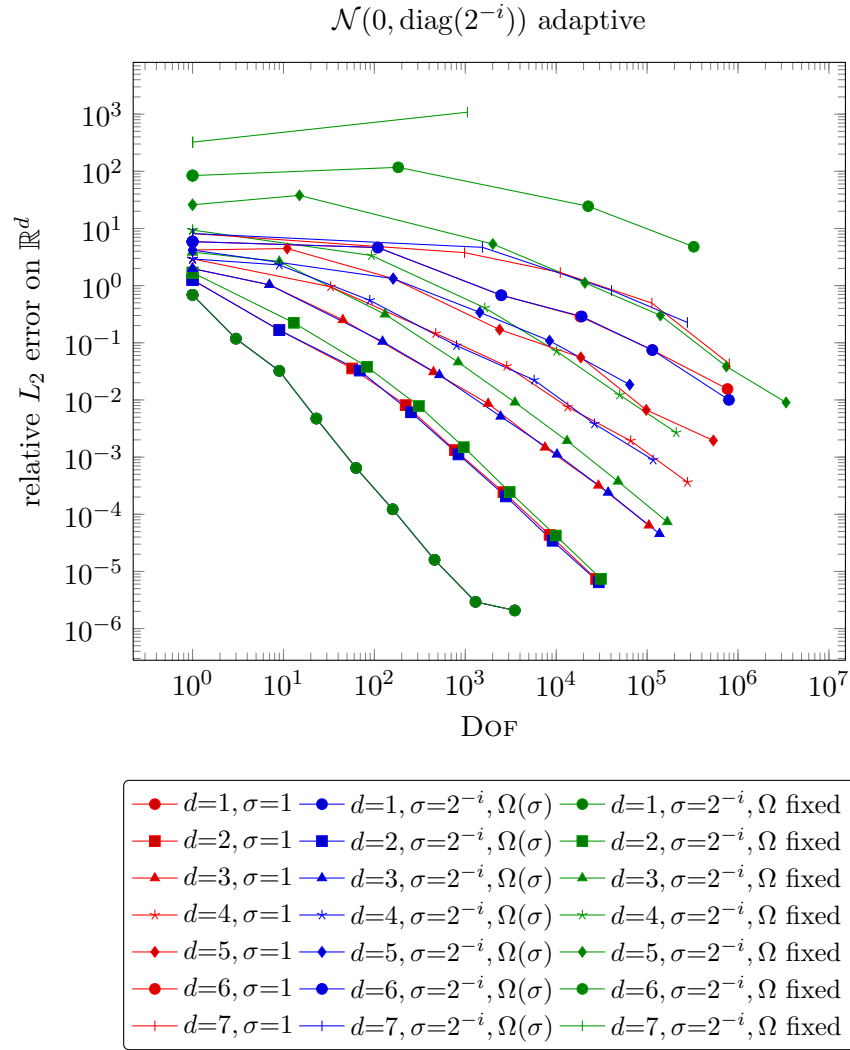


Figure 3.8: Adaptive grid refinement for the isotropic Gaussian on $[-4.9, 4.9]^d$ (red), the anisotropic Gaussian with $\sigma_i = 2^{-i}$ on $\otimes[-4.9\sqrt{\sigma_i}, 4.9\sqrt{\sigma_i}]$ (blue) and the same anisotropic Gaussian on $[-4.9, 4.9]^d$ (green).

grids to improve the distribution of degrees of freedom in the resulting simulation domain. The following section studies the application of more general transformations.

A Priori nonlinear Grid Grading

We investigate an a priori transformation which yields graded grids based on knowledge about the approximant. It is known that the inverse cumulative normal distribution is a good candidate for graded grids when normal distributions are involved, see [Hol08]

3.2 Case Study: Density Approximation and the Normal Distribution

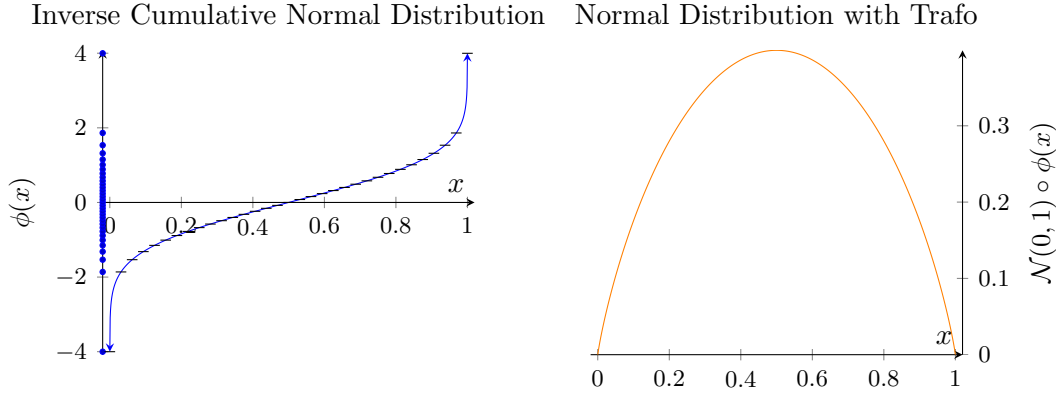


Figure 3.9: The inverse cumulative normal distribution and the resulting grid grading (left) and the transformed normal distribution (right)

and the references therein. Consequently, we consider the transformation

$$\phi: [0, 1]^d \rightarrow \mathbb{R}^d \quad (3.64)$$

such that $\phi^{-1}: \mathbb{R}^d \rightarrow [0, 1]^d$ is the cumulative normal distribution for variance 1, i.e.

$$\phi^{-1}(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt. \quad (3.65)$$

The parameters of the inverse cumulative normal distribution ϕ are chosen such that ϕ and our approximant fit together (using expected value $\mu_i = 0$ and variance $\sigma_i = 1$ for $i = 1, \dots, d$). Algorithms to evaluate the inverse cumulative normal distribution can be found in [Mor95b] whereas algorithms for the cumulative normal distribution are described in [Mor95a]. The effect of the inverse cumulative normal distribution is to use almost all of $[0, 1]^d$ for the area of \mathbb{R}^d in which almost all of the relevant features of f are situated whereas the tails of f will be mapped to a small boundary slice of $[0, 1]^d$. This is illustrated in Figure 3.9 (left): we see that $\phi(0 + \epsilon) \approx -4$ and $\phi(1 - \epsilon) \approx +4$ where for one dimension, almost all of the probability mass of f can be found. The grid points are sampled uniformly on the x axis of Figure 3.9 (left) which produces the graded grid shown on the y axis. Figure 3.9 (right) shows the resulting transformed Gaussian $f \circ \phi$ which is no longer strongly peaked but appears more like a parable.

We still need some sort of domain truncation since $\phi(0) = -\infty$ and $\phi(1) = \infty$. Motivated by our results of preceding sections, we choose $\Omega = [-4, 4]^d \subset \mathbb{R}^d$ and the transformed domain $\tilde{\Omega} = [0 + \epsilon, 1 - \epsilon]^d \subset [0, 1]^d$ such that $\phi_i(\epsilon) = -4$ and $\phi_i(1 - \epsilon) = +4$. We then transform $\tilde{\Omega}$ linearly to the unit cube and introduce vanishing boundary conditions of Dirichlet type as in the preceding sections. To avoid complicated integral transformations for norm evaluation, we restrict ourselves to error evaluation in the L_∞ norm on \mathbb{R}^d only, using a fine grid for the error estimation (three times uniformly refined).

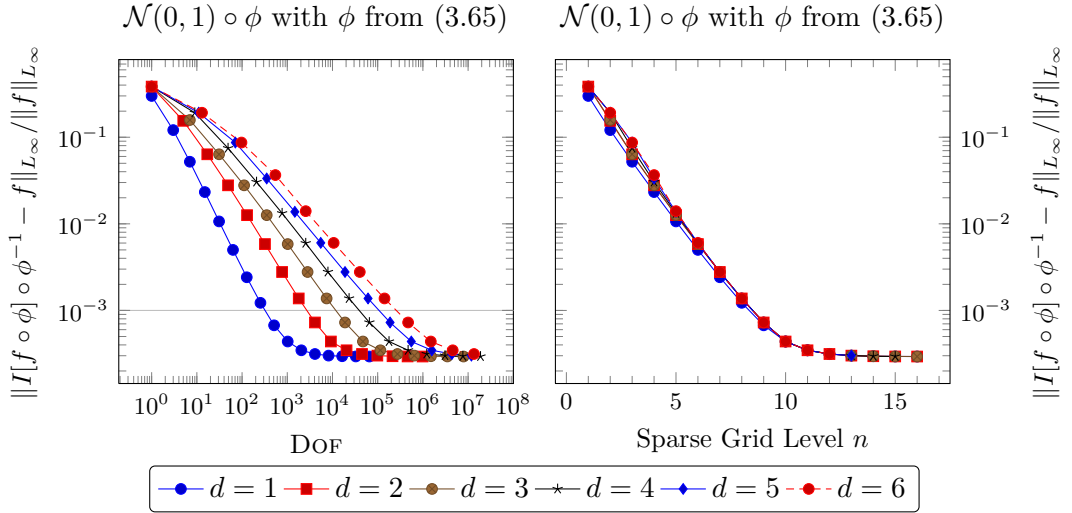


Figure 3.10: Interpolation results for the inverse cumulative normal distribution using the truncated domain $\Omega^{\text{cut}} = \phi^{-1}([-4, 4]^d) = [0+\varepsilon, 1-\varepsilon]^d$.

Figure 3.10 (left) shows interpolation results for the described experiment: we have degrees of freedom versus the relative L_∞ error

$$\frac{\|f - f_\Omega^h\|_{L_\infty[\Omega]}}{\|f\|_{L_\infty[\mathbb{R}^d]}} = \frac{\|f \circ \phi - f_{[0,1]^d}^h\|_{L_\infty[0,1]^d}}{\|f\|_{L_\infty[\mathbb{R}^d]}} \quad (3.66)$$

for dimensions $d = 1, \dots, 6$, this time using regular sparse grids to concentrate on the non-linear grid grading induced by ϕ . Besides the saturation effects originating in the boundary truncation, we see that data points with the same index among different dimensions are almost at the same y value. Data points represent sparse grid levels $n = 1, 2, 3, \dots$, so we get approximately the same error for every dimension if we keep n fixed. This is also depicted in Figure 3.10 (right) which shows the same results plotted against the level n instead of the degrees of freedom. Here, the effect is more obvious: the error does no longer depend on d as a result of our transformation. The increase of degrees of freedom has to be accounted for the log term in the sparse grid complexity only. This, in turn, grows sub-exponentially with respect to d (for fixed level n) as can be seen in Table 3.1: the table shows the required degrees of freedom to achieve a relative L_∞ accuracy of 10^{-3} for $d = 1, \dots, 6$, together with the respective semilogarithmic slopes. We see that these slopes decrease with d . This can be explained by the leading term of the cost complexity of sparse grids which is of the order $2^n n^{d-1} / (d-1)!$. The faculty $(d-1)!$ grows faster than c^d for any c and large dimension d , and we are effectively keeping n constant. Thus, the transformation reduces the overall complexity considerably. However, we have to pay for the singular nature near the end points: the slopes of Figure 3.10 (left) only indicate first order convergence for $d = 1$ and even lower rates for $d > 1$. Thus, we loose effectively one order with respect to the mesh width. This loss is caused by large errors in the tails of the distribution.

3.2 Case Study: Density Approximation and the Normal Distribution

Table 3.1: The degrees of freedom in Figure 3.10 (left) required for fixed relative L_∞ error of 10^{-3} .

d	DOF for $L_\infty = 10^{-3}$	\log_{10} -grad
1	323	–
2	2,708	0.92
3	11,847	0.64
4	39,035	0.52
5	107,106	0.44
6	257,936	0.38

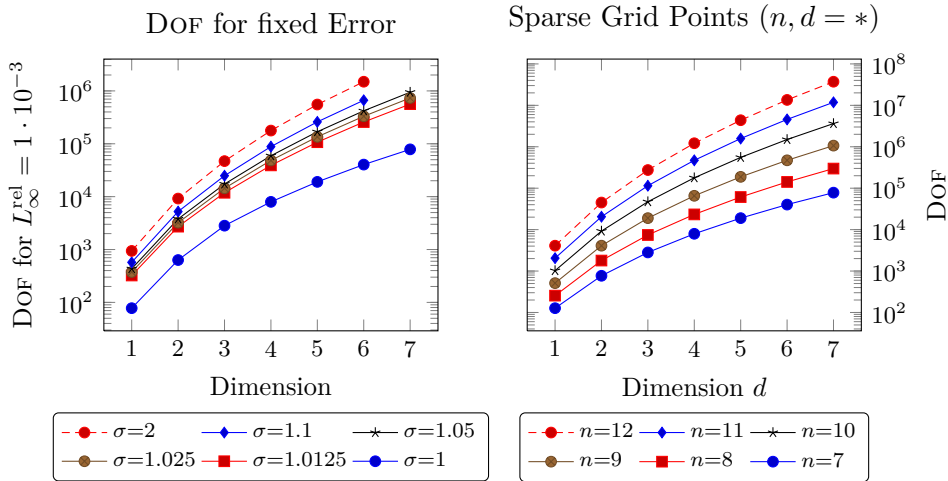


Figure 3.11: Results for the nonlinear domain transformation and different variances (left) and the related sparse grid complexity for fixed level n (right).

So far, our nonlinear transformation has been chosen to fit perfectly to our approximant; we have effectively plugged information about the solution into our transformation. This leads to the question what happens if the transformation is applied although it does not fit perfectly. To investigate in the parameter dependence, we analyze the performance of the *same* transformation applied to normal distributions with variances $\sigma \in \{0.5, 1, 1.0125, 1.025, 1.05, 1.1\}$ and common expected value $\mu = 0$. To this end, we choose again a simulation domain which is coupled to σ , namely $\Omega = [-4\sqrt{\sigma}, +4\sqrt{\sigma}]^d$ in order to maintain a small cut error. However, the transformation still assumes $\sigma = 1$.

The results exhibit the same qualitative behavior: the error depends only on the sparse grid discretization level n and, additionally, on σ – but it is independent of the dimension d . They also show the reduction of the convergence rate with respect to n pre-asymptotics. The only parameter which influences the outcome of this experiment is σ which is depicted in Figure 3.11 (left). The picture shows the required degrees of freedom to achieve a relative L_∞ error of 10^{-3} for the different normal distributions, plotted for $d = 1, \dots, 6$. We see that the lines grow, but they grow sub-exponentially since their

slopes become smaller with increasing d . This is, again, only an effect of the sparse grid complexity (its log term). Furthermore, larger variances σ require more degrees of freedom which results in a vertical offset. The observation that only the degrees of freedom depend on d whereas the error depends on n and σ motivates a visualization of the degrees of freedom for fixed level n and varying d . Such a visualization has exactly the same form as discussed for our experiments and is shown in Figure 3.11 (right): we see the sub-exponential growth with d for level $n = 7, 8, \dots, 14$.

We conclude that the cumulative normal distribution yields an a priori grid grading which concentrates near the mean value of the Gaussian, neglecting the tails. The low resolution near the tails reduces the overall convergence rate by approximately one order. The benefit is that its error becomes independent of the dimension, even if parameters do not fit perfectly. The cost complexity for fixed error depends on the low convergence rate and on the sparse grid cost complexity, which grows with d in its log term. Note that these sparse grid points carry significant information: the transformed function is still high dimensional (in fact, the highest order ANOVA term, compare Chapter 4, dominates the representation). The transformation relies on Gaussian decay in the tails and appears to be valuable if such decay is given.

On Non-Linear Problem Formulations

The observation that linear approximation tools like our sparse grid method exhibit exponential growth when it comes to resolution of high dimensional densities raises the questions if non-linear approaches can improve the situation. We discuss several approaches to combine sparse grid methods and non-linear methods and their application to density approximation (which is often formulated by means of the Fokker-Planck-Equation or a similar model).

One idea is to enrich the basis by something which is close to the solution and resolve the rest by means of a linear method, for example sparse grids. In our case, we could try to use a Gaussian basis function with properly chosen mean and variances, combined with either a multiplicative splitting Gaussian times rest or an additive splitting Gaussian plus rest. Then, the “rest” term can be approximated by means of sparse grids (compare the related spectral methods with multiplicative Gaussian splitting discussed in [MST05]). The additive approach becomes approximation of a difference of Gaussians unless the difference between the Gaussian basis function and the solution can be neglected completely. A difference of Gaussians is more complicated than a single Gaussian, so a sparse grid method will have at least the same (exponential) cost as for a single Gaussian.

A multiplicative approach of the form Gaussian times rest also yields a rest which is governed by the term $\exp(-x^T x)$ if we assume the simplest case of a Gaussian solution, approximated by Gaussian times rest. However, the rest of type $\exp(-x^T x)$ lives on a smaller scale and is thus of a different quality as our density approximation framework: consider the quotient of two Gaussians with almost the same variance in one dimension,

3.2 Case Study: Density Approximation and the Normal Distribution

$$\frac{(2\pi\sigma)^{-1/2} \exp(-\frac{x^2}{2\sigma})}{(2\pi\sigma(1+\epsilon))^{-1/2} \exp(-\frac{x^2}{2\sigma(1+\epsilon)})} = \exp(-\frac{1}{2} \frac{x^2}{\frac{\sigma}{\epsilon} + 1}) \cdot \sqrt{1+\epsilon}. \quad (3.67)$$

Consequently, the quotient is of Gaussian type, but its variance is considerably larger – and the relative error is measured against the target function, not the quotient. Thus, it is almost constant on the scale of interest. Furthermore, the quotient of two Gaussians with almost the same *mean* is dominated by

$$\frac{\exp(-\frac{x^2}{2})}{\exp(-\frac{(x+\epsilon)^2}{2})} = \exp(x\epsilon) \exp(\epsilon^2/2) \approx C \cdot (1 + \epsilon x) \quad (3.68)$$

for small ϵ , i.e. it is nearly linear if ϵ is small enough. On the other hand, the division by a rapidly decaying function has a bad absolute condition number and requires attention when applied together with boundary truncation schemes. Nevertheless, a multiplicative splitting which already contains essential parts of the solution might prove to be cheaper than a direct method. We come back to such a splitting in Section 4.4 where we consider nonlinear dimensionwise decompositions of product type, $f = \prod g_i$ times rest.

The key questions for any particular application are then: is the solution close (enough) to a Gaussian yet distant enough to require the “rest” term? And: can we find the Gaussian part? These questions need to be answered before applying specialized methods and are already part of our outlook.

Another promising non-linear approach is to approximate the log density instead of the density if it is always positive: for a Gaussian, the log density is a superposition of at most two-dimensional functions (see also the adaptive sparse grid illustration in Figure 2.13 on page 58). It might even allow dimension adaptive procedures which are subject of Chapter 4.

The price to transform a Fokker–Planck–Equation to a log density formulation is a non-linear term on the one hand and a complicated boundary condition on the other hand. We discuss why especially the boundary conditions make such an approach ineffective. Often, one uses the density normalization $\int_{\mathbb{R}} p(x) dx = 1$ and the implied decay $\lim_{\|x\| \rightarrow \infty} p(x) = 0$ to truncate the density to a sufficiently large domain. Providing function values on the boundary has several effects: first, vanishing boundary conditions are not compatible with log density approximation. Second, one cannot just “truncate” the boundary value of the logarithm: e^{-10} might be small relative to $\mathcal{O}(1)$, but -10 is significant. Even more, boundary values already carry 100% of the information if the function is actually a superposition of low dimensional contributions. For example, the function $f(x, y) = -x^2 - y^2$ can be represented with the cost of two one-dimensional grids as in Figure 2.13 on page 58. The required degrees of freedom can be fixed by inspecting boundary values, see also the excursion about Dirichlet conditions and dimension adaptive partial differential equations on page 147. Consequently, a log density formulation can result in a very simple structured solution (as for the Gaussian), but it moves the difficulty into the boundary conditions and the differential operator. In fact, the PDE solution might be zero if all information can be deduced from the boundary. In

the general case, the lack of precise boundary conditions for full space problems makes log density models in attractive.

3.2.4 Related Topics in the Area of Information Based Complexity

A more general discussion of d -dependent order coefficients is subject of the area Information Based Complexity (IBC). There, one studies the worst-case error of a problem over a complete class of functions. If the minimum cost complexity, measured in terms of either function evaluations or application of linear operations, for the worst-case error is not exponential, the problem class is said to be *tractable*. If the cost grows either exponentially in terms of constant to the d or in terms of the accuracy, ϵ^{-d} , the problem is said to be *intractable*, see [NW08].

In this section, we summarize results of IBC related to our case study. To this end, we provide a brief survey over results presented in [NW08] and [TW99].

The tractability of a problem class relies on the following ingredients: a criterion to describe non-exponential cost complexity, the definition of a unit ball (i.e. the norm), a choice which types of algorithms are allowed (like point evaluation etc.) and the definition of an error ϵ measured over the complete problem class for one fixed algorithm.

The non-exponential growth criteria for the cost complexity can be formalized by a non-decreasing function $T(\epsilon^{-1}, d)$ which is not exponential, i.e.

$$\lim_{x+y \rightarrow \infty} \frac{\log T(x, y)}{x + y} = 0. \quad (3.69)$$

If the cost $N(\epsilon, d)$ required for an error ϵ in dimension d is bounded by (a polynomial of) T , $N(\epsilon, d) \leq T(\epsilon^{-1}, d)$, the problem class is *tractable*. Here $T(x, y) = xy$ yields polynomial tractability which is equivalent to $N(\epsilon, d) \leq C\epsilon^{-p}d^q$ for all $\epsilon < 1$ and all d . For $T(x, y) = \exp((1 + \log x)(1 + \log y))$, the notion of tractability allows also super polynomial growth of $N(\epsilon, d)$ and is called *weak tractability*, compare [NW08].

The IBC literature commonly analyzes the worst case absolute error over the problem class X for a particular algorithm A_N taking N pieces of information,

$$\epsilon^{\text{wor}}(A_N) := \sup_{f \in X, \|f\| \leq 1} \| \|A(f) - A_N(f)\| \|, \quad (3.70)$$

where $A(f)$ is the correct result and $\| \cdot \|$ a proper norm of to compare the result. The minimum cost complexity is then

$$N(\epsilon, d) = \min\{N \mid \text{there exists } A_N \text{ with } \epsilon^{\text{wor}}(A_N) \leq \epsilon\}. \quad (3.71)$$

Other error measurements over the problem class use averaged errors or probabilistically selected sub-parts of the problem class. The relative error is only of limited use when its worst case is considered over a complete function class: the best bound which can be obtained is a worst case relative error of 100%. This problem occurs since relative errors are not limited to unit balls, so there can always be functions in the function class for which a fixed algorithm A_N assumes it is the constant zero (based on N bits

3.2 Case Study: Density Approximation and the Normal Distribution

of information). Thus, the worst case formulation over a complete function class is not applicable here, compare [NW08, Example 10].

Concerning the worst case error, several examples of problem classes with exponential cost complexity have been identified in the literature. An important example is classical approximation in Sobolev spaces of order r which yields a cost complexity of order $\Theta(c(d)\epsilon^{-d/r})$ for any algorithm. Such a complexity grows faster than any polynomial and is thus polynomially intractable, no matter which coefficient $c(d)$ enters the complexity. If the coefficient $c(d)$ decreases exponentially with d , the problem may become weakly tractable, compare [NW08, p. 11]. Another example of limited smoothness is the space of Lipschitz functions characterized by

$$\|f\| = \max\left(\sup_{x \in [0,1]^d} |f(x)|, \sup_{x,y \in [0,1]^d} \frac{|f(x) - f(y)|}{\|x - y\|_\infty}\right) < \infty \quad (3.72)$$

which is also intractable, i.e. there are elements requiring exponential cost for any algorithm. Surprisingly, the space of infinitely differentiable functions C^∞ is also polynomially intractable despite spectral convergence order if the unit ball with respect to either the L_2 norm or a norm involving L_2 and any higher order derivative is considered. If derivatives are considered, it is even weakly intractable as well, whereas it is weakly tractable with respect to the L_2 norm, compare [NW08, Example 4].

Note that all these intractability results tell nothing about a *particular* element of the underlying function spaces, they state an existence result about at least one bad element. Only positive tractability results allow guaranteed cost complexities for all elements.

The space $X^{q,2}$ with functions of bounded mixed derivatives is tractable with respect to ϵ as we have seen constructively in Lemma 2.1.16, see also [TW99, p. 38]. The log-terms are considered together with the involved dimension-dependent coefficients. However, for our case of the Gaussian and the sparse grid method, the coefficients grow exponentially with d , which is clearly not optimal.

There is one class of problems for which tractability with respect to both, ϵ and d can be shown: the class of weighted spaces in which functions are additive superpositions of inherently low dimensional components, see [NW08]. We will develop suitable algorithms for such a setting, together with appropriate sparse grid methods in Chapter 4 of this thesis. Such a framework allows polynomial cost complexity and can be dealt with efficiently.

3.2.5 Application in Moderate Dimensions: a Fokker–Planck–Example

Finally, we apply our results on sparse grid approximation to an example of practical relevance: the Fokker–Planck–Equation. The example will show both, features and limitations of sparse grids, in a scenario which fulfills the required smoothness assumptions. We choose the example simple enough to get an analytic reference yet at the same time relevant enough to allow conclusions about real-life examples of similar quality, and we discuss such applications.

We choose a linear Fokker–Planck–Example with deterministic initial value as example: the Ornstein–Uhlenbeck–Process. Thus, we simulate the dynamics of a stochastic

3 Sparse Grids and Moderate Dimensional Approximation – Two Case Studies

process which is completely described by its conditional probability density function $p(x, t | x_0, t_0)$, i.e. the probability of finding $X_t \in I$, provided we started with $X_{t_0} = x_0$, is given by

$$P(X_t \in I | X_{t_0} = x_0) = \int_I p(x, t | x_0, t_0) dx. \quad (3.73)$$

In general, the (non-linear) dynamics of a stochastic process $X_t \in \mathbb{R}^d$ which is undergoing white noise excitation can be described by a stochastic differential equation of the form³

$$dX_t = A(X_t, t)dt + B(X_t, t)dW_t, \quad X_0 = x_0, \quad (3.74)$$

with deterministic coefficient functions $A(\cdot, \cdot) \in \mathbb{R}^d$ and $B(\cdot, \cdot) \in \mathbb{R}^{d \times d'}$, d' -dimensional white noise modeled by a Wiener process $W_t \in \mathbb{R}^{d'}$, $d' \leq d$ and a (possibly random) initial value $x_0 \in \mathbb{R}^d$. The associated probability density function $p(x, t) := p(x, t | x_0, t_0)$ fulfills the Fokker–Planck–Equation

$$\frac{\partial p}{\partial t}(x, t) = -\nabla_x \cdot (A(x, t)p(x, t)) + \frac{1}{2}(\nabla_x \nabla_x^T) : (D(x, t)p(x, t)), \quad x \in \mathbb{R}^d \quad (3.75)$$

$$p(x, 0) = p_0(x), \quad \int p_0(x) dx = 1, \quad p_0(x) \geq 0 \quad (3.76)$$

where $D(x, t) := B(x, t)B(x, t)^T$ and $K : J = \sum_{i,j=1}^d K_{i,j}J_{i,j}$ for two matrices $K, J \in \mathbb{R}^{d \times d}$ is the euclidean inner product applied to the vector of matrix entries. Application of the chain rule yields the div – grad formulation

$$\begin{aligned} \frac{\partial p}{\partial t} - \frac{1}{2} \operatorname{div}(D \operatorname{grad} p) + \sum_{i=1}^d \left(A_i - \frac{1}{2} \sum_{j=1}^d \frac{\partial D_{ij}}{\partial x_j} \right) \frac{\partial p}{\partial x_i} \\ + \left(\sum_{i=1}^d \frac{\partial A_i}{\partial x_i} - \frac{1}{2} \sum_{i,j=1}^d \frac{\partial^2 D_{ij}}{\partial x_i \partial x_j} \right) p = 0, \quad x \in \mathbb{R}^d \end{aligned} \quad (3.77)$$

with initial value

$$p(x, 0) = p_0(x), \quad \int p_0(x) dx = 1, \quad p_0(x) \geq 0. \quad (3.78)$$

The Fokker–Planck–Equation is always linear with respect to p , yet it is called non-linear if the associated stochastic differential equation (3.74) depends non-linearly on X_t . More precisely, (3.77) is called linear (in narrow sense) if $A(x, t) = \alpha(t) + A(t)x$ and $B(x, t) = B(t)$, $D(x, t) = B(t)B(t)^T$. It is called non-linear for other choices of A and B .

The special case of a linear Fokker–Planck–Equation is closely related to our study on normal distributions: provided the initial value $p_0(x)$ is either a Gaussian or a delta peak, (3.74) can be solved analytically and $p(x, t) = \mathcal{N}(\mu(t), \Sigma(t))$ is a (time-dependent)

³Read it like ordinary differential equations $\frac{dX_t}{dt} = A(X_t, t) + B(X_t, t) \frac{dW_t}{dt}$ with irregular, white noise force $\xi_t = \frac{dW_t}{dt}$.

3.2 Case Study: Density Approximation and the Normal Distribution

normal distribution, see [Ött96, (3.55)-(3.58)]. The quantities $\mu(t)$ and $\Sigma(t)$ can be evaluated by means of ordinary integrals [Ött96, (3.57) and (3.58)]. Linear Fokker–Planck–Equations constitute one of the few cases where analytical solutions are known (compare [Ött96]). Typically, linear stochastic equations are the simplest models which are then generalized to non–linear variants.

One application example are equations of motion of movable objects which are coupled by springs, where the movement is caused by random force fields (offshore installations subjected to environmental forces), see [WB00] and the references therein. Here, the number of objects determines the dimension d . The simplest case of linear oscillations has a delta peak as initial value (since the start position is known) and a linear Fokker–Planck–Equation governing the dynamics, see [WB00]. Another example are non–Newtonian fluids: polymeric substances in a solution can be modeled using bead–chain–models, see [Ött96, BAH87]. The number of chains N defines the model resolution and the dimension d grows linearly like $3(N - 1)$; the coupled equations of motion can be described by a Fokker–Planck–Equation. The simplest model are Hookean Spring forces, yielding a linear Fokker–Planck–Equation and thus Gaussians as solution. Note that for lower dimensional noise, $d' < d$, (3.77) becomes degenerate parabolic, see also [SST08] on operators with non–degenerate characteristic form. We believe that an approximation tool like our sparse grid approach can be tested and verified for such linear models which motivates a linear model problem here.

We choose the linear Fokker–Planck–Equation associated with the Ornstein–Uhlenbeck–Process, given by

$$A(x, t) = -\theta \cdot x - \mu \cdot (1, \dots, 1)^T \in \mathbb{R}^d, \quad (3.79)$$

$$B(x, t) = \sigma \cdot \text{diag}(1, \dots, 1) \in \mathbb{R}^{d \times d} \quad (3.80)$$

and real parameters $\mu = 0.9$, $\sigma = 0.6$ and $\theta = 1$. The reference solution for $p_0(x) = \delta(x - x_0)$ is $p(x, t) = \mathcal{N}(\mu(t), \Sigma(t))$ with

$$\mu(t) = e^{-\theta t}(x_0 - \mu \mathbf{1}) + \mu \cdot \mathbf{1}, \quad \Sigma(t) = \frac{\sigma^2}{2\theta}(1 - e^{-2\theta t}) \cdot I. \quad (3.81)$$

Instead of the complicated delta function $\delta(x - x_0)$, we choose $p_0(x) = \mathcal{N}(\mu(t_0), \Sigma(t_0))$ with $x_0 = (2, \dots, 2)^T \in \mathbb{R}^d$ and $t_0 = 0.05$. A normal distribution as approximation to a delta initial value is also used in [WB00]. In our case, we have simply advanced the initial time to $t_0 > 0$ to keep the reference solution.

The discretization of a Fokker–Planck–Equation (3.77) with parameters (3.79) requires to handle the initial value, the unbounded domain \mathbb{R}^d , the time discretization and the space discretization. We employ the standard time discretization by means of the λ scheme [QSS01]

$$Lu^{(t+1)} + \frac{1}{\delta_t}u^{(t+1)} = \left(\frac{1}{\delta_t} - (1 - \lambda)L\right)u^{(t)} \quad (3.82)$$

which corresponds to the second order accurate Crank–Nicolson method for $\lambda = 1/2$ (note that the right–hand–side vanishes, so the term $(1 - \lambda)f^{(t)} + \lambda f^{(t+1)}$ is not necessary). Since the Crank–Nicolson method does not damp oscillations, we choose $\lambda = 1/2 + 1/100$

which will be slightly less than second order accurate in time but provides an L -stable solver.

The space discretization is realized similar to our interpolation results in the preceding sections: we use a domain which is sufficiently large to capture the dynamics of $p(x, t)$ for $t \in [0.05, 1]$ and truncate $p(x, t)$ to 0 on its boundary (making points on the boundary superfluous). In our case, we choose $\Omega = [-1, 4]^d$. The initial value is interpolated on a sparse grid. The elliptic equation (3.82) arising for every time step is discretized by means of a sparse grid Galerkin scheme. The details of the Galerkin scheme are described in [Feu05] for the particular case of the prewavelet basis, the ideas especially for fast algorithms go back to [Bun98] and [Bal94] (see Section 2.2.2 for an overview). Diagonal scaling yields optimal preconditioning for the Helmholtz problem [GO95] and we observed good iteration numbers for our case of linear coefficients as well (see also the L_2 orthonormal wavelet approach with optimal preconditioning taken in [DSS09]). Some of the matrix–vector–product algorithms are listed in Appendix A.3 as well.

The actual simulation is carried out for three different space discretizations:

1. The first space discretization uses regular sparse grids of level n for every single time step, combined with a time step of $\delta_t = 2^{-n}$.
2. The second approach relies on levelwise adaptive refinement as follows: for given target precision ϵ , the initial value is resolved using the adaptive interpolation of Algorithm 1, see page 33. However, a grid refinement means to increase the regular sparse grid level by one. Then, we compute time steps with the actual regular sparse grid. At the end of each time step, we apply a levelwise grid compression: if all local error indicators for the finest level(s) result in small contributions, we reduce the regular sparse grid level accordingly. The time step size remains fixed for every experiment, since the method is second order accurate, it has been chosen as $\delta_t := \sqrt{\epsilon}$ where ϵ is the space adaptive target threshold (which corresponds to the desired target precision). During time steps, no refinement loop is performed. This approach thus constitutes levelwise compression (coarser regular sparse grids with time).
3. The third method uses fully space adaptive grids: the initial value is interpolated adaptively according to Algorithm 1 up to ϵ , then a refinement loop is applied for every single time step. The refinement loop inserts new nodes, it does not throw nodes away. At the end of each time step, we compress the grid in the following way: every node which is considered to be relevant (up to ϵ) and all of its $2 \cdot d$ direct child nodes remain in the grid. This avoids oscillation of grids between successive time steps. As for the level wise approach 2), the time step size is fixed to $\delta_t := \sqrt{\epsilon}$ where ϵ is the space adaptive target threshold.

In all three cases, δ_t is independent of t . The results of this approach are shown in Figure 3.12: blue lines indicate the regular sparse grid experiment 1), green lines the levelwise compression approach 2) and red lines the space adaptive refinement 3). We plot the relative L_2 error, measured at end time $t = 1$, against the sum of all degrees of

3.2 Case Study: Density Approximation and the Normal Distribution

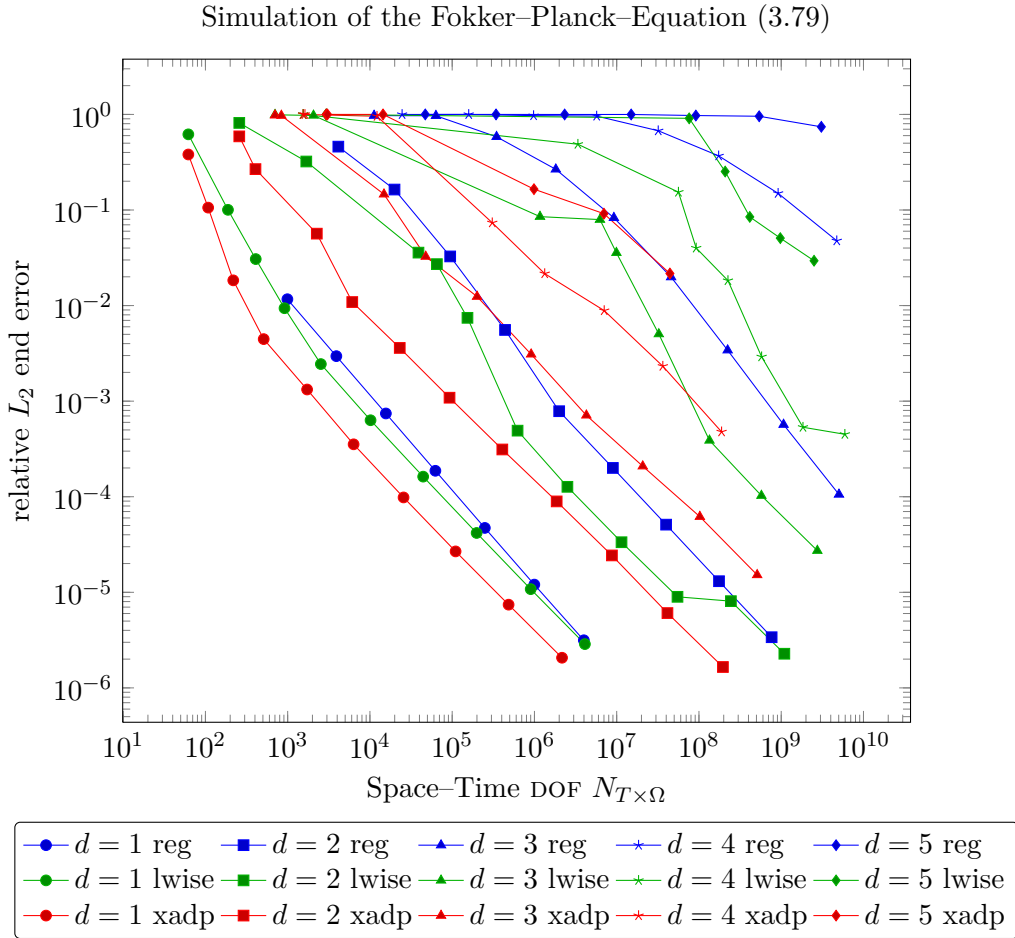











Figure 3.12: Convergence properties of regular sparse grids, levelwise adaptive sparse grids and space adaptive sparse grids for the solution of the parabolic problem (3.79).

freedom, $N_{T \times \Omega} = \sum_{i=0}^{K-1} N_{t_i}$, where K is the number of time steps ($K \approx (1 - 0.05)/\delta_t$) and N_{t_i} the number of grid points at the end of time step i . Figure 3.12 contains results for space dimensions $d = 1, 2, \dots, 5$, identified by markers. As expected, the regular sparse grid approach (blue) has the greatest offset since it wastes degrees of freedom in the tails of the distribution and since it keeps the level fixed. For $d = 1$, we find a convergence rate of 1.0, $d = 2$ has 0.9 and $d = 3$ has 1.1. Since the time discretization is of second order, this is what we expected: the space discretization has second order (and logs) and we use $\delta_t = h$; the number of time steps is thus $K = \mathcal{O}(1/\delta_t) = \mathcal{O}(2^n)$, resulting in cost $N_{T \times \Omega} = \mathcal{O}(2^n \cdot 2^n n^{d-1})$ and error $\mathcal{O}(\delta_t^2 + 2^{-2n} n^{d-1}) = \mathcal{O}(2^{-2n} n^{d-1})$, yielding a convergence rate of about 1 (up to logarithmic bounds).

The levelwise compression reduces the ϵ complexity since it uses less degrees of freedom for the same accuracy. However, its convergence rate is slightly below 1: it is around

0.9. Further experiments revealed that this is due to inaccurate balancing of δ_t and the space discretization: δ_t is too small. This is caused by differences between the finally achieved accuracy and the adaptive target threshold ϵ : for low accuracies, there is about a factor of 10 between them whereas higher accuracies thus have a relative L_2 error which is between 30 and 50 times larger than ϵ . Since $\delta_t = \sqrt{\epsilon}$, the convergence rate of the time discretization becomes sub-optimal⁴, a correction is possible by using step size control for the time discretization (which is beyond the scope of our example here). Levelwise grid compression based only on local error estimators might insert a complete level even if few points have large local errors (which can be seen for  at the end). The levelwise compression as such is easy to manage and allows efficient data structures; it is also considerably cheaper than the regular sparse grid method. It allows to reach dimension 4 or 5.

Finally, the fully space adaptive approach is most efficient in terms of degrees of freedom: it has the smallest total grid size and about the same convergence rate as the levelwise approach (the rate 0.9 instead of 1 is due to the same balancing issues between ϵ and δ_t as discussed above). Even though the adaptive approach has linear cost, it requires to solve every time step multiple times and involves more complicated data structures – leading to considerably higher runtime requirements *per node* as opposed to the other methods. Furthermore, prewavelet adaptivity requires additional transport nodes which carry no information, see the discussion Appendix A.4.1 for details. Space adaptive refinement saves one or two orders of magnitude with respect to degrees of freedom (not including transport nodes) compared to the levelwise approach for the cases $d = 4$ (compare  and ) and $d = 5$ (compare  and ) , and much more compared to the non-adaptive regular sparse grids (compare  with  and  with ).

In summary, our experiment shows that sparse grids can be used to approximate moderately dimensional problems, in our case up to five space dimensions and one time dimension. The domain truncation, combined with adaptive sparse grids, allows to handle full space diffusion problems. Together with our results of the preceding sections, dimensions around $d = 5$ (+1 for time) pose a natural limit of standard sparse grid approaches, i.e. when applied to inherently high dimensional functions like the Gaussian. The limit is caused by the task to capture the complete probability mass, which leads to exponentially growing dimension dependent complexity coefficients due to the domain size and the involved regularity norms. Qualitative improvements, i.e. approaches beyond $d = 5$, can only be expected if the approximant has more structure. Such structure is analysed in the following sections.

3.3 Case Study: Functions with Axis Parallel Structure

This case study employs a totally different type of structure than mix smoothness in order to obtain efficient high dimensional function approximations: it relies on axis

⁴The electronic version of this document provides this additional information by clicking onto data points in Figure 3.12.

parallel features of functions. Sparse grids resolve functions easily along the axis and neglect “diagonal” basis coefficients based on mix smoothness. If a function essentially consists of axis parallel contributions, we can obtain compression effects beyond the convergence rates of mix smoothness based sparse grids.

We start with a simple function depending only on one direction and increase the difficulty successively. In order to avoid diffusive effects (and thus diagonal contributions), we restrict the study to functions with sharp jumps in axis parallel directions. Clearly, the error order will be low (it will be something like $h^{1/2}$ with respect to the L_2 norm) – but the cost might be very low with respect to growing d . Thus, the result can have a favorable ϵ -complexity (cost/gain ratio). We study the potential and the limitations imposed by axis parallel structure.

3.3.1 Jump along a Hyperplane

We study the function

$$f(x_1, \dots, x_d) = \begin{cases} 0 & x_1 < 0.51 \\ 1 & x_1 \geq 0.51 \end{cases} \quad (3.83)$$

on $[0, 1]^d$ using adaptive sparse grid interpolation, see Figure 3.13 (left). Since its dynamics depends only on x_1 , we expect no dependency on d at all – although the jump is along a manifold of dimension $(d - 1)$. To employ such structure, we use sparse grids whose root is anchored at $(0, \dots, 0)$ instead of $(1/2, \dots, 1/2)$ combined with the hierarchical hat basis having the constant as coarsest level. In other words: we employ the space- and dimension-adaptive approach discussed in Section 4.3. We choose the L_2 error indicator for refinement and measure L_2 errors. The local refinement is done using a look-ahead of $q = 3$ in Algorithm 1 (see page 33) and the error is finally measured on a four times refined grid.

Results are shown in Figure 3.13 (right) as semi-logarithmic plot of grid size (degrees of freedom) versus relative L_2 error. We choose a logarithm base 2 for the error. The plot contains results for $d = 1, 2, \dots, 7$. We see that each yields the same curve as the one-dimensional case. Furthermore, we see a convergence rate of $1/2$ in the semilogarithmic plot, indicating an approximation order of exponential type,

$$\epsilon(N) = \mathcal{O}(2^{-1/2N}). \quad (3.84)$$

The rate is independent of d at all due to the anisotropic basis expansion which employs the constant as coarsest level. The result for $N = 38$, produced on a finest level of $n = 36$, is shown in Figure 3.13 (left): it contains only grid points on the line $x_2 = 0$. The markers are colored according to their logarithmic magnitude. Actually, the expansion uses only a couple of different basis coefficients which are either 0 or of the same magnitude, compare the color bar of Figure 3.13.

3.3.2 Sparse Grids, Jumps, and Overshooting

It should be noted that interpolation of jumps by means of sparse grids, even using adaptive sparse grids, leads to overshooting (or undershooting) effects unless the jump

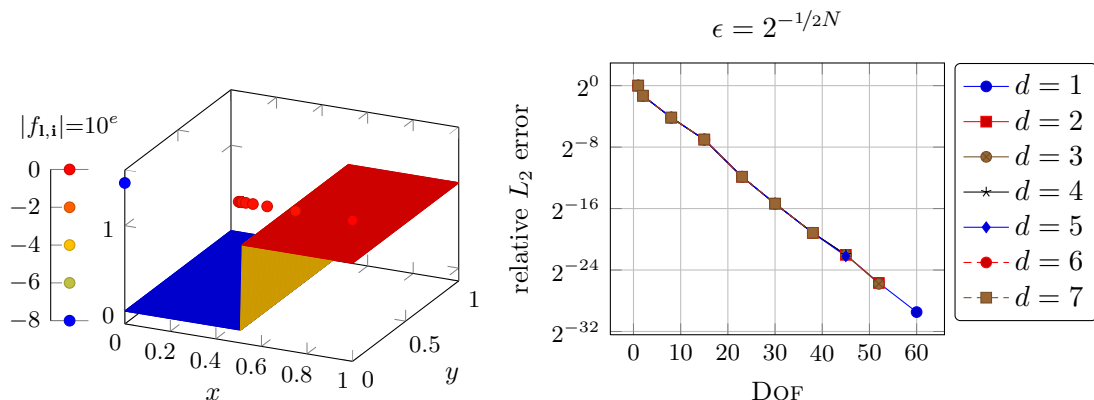


Figure 3.13: The jump function (left) and its convergence plot with adaptive sparse grids (right).

is (locally) resolved by a full grid. In other words: the interpolant near the jump is much larger or much smaller than the start- or end points of the jump. This section explains briefly why sparse grid structures imply overshoots, even if there are just local sparse grid structures and even if one uses piecewise constant basis functions: there will always be overshooting effects, on every scale. The only way to avoid overshooting is local full grid structure combined with either piecewise constant or piecewise linear splines.

These overshooting effects typically fall between visualization mesh width and are thus not easily seen (only in the error estimations). They occur also in the experiments below, namely for the hyper cube, the diagonal jump and the arc. Since overshoots are of the width of one mesh width and we measure L_2 errors of fine adaptive resolutions, we achieve convergence. The reason why (local) sparse grid structure yields overshoots is the hierarchical approach and the lack of smoothness on every scale, combined with the fact that adaptive refinement always reproduces local sparse grid structure (it inserts $2 \cdot d$ sons, not 3^d neighbors).

We consider a small example, which illustrates the key behavior on a coarse scale. We use a two-dimensional full grid of level 1 (3×3 points) and assign just one non-vanishing nodal value $n_{\mathbf{1},\mathbf{i}} = \delta_{(0,0|0,0),(1,\mathbf{i})}$. A sparse grid contains fewer grid points in direction of the diagonal. Consequently, we have to remove the middle point to simulate sparse grid structure. The resulting interpolant and its hierarchical basis coefficients are shown in Figure 3.14 (left) for a full grid and in Figure 3.14 (right) for the sparse grid: the hierarchical coefficient at $x = (1/2, 1/2)$ turns out to be important; it is $u_{(1,1|1,1)} = \frac{1}{4}$ and the error is thus 25%. Such a situation is the common case for jumps through sparse grids: patterns like these occur on all mesh widths somewhere along the jump (the adaptive refinement always reproduces it). The point is that anisotropic basis functions with large support (in our example the ones with coefficient $-1/2$) are used to interpolate the missing value by means of differences and sums. But since there is no decay of basis coefficients, the interpolated value has a large error – which is reproduced on all scales in a self-similar way by the adaptivity.

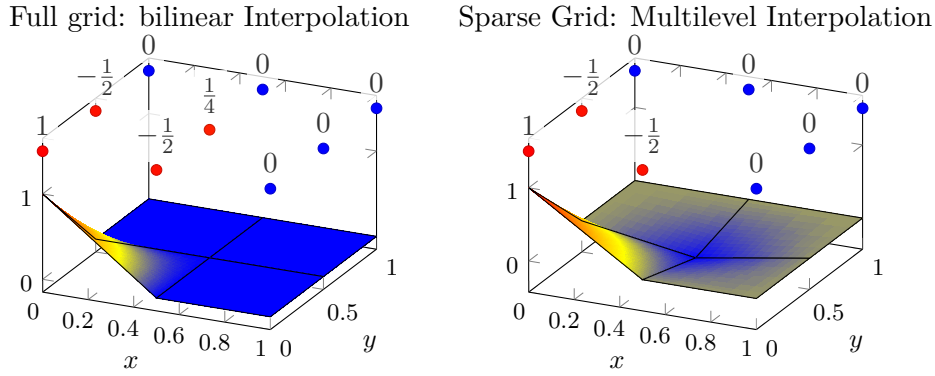


Figure 3.14: Illustration of multilevel interpolation schemes causing overshooting in sparse grids (right) but not in full grids (left).

Note that piecewise constant bases (which are typically used due to their stability in full grid schemes) are even worse in sparse grids: the same test repeated for piecewise constant bases lead to errors of 100%. Experiments with jumps along the diagonal (see below) for piecewise constant bases easily yields overshoots of 200% or more since the complete jump might be added multiple times in the multilevel differencing scheme. Adaptive sparse grid refinement only moves the effect to a vanishingly small mesh width.

3.3.3 Jump along a Hypercube

Our next experiment is to use jumps on the boundary of a cube, $f(x) = \chi_{[0.21,0.81]^d}(x)$, using the same method as for Section 3.3.1.

Figure 3.15 (top left) shows the result obtained with $N = 1993$ grid points (placed only inside of the unit cube, not on the boundary) with finest mesh width 2^{-34} (level $n = 34$). The grid is a product of the one-dimensional result of Figure 3.13, and it has comparable approximation properties. The approximation properties are no longer exponential, but they are still better than algebraic. This is depicted in Figure 3.15 (top right) where the degrees of freedom are plotted against the relative L_2 error in a double logarithmic scale. It is better than algebraic. Figure 3.15 (bottom left) shows the error plotted against the finest level n in a semi logarithmic plot, which indicates an error of $\epsilon(n) = 2^{-1/2n}$. Figure 3.15 (bottom right) shows the finest level n plotted versus the degrees of freedom which reveals $N(n, d) = n^{k(d)}$ with a d -dependent exponent $k(d) \geq 1$. Estimating slopes of Figure 3.15 (bottom right) yields $k(1) \approx 1$, $k(2) \approx 2.4$, $k(3) \approx 3.6$ and $k(4) \approx 5$. It looks like linear growth of $k(d)$, perhaps $k(d) \approx d + 1$. Thus, we find poly-logarithmic cost $N(n, d) = \mathcal{O}(n^{d+1})$ and accuracy $\epsilon(n, d) = 2^{-1/2n}$ and so

$$\epsilon(N) = \mathcal{O}(2^{-1/2N^{1/(d+1)}}). \quad (3.85)$$

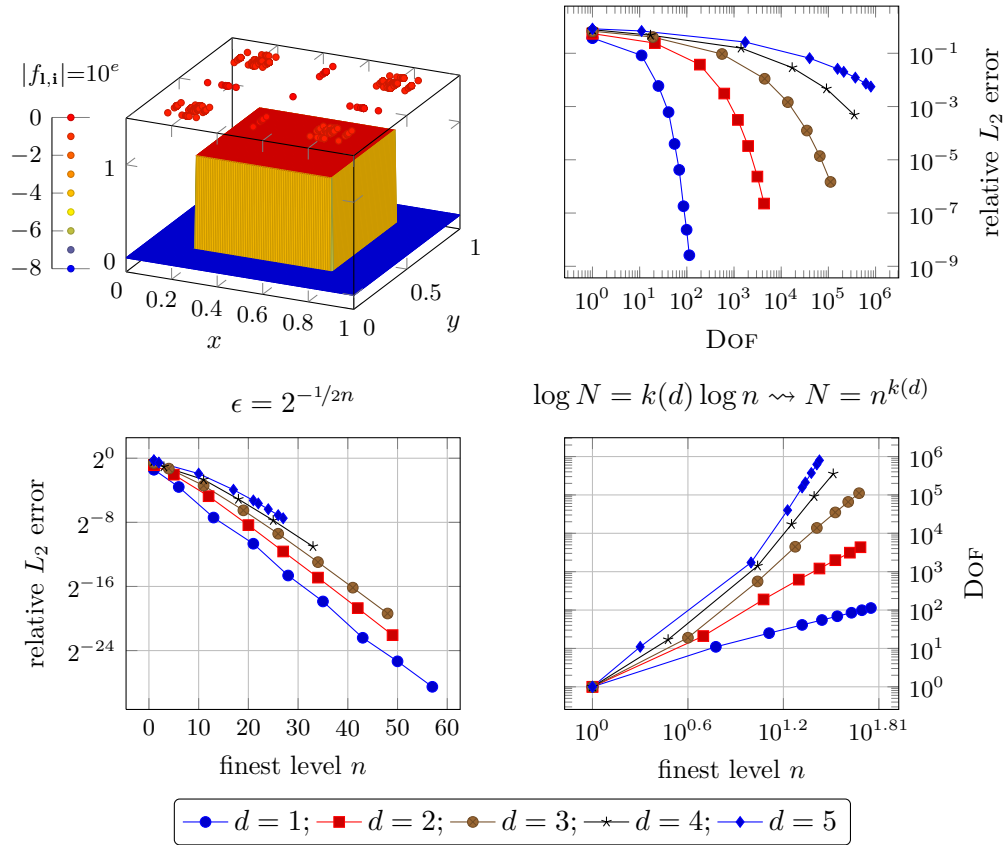


Figure 3.15: The box function and its adaptive grid (top left), its error plotted versus degrees of freedom (top right), its error plotted against the finest level n (bottom left) and its cost plotted against level n (bottom right).

3.3.4 Diagonal Structure and Sparse Grids

Having seen that axis parallel structure allows to improve the convergence rate qualitatively, we study the limitations arising by diagonal structure. We consider a jump along the diagonal unit vector $v = 1/\sqrt{d}(1, \dots, 1)^T \in \mathbb{R}^d$ given by

$$f(x) = f_n(x^T v), \quad f_n: \mathbb{R} \rightarrow \mathbb{R}, \quad f_n(d) = \begin{cases} 0 & x < 0.6 \\ 1 & x \geq 0.6. \end{cases} \quad (3.86)$$

We compute the adaptive sparse grid interpolant as before. The resulting function for $\epsilon = 3.2 \cdot 10^{-3}$ and its adaptive grid is shown in Figure 3.16 (left), together with the relative L_2 error plotted versus degrees of freedom in a double logarithmic scale in Figure 3.16 (right). The interpolant exhibits overshooting effects along the jump as discussed in Section 3.3.2; it is strongly refined along the diagonal and has only the ancestors in axis parallel directions. The convergence plot shows algebraic rates which are about 0.4 for the two-dimensional case, 0.2 for the case $d = 3$ and about 0.1 for the case $d = 4$.

3.3 Case Study: Functions with Axis Parallel Structure

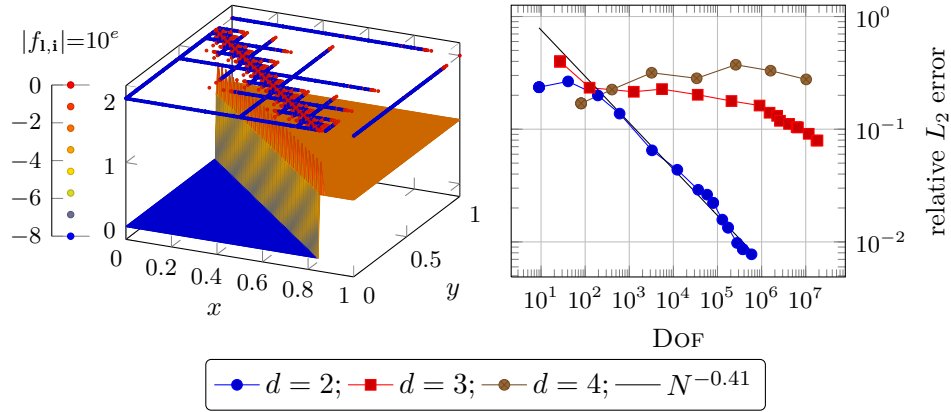


Figure 3.16: Jump along the diagonal (left) and its convergence plot with adaptive sparse grids (right)

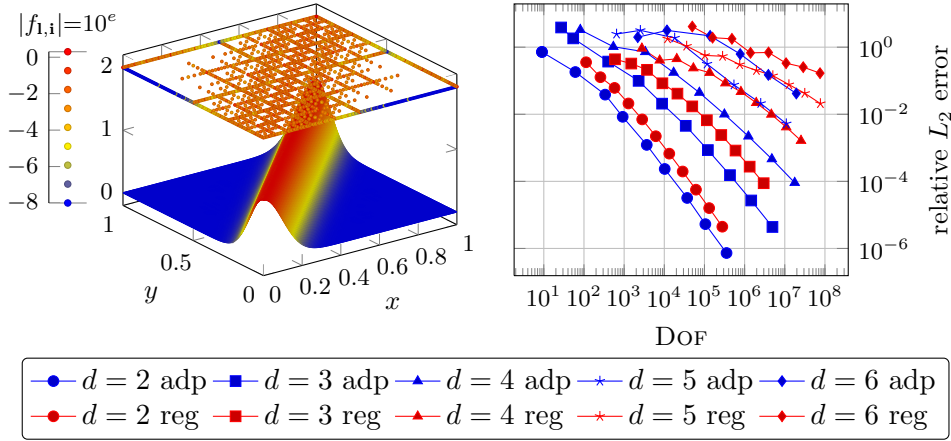


Figure 3.17: Smooth diagonal structure allows (just) sparse grid rates.

Jumps across the diagonal require locally full grid structure, and the adaptive refinement generates it – although it also inserts fine local sparse grid structure (which produces the overshooting effect).

It should be stressed that diagonal structure is not necessarily bad – the sparse grid compression based on mix smoothness works as expected. This is demonstrated in Figure 3.17 for the smooth problem

$$f(x) = \exp\left(-\frac{\|x^T v v - x\|^2}{s}\right), \quad s = \frac{1}{100}, \quad v = \frac{1}{\sqrt{d}}(1, \dots, 1)^T. \quad (3.87)$$

Figure 3.17 (left) shows the approximated function and Figure 3.17 (right) the resulting convergence results obtained for adaptive sparse grids (blue) and for regular sparse grids (red). Here, common markers indicate the same dimension. We see that the adaptive convergence rates are the same as the ones for regular sparse grids as expected for a

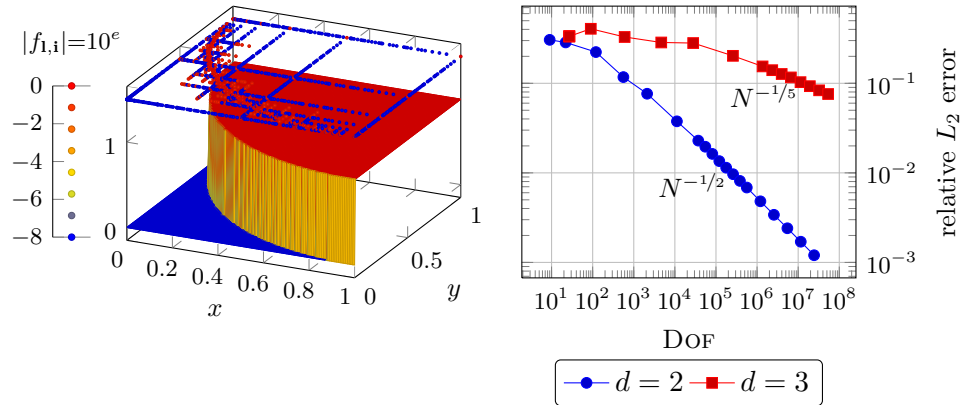


Figure 3.18: Adaptive interpolation of an arc and its convergence plot.

smooth problem. While the convergence rates are as expected, the concentration of information allows to reduce the overall degrees of freedom if one uses adaptivity.

3.3.5 Jump along an Arc

Finally, we consider the adaptive resolution of an arc. An arc has parts which are parallel to the axis, but it also has parts crossing the diagonals. The diagonals have less special structure, they need to be approximated using the methods described in Section 2.1.3. The question is thus, whether an approximation of an arc can benefit from locally axis-parallel parts or if the diagonal parts dominate.

We consider an arc in two dimensions and a part of a sphere for $d = 3$. More precisely, we use the characteristic function of a ball,

$$f(x) := \chi_{\{x \mid \|x-c\| \leq 1\}}(x), \quad (3.88)$$

where $c := (1, \dots, 1)^T$ and $\|\cdot\| = \|\cdot\|_2$ is the euclidean norm.

Approximation results are shown in Figure 3.18: on the left, we see the approximated function on $[0, 1]^2$, together with its adaptive grid and Figure 3.18 (right) shows the convergence properties for $d = 2$ and $d = 3$. The grid is highly adapted near the edge, and coarse everywhere else. We only see the hanging nodes, i.e. ancestors of fine levels. Furthermore, we see how axis-parallel structure reduces the amount of grid points near $x_1 > 0.8$ and near $x_2 > 0.8$. Note that again, the basis coefficients are either of order $\mathcal{O}(1)$ or they vanish. In fact, there are just five different values, $|f_{1,i}| \in \{1, 3/4, 1/2, 1/4, 0\}$ for the two-dimensional case. The convergence plot is a line, indicating an algebraic convergence rate. The slopes for $d = 2$ approaches $1/2$ whereas the slope for $d = 3$ approaches $1/5$. Note that the same experiment repeated for $d = 4$ yields almost no measurable error reduction for a similar amount of grid points. We conclude that diagonal parts dominate the complexity.

Further experiments with rapidly decaying refinement weights or other unit balls (for example $\|\cdot\|_6$) yielded the same quality and the same rates.

The result of our second case study can be summarized as follows: if the approximant varies at least locally along the *diagonal*, only the smoothness based representation results (of Section 2.1.3) hold. However, if the approximant exhibits only axis-parallel dynamics, few anisotropic basis functions cover a lot of the function and adaptive refinement can increase the effectiveness significantly.

3.4 Summary: Benefits and Limitations of Sparse Grids for Moderate Dimensional Problems

Our first case study on approximation of density functions defined on \mathbb{R}^d has two results: the first is the formulation and quantitative analysis of inherent limitations of sparse grids and the second is their superiority over full grid methods which allows sparse grids to reach space dimensions $d = 5$ or $d = 6$. The limitation arises in form of exponentially growing order coefficients, analysed representatively for the Gaussian as unimodal, smooth density function. It occurs for every mean (position) and covariance (width) of the Gaussian due to the necessity to increase the simulation domain together with the covariance. It is closely related to relative errors (in our case L_2 - and energy errors, but it also arises for L_∞ approximation), but may be reduced by absolute errors or rescaled norms. Further experiments indicated that the results can be generalized to piecewise constant bases and hierarchical polynomial sparse grid bases of higher order [Ach03, Bun98] which have higher convergence rates and higher order coefficients [Bun98] (see also Appendix A.2 for these bases). We conclude that sparse grids, applied to density approximation problems (for example the Fokker-Planck-Equation) are inherently limited to moderate dimensions of, say, up to $d = 6$. Their advantage is to go beyond limitations of alternative full grid based spline methods.

Our second case study is a first step towards different function spaces: axis parallel structure can, to some extent, reduce cost for d -dimensional problems. As soon as diagonal parts contribute to the function, this advantage is lost. The results motivate a systematic approach on axis parallel structure and constitutes a transition to dimension adaptive approximation, an approach which is subject of the following chapter.

4 Low Effective Dimensionality – A Dimension Adaptive Method

We have seen in Chapter 3 that approximation of functions is not a simple task if the dimension d grows: linear approximation tools suffer from exponential growth either in terms of a coefficient raised to the d as in the sparse grid case or even in terms of one-dimensional costs raised to the d for full grid methods. This leads to the question whether there are function spaces for which qualitative improvements can be expected.

In this chapter, we study function spaces which have more structure: they combine smoothness assumptions with inherently low dimensional structure. Here, low dimensional structure is formalized in an axis parallel manner to refine our results of Section 3.3 on axis parallel functions: we study linear decompositions of a d -dimensional function into its contributions from different groups of directions. The motivation behind such an approach is that the relevant parts of the function can be captured by few varying directions although such a structure might not be obvious.

The result is closely related to the already mentioned *weighted spaces* known from integration and Information Based Complexity – if sets of directions are of decaying importance according to specific real weights, the cost for an approximation up to prescribed error does *not* grow exponentially in the nominal dimension d . Spaces weighted with respect to the L_2 inner product have been studied in [NW08, Example 6], together with conditions on the weights to ensure non-exponential cost complexities. A similar approach based on Sobolev regularity, formulated by means of reproducing kernel Hilbert spaces, has been studied in [Gri06, section 1.3.3], see also the references cited therein. A further approach for the case of weights for mixed first derivatives has been elaborated in [Hol08, section 4.2] for the particular application of integration: for given weights, optimal sparse grid quadrature algorithms are derived and analyzed.

We study spaces with weighted second mixed derivatives. If the weights are known in advance, we apply a similar procedure as [Hol08] in order to derive new optimal sparse grid spaces based on a priori estimates in Section 4.2. If functions are known to live in a weighted space, but the weights are not known, Section 4.3 develops new a posteriori, dimension adaptive refinement routines for the approximation problem. The method may also be applied if sets of directions are weighted and the function is only partially smooth. In such a case, the method of Section 4.3 can rely on space *and* dimension adaptive refinement.

Before we come to weighted spaces, we discuss the underlying dimension decomposition framework of ANOVA type to formalize what we mean by different sets of directions. We also elaborate the relation between the ANOVA decomposition and our multi-level basis expansions to derive fast *discretized ANOVA decompositions* in Section 4.1.3.

4.1 From ANOVA Decompositions to Sparse Grids

We consider a decomposition of a high dimensional function f into a superposition of lower dimensional components of the form

$$\begin{aligned} f(x_1, \dots, x_d) = & f_0 + \sum_{j_1} f_{j_1}(x_{j_1}) + \sum_{j_1 < j_2} f_{j_1, j_2}(x_{j_1}, x_{j_2}) \\ & + \sum_{j_1 < j_2 < j_3} f_{j_1, j_2, j_3}(x_{j_1}, x_{j_2}, x_{j_3}) + \dots + f_{1, \dots, d}(x_1, \dots, x_d). \end{aligned} \quad (4.1)$$

The first component f_0 is a constant, followed by one–dimensional contributions f_{j_i} , two–dimensional ones and so on until finally a fully d –dimensional component $f_{1, \dots, d}$ is added. Decompositions of this type go back to [Hoe48] and are well known in statistics under the name analysis of variance (ANOVA), see [ES81].

Dimension decompositions are successful for high dimensional integration applications. The success of Quasi-Monte-Carlo integration methods applied to problems of computational finance and the low effective dimension of such integrands is discussed in [SW98]. Generally, integration for functions with either decaying or vanishing high order terms in (4.1) is possible in polynomial time and thus tractable, see also [NW08, Example 6]. Furthermore, sparse grid quadrature methods have been proposed in [GG03] and successfully applied to problems of computational finance [Hol08].

We will see that substantial improvements of the *approximation* problem are also possible for functions of decaying (or vanishing) high dimensional ANOVA components. Furthermore, we develop efficient approximation methods to identify and employ such a structure and discuss necessary modifications compared to quadrature methods [GG03].

4.1.1 ANOVA–like Decompositions

The decomposition (4.1) is a finite expansion into $\sum_{i=0}^d \binom{d}{i} = 2^d$ summands which we study in detail, following [Gri06].

The key idea is to work in an axis parallel manner: we select particular subsets of directions in a superposition model. We denote a d –dimensional tensor product function space by $V^{(d)}$ and work with functions $f: [0, 1]^d \rightarrow \mathbb{R}$. Let μ be a product measure with unit mass,

$$d\mu(x) = \prod_{j=1}^d d\mu_j(x), \quad \int_0^1 d\mu_j(x) = 1. \quad (4.2)$$

We start with the one–dimensional case where the decomposition consists only of constants and one–dimensional contributions,

$$V^{(1)} = \mathbf{1} \oplus W^{(1)}. \quad (4.3)$$

Here, $\mathbf{1} = \text{span}\{1\}$ denotes the one–dimensional space of constant functions and $W^{(1)}$ denotes the complement space of $\mathbf{1}$ in $V^{(1)}$. We define the splitting by the projection

4.1 From ANOVA Decompositions to Sparse Grids

$P: V^{(1)} \rightarrow \mathbf{1}$,

$$Pf(x) = \int_0^1 f(x) d\mu(x) \quad (4.4)$$

and its complementary projector $(I - P): V^{(1)} \rightarrow W^{(1)}$. A first example for such a projector is the conventional Lebesgue measure $d\mu(x) = dx$ which leads to the integral average $f_0 = Pf(x) = \int_0^1 f(x) dx$ and $f_1(x) = (I - P)f(x) = f(x) - \int_0^1 f(x) dx$. In this case, $W^{(1)} = (I - P)(V^{(1)})$ is the orthogonal complement of $\mathbf{1}$ in $V^{(1)}$ with respect to the inner product $(f, g) = \int_0^1 f(x)g(x) dx$ since per construction $(f_1, \mathbf{1}) = 0$.

Another example is the Dirac measure located at a point $a \in [0, 1]$, i.e. $d\mu(x) = \delta(x - a) dx$, which results in a simple evaluation of f at a ,

$$f_0 = Pf(x) = \int_0^1 f(x)\delta(x - a) dx = f(a), \quad (4.5)$$

and the complement

$$f_1 = (I - P)f(x) = f(x) - f(a). \quad (4.6)$$

The approach is the same as we already used it for non-homogeneous boundary conditions of classical sparse grids in Section 2.1.4: there, we employed the anchor $a = 0$ and the Dirac measure to get a boundary splitting.

Note that due to the unit mass of $d\mu(x)$, both P and $(I - P)$ are indeed projections: it holds $P(Pf(x)) = \int \int f(x) d\mu(x) d\mu(x) = Pf(x) \cdot 1$ and $(I - P)^2 f = (I - P)f$ as well. In fact, one can define splittings just by means of projectors, it is not necessary to rely on integration as in (4.2), compare [KSWW09].

Now, we consider the d -dimensional case: The one-dimensional splitting introduces a natural decomposition of the d -dimensional function space $V^{(d)}$ by a tensor product construction

$$V^{(d)} = \bigotimes_{i=1}^d (\mathbf{1}_i \oplus W_i) \quad (4.7)$$

$$= \mathbf{1}_1 \otimes \cdots \otimes \mathbf{1}_d \quad (4.8)$$

$$\oplus \bigoplus_{j_1=1}^d \mathbf{1}_1 \otimes \cdots \otimes W_{j_1} \otimes \cdots \otimes \mathbf{1}_d \quad (4.9)$$

$$\oplus \bigoplus_{j_1 < j_2} \mathbf{1}_1 \otimes \cdots \otimes W_{j_1} \otimes \cdots \otimes W_{j_2} \otimes \cdots \otimes \mathbf{1}_d \quad (4.10)$$

$$\oplus \cdots \quad (4.11)$$

$$\oplus W_1 \otimes \cdots \otimes W_d. \quad (4.12)$$

Here, we use the subscript j in $\mathbf{1}_j$ and W_j merely to indicate the respective coordinate direction for explanatory reasons. This yields a decomposition of $f \in V^{(d)}$ of the desired type,

$$f(x_1, \dots, x_d) = f_0 + \sum_{j_1} f_{j_1}(x_{j_1}) + \cdots = \sum_{u \subseteq \{1, \dots, d\}} f_u(x_u), \quad (4.13)$$

where x_u denotes the variables x_i for $i \in u$. If we denote the subspace with all factors W_j , $j \in u$ and $\mathbf{1}_k$, $k \in \{1, \dots, d\} \setminus u$ by W_u , we find

$$V^{(d)} = \bigoplus_{u \subseteq \{1, \dots, d\}} W_u \quad (4.14)$$

and $f_u \in W_u$. Due to the power set construction, we have $\sum_{i=0}^d \binom{d}{i} = 2^d$ many terms in the expansion. The decomposition is unique for a fixed choice of the one-dimensional projector $P: V^{(1)} \rightarrow \mathbf{1}$. Associated is the identity

$$I^{(d)} = \bigotimes_{j=1}^d (P_j + (I_j - P_j)) \quad (4.15)$$

$$= \sum_{u \subseteq \{1, \dots, d\}} \left(\prod_{k \in \{1, \dots, d\} \setminus u} P_k \right) \cdot \left(\prod_{j \in u} (I_j - P_j) \right) \quad (4.16)$$

$$=: \sum_{u \subseteq \{1, \dots, d\}} P_u \quad (4.17)$$

where P_j and $(I_j - P_j)$ are the one-directional projection operators for the j th coordinate direction, compare (4.4). Note that the literature uses different notations for P_u : in our notation, we have $f_u = P_u f$, following [Gri06]. There is an alternative notation which uses $\prod_{j \in u} P_j$ instead, see for example [Hol08] or [KSWW09].

The lowest order term $P_\emptyset = \prod_{j=1}^d P_j$ yields the integral of f (measured with $d\mu(x)$) whereas P_u for $u \neq \emptyset$ eliminates only directions $j \in \{1, \dots, d\} \setminus u$ and varies in directions $j \in u$. The single terms f_u of the decomposition can be computed by successive application of projections and subtraction of low order terms, more precisely, by the recursion

$$f_u(x_u) = \left(\prod_{j \in \{1, \dots, d\} \setminus u} P_j f \right)(x_u) - \sum_{v \subset u} f_v(x_v), \quad (4.18)$$

compare [Gri06]. They are unique and it holds $P_j f_u = 0$ for $j \in u$, compare [KSWW09]. A non-recursive formula for f_u has been presented in [KSWW09]:

$$f_u(x_u) = \sum_{v \subseteq u} (-1)^{|u|-|v|} \left(\prod_{j \in \{1, \dots, d\} \setminus v} P_j f \right)(x_v). \quad (4.19)$$

The decomposition for the Lebesgue measure $d\mu(x) = dx$ resembles the well known ANOVA decomposition used in statistics, see [ES81, Wah90], and the references therein. The simpler Dirac measure $d\mu(x) = \delta(x - a) dx$ anchored at a point $a \in [0, 1]^d$ is called anchor ANOVA decomposition [SWW04], [DSWW04] or cut HDMR [RA99]. For an overview of more generalization, we refer to [Gri06].

4.1.2 Axis Parallel Structure and Low Effective Dimension

ANOVA type decompositions are possible for every function for which the required integrals are defined – it is actually just an intelligent differencing scheme to subtract

lower order components. However, the decomposition is only useful if components of higher dimensionality can be neglected or treated with reduced cost complexity. For functions for which (4.1) decays rapidly with $|u|$, considerable savings can be expected since only superpositions of low dimensional functions govern the essential features. If a function has finite order weights, i.e. $f_u \equiv 0$ for $|u| > d_s$, the case becomes even simpler.

We use this superposition approach¹ to define the *effective dimensionality*: a function is said to be *effectively low dimensional* (of dimension d_s),

- a) if it is of finite order d_s , i.e. $f_u \equiv 0$ for $|u| > d_s$, or
- b) when we find $\|f - \sum_{|u| \leq d_s} f_u\| \leq \epsilon$ for a particular choice of ϵ and $\|\cdot\|$.

One common choice for the Lebesgue ANOVA is to use the L_2 norm. Then, the orthogonality $(f_u, f_v) = 0$ for $u \neq v$ yields

$$\|f - \sum_{|u| \leq d_s} f_u\|^2 = \|f\|^2 - 2(f, \sum_{|u| \leq d_s} f_u) + \|\sum_{|u| \leq d_s} f_u\|^2 \quad (4.20)$$

$$= \sum_{u \subseteq \{1, \dots, d\}} \|f_u\|^2 - 2 \sum_{|u| \leq d_s} \|f_u\|^2 + \sum_{|u| \leq d_s} \|f_u\|^2 \quad (4.21)$$

$$= \sum_{|u| > d_s} \|f_u\|^2 \leq \epsilon^2. \quad (4.22)$$

In statistics, $\|g\|^2$ is the *variance* of g which motivates the name ‘‘Analysis of Variance’’ (ANOVA). A common choice is $\epsilon^2 := 0.01\|f\|^2$, i.e. one percent of the total variance, compare [Hol08].

Since our dimension decompositions are defined on subsets of axes (directions), the notion of effective dimension is inherently axis parallel. The hope is to find problem formulations in which either finite order weights with $d_s \ll d$ or at least some sort of decay with $|u|$ can be achieved and exploited.

4.1.3 Sparse Grids as Discretized ANOVA Decomposition

Computing terms of an ANOVA decomposition by means of either the recursive formula (4.18) or the iterative variant (4.19) is a time consuming task involving a lot of partial integrations. It is particularly expensive if more than just one evaluation point x is desired – the complete operation has to be done for each point x *separately*.

In this thesis, we propose to apply an approximation based approach which relies on particular basis representations. Choosing the correct basis means choosing the correct projector – and thus an ANOVA decomposition. It also works the other way round: given a particular ANOVA decomposition, defined by its projectors, we can create a suitable basis which discretizes the decomposition in a natural way. Finally, a given basis expansion automatically yields discretized ANOVA components. The existence of a relation between sparse grids and dimension decompositions is known and motivated

¹See also [Hol08] for and the references therein for truncation based approaches to define effective dimensionality.

the development of dimension adaptive tools: [Heg03] motivated a sparse grid approach as dimension decomposition which works “similar” to ANOVA decompositions. He proposed an adaptive work flow which has been applied to high dimensional integration [GG03]. Later, [Gar04] applied the mechanism to machine learning, based on the hierarchical hat basis with constant on level -1 which we will find to be a discretization of the anchor ANOVA. The contribution in this chapter is to formulate the ANOVA decomposition by means of basis properties which are determined by projectors. Thus, we find a constructive approach to compute, analyze and visualize ANOVA components in an effective way, which improves the common evaluation methods (4.18) and (4.19). In comparison to approaches in [GG03, Gar04], the new method presented here allows space- and dimension adaptivity. It has a direct relation to ANOVA decompositions.

To this end, we consider a finite dimensional subspace $V_h^{(d)} \subset V^{(d)}$ of a tensor product space $V^{(d)}$ for which we study basis expansions. We start with a one-dimensional splitting of $V_h^{(1)}$ into constant and rest as before,

$$V_h^{(1)} = \mathbf{1} \oplus W_h^{(1)}, \quad (4.23)$$

where $W_h^{(1)}$ is spanned by a properly chosen basis,

$$V_h^{(1)} = \text{span}\{1\} \oplus \text{span}\{\phi_1, \dots, \phi_n\}. \quad (4.24)$$

The key idea is to choose the basis ϕ_1, \dots, ϕ_n such that $P(\phi_i) = 0$. Thus, any one-dimensional function $f^h \in V_h^{(1)}$ represented as linear combination of $\{1, \phi_1, \dots, \phi_n\}$,

$$f^h = a_0 + \sum_{i=1}^n a_i \phi_i, \quad (4.25)$$

fulfills $Pf^h = a_0 + \sum a_i P\phi_i = a_0 \in \text{span}\{1\}$ and

$$(I - P)f^h = f^h - a_0 \in W_h^{(1)}. \quad (4.26)$$

For reasons of convenience, we define $\phi_0 := 1$ and write $V_h^{(1)} = \text{span}\{\phi_0, \dots, \phi_n\}$.

The d -dimensional case follows using the same tensor product approach as in (4.12),

$$V_h^{(d)} = \bigotimes_{j=1}^d (\mathbf{1}_j \oplus W_{h,j}) = \bigotimes_{j=1}^d (\text{span}\{1\} \oplus \text{span}\{\phi_1, \dots, \phi_n\}). \quad (4.27)$$

This leads to the d -dimensional tensor product basis for $V_h^{(d)}$,

$$\phi_{i_1, \dots, i_d}(x) = \prod_{j=1}^d \phi_{i_j}(x_j), \quad (4.28)$$

which fulfills $\phi_{0, \dots, 0} = 1$. The same reasoning as before now leads to

$$f_u^h = P_u f^h \in W_u^h \quad (4.29)$$

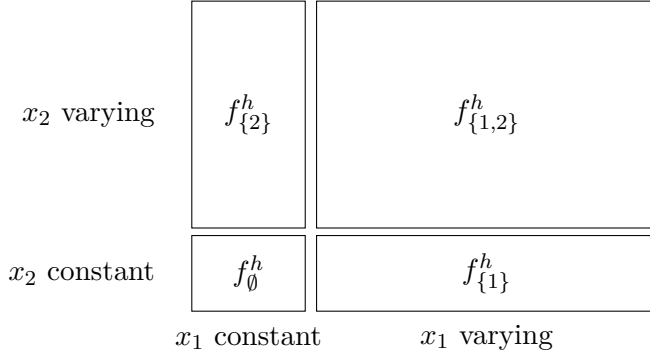


Figure 4.1: Illustration of a two-dimensional ANOVA decomposition.

where

$$P_u = \left(\prod_{j \in \{1, \dots, d\} \setminus u} P_j \right) \cdot \left(\prod_{j \in u} (I_j - P_j) \right) \quad (4.30)$$

and W_u^h is the associated tensor product consisting of constants in directions $j \in \{1, \dots, d\} \setminus u$ and varying terms in directions $j \in u$. Due to the one-dimensional compatibility between basis and projector, we get d -dimensional approximations (more precisely, $|u|$ -dimensional ones) $f_u^h \approx f_u$ which converge for $n \rightarrow \infty$.

Before we actually choose basis sets matching the classical ANOVA decomposition or the anchor ANOVA, we note how to obtain f_u^h from a given representation f^h . We assume we have a basis expansion

$$f^h = \sum_{i_1, \dots, i_d \geq 0} a_{i_1, \dots, i_d} \phi_{i_1, \dots, i_d}. \quad (4.31)$$

We define the ANOVA component for a multi index (i_1, \dots, i_d) to be

$$u(i_1, \dots, i_d) := \{j \mid i_j \neq 0\}, \quad (4.32)$$

i.e. $u(i_1, \dots, i_d)$ contains only those directions for which the one-dimensional basis function ϕ_{i_j} is not the constant. Since we have a basis, it follows that

$$f_u^h = \sum_{\substack{i_1, \dots, i_d \geq 0 \\ i_j \neq 0 \text{ for } j \in u}} a_{i_1, \dots, i_d} \prod_{k \in u} \phi_{i_k}. \quad (4.33)$$

As consequence of (4.33), we only need to compute a basis expansion (for example by interpolation) and get the decomposition and all its components f_u^h for free. We illustrate such a splitting using boxes as in Figure 4.1: the 2^d components are visualized using the constant and rest power set.

We will now present basis sets which are suitable to compute the classical ANOVA decomposition and the anchor ANOVA decomposition with anchor $a = (0, \dots, 0)$.

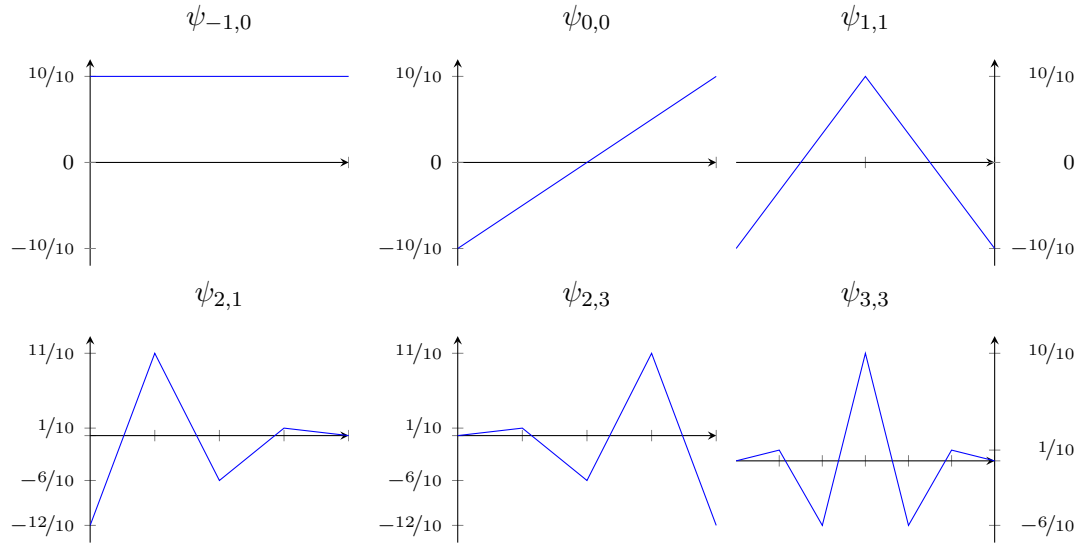


Figure 4.2: Prewavelet basis functions for selected multi-indices in the set $(l, i) \in \{(-1, 0), (0, 0), (1, 1), (2, 1), (2, 3), (3, 3)\}$ (from top left to bottom right).

The Prewavelet Basis as Natural Discretization of The ANOVA Decomposition

The classical ANOVA decomposition is characterized by the Lebesgue measure $d\mu(x) = dx$ and thus the one-dimensional projector $Pf = \int_0^1 f(x) dx$. Consequently, we need a basis set $\{\phi_1, \dots, \phi_n\}$ for which $P\phi_i = 0$ for $1 \leq i \leq n$, i.e. with vanishing zeroth moment $\int_0^1 \phi_i dx = 0$. The complete basis for $V_h^{(1)}$ is then given by $1, \phi_1, \phi_2, \dots$ which constitutes a hierarchical decomposition with “coarsest scale” $\phi_0 = 1$. Vanishing moments are common for wavelet bases: any wavelet basis expansion allows discretized Lebesgue ANOVA representations. The most simple wavelet is the piecewise constant Haar wavelet [Haa10] (see Appendix A.2.4 for an implementation) which is first order accurate. Here, we employ the second order piecewise linear Neumann prewavelet basis $\{\psi_{l,i}\}$ presented in [GO95]. As a hierarchical basis, it is defined on different levels l , so we switch the indexing to level and space indices of the form $j \equiv (l, i)$. The constant gets the level index $l = -1$ and space index 0, i.e. $\psi_{-1,0} := 1$. Any level index $l \geq 0$ makes up the non-constant part $W_h^{(1)}$. The Neumann prewavelet basis is illustrated in Figure 4.2: the first basis function is the constant, the function on level $l = 0$ is linear (it gets space index 1 such that $x_{0,1} = 1$ is the right boundary), level $l = 1$ is a hat function and starting with $l = 2$, we get spline functions made up using multiple adjacent hat functions. Besides the required orthogonality property $\int_0^1 \psi_{l,i} \psi_{-1,0} dx = \int_0^1 \psi_{l,i} dx = 0$ for $(l, i) \neq (-1, 0)$, the prewavelet basis also fulfills $\int_0^1 \psi_{l,i} \psi_{k,j} dx = 0$ for $l \neq k$. As a hierarchical basis, it allows a sparse grid construction for the d -dimensional case which reduces the required degrees of freedom considerably compared to the full tensor product $i_1, \dots, i_d = 0, \dots, n$ provided f has bounded second mixed derivatives. Furthermore, there exist fast $\mathcal{O}(N)$ algorithms to transform nodal values $f(x_{l,i})$ to a prewavelet rep-

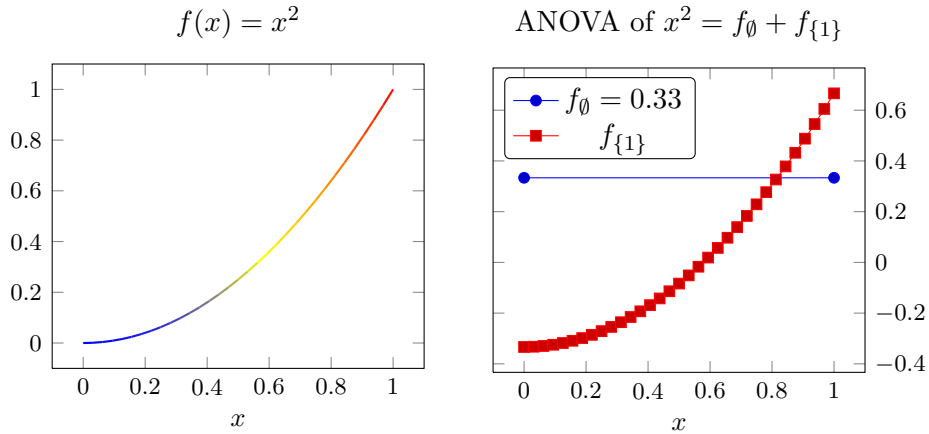


Figure 4.3: One-dimensional example of the discretized ANOVA decomposition by means of the prewavelet basis.

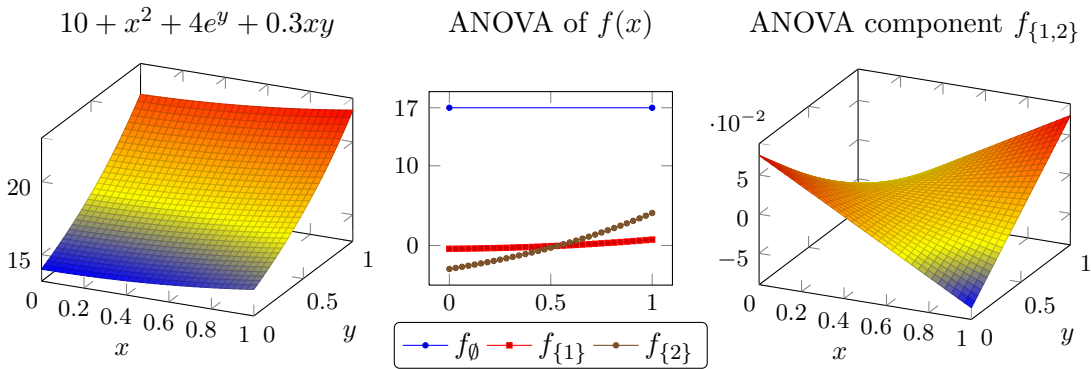


Figure 4.4: The function $f(x, y) = 10 + x^2 + 4e^y + 0.3xy$ and its discretized ANOVA decomposition.

resentation and backwards. These algorithms are presented in detail in Appendix A.2.3, together with a rigorous definition and implementational details.

A one-dimensional example for the decomposition is shown in Figure 4.3: the left picture shows the graph of $f(x) = x^2$ and the right its two ANOVA components f_\emptyset and $f_{\{1\}}$. The constant is $\int x^2 dx = 1/3$, represented by the single basis coefficient $a_{-1,0}$. Figure 4.4 (left) shows the graph of $f(x, y) = 10 + x^2 + 4e^y + 0.3xy$ and its ANOVA components $f_\emptyset, f_{\{1\}}, f_{\{2\}}$ (middle) and $f_{\{1,2\}}$ (right). We see that the main contributions come from $f_{\{2\}}$ and $f_{\{1\}}$; the highest order component is small. Note that it is not equal to $0.3xy$ due to the difference construction involved.

A further example of high practical relevance is the density of a normal distribution, shown in Figure 4.5 (left), along with its components (middle and right). The normal distribution requires the highest order term for any reasonable accuracy. In fact, the “mean effective dimension” of the normal distribution grows linearly with the nominal

4 Low Effective Dimensionality – A Dimension Adaptive Method

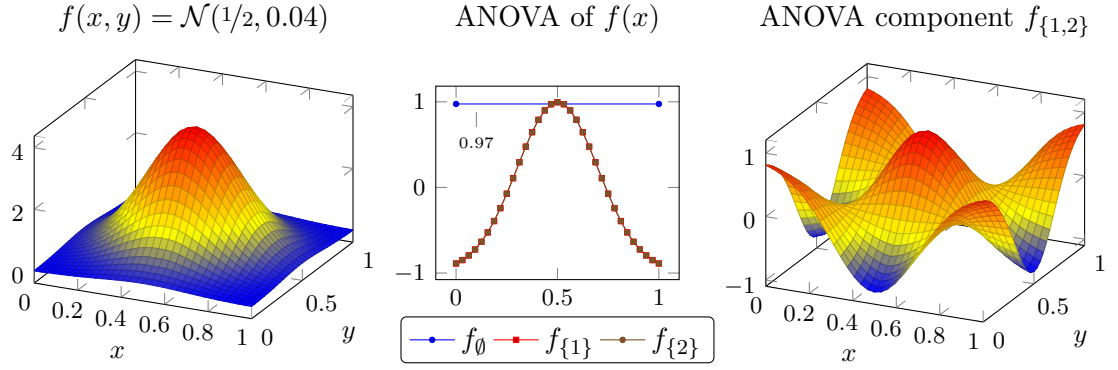


Figure 4.5: The normal distribution $\mathcal{N}(\mu, \sigma)$ with $\mu = (1/2, 1/2)$ and $\sigma = \text{diag}(0.04, 0.04)$ and its discretized (Lebesgue) ANOVA decomposition.

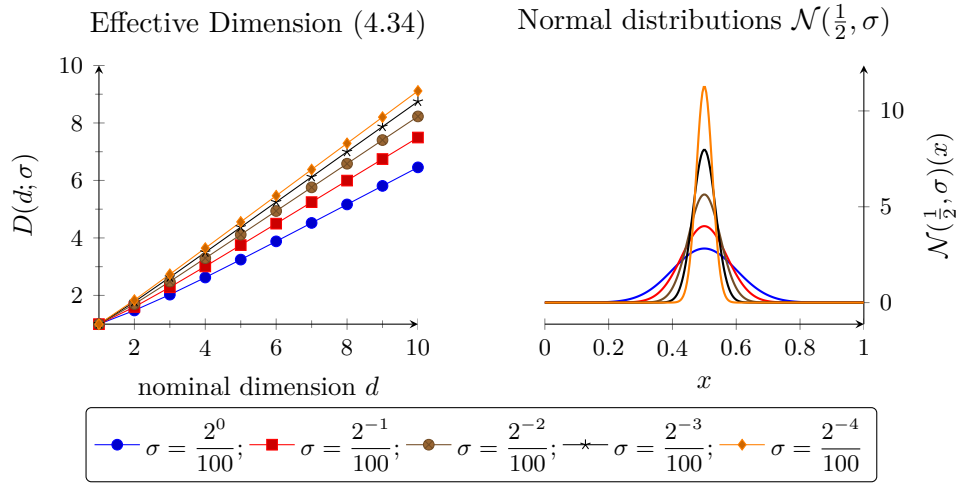


Figure 4.6: The mean effective dimension $D(d, \sigma)$, (4.34), for different normal distributions.

dimension. Here, we refer to the variance–based definition of mean effective dimension according to [Owe03]: For any tensor product function $g(x) = \prod_{j=1}^d g_j(x_j)$, the mean effective dimension is

$$D = \frac{\sum_{j=1}^d \gamma_j^2 / (\gamma_j^2 + \mu_j^2)}{1 - \prod_{j=1}^d \mu_j^2 / (\gamma_j^2 + \mu_j^2)}, \quad \mu_j = \int_0^1 g_j(x) dx, \quad \gamma_j^2 = \int_0^1 (g_j(x) - \mu_j)^2 dx, \quad (4.34)$$

compare [Owe03]. Figure 4.6 shows the mean effective dimension $D = D(d, \sigma)$ for densities of normal distributions $\mathcal{N}(1/2, \sigma)$ and $\sigma = 1/100 \cdot 2^i$, $i = 0, -1, -2, -3, -4$. We see that the effective dimension $D(d, \sigma)$ grows linearly with the nominal dimension. This property yields a further intuition why the approximation of Gaussians is difficult.

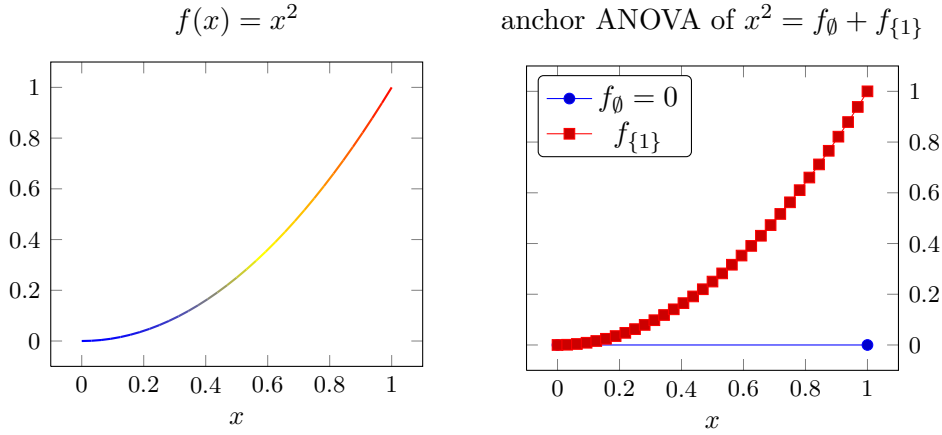


Figure 4.7: One-dimensional example of the discretized anchor ANOVA decomposition by means of the hierarchical hat basis.

Two Basis Systems For The Anchor ANOVA

We study the case of a discretized anchor ANOVA with anchor $a = 0$, i.e. we deal with the one-dimensional projector $Pf(x) = \int_0^1 f(x)\delta(x) dx = f(0)$. A basis for the splitting $V_h^{(1)} = \mathbf{1} \oplus W_h^{(1)} = \text{span}\{1, \phi_1, \dots, \phi_n\}$ needs to fulfill $P\phi_i = 0$. This property is fulfilled by the hierarchical hat basis with constant on the left boundary. We have already seen the corresponding decomposition in Section 2.1.4 where we investigated sparse grid error estimates for the case of non-homogeneous boundary conditions; it is actually the same as the anchor ANOVA decomposition. Consequently, we only need to compute the basis representation for the hierarchical hat basis in order to get anchor ANOVA components.

This is illustrated in Figure 4.7 for $f(x) = x^2$ and in Figure 4.8 for $f(x, y) = 10 + x^2 + 4e^y + 0.3xy$. Note the differences between the classical ANOVA decomposition of the same functions (illustrated in Figure 4.3 and Figure 4.4, respectively).

A further, straight-forward and readily implementable example which is compatible with the anchor ANOVA decomposition is a spline basis of the form $\{\phi_0, \phi_1, \dots, \phi_N\}$,

$$V_i^h = \underbrace{\text{span}\{\text{---}\}}_{\text{constant}} \oplus \underbrace{\text{span}\{\text{zigzag basis functions}\}}_{\text{non-constant}} \quad (4.35)$$

which fulfills $P\phi_i = \phi_i(0) = 0$ for $i > 0$. A one-dimensional basis expansion $f^h = \sum_{i=0}^N a_i \phi_i$ is given by $a_0 = f(0)$ and $a_i = f(x_i) - f(0)$, $i > 0$. The tensor product approach discussed in Section 4.1.3 yields the full grid discretization

$$f^h = \sum_{0 \leq i_1, \dots, i_d \leq N} a_{i_1, \dots, i_d} \phi_{i_1, \dots, i_d} \quad (4.36)$$

which can be computed by applying the one-dimensional formulas for a_i successively along the different coordinate directions. The discretized anchor ANOVA terms can be

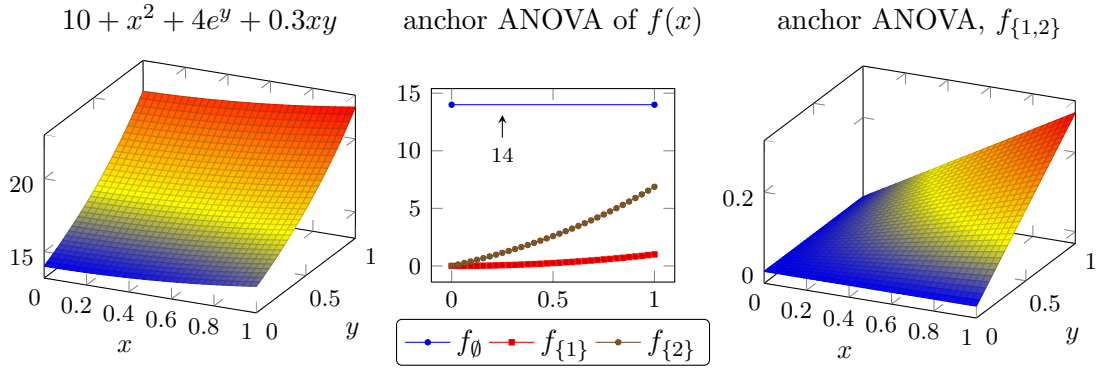


Figure 4.8: The function $f(x, y) = 10 + x^2 + 4e^y + 0.3xy$ and its discretized anchor ANOVA decomposition.

read off using

$$f_u^h = \sum_{\substack{0 \leq i_1, \dots, i_d \leq N \\ i_j \neq 0 \text{ for } j \in u}} a_{i_1, \dots, i_d} \prod_{k \in u} \phi_{i_k}. \quad (4.37)$$

This approach is equivalent to classical full grids for every component f_u , with total complexity $(N+1)^d$ and accuracy $\|f - f^h\|_a = \mathcal{O}(N^{-2})$ for $a \in \{2, \infty\}$ if f has bounded second derivatives (we studied the case of mixed derivatives in Lemma 2.1.7). However, if we know in advance that $f_u = 0$ for $|u| > d_s$, we can eliminate the associated coefficients and do the same job, i.e. we compute

$$f^h = \sum_{\substack{0 \leq i_1, \dots, i_d \leq N \\ |\{i_j \neq 0\}| < d_s}} a_{i_1, \dots, i_d} \phi_{i_1, \dots, i_d}. \quad (4.38)$$

In this case, the complexity is reduced to exponential growth in d_s ,

$$\binom{d}{d_s} (N+1)^{d_s} \leq \left(\frac{d \cdot e}{d_s}\right)^{d_s} (N+1)^{d_s}. \quad (4.39)$$

Note that (4.38) allows the same computation of a_i as before, we only need to check whether components are needed. The constant times rest splitting allows this simple sort of algorithm.

A similar full grid approach could be implemented for the Neumann prewavelet which discretizes the classical ANOVA decomposition (although its transformation is more involved, see Appendix A.2.3).

4.2 Weighted Spaces and A Priori Optimized Sparse Grids

In the following sections, we consider methods to exploit functions in which higher dimensional terms of the dimension decomposition are either small or vanish completely. If such structure is known in advance, specialized approximation methods can be derived

a priori, provided additional knowledge about the smoothness of the ANOVA terms is known. This section presents new results about sparse grid spaces for functions in which the terms of our dimension decomposition can be weighted with real weights and have bounded second mixed derivatives. We start with the case where all weights are known in advance. Section 4.3 will then deal with a posteriori grid adaptation, i.e. with the case when weighted structure is to be expected without explicit knowledge of the weights.

Our approach is based on the ideas presented in Section 4.1.3: the discretized method should represent the splitting into constant and rest, $V^{(1)} = \mathbf{1} \oplus W^{(1)}$, which is characterized by a one-dimensional projector $P: V^{(1)} \rightarrow \mathbf{1}$. Any tensor product basis built using one-dimensional basis functions $\{1, \phi_1, \dots, \phi_N\}$ of $V^{(1)}$ with $P\phi_i = 0$ will automatically yield a compatible, discretized ANOVA decomposition, see Section 4.1.3 for details.

While ANOVA decompositions are based on sets of directions, we now need to discretize each set of directions. We have already seen how to discretize the anchor ANOVA decomposition if its components have bounded second derivatives in Section 4.1.3.

If the functions satisfy mix smoothness, we can employ sparse grid structure. This is particularly interesting if dimension adaptive procedures are necessary to find the effective dimension, for example a parameter like d_s , automatically. Furthermore, the exponential growth of the form N^d can be reduced by sparse grids, thereby allowing higher (effective) dimensions. We will discuss sparse grid based approximation spaces in the following sections.

4.2.1 Weighted Mix Spaces

We will now formalize weighted spaces suitable for sparse grid approximation. In comparison to the literature cited at the beginning of Chapter 4, our approach is based on weighted second mixed derivatives which govern the approximation properties. The approach is similar to the construction presented in [Hol08, section 4.2] where optimized sparse grid methods for the problem of integration in weighted spaces have been derived.

Let $\Gamma := \{\gamma_u \mid u \subseteq \{1, \dots, d\}\}$ be a non-empty set of real weights, $\gamma_u \geq 0$. Then, we define the space of functions with weighted dimension decomposition

$$H_a^\Gamma := \{f \in X^{q,2}[0, 1]^d \mid |||f|||_a < \infty\} \quad (4.40)$$

for either $a = \infty$ or $a = 2$ with the seminorm

$$|||f|||_a := \sum_{\substack{u \subseteq \{1, \dots, d\} \\ \gamma_u \neq 0}} \frac{1}{\gamma_u} |f_u|_{2,a}. \quad (4.41)$$

Here, $f = \sum_{u \subseteq \{1, \dots, d\}} f_u$ is a dimension decomposition with uniquely determined terms f_u . The special case $\gamma_u = 0$ is allowed: in this case, we require $|f_u|_{2,a} = 0$ (i.e. we use the notation $0/0 = 0$). For reasons of simplicity, we also assume $f_u \equiv 0$ for $\gamma_u = 0$. The seminorm

$$|f_u|_{2,a} := \|D_u^2 f_u\|_a \quad (4.42)$$

is the norm of the second mixed derivatives of f_u in the set of direction u ,

$$D_u^2 f_u = \prod_{j \in u} \frac{\partial^2}{\partial x_j^2} f_u. \quad (4.43)$$

Since per construction, $\frac{\partial}{\partial x_j} f_u = 0$ for $j \notin u$, we will occasionally treat f_u as function $f_u: [0, 1]^{|u|} \rightarrow \mathbb{R}$. Note that

$$|f_u|_{\mathbf{2},a} = \gamma_u (\|f\|_a - \sum_{v \neq u} \frac{1}{\gamma_u} |f_v|_{\mathbf{2},a}) \leq \gamma_u \|f\|_a, \quad (4.44)$$

so γ_u weights the part of $|f_u|_{\mathbf{2},a}$ relative to $\|f\|_a$.

4.2.2 Sparse Grids with Optimized Cost/Gain Ratio

Unweighted sparse grid spaces can be derived as result of an optimization process which balances local cost and local benefit, see [BG04] or [Bun98]. Such an optimization takes place on inner grid points, i.e. in the highest dimensional component of an ANOVA decomposition. The optimization procedure, especially its reduction to *local* information, can be applied to each term of a dimension decomposition as well if the constant plus rest splitting is respected. Our approach will use this idea: we apply the local optimization procedure to each decomposition term separately and re-use results known for classical sparse grids. The coupling between separated optimizations can be achieved using a single reference value. The following steps are thus the definition of local cost–benefit ratios on each decomposition term followed by the definition of a reference value which couples the local ratios in order to optimize the overall cost. We will then analyze the global cost and error.

We make use of the special notation $l = -1$ for the constant basis function as we did before, so the one–dimensional hierarchy starts with $l = -1$ for the constant, followed by $l = 0$ for the linear basis function and the remaining ones with $l > 0$ as for the unweighted case. This is to maintain consistency with the notation for inner nodes. In terms of basis functions, we have thus $\phi_{-1,0} := 1$ as the only basis function on level $l = -1$ which is attached to the left boundary and $\phi_{0,1} := x$ for level $l = 0$, attached to the right boundary. Since the constant plays a special role, we exclude it from the often important sum of all level indices and write for $\mathbf{l} = (l_1, \dots, l_d)$

$$|\mathbf{l}|_1^* := \sum_{\substack{j=1 \\ l_j \neq -1}}^d l_j. \quad (4.45)$$

To identify the ANOVA term f_u which contains a given level index \mathbf{l} , we write

$$u(\mathbf{l}) := \{j \mid l_j \text{ is a non-constant part} \Leftrightarrow l_j \neq -1\}. \quad (4.46)$$

Thus, an element $f_{\mathbf{l}}$ defined by a multilevel basis representation

$$f = \sum_{\mathbf{l} \in (\mathbb{N} \cup \{-1, 0\})^d} f_{\mathbf{l}} =: \sum_{\mathbf{l} \in \mathbf{L}^d} f_{\mathbf{l}} \quad (4.47)$$

4.2 Weighted Spaces and A Priori Optimized Sparse Grids

as in (2.85) is uniquely associated with the ANOVA component $f_{u(1)}$. Furthermore, f_u can be represented by a specific subset of the $f_{\mathbf{1}}$,

$$f_u = \sum_{\substack{\mathbf{1} \in \mathbf{L}^d \\ u(\mathbf{1})=u}} f_{\mathbf{1}}. \quad (4.48)$$

Note that each $f_{\mathbf{1}}$ in $f = \sum_{\mathbf{1}} f_{\mathbf{1}}$ is formally defined in d dimensions, so f_u is also formally defined on $[0, 1]^d$. But since all directions $j \notin u$ are constant, we can regard it as $|u|$ -dimensional function and write

$$f_u = \sum_{\mathbf{1} \in \mathbf{L}^{|u|}} f_{u,l}. \quad (4.49)$$

We present the optimization for the hierarchical hat basis which discretizes the anchor ANOVA decomposition. Since the hat basis fulfills $\phi_{l,i}(0) = 0$ for every basis function except for the constant $\phi_{-1,0} = 1$, and furthermore $\phi_{l,i}(1) = 0$ for all but the constant and the linear, we have almost homogeneous boundary conditions. Thus, we start with the case of homogeneous boundary conditions and come back to non-homogeneous boundary conditions later. To this end, we assume the coefficient for the linear basis function, $f_{0,1}$, vanishes and the associated basis function at $l = 0$ will not be considered in the following reasoning.

We are now in a position to apply estimates for $|u|$ -dimensional functions with vanishing boundary conditions. We have already studied this case thoroughly in Section 2.1.2 for the hierarchical hat basis, so we will only summarize the relevant properties here. It holds by means of Lemma 2.1.6 that

$$\|f_{u,l}\|_{L_2} \leq 3^{-|u|} \cdot 2^{-2|\mathbf{1}_1^*|} \cdot |f_u|_{\mathbf{2},2} \quad (4.50)$$

and

$$\|f_{u,l}\|_{L_\infty} \leq 2^{-|u|} \cdot 2^{-2|\mathbf{1}_1^*|} \cdot |f_u|_{\mathbf{2},\infty} \quad (4.51)$$

for level components of a fixed ANOVA component f_u . Due to the weighting property $|f_u|_{\mathbf{2},a} \leq \gamma_u \|f\|_a$ for $a = 2$ and $a = \infty$, we can deal with both cases simultaneously using constants $c^{(\infty)} := -1$ and $c^{(2)} := -\text{ld } 3$ and

$$\|f_{u,l}\|_{L_a} \leq 2^{-2|\mathbf{1}_1^*| + c^{(a)} \cdot |u| + \text{ld } \gamma_u} \cdot \|f\|_a. \quad (4.52)$$

Furthermore, we find

$$\|f_{u,l}\|_E \leq (2 \cdot 12^{(|u|-1)/2})^{-1} \cdot 2^{-2|\mathbf{1}_1^*|} \cdot \left(\sum_{j \in u} 2^{2l_j} \right)^{1/2} \cdot \gamma_u \|f\|_\infty \quad (4.53)$$

$$= 2^{-2|\mathbf{1}_1^*| + c^{(E)} \cdot |u| + \tilde{c}^{(E)} + \text{ld } \gamma_u} \cdot \left(\sum_{j \in u} 2^{2l_j} \right)^{1/2} \cdot \|f\|_\infty \quad (4.54)$$

with $c^{(E)} := -1/2 \text{ld } 3 - 1$ and $\tilde{c}^{(E)} := 1/2 \text{ld } 3$.

On the other hand, we know the cost of one complete level \mathbf{l} which is the number of different basis functions,

$$C(\mathbf{l}) := |W_{\mathbf{l}}| = \prod_{\substack{j=1 \\ l_j > 0}}^d 2^{l_j-1} = \prod_{\substack{j=1 \\ l_j \neq -1}}^d 2^{l_j-1} = 2^{|\mathbf{l}|_1^* - |u(\mathbf{l})|} \quad (4.55)$$

where the second equality employs that $l_j = 0$ is not possible. The special case for the linear basis function on level $l = 0$ is discussed later. The local cost can also be written as

$$C(\mathbf{l}) = 2^{n(\mathbf{l})-1} \quad (4.56)$$

where we use $n(\mathbf{l}) := |\mathbf{l}|_1^* - |u(\mathbf{l})| + 1$ to denote the scalar level of a multi-index with respect to its ANOVA component $u(\mathbf{l})$ with the special case $n(-1, \dots, -1) := 0$ (in consistency with (2.18)).

For example, the level $l = (1, \dots, 1)$ belongs to the highest order component $u(\mathbf{l}) = \{1, \dots, d\}$ and has $n(\mathbf{l}) = 1$. A level like $l = (-1, 1, \dots, 1)$ belongs to $u(\mathbf{l}) = \{2, \dots, d\}$ and also has scalar level $n(\mathbf{l}) = 1$, but in its ANOVA component $u(\mathbf{l})$.

Given a set of level indices, we can use the local contributions $\|f_{u, \mathbf{l}}\|$ and associated costs $C(\mathbf{l})$ to compute the global cost and accuracy for f . Before actually doing so, we choose the set of levels a priori as optimal choice with respect to our weights and the mix smoothness. To this end, we search for an optimal approximation space $V^{(\text{opt})}$, defined by a set of level indices, which has the smallest possible error for prescribed work count w . The space $V^{(\text{opt})}$ is chosen as result of the optimization procedure

$$\underbrace{\max_{f \in X_0^{q,2}: \|f\|=1} \|f - f_{V^{(\text{opt})}}\|}_{\text{worst case in } V^{(\text{opt})}} = \min_{U \subset V^{(d)}: |U|=w} \underbrace{\max_{f \in X_0^{q,2}: \|f\|=1} \|f - f_U\|}_{\text{worst case in } U} \quad (4.57)$$

which uses the (relative) optimum over all functions with bounded second mixed derivatives and homogeneous boundary conditions, $f \in X_0^{q,2}$. Note that it is not the optimum for one particular function, but for all elements in that space. Reformulating the optimization problem in terms of local cost and local benefits $B(\mathbf{l})$ reveals a surprisingly simple algorithm to compute the optimal solution, see [BG04]: the global solution can be obtained using local cost–benefit ratios $B(\mathbf{l})/C(\mathbf{l})$ which are below a reference value depending on the prescribed work count. For details of this reasoning, we refer the interested reader to [BG04]. We will turn our attention to the local cost–benefit ratios and a reference value suitable for our weighted dimension decomposition. In a sparse grid decomposition $f^h = \sum_{\mathbf{l}} f_{\mathbf{l}} = \sum_{\mathbf{l}} f_{u(\mathbf{l}), \mathbf{l}}$, the local benefit of a level index \mathbf{l} depends crucially on $f_{u(\mathbf{l}), \mathbf{l}}$. Consequently, we define the local benefit as upper bound on $\|f_{u(\mathbf{l}), \mathbf{l}}\|^2$. We obtain

$$B^{(a)}(\mathbf{l}) := 2^{-4|\mathbf{l}|_1^* + 2c^{(a)} \cdot |u(\mathbf{l})| + 2\text{ld} \gamma_{u(\mathbf{l})}} \cdot \|f\|_a^2 \quad (4.58)$$

for the L_∞ norm ($a = \infty$) and the L_2 norm ($a = 2$) and furthermore

$$B^{(E)}(\mathbf{l}) := 2^{-4|\mathbf{l}|_1^* + 2c^{(E)} \cdot |u(\mathbf{l})| + 2\tilde{c}^{(E)} + 2\text{ld} \gamma_{u(\mathbf{l})}} \cdot \left(\sum_{j \in u(\mathbf{l})} 2^{2l_j} \right) \cdot \|f\|_\infty^2 \quad (4.59)$$

4.2 Weighted Spaces and A Priori Optimized Sparse Grids

for the energy norm.

The actual optimization of cost versus benefit can now be done by considering the cost–benefit ratio

$$\text{cbr}^{(a)}(\mathbf{l}) := \frac{B^{(a)}(\mathbf{l})}{C(\mathbf{l})} \quad (4.60)$$

for every local contribution \mathbf{l} , compare [BG04]. The global optimization procedure presented in [BG04] yields an optimal subspace if all level indices \mathbf{l} fulfilling

$$\text{cbr}^{(a)}(\mathbf{l}) \geq \omega \quad (4.61)$$

for a reference cost–benefit ratio ω are taken into account. The value ω is yet to be determined. Note that due to our reasoning above, the entity $\text{cbr}^{(a)}(\mathbf{l})$ depends solely on *one* ANOVA component, namely on $f_{u(\mathbf{l})}$. The reference value will now balance the different terms, thus, it combines the isolated local entities to a global criterion and relates different ANOVA components to each other. The reference value ω should somehow depend on the prescribed work count motivated at the beginning of this section. We follow the reasoning in [BG04] and use a specific scalar level n to fix the reference value. While [BG04] uses the sparse grid level n and $\omega := \text{cbr}^{(a)}(\bar{\mathbf{l}})$ with one specific level index $\bar{\mathbf{l}}$ belonging to the finest resolution in such a sparse grid, $n(\mathbf{l}) = n$, we will here use a specific level index $\bar{\mathbf{l}}$ which belongs to the finest resolution used for the ANOVA component with largest weight. More precisely, let

$$\bar{u} := \underset{\substack{u \subseteq \{1, \dots, d\} \\ u \neq \emptyset}}{\text{argmax}} \{ \gamma_u \} \quad (4.62)$$

be the ANOVA component with largest weight. If the maximum is not unique, one of the matching components of largest dimensionality should be chosen. Then, we define $\bar{\mathbf{l}}$ using

$$\bar{l}_i := \begin{cases} -1 & i \notin \bar{u}, \\ n & i = \min(\bar{u}), \\ 1 & \text{else} \end{cases} \quad (4.63)$$

where $n \in \mathbb{N}$ is a given parameter indicating the total work count (a scalar level). Note that $\bar{\mathbf{l}}$ is associated with a $|\bar{u}|$ -dimensional grid and we have $|\bar{\mathbf{l}}|_1^* = n + |\bar{u}| - 1$ and thus $n(\bar{\mathbf{l}}) = n$. We define the reference value ω as

$$\omega := \text{cbr}^{(a)}(\bar{\mathbf{l}}) \quad (4.64)$$

and obtain thus the optimal sparse grid for the weighted anchor ANOVA decomposition by using all level indices \mathbf{l} fulfilling

$$\text{cbr}^{(a)}(\mathbf{l}) \geq \text{cbr}^{(a)}(\bar{\mathbf{l}}). \quad (4.65)$$

The result is stated in

Lemma 4.2.1 (A priori optimized sparse grids for weighted mix spaces). *Let $\Gamma := \{\gamma_u \geq 0 \mid u \subseteq \{1, \dots, d\}\}$ be a non-empty set of weights and $f \in H_a^\Gamma$ for either L_∞ ($a = \infty$) or L_2 ($a = 2$) where H_a^Γ is based on the anchor ANOVA decomposition $f = \sum f_u$ of f with $\gamma_u = 0 \Rightarrow f_u \equiv 0$. Furthermore, let $\bar{u} = \operatorname{argmax}_{u \subseteq \{1, \dots, d\}, u \neq \emptyset} \{\gamma_u\}$ with the special cases discussed for (4.62).*

Then, the sparse grid space

$$V^{n, \Gamma, a} := \{f^h = \sum_{\mathbf{l} \in \mathbf{L}_{(a)}^n} f_{\mathbf{l}}\} = \bigoplus_{\mathbf{l} \in \mathbf{L}_{(a)}^n} W_{\mathbf{l}} \quad (4.66)$$

spanned by the hierarchical hat basis has optimal cost-benefit ratio with respect to approximation in the $\|\cdot\|_a$ norm, $a \in \{\infty, 2, E\}$, if $\mathbf{L}_{(a)}^n$ is chosen as

$$\mathbf{L}_{(\infty)}^n := \{\mathbf{l} \mid |\mathbf{l}_1^*| + \frac{1}{5}|u(\mathbf{l})| - \frac{2}{5} \operatorname{ld} \gamma_{u(\mathbf{l})} \leq n + 1 \frac{1}{5} |\bar{u}| - 1 - \frac{2}{5} \operatorname{ld} \gamma_{\bar{u}}\}, \quad (4.67)$$

$$\begin{aligned} \mathbf{L}_{(2)}^n &:= \{\mathbf{l} \mid |\mathbf{l}_1^*| + \frac{1}{5} \operatorname{ld} \frac{9}{2} |u(\mathbf{l})| - \frac{2}{5} \operatorname{ld} \gamma_{u(\mathbf{l})} \\ &\leq n + |\bar{u}| \cdot (1 + \frac{1}{5} \operatorname{ld} \frac{9}{2}) - 1 - \frac{2}{5} \operatorname{ld} \gamma_{\bar{u}}\}, \end{aligned} \quad (4.68)$$

$$\begin{aligned} \mathbf{L}_{(E)}^n &:= \{\mathbf{l} \mid |\mathbf{l}_1^*| - \frac{1}{5} \operatorname{ld} (\sum_{j \in u(\mathbf{l})} 4^{l_j}) - \frac{2}{5} \operatorname{ld} \gamma_{u(\mathbf{l})} + \frac{1}{5} \operatorname{ld} 6 \cdot |u(\mathbf{l})| \\ &\leq n - \frac{1}{5} \operatorname{ld} (4^n + 4|\bar{u}| - 4) - \frac{2}{5} \operatorname{ld} \gamma_{\bar{u}} + (1 + \frac{1}{5} \operatorname{ld} 6) \cdot |\bar{u}| - 1\}. \end{aligned} \quad (4.69)$$

Proof. The sub spaces selection criteria follow from $\operatorname{cbr}^{(a)}(\mathbf{l}) \geq \operatorname{cbr}^{(a)}(\bar{\mathbf{l}})$ as elaborated below, the global optimality from the reasoning in [BG04].

For $a = \infty$ and $a = 2$, we find

$$\operatorname{cbr}^{(a)}(\mathbf{l}) := \frac{B^{(a)}(\mathbf{l})}{C(\mathbf{l})} = 2^{-5|\mathbf{l}_1^*| + |u(\mathbf{l})| \cdot (1 + 2c^{(a)}) + 2 \operatorname{ld} \gamma_{u(\mathbf{l})}} \cdot \|f\|_a^2 \quad (4.70)$$

and consequently

$$\operatorname{cbr}^{(a)}(\bar{\mathbf{l}}) = 2^{-5n + 5 + |\bar{u}| \cdot (-4 + 2c^{(a)}) + 2 \operatorname{ld} \gamma_{\bar{u}}} \|f\|_a^2. \quad (4.71)$$

The criterion $\operatorname{cbr}^{(a)}(\mathbf{l}) \geq \operatorname{cbr}^{(a)}(\bar{\mathbf{l}})$ simplifies to

$$|\mathbf{l}_1^*| + |u(\mathbf{l})| \cdot (-\frac{1}{5} - \frac{2}{5}c^{(a)}) - \frac{2}{5} \gamma_{u(\mathbf{l})} \leq n - 1 + |\bar{u}| (1 - \frac{1}{5} - \frac{2}{5}c^{(a)}) - \frac{2}{5} \gamma_{\bar{u}}. \quad (4.72)$$

For $a = \infty$ and $c^{(\infty)} = -1$, the left hand side contains the summand $|u(\mathbf{l})| \cdot 1/5$ whereas the right hand side becomes $|\bar{u}| \cdot 11/5$. For $a = 2$ and $c^{(2)} = -\operatorname{ld} 3$, we find $|u(\mathbf{l})| \cdot 1/5 \cdot \operatorname{ld}(9/2)$ and $|\bar{u}| \cdot (1 + 1/5 \cdot \operatorname{ld}(9/2))$, respectively.

4.2 Weighted Spaces and A Priori Optimized Sparse Grids

The energy norm ratio is

$$\text{cbr}^{(E)}(\mathbf{1}) = 2^{-5} |\mathbf{1}_1^*| \cdot (1+2c^{(E)}) + 2\bar{c}^{(E)} + 2 \text{ld} \gamma_{u(\mathbf{1})} \cdot \left(\sum_{j \in u(\mathbf{1})} 2^{2l_j} \right) \cdot \|f\|_\infty^2 \quad (4.73)$$

with reference quantity

$$\text{cbr}^{(E)}(\bar{\mathbf{1}}) = 2^{-5n + |\bar{u}| \cdot (-4 + 2c^{(E)}) + 5 + 2\bar{c}^{(E)} + 2 \text{ld} \gamma_{\bar{u}} + \text{ld}(4^n + 4|\bar{u}| - 4)} \cdot \|f\|_\infty^2. \quad (4.74)$$

Thus, we find the criterion

$$\begin{aligned} |\mathbf{1}_1^*| + |u(\mathbf{1})| \cdot \left(-\frac{1}{5} - \frac{2}{5}c^{(E)} \right) - \frac{2}{5} \text{ld} \gamma_{u(\mathbf{1})} - \frac{1}{5} \text{ld} \left(\sum_{j \in u(\mathbf{1})} 2^{2l_j} \right) \leq \\ n + |\bar{u}| \cdot \left(1 - \frac{1}{5} - \frac{2}{5}c^{(E)} \right) - 1 - \frac{2}{5} \text{ld} \gamma_{\bar{u}} - \frac{1}{5} \text{ld}(4^n + 4|\bar{u}| - 4) \end{aligned} \quad (4.75)$$

with $c^{(E)} = -1/2 \text{ld} 3 - 1$ and thus $-1/5 - 2/5 c^{(E)} = 1/5 \text{ld} 6$. \square

Besides the criteria for the subspace splittings, we also find explicit criteria for every ANOVA component:

Lemma 4.2.2. *Under the assumptions of Lemma 4.2.1, a given hierarchical hat basis representation $f^h = \sum_{\mathbf{l} \in \mathbf{L}_{(a)}^n} f_{\mathbf{l}}$ directly yields a discretized anchor ANOVA $f^h = \sum_{u \subseteq \{1, \dots, d\}} f_u^h$ by using a subset of $\mathbf{L}_{(a)}^n$ to compute the single components as follows:*

$$f_u^h = \sum_{\substack{\mathbf{l} \in \mathbf{L}_{(a)}^n \\ u(\mathbf{l})=u}} f_{\mathbf{l}}. \quad (4.76)$$

For $a \in \{\infty, 2\}$, this is equivalent to

$$f_u^h = \sum_{\substack{u(\mathbf{l})=u \\ |\mathbf{l}_1^*| \leq n^{(a)}(n, \gamma, u, \bar{u}) + |u| - 1}} f_{\mathbf{l}} \quad (4.77)$$

with the scalar, weighted levels

$$n^{(\infty)}(n, \gamma, u, \bar{u}) := n + \lfloor \frac{1}{5} (|\bar{u}| - |u|) + \frac{2}{5} \text{ld} \frac{\gamma_u}{\gamma_{\bar{u}}} \rfloor \in \mathbb{Z}, \quad (4.78)$$

$$n^{(2)}(n, \gamma, u, \bar{u}) := n + \lfloor (1 + \frac{1}{5} \text{ld} \frac{9}{2}) (|\bar{u}| - |u|) + \frac{2}{5} \text{ld} \frac{\gamma_u}{\gamma_{\bar{u}}} \rfloor \in \mathbb{Z}. \quad (4.79)$$

For the energy optimized components, we find

$$f_u^h = \sum_{u(\mathbf{l})=u} f_{\mathbf{l}} \quad (4.80)$$

$$|\mathbf{l}_1^*| - \frac{1}{5} \text{ld} \left(\sum_{j \in u(\mathbf{l})} 4^{l_j} \right) \leq n^{(E)}(n, \gamma, u, \bar{u}) + |u| - 1 - \frac{1}{5} \text{ld}(4^n + 4|u| - 4)$$

with

$$n^{(E)}(n, \gamma, u, \bar{u}) := n + \lfloor \frac{2}{5} \text{ld} \frac{\gamma_u}{\gamma_{\bar{u}}} + \frac{1}{5} (6 + \text{ld} 3) (|\bar{u}| - |u|) + \frac{1}{5} \text{ld} \left(\frac{4^n + 4|u| - 4}{4^n + 4|\bar{u}| - 4} \right) \rfloor \in \mathbb{Z}. \quad (4.81)$$

Proof. The hierarchical hat basis is a discretized anchor ANOVA decomposition due to the argumentation in Section 4.1.3.

Concerning the formulas, we find from Lemma 4.2.1 for *fixed* u (independent of \mathbf{l}) and $a \in \{2, \infty\}$ that

$$|\mathbf{l}_1^*| \leq n - 1 + |u| - |u| + |\bar{u}| \left(1 - \frac{1}{5} - \frac{2}{5}c^{(a)}\right) - |u| \left(-\frac{1}{5} - \frac{2}{5}c^{(a)}\right) + \frac{2}{5}(\text{ld } \gamma_u - \text{ld } \gamma_{\bar{u}}) \quad (4.82)$$

$$= n - 1 + |u| + (|\bar{u}| - |u|) \left(1 - \frac{1}{5} - \frac{2}{5}c^{(a)}\right) + \frac{2}{5} \text{ld } \frac{\gamma_u}{\gamma_{\bar{u}}}. \quad (4.83)$$

Since $|\mathbf{l}_1^*| \in \mathbb{N}_0$, we can apply gauss brackets and find the assertion using the constants $c^{(a)}$ as in the proof for Lemma 4.2.1.

The proof for energy grids works by introducing the additional summands $\pm|u|$ and $\pm\frac{1}{5} \text{ld}(4^n + 4|u| - 4)$. □

The theorem yields a grid consisting of regular sparse grids for every ANOVA component where each component has its separate level. This can be seen by substituting $|u|$ by d and $n^{(a)}(n, \gamma, u, \bar{u})$ by n : the substitution in (4.77) yields the classical sparse grid selection criteria $|\mathbf{l}| \leq n + d - 1$ and the substitution in (4.80) yields the corresponding selection criteria for energy optimal grids, compare definition 2.1.1 on page 22. The level $n^{(a)}(n, \gamma, u, \bar{u})$ depends on the component's dimension and its weight, but it is still an integer like the well-known sparse grid level n .

Note that the sparse grid optimization procedure of [BG04] follows from Lemma 4.2.1 as special case, namely for $\gamma_{1, \dots, d} = 1$ and $\gamma_u = 0$ for $|u| < d$: in this case, boundary components $|u| < d$ are skipped and the reference level \bar{l} is the same as in [BG04], namely $\bar{l} = (n, 1, \dots, 1)$. The weights disappear and the $|\bar{u}|$ and $|u|$ terms cancel each other.

Note furthermore, that a component with $\gamma_u = 0$ will not receive any points since $\text{ld } \gamma_u = -\infty$.

Lemma 4.2.1 does not assume normalized weights. If we introduce $\tilde{\gamma}_u := \gamma_u / \gamma_{\bar{u}}$, we find $0 \leq \tilde{\gamma}_u \leq 1$ and $\tilde{\gamma}_{\bar{u}} = 1$. Thus, the normalized weights $\tilde{\gamma}_u$ allow simplifications of the notation.

Note that a practical realization of Lemma 4.2.1 will also need some sort of hanging node condition: for every level \mathbf{l} on component $u(\mathbf{l})$, its hierarchical ancestors on lower dimensional ANOVA components $v \subset u(\mathbf{l})$ should be inserted as well.

Non Homogeneous Boundary Conditions

So far, we have studied the case of an anchor ANOVA in which every component f_u exhibits homogeneous boundary conditions of the type $j \in u, x_j \in \{0, 1\} \Rightarrow f_u(x) = 0$. We will now discuss the relevant techniques to generalize our results to non-homogeneous boundary conditions. This involves a strategy to select boundary levels and care when it comes to global error analysis.

The selection of boundary levels could be done by means of the local cost–benefit optimization as before, but there is a simpler approach: since any non–vanishing component will, in general, need at least level 1, the boundary level 0 is inserted anyway to fulfill hanging node conditions. Thus, we exclude level 0 from the subspace selection explicitly and insert it if and only if it is required due to hanging node conditions (a descendant is inserted).

To allow error analysis, we proceed as in our derivation of sparse grids in Section 2.1.4: we introduce a boundary splitting and operate on each boundary separately. Our task here is actually just a special case of the results derived in Section 2.1.4 since we have already dealt with the left boundary by means of the ANOVA decomposition.

The idea is as follows: the anchor ANOVA decomposition can be formulated in terms of the hierarchical basis. A component f_u fulfills $P_j f_u = 0$ for every $j \in u$ where P_j is the associated ANOVA projector. In terms of the hierarchical basis, this means we have *no* contributions of $\phi_{l,i}$ for $l_j = -1$, $j \in u$. The idea of Section 2.1.4 is to introduce a three–term–splitting into left boundary (constant), right boundary (linear) and rest such that the one–dimensional space is split uniquely into $\mathbf{1} \oplus \text{lin} \oplus \widetilde{W}$ by corresponding projectors. Since f_u does not have contributions from the constant, we will split it only into right boundary and rest. Since the rest terms satisfy homogeneous boundary conditions with respect to the varying directions, we can then apply all theorems separately to the rest terms.

In detail, let $f_u = \sum_{\mathbf{l} \in \{-1,0,*\}^{|u|}} \bar{f}_{u,\mathbf{l}}$ be the unique three–term–decomposition of f_u according to Definition 2.1.2. As discussed, the constant contributions vanish since f_u is an ANOVA component compatible to the three–term–decomposition. Thus, $f_u = \sum_{\mathbf{l} \in \{0,*\}^{|u|}} \bar{f}_{u,\mathbf{l}}$. We extend the definition of $|f_u|_{\mathbf{2},a}$ accordingly to

$$|f_u|_{\mathbf{2},a} := \max_{\substack{\mathbf{l} \in \{0,*\}^{|u|} \\ |\mathbf{l}^{(*)}| > 0}} |f_{u,\mathbf{l}^{(*)}}|_{\mathbf{2},a} \quad (4.84)$$

to measure mix smoothness on lower dimensional boundary manifolds as well, compare with the seminorm of Lemma 2.1.16. Remember from (2.88) that $\mathbf{l}^{(*)} = \{j \mid l_j = *\}$, so $D_{\mathbf{l}^{(*)}}^2$ applies only in directions belonging to the varying parts. Then, we build the weighted mix spaces as before, but with the new seminorm. Note that weights apply to $|f_u|_{\mathbf{2},a}$ as before, not to the underlying boundary decomposition.

Finally, we apply Lemma 2.1.16 to find

Lemma 4.2.3 (Single component error with non–homogeneous boundaries). *Let f_u be the anchor ANOVA component for u of f , $f \in H_a^\Gamma$. Let f_u^h be a regular sparse grid interpolant of f_u on level $n \in \mathbb{N}$ and assume that² either $\bar{f}_{u,\mathbf{0}} = 0$ or $\bar{f}_{u,\mathbf{0}} - \bar{f}_{u,\mathbf{0}}^h = 0$.*

²This is true if the linear basis function on level $\mathbf{0} = (0, \dots, 0)$ is inserted as hanging node. The weighted space selection is based on second derivatives, so linear functions need extra subspace selection handling. But usually, level 1 is necessary anyway, and level 0 as hanging node has no error.

Then, the error can be estimated by

$$\|f_u - f_u^h\|_a \leq \sum_{\substack{\mathbf{l} \in \{0, *\}^{|u|} \\ |\mathbf{l}^{(*)}| > 0}} c_a^{|\mathbf{l}^{(0)}|} \cdot C_{a, |\mathbf{l}^{(*)}|} \cdot 2^{-2n} \cdot |f_{u, \mathbf{l}^{(*)}}|_{2, a} \cdot A(|\mathbf{l}^{(*)}|, n) \quad (4.85)$$

$$< 2^{|u|} \cdot \underbrace{\max_{j=0, \dots, |u|-1} c_a^j \cdot \max_{j=1, \dots, |u|} C_{a, j}}_{=: C_a(|u|)} \cdot 2^{-2n} \cdot |f_u|_{2, a} \cdot A(|u|, n) \quad (4.86)$$

$$= \mathcal{O}(2^{-2n} n^{|u|-1}). \quad (4.87)$$

The constants c_a and $C_{a, j}$ are as in Lemma 2.1.16, i.e. $c_\infty = 1$, $c_2 = 3^{-1/2}$ (norm of the linear basis functions in one dimension) and $C_{\infty, j} = 2 \cdot 8^{-j}$, $C_{2, j} = 2 \cdot 12^{-j}$ (dimension dependent coefficients of inner sparse grid errors in dimension j). Thus, we have $C_\infty(m) = 2^m/4$ and $C_2(m) = 2^m/6$.

Note that the constants could be improved, compare Lemma 2.1.16.

Proof. The proof follows directly from Lemma 2.1.16 since we can regard f_u as $|u|$ -dimensional function. Furthermore, $|f_u|_{2, a}$ as defined in (4.84) is compatible with the requirements of Lemma 2.1.16. However, instead of $3^{|u|} - 2^{|u|}$, the number of summands here is

$$\sum_{\substack{\mathbf{l} \in \{0, *\}^{|u|} \\ |\mathbf{l}^{(*)}| > 0}} 1 = 2^{|u|} - 1 < 2^{|u|}. \quad (4.88)$$

□

Global Cost And Error

We are now in a position to analyze global cost and accuracy for our weighted sparse grid $V^{n, \Gamma, a}$.

Lemma 4.2.4 (Global Cost). *The number of basis functions, i.e. the cost of $V^{n, \Gamma, a}$ for $a \in \{2, \infty\}$, is bounded by*

$$|G^{n, \Gamma, a}| \leq \sum_{u \subseteq \{1, \dots, d\}} (-2)^{|u|} + 2^n \cdot 2^{|\bar{u}| \cdot C_n^{(a)}} \cdot \sum_{u \subseteq \{1, \dots, d\}} 2^{|u| \cdot (1 - C_n^{(a)})} \cdot \left(\frac{\gamma_u}{\gamma_{\bar{u}}} \right)^{2/5} \sum_{i=0}^{|u|-1} \binom{n^{(a)}(n, \gamma, u, \bar{u}) - |u| + 1}{i} \cdot 2^{|u| - i - 1}; \quad (4.89)$$

$$|G^{n, \Gamma, a}| = \mathcal{O}\left(2^n \sum_{u \subseteq \{1, \dots, d\}} \frac{(\gamma_u)^{2/5}}{\gamma_{\bar{u}}} (n^{(a)}(n, \gamma, u, \bar{u}))^{|u|-1}\right). \quad (4.90)$$

Here, $C_n^{(\infty)} := 11/5$ and $C_n^{(2)} := (1 + 1/5 \text{ld } 9/2)$ are constants such that

$$n^{(a)}(n, \gamma, u, \bar{u}) = n + \lfloor C_n^{(a)} (|\bar{u}| - |u|) + \frac{2}{5} \text{ld } \frac{\gamma_u}{\gamma_{\bar{u}}} \rfloor. \quad (4.91)$$

4.2 Weighted Spaces and A Priori Optimized Sparse Grids

Proof. It holds

$$|G^{n,\Gamma,a}| = 1 + \sum_{\substack{u \subseteq \{1,\dots,d\} \\ |u| > 0}} |G_{n^{(a)}(n,\gamma,u,|u|)}^{|u|,\text{right}}| \quad (4.92)$$

since for each non-empty u , we have a *regular* sparse grid $G_n^{d,\text{right}}$ of dimension $|u|$ and level $n^{(a)}(n, \gamma, u, |u|)$ with boundary points on right boundaries, but no boundary points on the left. The left boundary points make up the constant term and are part of lower dimensional contributions until finally, the term for $u = \emptyset$ contains just one grid point on $(0, \dots, 0)$.

The cost for one sparse grid which contains boundary points on the right, but none of them on the left, can be estimated similarly as in Lemma 2.1.2 where we dealt with all boundaries. We find, with the same reasoning as in Lemma 2.1.2,

$$|G_n^{d,\text{right}}| = \sum_{j=0}^d \binom{d}{j} |G_j^d| < |G_n^d| \sum_{j=0}^d \binom{d}{j} = 2^d |G_n^d|. \quad (4.93)$$

Furthermore, we know from (2.22) that

$$|G_n^d| = (-1)^d + 2^n \sum_{i=0}^{d-1} \binom{n+d-1}{i} (-2)^{d-1-i} = 2^n \frac{n^{d-1}}{(d-1)!} + \mathcal{O}(n^{d-2}). \quad (4.94)$$

By substituting both inequalities into (4.92) and using $\lfloor x \rfloor \leq x$, we obtain

$$\begin{aligned} |G^{n,\Gamma,a}| &\leq 1 + \left(\sum_{|u|>0} (-2)^{|u|} \right) + 2^n 2^{C_n^{(a)}|\bar{u}|} \\ &\cdot \left(\sum_{|u|>0} 2^{|u|(1-C_n^{(a)})} \cdot \left(\frac{\gamma_u}{\gamma_{\bar{u}}}\right)^{2/5} \cdot \sum_{i=0}^{|u|-1} \binom{n^{(a)}(\cdot) + |u| - 1}{i} (-2)^{|u|+1-i} \right) \end{aligned} \quad (4.95)$$

which fulfills the n asymptotics

$$|G^{n,\Gamma,a}| = \mathcal{O}\left(2^n \sum_u \left(\frac{\gamma_u}{\gamma_{\bar{u}}}\right)^{2/5} (n^{(a)}(n, \gamma, u, \bar{u}))^{|u|-1}\right) \quad (4.96)$$

for fixed d as stated. □

Lemma 4.2.5 (Global Error). *Let $V^{n,\Gamma,a}$ be the weighted sparse grid space defined in Lemma 4.2.1 and H_a^Γ with support for boundary seminorms as in (4.84) (i.e. plug (4.84) into the definition of $\|f\|_a$, (4.41)). Let $f \in H_a^\Gamma$ be a function with bounded second mixed derivatives in each anchor ANOVA component.*

Then, it holds for the interpolant $f^h \in V^{n,\Gamma,a}$ of f that

$$\begin{aligned} \|f - f^h\|_{L_a} &\leq 4 \cdot 2^{-2n} \cdot 2^{-2C_n^{(a)}|\bar{u}|} \cdot \|f\|_a \cdot \sum_{u \subseteq \{1,\dots,d\}} C_a(|u|) 2^{2C_n^{(a)}|u|} \\ &\cdot \gamma_u \cdot \left(\frac{\gamma_u}{\gamma_{\bar{u}}}\right)^{-4/5} \cdot A(|u|, n^{(a)}(n, \gamma, u, |u|)) \end{aligned} \quad (4.97)$$

4 Low Effective Dimensionality – A Dimension Adaptive Method

where $C_n^{(\infty)} = 11/5$, $C_n^{(2)} = (1 + 1/5 \text{ld} 9/2)$ are coefficients arising in $n^{(a)}$ and $C_\infty(m) = 2^m/4$, $C_2(m) = 2^m/6$ are the coefficients of boundary and inner regular sparse grid error estimates (see Lemma 4.2.3).

If we are only interested in the n -asymptotics, the error is of the order

$$\|f - f^h\|_{L_a} = \mathcal{O}(2^{-2n} \sum_{u \subseteq \{1, \dots, d\}} \gamma_u \left(\frac{\gamma_u}{\gamma_{\bar{u}}}\right)^{-4/5} \cdot (n + 2/5 \text{ld} \frac{\gamma_u}{\gamma_{\bar{u}}})^{|u|-1}). \quad (4.98)$$

Proof. We have

$$\|f - f^h\|_{L_a} = \left\| \sum_{u \subseteq \{1, \dots, d\}} (f_u - f_u^h) \right\|_{L_a} \leq \sum_u \|f_u - f_u^h\|_{L_a}. \quad (4.99)$$

Furthermore, by Lemma 4.2.3 and the relation for n^a of Lemma 4.2.4, we find

$$\begin{aligned} \|f_u - f_u^h\|_a &\leq C_a(|u|) \cdot 2^{-2n^{(a)}(n, \gamma, u, \bar{u})} \cdot |f_u|_{\mathbf{2}, a} \cdot A(|u|, n^{(a)}(n, \gamma, u, \bar{u})) \\ &\leq C_a(|u|) \cdot 2^{-2(n + \lfloor C_n^{(a)} \cdot (|\bar{u}| - |u|) + 2/5 \text{ld} \frac{\gamma_u}{\gamma_{\bar{u}}})} \cdot \gamma_u \|f\|_a \cdot A(|u|, n^{(a)}(\cdot)) \\ &\leq 4 \cdot C_a(|u|) \cdot 2^{-2n} \cdot 2^{-2C_n^{(a)} \cdot (|\bar{u}| - |u|)} \cdot \gamma_u \left(\frac{\gamma_u}{\gamma_{\bar{u}}}\right)^{-4/5} \cdot \|f\|_a \cdot A(|u|, n^{(a)}(\cdot)) \end{aligned}$$

where we employed $2^{-2\lfloor x \rfloor} \leq 4 \cdot 2^{-2x}$. Inserting the expression completes the proof. \square

4.2.3 Examples of Weighted Sparse Grid Spaces

Figure 4.9 shows a classical sparse grid for dimension $d = 3$ up to $n = 8$, i.e. the case $\gamma_{1,2,3} = 1$ and $\gamma_u = 0$ for $|u| < 3$. It shows level components l_1 , l_2 and l_3 and the subspaces W_l which make up the sparse grid. Here, levels with common scalar level $n(\mathbf{l})$ have the same color. Note that subspaces for $l_i = -1$ and $l_i = 0$ are inserted up to the same level as for inner points.

Figure 4.10 (left) shows the weighted sparse grid $G^{n, \Gamma, \infty}$ for $n = 8$ and $\Gamma = \{\gamma_2 = 8, \gamma_3 = 16, \gamma_{1,2} = 1, \gamma_{1,3} = 0.3, \gamma_{2,3} = 0.001\}$. It consists of all one- and two-dimensional components, but the highest order term has no degrees of freedom since $\gamma_{1,2,3} = 0$. Note that the component for $u = \{1\}$ is present up to the same resolution as the adjacent higher dimensional terms $\{1, 2\}$ and $\{1, 3\}$ due to the hanging node definition 2.1.3. Figure 4.10 (right) shows the weighted sparse grid $G^{n, \Gamma, \infty}$ for $\Gamma = \{\gamma_1 = 0.2, \gamma_2 = 1, \gamma_{1,2} = 0.01, \gamma_{2,3} = 0.1, \gamma_{1,2,3} = 0.001\}$. It contains few levels of the highest order term as well.

Product Weights

A special case of weighted spaces are those with product weights. We assume we are given non-negative parameters $\beta \geq 0$ and $\alpha_i \geq 0$, $i = 1, \dots, d$, and weights of the form

$$\gamma_u = \prod_{j \in u} \gamma_j, \quad \gamma_j := 2^{-5/2(\alpha_j + \beta)}. \quad (4.100)$$

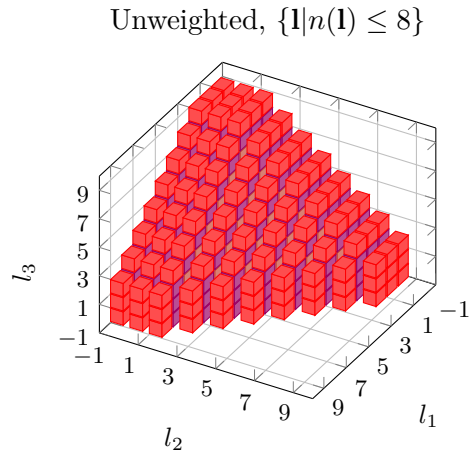


Figure 4.9: The subspace decomposition for a regular sparse grid including boundaries in three dimensions.

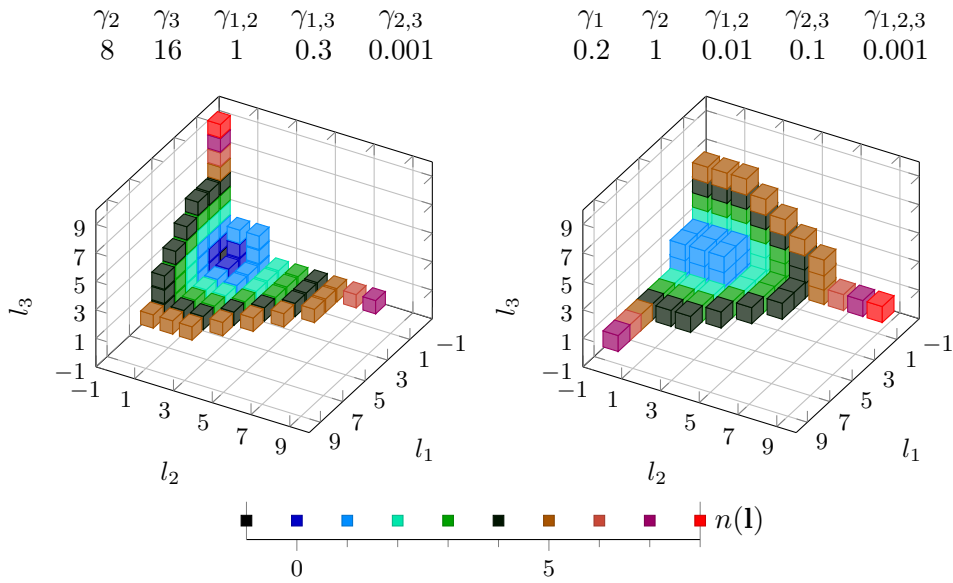


Figure 4.10: The subspace decompositions for two weighted sparse grids with respect to L_∞ using $n = 8$.

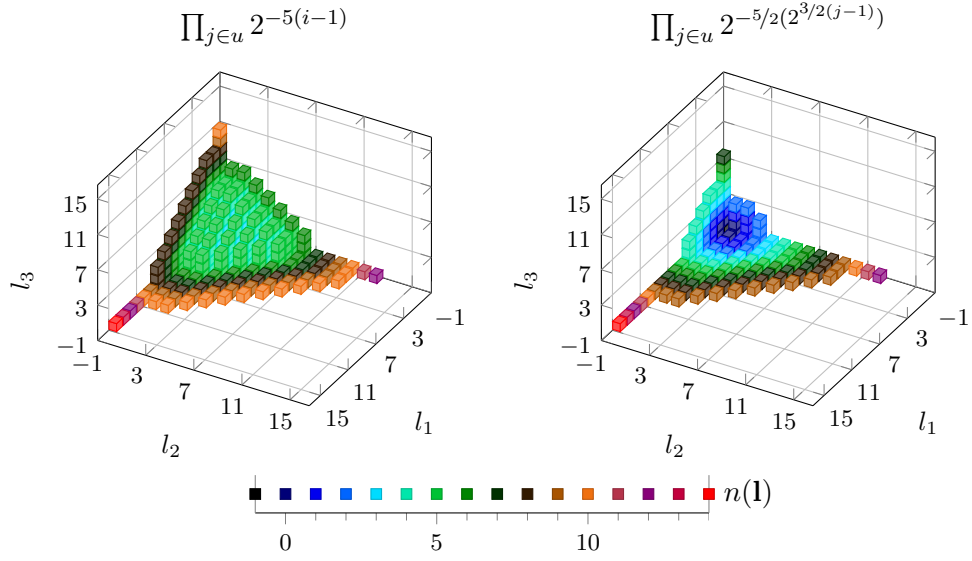


Figure 4.11: The sparse grid subspace decompositions for product weights once with linear exponent (left) and once with double-exponential decay (right) up to maximum weighted level $n = 14$.

Thus, the weights allow anisotropic parameters α_i and isotropic parameters β . The choice of prefactors in the exponent allows simplifications for our subspace selection criteria. For example, the criterion for L_∞ simplifies from

$$|\mathbf{l}_1^* + \frac{1}{5}|u(\mathbf{l})| - \frac{2}{5} \text{ld } \gamma_{u(\mathbf{l})} \leq n + 1 \frac{1}{5} |\bar{u}| - 1 - \frac{2}{5} \text{ld } \gamma_{\bar{u}} \quad (4.101)$$

to

$$|\mathbf{l}_1^* + |u(\mathbf{l})|(\frac{1}{5} + \beta) + \sum_{j \in u(\mathbf{l})} \alpha_j \leq n + \frac{1}{5} + \min_j \alpha_j + \beta \quad (4.102)$$

since $\bar{u} = \text{argmax}_{\{j\}, 1 \leq j \leq d} \gamma_{\{j\}}$.

Figure 4.11 (left) shows systematic decay relying on $\alpha_j = 2j$ and $\beta = -2$ for maximum level $n = 14$. Even faster decay is shown in Figure 4.11 (right) where double exponential order is established by $\alpha_j = 2^{3/2(j-1)}$ and $\beta = 0$, i.e.

$$\gamma_j = 2^{-5/2(2^{3/2(j-1)})}. \quad (4.103)$$

Here, the highest order component $f_{1,2,3}$ has no points for $n = 14$; there are about 10 levels less for increasing directions.

Energy Norm Optimized Weighted Grids

We conclude our example section with an example of energy norm optimized weighted grids according to Lemma 4.2.1. Figure 4.12 (left) shows the unweighted case of a three-dimensional energy optimized sparse grid. The subspace splitting has higher resolution

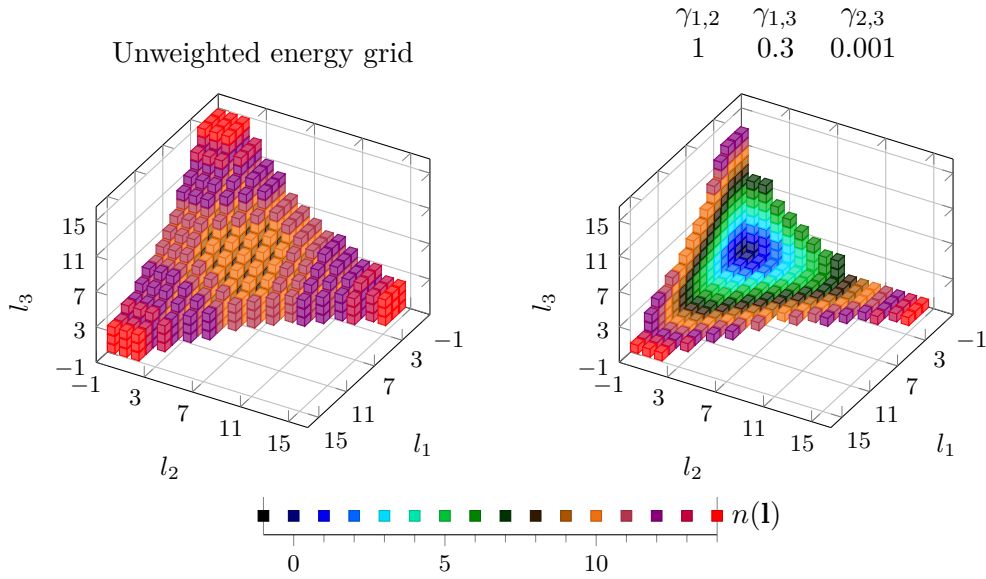


Figure 4.12: Energy optimal, unweighted sparse grid (left) and weighted energy optimal grid (right) for weights $\Gamma = \{\gamma_{1,2} = 1, \gamma_{1,3} = 0.3, \gamma_{2,3} = 0.001\}$, each on level $n = 14$.

near the axes and coarse resolution in the diagonals. For even larger levels, the diagonals are of considerably less importance than the axis, which eliminates the asymptotic log term n^{d-1} known for regular sparse grids. Figure 4.12 (right) shows the weighted subspace splitting $G^{n,\Gamma,E}$ with $n = 14$, $\Gamma = \{\gamma_{1,2} = 1, \gamma_{1,3} = 0.3, \gamma_{2,3} = 0.001\}$. It is a superposition of two-dimensional structures of different relative importance. Each one in turn exhibits the energy optimal subspace pattern with axis parallel dominance.

4.2.4 Summary of A Priori Grid Optimization

Our approach allows to generalize the method to derive spaces with optimal cost/benefit ratio introduced in [BG04] for regular sparse grids to the case of functions whose ANOVA terms are weighted in a known way. The derivation as such holds for the anchor ANOVA decomposition for which we provide boundary error estimates based on the closely related hierarchical hat basis. Note that we expect similar results for other types of ANOVA decompositions as well. For example, the classical ANOVA decomposition for the Lebesgue measure can be discretized in a natural way using the Neumann prewavelet as discussed in Section 4.1.3. This, in turn, is also a piecewise linear spline basis which spans the same space as the hat basis for the unweighted case. Therefore, it exhibits the same approximation properties.

The a priori grid optimization, together with the correct choice of basis functions, allows to work with ANOVA decompositions and their components in a cost-effective way. From an algorithmic point of view, one only needs to determine the required set

of levels and apply the hierarchical transformations to compute a discretized ANOVA decomposition (see Section 4.1.3). The required set of levels consists of all levels up to some maximum value $n^{(a)}(n, \gamma, u, \bar{u})$ which depends mainly on the prescribed work count n , the weights and the ANOVA component index u . It can be built successively by starting with a coarse level $n(\mathbf{1}) = 1$, inserting further level components until finally the criterion stops the procedure.

4.3 A Posteriori Adaptive Sparse Grids in Weighted Spaces

Having seen how sparse grid spaces can be constructed based on a priori knowledge, we will now address the topic of a posteriori dimension adaptive grid refinement. Thus, we assume a different relative importance between different (sets of) directions. Adaptive refinement methods are quite common for irregular domains or to resolve singularities and we have already seen examples of this sort of adaptivity for adaptive interpolation or approximation in Section 2.2.1. What we intend here has a different quality: our goal is to reduce cost for coupled additive interactions among the input variables, not to compensate missing smoothness. Based on approaches taken in the literature, we build a new, improved dimension adaptive tool which is explicitly based on ANOVA techniques. The basics of the approach go back to an adaptive interpolation method presented by [Heg03]. The method has been elaborated and applied to problems of high dimensional integration in [GG03] and for machine learning in [Gar04]. The key idea is to work with complete subspaces $W_{\mathbf{l}}$ for multi-indices \mathbf{l} , just as we did for our a priori optimization: if (elements of) a subspaces $W_{\mathbf{l}}$ contribute a great part to the target quantity, all its basis functions are inserted into the finite basis. Such a sort of block-adaptivity is described by the term dimension-adaptive: only the directions contributing to $W_{\mathbf{l}}$ (thus, all $l_j > -1$) are relevant, up to levels l_j . The term space adaptivity may include the same, but the emphasize of space adaptivity is the ability to employ only parts of $W_{\mathbf{l}}$ if needed.

Dimension adaptive refinement will usually start with a coarse resolution in space and few directions. Then, some sort of error estimation (or indication) chooses important subspaces iteratively. Here, the error estimation can involve information of complete subspaces, not just single grid points. The methods described for integration in [GG03], approximation in [Heg03] and function reconstruction in [Gar04] typically allow full grid methods applied to single subspaces which are finally combined by specific additive recombination rules.

In the following we analyze the common dimension adaptive procedure. We formulate error indicators and discuss the refinement strategy. Furthermore, we propose a new variant based on new admissibility (“hanging node”) conditions in order to improve the reliability of the approach in the context of dimension adaptive approximation. We present examples of adaptive function interpolation and dimension adaptive solution of partial differential equations. Furthermore, we compare approaches based on the anchor ANOVA with those for the classical ANOVA decomposition.

4.3.1 ANOVA–based Dimension Adaptive Refinement

Any adaptive refinement procedure, be it space adaptive or dimension adaptive, needs a criterion to decide whether the global accuracy is acceptable (a *termination condition*), a method to decide *where* to refine the index set (local error estimator or -indicator) and strategies *how* to refine the index set in the neighborhood of high local errors. The particular case of dimension adaptivity also needs a description how relations between different dimensions shall be formalized (the *type* of dimension decomposition) and specialized data structures and adopted formal admissibility criteria. While the admissibility of a space adaptive grid is usually merely a technical condition to avoid hanging nodes, dimension adaptive grid structures require admissibility conditions to control the interaction between different sets of directions. Can we insert a new set of directions although its lower dimensional “adjacent” neighbors do not contribute to the representation? The answer has impact beyond merely technical issues and is part of our consideration.

To realize a dimension adaptive approximation tool, we propose to use the space adaptive error indication of Section 2.2.1 combined with a discretized ANOVA decomposition as in Section 4.1.3 and an admissibility criterion. Choosing the hierarchical hat basis with constant on level -1 immediately discretizes the anchor ANOVA decomposition with anchor $a = 0$ whereas the Neumann prewavelet basis yields the integral based (Lebesgue) ANOVA, see Section 4.1.3 for details. The error indicators of Section 2.2.1 use weighted basis coefficients $|f_{\mathbf{l},\mathbf{i}}| |\phi_{\mathbf{l},\mathbf{i}}|$ to identify regions of large local variation. Since every multi-index (\mathbf{l}, \mathbf{i}) is uniquely associated with its ANOVA component $u(\mathbf{l}) = \{j \mid l_j \neq -1\}$, we can easily implement (space) adaptive refinement inside of single ANOVA components: simply refine a node with large error indicator by inserting its sons belonging to the same ANOVA component (i.e. sons in directions $j \in u(\mathbf{l}) \Leftrightarrow l_j \neq -1$). Complementing such an approach with an admissibility criterion which controls the interrelation between ANOVA components thus yields a space- and dimension adaptive refinement procedure. One refinement step consists of the parts:

1. Determine the set of indices I_{check} which should be considered for refinement according to the local error indicator. In our case, this is

$$I_{\text{check}} = \{(\mathbf{l}, \mathbf{i}) \in G \mid |f_{\mathbf{l},\mathbf{i}}| |\phi_{\mathbf{l},\mathbf{i}}| \geq \epsilon, \text{ at least one of the } 2d \text{ sons of } (\mathbf{l}, \mathbf{i}) \text{ is not in } G\} \quad (4.104)$$

with obvious modifications for relative errors or look-ahead strategies as in Section 2.2.1.

2. For every element $(\mathbf{l}, \mathbf{i}) \in I_{\text{check}}$ and every direction $m = 1, \dots, d$, check whether the insertion of the two sons in direction m , $(l_m + 1, 2i_m \pm 1)$, is admissible. If so, insert them.
3. Update the approximant f on the new grid (to recompute error indicators).

Steps (1.) – (3.) are iterated until the termination condition is satisfied. Possible termination conditions are of wavelet compression type (no further point has been inserted) as in [Gri98] or some indication involving a global error estimate ([GG03, Gar04]).

Note that the dimension adaptive method employed in [GG03] and [Gar04] uses a slightly different approach for (1.): they compute error indicators for whole subspaces W_1 and the set of refinement candidates I_{check} is the refineable subspace with *largest* error indicator (and all its associated basis functions). Choosing just the largest contribution yields a depth–first search through the hierarchical search space: the largest error is considered first, refined, then possibly refined again and so on. Our approach (4.104) yields a breadth–first search since it refines a complete region of large local errors simultaneously. Provided we follow the wavelet–compression approach which inserts all local contributions of magnitude larger than ϵ , the outcome of both ways is the same – only the number of loop iterations varies. We choose the breadth–first–search strategy here since the update–step (3.) may need to recompute every value $f_{1,i}$ after a refinement, including already existing grid points. For example, this is necessary for the Lebesgue ANOVA or for the solution of a PDE. In such a case, the reduced number of iterations of the breadth–first–search approach appears to be more adequate (note that both, [GG03] and [Gar04] have incremental update rules for step (3.)).

Thus, we are left with a choice of admissibility condition (2.). The admissibility criterion for pure space adaptive sparse grid refinement is simple: we insert every candidate along with all its hierarchical ancestors (typically one per direction followed by a recursion). The grid is thus *made* admissible.

The dimension adaptive procedures in [GG03] and [Gar04] are characterized by a different condition to reduce the number of candidates: any new node must be inserted *after* all its ancestors have been inserted into the index set. Even more: a new node is inserted only if all its backward neighbors (direct fathers) have already been checked to be relevant and exist in the index set (formulated by means of an active index set). This strict checking policy ensures a systematic additive grid. It has the advantage that higher order components are not directly inserted as for the simple space adaptive policy. However, it turns out to be too restrictive: for the approximation problem, any nonzero function with vanishing low order ANOVA components will cause the procedure to terminate with an error of 100%. For example, using the anchor ANOVA decomposition and its corresponding hierarchical hat basis to resolve $f(x, y) = x^2 \cdot y^2$ adaptively will fail, no matter which initial level is chosen. This is because

$$f = \underbrace{f_0}_{=0} + \underbrace{f_{\{1\}}}_{=0} + \underbrace{f_{\{2\}}}_{=0} + f_{\{1,2\}} \quad (4.105)$$

has vanishing low order terms. The failure is immediately clear if the coarsest level just contains the constant since in such a case, the error indicator fails in step (1.). But the failure occurs even if we start on a larger level. This is illustrated in Figure 4.13. The first picture shows the initial grid. In this case, we choose an initial resolution of $n_0 = 5$ and insert all subspaces of a regular sparse grid $n(\mathbf{l}) \leq 5$ using $n(\mathbf{l})$ as in (2.18) (note that the boundary indices $-1, 0$ also have level 5 in their respective lower dimensional manifold). The refinement correctly inserts points in the upper right half, but the tails with $l_2 > 5$ and $l_1 > 5$ are not admissible since $f_{-1,-1} = f_{-1,l_2} = f_{l_1,-1} = 0$ for every l_1, l_2 (indicated by color in Figure 4.13). However, the correct grid would include these tails as well.

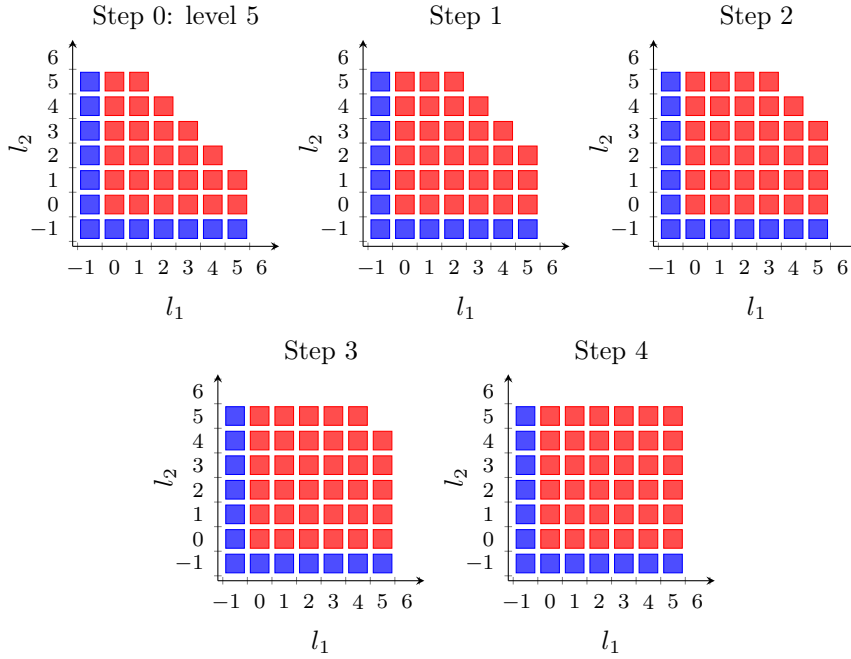


Figure 4.13: The subspace sequence generated by the strict checking heuristics applied to $f(x, y) = x^2 \cdot y^2$ starting with a regular sparse grid on level 5. Here, ■ indicates a vanishing contribution $f_l \equiv 0$ whereas ■ indicates a non-vanishing contribution $f_l \neq 0$.

This effect is *not* caused by the error indication since subspaces of higher level have been identified to be relevant. It is caused by the fact that subspaces of high levels cannot be refined since their ancestors on lowest ANOVA order are irrelevant. Thus, we need to improve the procedure in order to get a dimension adaptive approach for approximation problems.

We propose a new refinement variant to improve the early termination issue. It should be somewhere between the strict checking policy and the no-check policy used by pure space adaptivity. Our idea is that insertion of child nodes should *always* be admissible provided the child and the father are in the same ANOVA component. As long as this is the case, any required ancestors on lower levels should be inserted to avoid hanging nodes, just as for the no-check policy (this includes ancestors on lower dimensional ANOVA components). Thus, inside of an ANOVA component, the new strategy yields the same result as the pure space adaptive condition: if the error in an ANOVA component is large, refine the grid associated to this particular component. But we propose to complement it with an even more restrictive approach when it comes to the insertion of children belonging to *higher* dimensional ANOVA components: such children are never admissible. The advantage is immediately clear: once at least one point exists for an ANOVA component, it will be refined according to the well-known space adaptive procedures. Furthermore, the strategy maintains an existing compact low order repre-

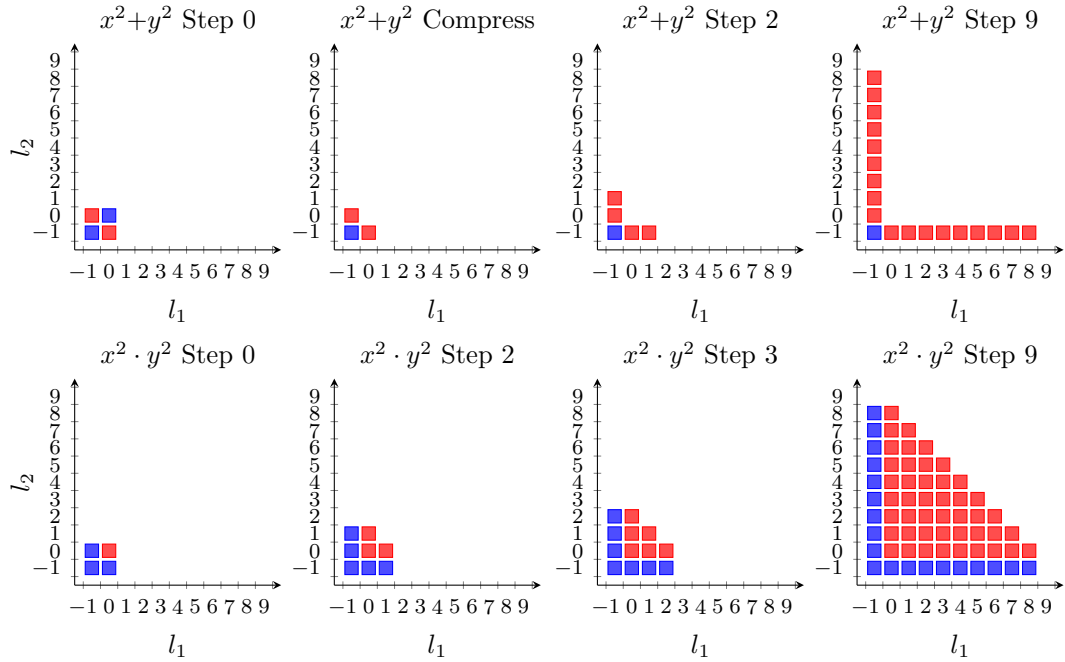


Figure 4.14: Dimension adaptive refinement with the new ANOVA based admissibility criterion for $f(x, y) = x^2 + y^2$ (top row) and $f(x, y) = x^2 \cdot y^2$ (bottom row). Here, \blacksquare indicates a vanishing contribution $f_l \equiv 0$ whereas \blacksquare indicates a non-vanishing contribution $f_l \neq 0$.

sentation without inserting as much points as the purely space adaptive condition. This is illustrated in Figure 4.14 (top) and Figure 4.14 (bottom): the top pictures show the subspace refinement procedure for a sum of two one-dimensional functions. The cost is essentially two times the cost for the one-dimensional contributions. The bottom pictures show the same function as in Figure 4.13, $f(x, y) = x^2 \cdot y^2$, this time started on level $n_0 = 0$. It refines as it ought to since the low order components are inserted by the admissibility criterion. Note that the initial grid uses $n_0 = 0$ (consisting of four subspaces, each with one basis function) in both cases. This is a typical property of the new heuristics: it requires an initial guess, an upper bound, on the largest allowed size for sets of directions, $|u|$, combined (as usual) with an initial level for each of these subsets. We propose to provide the initial grid at level n_0 up to ANOVA components of some prescribed dimension $q \leq d$. The first refinement step should then *compress* the grid, i.e. it should use the local error indicators to decide which of the initial components are non-zero. This *boolean check* is possible on a coarse resolution; its purpose is just to check for $f_u \equiv 0$ or $f_u \neq 0$. It is visible on Figure 4.14 as well: the compress step for Figure 4.14 (top) removes the subspace $\mathbf{l} = (0, 0)$ belonging to $f_{\{1,2\}}$ and the additive refinement continues afterwards. The choice $q = d$ allows fully automatic selection of admissible ANOVA components (using the initial resolution n_0). Note that $n_0 = 0$ requires already 2^q points, $n_0 = 1$ requires 3^q points. The case $n_0 > 1$ with $q = d$ is

just a normal sparse grid with boundary points and $q < d$ yields a set union of $\binom{d}{q}$ many sparse grids of dimensionality q .

Note that the initial compression step at level n_0 can only be avoided by a priori knowledge about the admissible ANOVA components. Even knowledge of the *exact* low order ANOVA terms does not indicate whether higher orders vanish or not. We need at least level 0 to get information regarding higher order terms. For example, if we know f_\emptyset in a splitting $f(x) = f_\emptyset + f_{\{1\}}(x)$ into constant and rest, we know nothing about $f_{\{1\}}$; we need at least some sort of gradient to decide whether it is constant or not. But the gradient is related to hierarchical coefficients at level 0 (the linear basis function)! If we investigate also level 1, we get information about second derivatives due to Lemma 2.1.5. This motivates why a fully automatic approach (without a priori indication) needs at least level $n_0 = 0$ to decide whether higher order components vanish.

Algorithm 12 Dimension (and space-) adaptive Approximation

Input: Initial Grid $G^{(0)}$ containing a priori guesses about required ANOVA components of initial (coarse) resolution

Input: Adaptive threshold $\epsilon = \epsilon^{(\text{abs})}$ or $\epsilon = \epsilon^{(\text{rel})}$

Input: A “set values” algorithm (can be interpolation or a PDE)

Output: a grid G and the approximated function f on G

- 1: $i = 0$
 - 2: $f^{(0)} := \text{set values on } G^{(0)}$
 - 3: $G^{(1)} := \text{compress } G^{(0)}$ based on local error indicators of $f^{(0)}$
 - 4: $i := i + 1$
 - 5: **repeat**
 - 6: $f^{(i)} := \text{set values on } G^{(i)}$
 - 7: $G^{(i+1)} := \text{refine } G^{(i)}$ based on local error indicators of $f^{(i)}$:
 Insert child nodes if and only if their father had a large error contribution and they are in the same ANOVA component of their father
 - 8: $i := i + 1$
 - 9: **until** $G_{\text{result}}^{(i)} \setminus G_{\text{result}}^{(i-1)} = \emptyset$
 - 10: **return** $G := G^{(i)}$ and $f := f^{(i)}$
-

We summarize the resulting dimension adaptive (or even space- and dimension adaptive) procedure in Algorithm 12. On input, we expect an initial grid G^0 , the target threshold ϵ and a “set values” algorithm (for interpolation, this is just a hierarchical transformation to the desired basis). So far, the input is the same as for the space adaptive Algorithm 1 on page 33, with the following specialities: first, the grid’s anchor should be the point associated with the lowest order ANOVA component, the constant. Here, the grid’s anchor refers to the point where all adaptive algorithms start (see Appendix A.1 for details). The grid’s anchor and all its ancestors constitute the *minimal* grid size in order to fulfill hanging node conditions. Note that pure space refinement (Algorithm 1) usually employs the middle point $(\mathbf{1}, \mathbf{i}) = (1, \dots, 1 | 1, \dots, 1)$ as anchor since

such a structure allows to discard all boundary nodes in favor of inner nodes³. For dimension adaptive refinement, the root node (anchor) is necessarily the node associated with the constant: $(\mathbf{l}, \mathbf{i}) = (-1, \dots, -1 | 0, \dots, 0)$. The minimal grid size is thus 1 (the constant) since there is no ancestor to insert. A further speciality of the input data is that the “set values” algorithm needs to provide a particular basis. The basis fixes the type of ANOVA decomposition as elaborated in Section 4.1.3, for example the hat basis with constant on level -1 as discretized anchor ANOVA or the Neumann prewavelet as discretized Lebesgue ANOVA.

The initial step of Algorithm 12 is to determine all non-vanishing ANOVA components $f_u \neq 0$. The compression as such is nothing special, it can be realized as Algorithm 26 in Appendix A.4. However, it has impact on the outcome: only the remaining ANOVA components will be considered for adaptive refinement, all discarded once remain discarded during Algorithm 12. Consequently, G^0 should be chosen carefully, for example as a regular, d -dimensional sparse grid of small, but not too small, level (say, $n_0 = 2$) unless a priori knowledge allows to reduce the initial (effective) dimension.

The remaining refinement loop is almost the same as for pure space adaptivity (compare Algorithm 1). The only difference is the insertion of child nodes: once a node has been found to have large local contribution $|f_{\mathbf{l}, \mathbf{i}}| \|\phi_{\mathbf{l}, \mathbf{i}}\| \geq \epsilon$, we insert all $2 \cdot |u(\mathbf{l})|$ child nodes (2 per direction $j \in u(\mathbf{l})$) which belong to the same ANOVA component $u(\mathbf{l})$. This decision is algorithmically simple: we insert child nodes in direction j if and only if $l_j \neq -1$. This ensures that we keep the ANOVA structure determined in line 3. Note that insertion of child nodes involves insertion of all ancestors (recursively) in order to maintain the grid structural assumptions (to maintain a tree). Details on the single step of line 7 can be found in Algorithm 25 in Appendix A.4.

The algorithm may be modified with look-ahead strategies as discussed for Algorithm 1 in a straight-forward way. After all, it constitutes not much more than a classical sparse grid space adaptive procedure applied to preselected ANOVA components and a method to select these components automatically, combined with an admissibility criterion based on ANOVA theory. The last iteration could be improved by adding a final compression step: the error indicator stops refinement when it finds only irrelevant points. These points could be removed.

4.3.2 Coupled Space- and Dimension Adaptive Refinement

As already hinted, Algorithm 12 supports both, space- and dimension adaptive refinement although our focus is on the peculiarities of dimension adaptive refinement so far. Now, suppose we have a locally unsmooth function as input for Algorithm 12. Suppose further that the “set values” algorithm supports adaptivity. Then, the initial grid compression can be applied just as for a space adaptive grid – only the tree’s root needs to be respected. Since the outcome of the grid compression in line 3 is used as boolean decision “ $f_u \equiv 0$ ” or “ $f_u \neq 0$ ” anyway, the space adaptivity does not hurt here. Furthermore, since we choose the same local error indicator as for space adaptive refinement,

³Without boundary nodes, the minimal grid size is thus 1. With boundary nodes, the minimal grid size is 3^d .

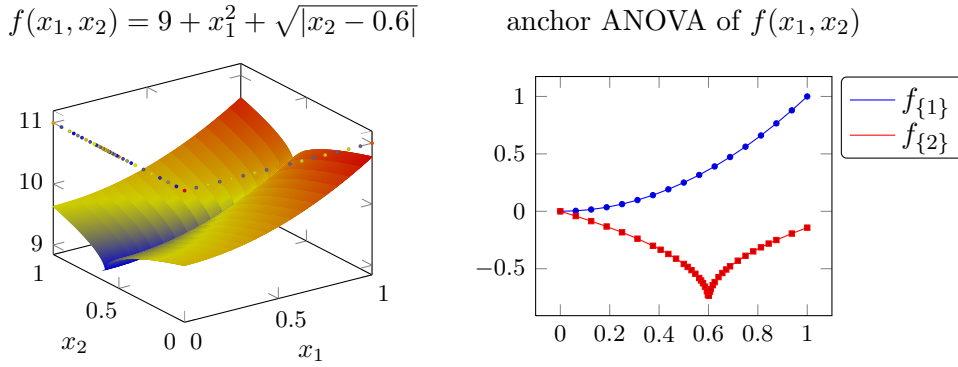


Figure 4.15: Space- and dimension adaptive resolution of $f(x_1, x_2) = 9 + x_1^2 + \sqrt{|x_2 - 0.6|}$ and the resulting grid (left), together with its anchor ANOVA components $f_{\{1\}}$ and $f_{\{2\}}$ and the respective parts of the grid (right). The component $f_{\{1,2\}}$ vanishes.

$|f_{\mathbf{l}, \mathbf{i}}| \|\phi_{\mathbf{l}, \mathbf{i}}\| \geq \epsilon$, we can also safely perform space adaptive refinement in line 7 – as long as we respect the fixed ANOVA structure and insert child nodes only in existing ANOVA components.

The distinction whether we allow space adaptivity or use just dimension adaptive procedures is now merely a technical issue: space adaptive refinement requires a different data structure than the simpler dimension adaptivity. Consequently, space adaptive refinement allows effective representations of irregular functions, but it is slightly slower when applied to a smooth function. Purely dimension adaptive data structures allow fast representation of smooth functions of low effective dimension, but are inefficient for irregular functions of low effective dimension. A purely dimension adaptive approach will insert the complete subspace $W_{\mathbf{l}}$ once at least one large contribution $(\mathbf{l}, \mathbf{i}) \in W_{\mathbf{l}}$ has been identified. Variations of this strategy which include the cost $|W_{\mathbf{l}}|$ have been discussed in [GG03]. Here, we realize pure dimension adaptive refinement by insertion of complete subspaces if at least one element of a subspace is relevant according to local error estimation.

An example of space- and dimension adaptive refinement is shown in Figure 4.15, which contains the graph of $f(x, y) = 9 + x^2 + \sqrt{|y - 0.6|}$ and its adaptive grid (left) determined with $\|\cdot\| = \|\cdot\|_{L_2}$, $\epsilon = 10^{-3}$ and the hierarchical hat basis. We see that only grid points on the axes are used. Consequently, $(l_1, l_2) > (-1, -1) \Rightarrow f_{l_1, l_2} = 0 \Rightarrow f_{\{1,2\}} \equiv 0$. The two one-dimensional components $f_{\{1\}}^h$ and $f_{\{2\}}^h$ are shown in Figure 4.15 (right), together with their grid points. They resemble $f_{\{1\}} = x^2$ and $f_{\{2\}} = \sqrt{|y - 0.6|}$, respectively. We see that $f_{\{1\}}^h$ has been discretized by means of regular grids whereas $f_{\{2\}}^h$ has an adaptive grid as expected. Note that these grids are nothing but the grid slices $x_2 = 0$ (for $f_{\{1\}}$) and $x_1 = 0$ (for $f_{\{2\}}$) of Figure 4.15 (left).

4.4 On Non–Linearly Weighted Axis Parallel Approaches

We discuss generalizations of ANOVA decompositions where a function takes the role of the constant. Instead of a one–dimensional splitting into constant and rest, $\mathbf{1} \oplus W$, as for the standard ANOVA construction (4.3), we consider splittings $V = \text{span } g \oplus W$ where $g(x): [0, 1] \rightarrow \mathbb{R}$ characterizes important features of V . The idea is to incorporate knowledge about the target function f into account and apply the same tensor product construction as in Section 4.1.1: if we know that f can be approximated by a rank–1 representation, $f(x) \approx \prod_{j=1}^d g_j(x_j)$, (compare [MB05]), we can use the $g_j(x_j)$ to derive a generalized ANOVA which consists of $\prod g_j$ as lowest order term (“constant”) and rest terms, defined by subsets of $\{1, \dots, d\}$ as before.

A decomposition of this type is applied successfully in [GH10] for applications of computational chemistry: if $\prod g_j$ already covers most of f and the rest terms have low effective dimension, the method allows to reduce the cost considerably, comparable to the classic ANOVA.

We sketch such a non–linear product ANOVA decomposition in the following. As in Section 4.1.1, we define a decomposition by means of one–dimensional projectors and a tensor product. Given a set of non–vanishing weighting functions $g_j: [0, 1] \rightarrow \mathbb{R}$, $j = 1, \dots, d$, a possible set of projectors is given by the orthogonal approach

$$P_j^{(1)} f(x) := \frac{(f, g_j)}{(g_j, g_j)} \cdot g_j(x) \quad (4.106)$$

for an inner product (\cdot, \cdot) . Due to the normalization, we find the projector property

$$P_j^{(1)} [P_j^{(1)} f] = \frac{\left(\frac{(f, g_j)}{(g_j, g_j)} g_j, g_j\right)}{(g_j, g_j)} g_j(x) = P_j^{(1)} f \quad (4.107)$$

and the associated rest projector $(I - P_j^{(1)})f$ to define $V = \text{span } g_j \oplus W_j$. Another possible projector can be obtained by means of integration,

$$P_j^{(1)} f := \int f(x) d\mu_j(x) \cdot \frac{g_j(x)}{\int g_j d\mu_j(x)}. \quad (4.108)$$

In general, any projector of type $P_j f(x) = g_j(x) P_j^* f$ where P_j^* is a linear projector which maps to a constant is feasible, provided it satisfies the normalization property $P_j^*[g_j] = 1$ and we get a one–dimensional splitting.

Since projectors of this sort operate only on one direction, they can be combined by means of the tensor product identity (4.15), resulting in

$$I^{(d)} = \sum_{u \subseteq \{1, \dots, d\}} P_u \quad (4.109)$$

with $P_u = \left(\prod_{k \in \{1, \dots, d\} \setminus u} P_k\right) \cdot \left(\prod_{j \in u} (I_j - P_j)\right)$. Consequently, any function can be represented by $f(x) = \sum_u f_u(x)$ with

$$f_u(x) = \sum_{v \subseteq u} (-1)^{|u|-|v|} \left(\prod_{j \in \{1, \dots, d\} \setminus v} P_j f(x) \right), \quad (4.110)$$

4.4 On Non-Linearly Weighted Axis Parallel Approaches

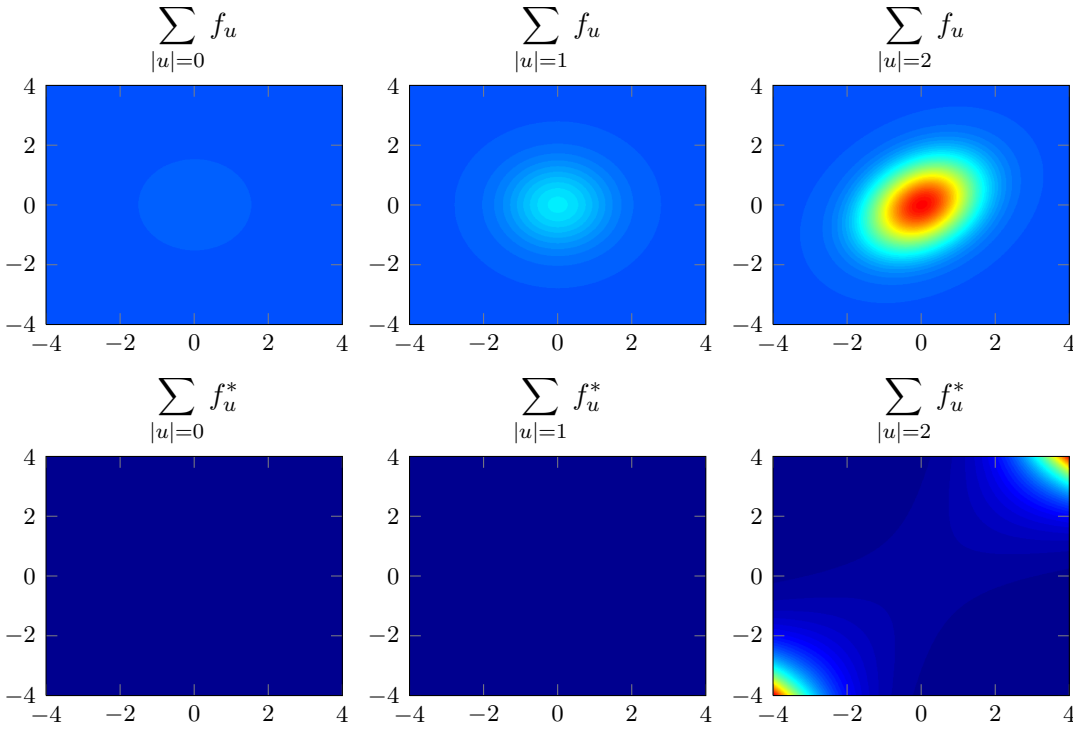


Figure 4.16: The factorized decomposition (4.112) applied to a Gaussian with $\mu = (0, 0)^T$ and $\sigma = \begin{bmatrix} 1 & 0.3 \\ 0.3 & 1 \end{bmatrix}$ where $g_j = \mathcal{N}(0, 1)$. The top row shows $\sum_{|u|=i} f_u$ for $i = 0$ (left), $i = 1$ (middle) and $i = 2$ (right); the bottom row shows $\sum_{|u|=i} f_u^*$ (i.e. without $g(x)$).

where f_u now depends on all components of x , not just x_u , compare (4.19). Provided the weights do not vanish at x , we can factorize f_u using

$$f_u(x) = g(x) \cdot \sum_{v \subseteq u} (-1)^{|u|-|v|} \frac{1}{g(v)(x)} \cdot \left(\prod_{j \in \{1, \dots, d\} \setminus v} P_j^* f \right)(x) =: g(x) \cdot f_u^*(x) \quad (4.111)$$

and we find

$$f = g \cdot \sum_{u \subseteq \{1, \dots, d\}} f_u^*. \quad (4.112)$$

Thus, the splitting is useful if $\sum_u f_u^*$ decays rapidly with $|u|$.

Consequently, any function with dominating rank-1 representation $\prod g_j$ and low dimensional rest terms f_u^* allows effective representations, even if the classical ANOVA is high dimensional. The rank-1 representation needs to be determined either from a priori knowledge or by means of a non-linear optimization (see [MB05]).

Note that the Gaussian with diagonal covariance studied in Section 3.2 yields $f_u^* \equiv 0$ for $|u| > 0$ and $f_\emptyset^* = 1$ since it resembles its rank-1 approximation.

Table 4.1: Decomposition errors of (4.112) for the case $d = 2$ (left) and $d = 3$ (right). The case $d = 2$ corresponds to the experiment of Figure 4.16 whereas the case $d = 3$ uses the three-dimensional Gaussian (4.114).

$ u \leq *$	L_2/r	L_∞/r	$ u \leq *$	L_2/r	L_∞/r
0	0.98	0.99	0	1	1
1	0.77	0.78	1	0.96	0.97
2	0	0	2	0.71	0.71
			3	0	0

Decompositions of type (4.112) exist as long as the projector can be applied and the g_j do not vanish. However, it may require the highest order term $f_{\{1, \dots, d\}}^*$. For example, a Gaussian with non-diagonal covariance requires $f_{\{1, \dots, d\}}^*$: the diagonal parts make up the rank-1 representation and the non-diagonal entries are represented by $f_{\{1, \dots, d\}}^*$. This can be seen in Figure 4.16 for the case $d = 2$ and the Gaussian $\mathcal{N}(\mu, \sigma)$ with $\mu = (0, 0)^T$ and $\sigma = \begin{bmatrix} 1 & 0.3 \\ 0.3 & 1 \end{bmatrix}$: the top row shows $\sum_{|u|=i} f_u(x)$ for $i = 0, 1, 2$ whereas the bottom row shows the factorial representation $\sum_{|u|=i} f_u^*$. The components have been computed by a brute force implementation of (4.111). Clearly, the highest order term for $i = 2$ dominates the representation. This is quantified in Table 4.1 which shows the low order error $\|f - f^{(i)}\|$ for

$$f^{(i)} := \sum_{|u| \leq i} f_u, \quad i = 0, 1, 2, \dots, d. \quad (4.113)$$

Besides the two-dimensional case, it also contains results for $d = 3$, this time with

$$\sigma = \begin{bmatrix} 1 & 0.3 & 0.5 \\ 0.3 & 1 & 0 \\ 0.5 & 0 & 1 \end{bmatrix}. \quad (4.114)$$

For both, $d = 2$ and $d = 3$, the error⁴ is of order $\mathcal{O}(1)$ for $i < d$.

We conclude that even the rank-1 nonlinear ANOVA leads to expensive approximations of the Gaussian, and probably for more difficult density functions as well.

4.5 Numerical Experiments in High Dimensions

We will now verify the new dimension adaptive approximation tool on a couple of model problems with inherently low-dimensional structure. We cover the case of dimension adaptive interpolation procedures and dimension adaptive solution of partial differential equations.

⁴The error has been computed on a grid, without measuring the interpolation error.

4.5.1 Remarks on Error Estimation and Pointwise Operations

We use the same error estimation routines as in Section 3.2.2, that means either evaluation on a sufficiently finer grid or using the absolute precision tensor product quadrature (3.58) if possible. Both methods provide reasonable accuracy for L_2 (or energy norm) evaluations.

Note that evaluation of L_∞ errors should be done on high dimensional grids or not at all: the basis points of dimension adaptive grids are usually unsuitable to approximate pointwise operations. For example, $f(x, y) = x + y$ can be represented exactly using the two-point dimension adaptive set of levels $\{(0, -1|0, 0), (-1, 0|0, 0)\}$ which constitutes level 0 of the two ANOVA components $u = \{1\}$ (since $l_1 \neq -1$ for the first point) and $u = \{2\}$ (since $l_2 \neq -1$ for the second point). However, we find

$$\max\{|f(1, 0)|, |f(0, 1)|\} = 1 \neq \|f\|_{L_\infty} = |f(1, 1)| = 2. \quad (4.115)$$

Any pointwise operation might need high-dimensional grids and may not benefit from dimension adaptive compression. This should be kept in mind when working with dimension adaptive representations together with L_∞ norms or the function algebra of Section 2.3. Consequently, we prefer the L_2 norm for error estimation.

4.5.2 Dimension Adaptive Interpolation

We start with examples of functions for which higher order ANOVA components vanish, i.e. $|u| > q \Rightarrow f_u \equiv 0$. Our model problem is a superposition of tensor products involving up to q factors,

$$f(x) = \sum_{\substack{u \subseteq \{1, \dots, d\} \\ |u|=q}} \prod_{j \in u} g(x_j), \quad (4.116)$$

using the single univariate function $g(x) = B(\alpha, \beta)^{-1} x^{\alpha-1} (1-x)^{\beta-1}$ with $\alpha = 2$ and $\beta = 5$.

What we expect is a method which is comparable to a smooth q -dimensional one with respect to its accuracy and the cost of $\mathcal{O}\binom{d}{q} \cdot N_q$ where N_q is the cost for one smooth q -dimensional problem.

The dimension adaptive procedure starts without a priori knowledge by inserting *all* points on level $n_0 = 1$, compresses these 3^d points as described in Algorithm 12 and continues with the identified non-vanishing ANOVA components. It employs relative thresholds $\epsilon = \epsilon^{(\text{rel})} \cdot \|f_h^{(i)}\|_{L_2}$ during each refinement step and computes $\|f_h^{(i)}\|_{L_2}$ on the i th grid⁵. We use a purely dimension adaptive approach by inserting full subspaces W_1 to resolve large local errors until finally all local errors larger than the prescribed threshold have been found. The final degrees of freedom N are considered as cost measure and the relative L_2 error as gain measure. Note that the runtime complexity is $\mathcal{O}(3^d + N)$ for a smooth problem since the refinement loop of Algorithm 12 applies first its compression of

⁵Note that L_2 norm evaluations in the semi-orthogonal Neumann prewavelet basis is considerably faster than for the hat basis. We switched basis representations for this task, similarly for L_2 error measurement.

4 Low Effective Dimensionality – A Dimension Adaptive Method

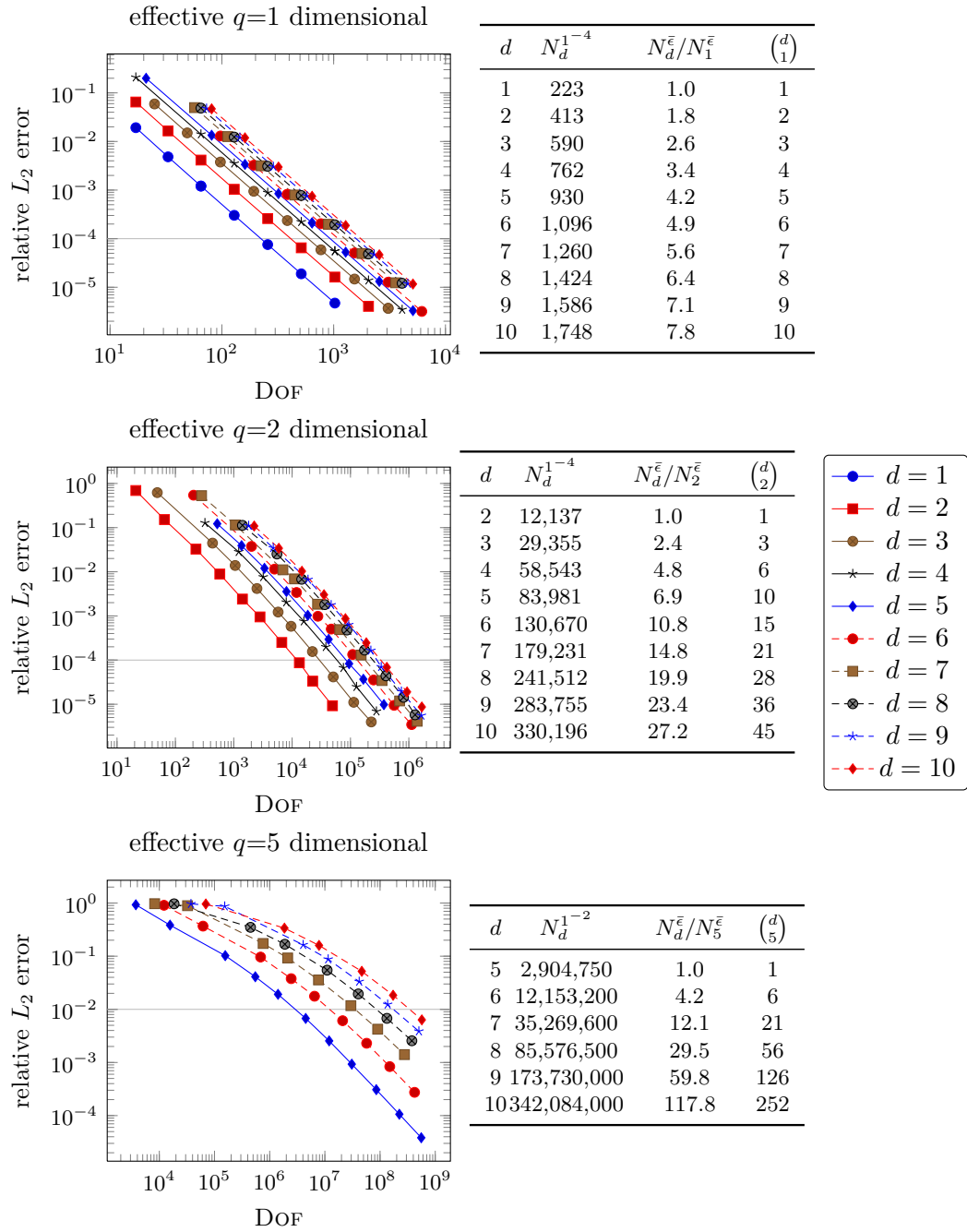


Figure 4.17: Dimension adaptive interpolation results for additive superposition models (4.116) of order $q = 1, 2, 5$ and the respective cost increase factors for a fixed error of $\bar{\epsilon}$ (right column).

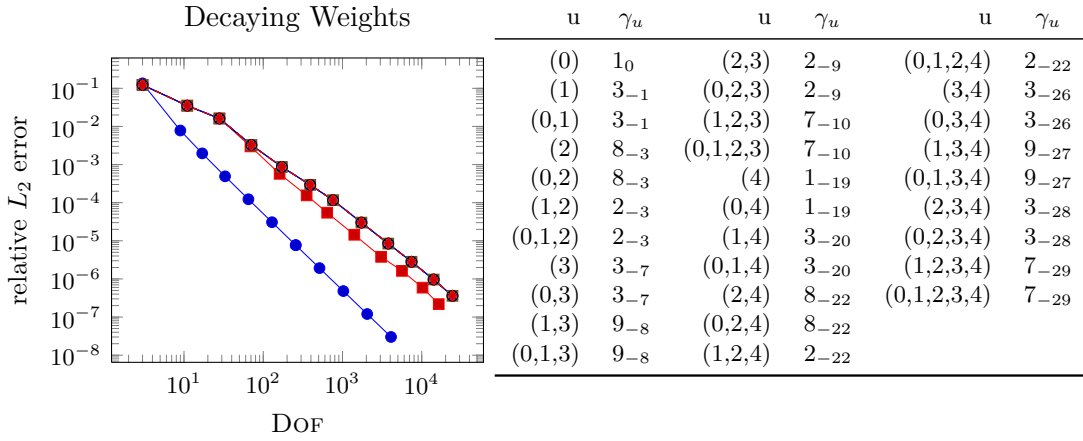


Figure 4.18: Dimension adaptive interpolation of the weighted superposition (4.118) for dimensions $d = 1, 2, \dots, 10$ and the associated weights for the case $d = 5$ (ordered by magnitude). See Figure 4.19 for dimension adaptive index sets.

$n_0 = 1$ and the rest is like a geometric series, compare Lemma 2.2.1. Our first examples use the anchor ANOVA decomposition, i.e. the hierarchical hat basis.

Results for a superposition of order $q = 1$ are shown in Figure 4.17 (top), for order $q = 2$ in Figure 4.17 (middle) and for $q = 5$ in Figure 4.17 (bottom). We see the final degrees of freedom plotted versus the relative L_2 error, for dimensions $d = q, \dots, 10$ on the left. The results for $d > q$ exhibit the same slope as the one for $d = q$, at slightly larger cost. A further analysis confirms that the magnitude of the slopes is as expected. The tables in Figure 4.17 (right column) indicate the cost growth for fixed relative error. It contains the x coordinates of intersection points $(N_d^{\bar{\epsilon}}, \bar{\epsilon})$ between a line passing through $\bar{\epsilon}$ which is parallel to the x axis. We choose $\bar{\epsilon} = 10^{-4}$ for $q = 1, 2$ and $\bar{\epsilon} = 10^{-2}$ for $q = 5$. Furthermore, it shows the value $\binom{d}{q}$ and the postprocessed value $N_d^{\bar{\epsilon}}/N_q^{\bar{\epsilon}}$ which indicates the d -dependent cost increase for fixed $\bar{\epsilon}$. We see that indeed, $N_d^{\bar{\epsilon}}/N_q^{\bar{\epsilon}} = \mathcal{O}(\binom{d}{q})$. Since some grid points can be shared, it is slightly more effective than an isolated superposition model with respect to its degrees of freedom. Thus, we find a method which can automatically exploit additive superposition structure in a cost-optimal way.

Our next experiment is to employ dimension adaptive interpolation of a function with rapidly decaying ANOVA components, i.e. a function for which $\|f_u\|$ approaches zero. Again, we employ the discretized anchor ANOVA decomposition (the hierarchical hat basis). We choose the univariate Cauchy distribution

$$g(x) := \frac{1}{\gamma\pi} \left(1 + \left(\frac{x - x_0}{\gamma}\right)^2\right)^{-1} \quad (4.117)$$

with $\gamma = 1/2$ and $x_0 = 0.8$ and real weights $w_i = 2 \cdot 2^{-2^{3/2}(i-1)}$ to create a weighted

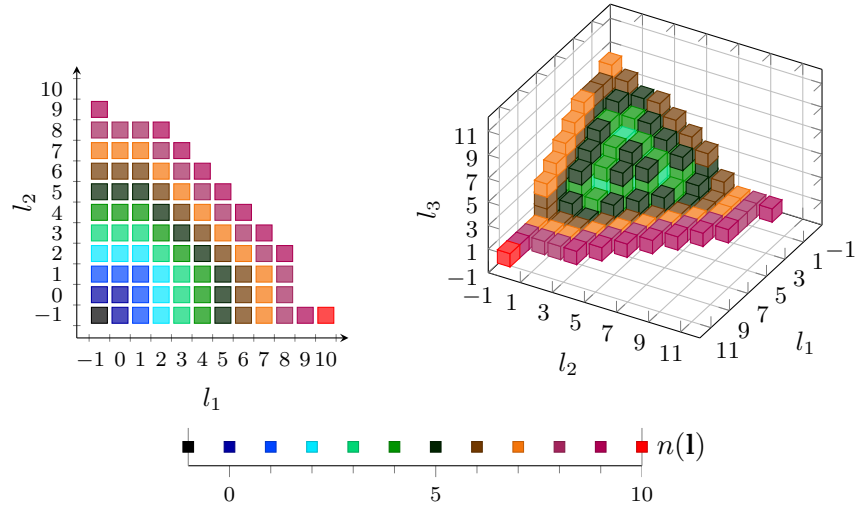


Figure 4.19: Dimension adaptive index sets for the experiment of Figure 4.18 in dimensions $d = 2$, $d = 3$. Each level $n(\mathbf{l})$ has the same color.

superposition

$$f(x) = \sum_{u \subseteq \{1, \dots, d\}} \prod_{j \in u} w_j g(x_j). \quad (4.118)$$

The dimension adaptive interpolation result is presented in Figure 4.18: it contains error estimations for $d = 1, 2, \dots, 10$ together with the weights $\prod_{j \in u} w_j$ ordered by magnitude. The associated dimension adaptive index sets are shown in Figure 4.19 for $d = 2$ and $d = 3$. Note the significant decay seen in the sparse grid levels (indicating by color) for the different directions.

A Comparison of Anchor- and Integral Based Decompositions

Having seen that the dimension adaptive procedure works well for an anchored dimension decomposition, we will now investigate the classical (Lebesgue) ANOVA decomposition. To this end, we repeat the same procedure for the Neumann prewavelet basis which constitutes a discretized Lebesgue ANOVA decomposition. The prewavelet experiments corresponding to Figure 4.17 and Figure 4.18 yielded almost identical results as for the hat basis: the plot lines were almost on top of each other (and are thus not shown here). Note that both, hat basis and prewavelet basis, span the same spaces if the index sets are the same and there is no space adaptivity (there are also fast invertible basis transformations, see Appendix A.2). Consequently, a function representable with terms up to a finite order $q < d$ with respect to the anchor ANOVA will also need at most the same order q for the Lebesgue ANOVA (see also [KSWW09] for this minimality property). Yet, there are differences in the magnitude of ANOVA components which is

anchor ANOVA ($ G = 36,353$)			Lebesgue ANOVA ($ G = 40,449$)		
$ u $	$\frac{\sum \ f_u\ _{L_2}^2}{\ f\ _{L_2}^2}$	$ G_u $	$ u $	$\frac{\sum \ f_u\ _{L_2}^2}{\ f\ _{L_2}^2}$	$ G_u $
0	0	1	0	0.698	1
1	0	512	1	0.285	1,024
2	0.082	35,840	2	0.017	39,424
3-8	0	0	3-8	0	0

Table 4.2: Squared L_2 norms (“variances”) of different ANOVA components applied to the interpolation problem of Figure 4.17 with $q = 2$, $d = 8$ and relative L_2 error of $1.8 \cdot 10^{-3}$, once for the hat basis (left) and once for the prewavelets (right).

presented in Table 4.2: we collect the value

$$\frac{1}{\|f^h\|_{L_2}^2} \sum_{\substack{u \subseteq \{1, \dots, d\} \\ |u|=k}} \|f_u^h\|_{L_2}^2, \quad k = 0, 1, \dots, d \quad (4.119)$$

to quantify the magnitude of components with dimension k (relative to the overall squared L_2 norm). The components belong to the experiment of Figure 4.17 with finite superposition dimension $q = 2$, nominal dimension $d = 8$ and relative L_2 error $1.8 \cdot 10^{-3}$. They are computed once for the hat basis and once for the prewavelets. The hat basis result has slightly less degrees of freedom here which is due to the different error indicators. As already noted, both use only components up to order q – but the anchored decomposition (hat basis) concentrates all of its information in the high-dimensional components whereas the Lebesgue decomposition (prewavelets) concentrates most of its information in low-dimensional components.

Note that we have $\|f^h\|_{L_2}^2 = \sum_{u, v \subseteq \{1, \dots, d\}} (f_u^h, f_v^h) = \sum_{u \subseteq \{1, \dots, d\}} \|f_u^h\|_{L_2}^2$ for the prewavelet basis due to the orthogonality $(f_u^h, f_v^h) = 0$ for $u \neq v$. Thus, the sum of all component norms yields the overall L_2 norm. This decomposition of the squared L_2 norm is the origin of the term ANOVA (analysis of variance) since the variance of a function is just the integral of its square.

Coming back to Table 4.2, we find that the prewavelet decomposition has 70% of its variance in the lowest order term for this particular example, 29% in the linear terms for $|u| = 1$ and just 1% in its two-dimensional terms. The hat basis variant does not allow percentages, but we see that its lower order terms vanish for this example.

We present the same postprocessing for the decaying weights experiment of Figure 4.18 in Table 4.3: again we observe the compression effect of prewavelets which tends to huge parts of the total variance in low order components in comparison to the anchored approach.

We conclude that the Lebesgue ANOVA compresses its information in the lower order components whereas the Anchor ANOVA tends to large high order components. Speaking in terms of the related bases, the prewavelets yield better compression and may be

anchor ANOVA ($ G = 392$)			Lebesgue ANOVA ($ G = 392$)		
$ u $	$\frac{\sum \ f_u\ _{L_2}^2}{\ f\ _{L_2}^2}$	$ G_u $	$ u $	$\frac{\sum \ f_u\ _{L_2}^2}{\ f\ _{L_2}^2}$	$ G_u $
0	0.133	1	0	0.916	1
1	0.268	208	1	0.084	208
2	0.002	183	2	$1 \cdot 10^{-4}$	183
3–8	0	0	3–8	0	0

Table 4.3: Squared L_2 norms (“variances”) of different ANOVA components of the weighted experiment of Figure 4.18 with $d = 8$ and relative L_2 error of $2.92 \cdot 10^{-4}$, once for the hat basis (left) and once for the prewavelets (right).

better when it comes to finding a minimal grid. For the case of pure dimension adaptive refinement, both bases span the same space and there exist fast transformations between them (see Appendix A.2). Even for space- and dimension adaptive grids, the spaces are close to each other, although prewavelets need temporary transport nodes to be inserted to perform adaptive algorithms (compare Appendix A.2). The differences are that the anchor ANOVA (hat basis) is simple to implement and to handle, but especially L_2 best approximation or energy projection (PDEs) are expensive due to large condition numbers and runtime of $2^d N$ for the matrix vector products (compare Section 2.2.2). The prewavelets are more involved, yet they allow fast best approximation due to small condition numbers and runtime $\mathcal{O}(dN)$ (or $\mathcal{O}(d^2 N)$) for matrix multiplications and they have better error estimators, compare [GO95]. In practice, the performance of adaptive grid refinement is often comparable (if not the same) with respect to degrees of freedom versus accuracy. Even if prewavelets result in less degrees of freedom, the hat basis often has the same performance with respect to grid size versus error (since the hat basis does not need temporary transport nodes). Thus, both are useful tools for dimension (and space-) adaptive approximation, with differences with respect to runtime requirements and implementational complexity.

4.5.3 A Dimension Adaptive PDE Solver

We will now apply our dimension adaptive refinement strategy to the solution of partial differential equations. The refinement loop of Algorithm 12 remains unchanged, only the abstract “set values” interface needs to be exchanged. The aim of this section is to point out relevant aspects of such an application and to demonstrate its feasibility.

A solver for PDEs needs to solve a (potentially large) linear system for which a good preconditioner and appropriate methods for matrix assembly or -multiplication combined with a properly assembled right-hand-side are necessary. We use a Galerkin formulation together with fast matrix multiplication routines, an approach which is built on top of the results presented in [Feu05]. Fast matrix multiplication routines for sparse grids rely on the unidirectional principle developed in [Bal94] and elaborated in [Bun98] (see also [Ach03] and [Feu05] for detailed information). This principle employs tensor

product structure and a dimension–recursive splitting into upper- and lower triangular, one–directional matrices. The scheme requires only an implementation of one–dimensional algorithms, just as for the basis transformations. The runtime complexity of the resulting matrix–vector product routine is *linear* with respect to the grid size, together with a dimension–dependent runtime factor $c(d)$ which is at least $c(d) = \Omega(d)$ but usually just bounded by $c(d) = \mathcal{O}(2^d)$. The upper bound can only be avoided if the stiffness matrix has low dimensional product–type variable coefficients and the basis provides some sort of L_2 orthogonality such that the mass matrix becomes (block–) diagonal as elaborated in [Feu05]. A complete discussion of these algorithms is beyond the scope of this thesis and can be found in [Feu05] (see also Appendix A.3 for some commonly required Algorithms).

We suppose we are given the Neumann model problem

$$-\Delta f = h \quad \text{on } [0, 1]^d, \quad (4.120)$$

$$\frac{\partial}{\partial n} f = g \quad \text{on } \partial[0, 1]^d, \quad (4.121)$$

$$f(0, \dots, 0) = f_0 \in \mathbb{R} \quad (4.122)$$

with given right–hand–side h and Neumann condition g . The Dirichlet corner fixes the remaining degree of freedom of the Neumann problem.

Excursion It should be noted that boundary conditions of Dirichlet type on $\partial[0, 1]^d$ are only useful for functions with non–vanishing highest order component, $f_{\{1, \dots, d\}}(x_1, \dots, x_d) \neq 0$. In other words: dimension adaptive procedures are unsuitable for pure Dirichlet conditions. If the highest order term vanishes, the Dirichlet condition already contains 100% of the solution. This statement follows from the three–term boundary splitting stated in Lemma 2.1.13: we can decompose the solution uniquely into boundary terms and *one* inner term, which can be written as $f = f_I + f_B$ where f_B is determined by the boundary Dirichlet conditions and $f_I|_\Gamma \equiv 0$. Now if, $f_{\{1, \dots, d\}} \equiv 0$, we can conclude that the highest order term for *any* ANOVA decomposition vanishes, including the three–term splitting of Lemma 2.1.13 and thus $f_I \equiv 0$. As a consequence, a pure Dirichlet problem does not possibly allow successful dimension adaptive approaches.

By greens formula, we find the weak form of (4.120),

$$\int \nabla f \cdot \nabla v \, dx = \int h \cdot v \, dx + \oint_{\partial[0, 1]^d} g \cdot v \, ds \quad (4.123)$$

$$f(0, \dots, 0) = f_0 \quad (4.124)$$

which we discretize on a sparse grid space with either hierarchical hat basis (and constant on level -1) or the Neumann prewavelet as before. The Dirichlet corner can be discretized by fixing the single basis coefficient $f_{-1, \dots, -1} := f_0$. We have to treat each

basis coefficient (including those on the boundary) as unknowns, so the diffusion matrix is of different shape as for the Dirichlet problem. A representation in the Neumann prewavelet yields mesh-width independent condition numbers by a simple diagonal scaling, compare [GO95]. Furthermore, the Neumann prewavelet is semiorthogonal with respect to the L_2 inner product in the sense that $(\phi_{\mathbf{l},\mathbf{i}}, \phi_{\mathbf{k},\mathbf{j}})_{L_2} = 0$ for $\mathbf{l} \neq \mathbf{k}$, $\mathbf{l}, \mathbf{k} = -1, 0, 1, \dots$. As remarked above, this allows to compute the complete matrix–vector product in time $\mathcal{O}(d^2N)$, or, by employing the inverse mass matrix, in time $\mathcal{O}(dN)$, see [Feu05] for details. The memory requirements are $\mathcal{O}(N)$.

In general, the right–hand–side can be approximated by interpolating h and g on appropriate grids and computing the inner– and boundary integrals exactly in time $\mathcal{O}(dN)$ as proposed in [Bun98], see also [Feu05]. This sparse grid interpolation scheme constitutes a quadrature method. In case the right–hand–side has product structure as in our model below, one may employ one–dimensional quadrature routines to integrate against the product type test functions. A key aspect for the more general interpolation approach is to employ dimension adaptive grids as well. Fortunately, the case of constant coefficients allows to employ the same dimension adaptive grid as for the solution: the derivative $\frac{\partial}{\partial x_i} f_u(x_u)$ of an ANOVA component $u \subseteq \{1, \dots, d\}$ can be represented on the same grid as f_u . We verify immediately that $i \notin u \Rightarrow \frac{\partial}{\partial x_i} f_u(x_u) \equiv 0$. If f_u is linear in direction i , $\frac{\partial}{\partial x_i} f_u$ becomes independent of x_i (it is in a lower ANOVA component and thus in the grid). In all other cases, it remains in the same ANOVA component, perhaps requiring a larger level. Thus, the right–hand–side of a linear differential equation with constant coefficients can be interpolated on the same ANOVA pattern as the solution. In case of variable coefficients which depend on $j \in v$, one might need to use further ANOVA components $u \cup v$. Thus, the right–hand–side can be assembled using the interpolant of either h or g , where the latter case is only evaluated on the boundary.

The dimension adaptive procedure is applied according to Algorithm 12: we initialise a grid of level 1 (without incorporating prior knowledge about the effective dimension), solve the PDE on these 3^d points and compress the result. The remaining ANOVA components are considered to be the relevant ones, so refinement will be applied if and only if child nodes belong to the same ANOVA component as their father.

To demonstrate the usefulness of the method, we consider the following model problem: Let $q = 4$ be a fixed, finite superposition dimension and

$$g(x; x_0, \gamma) := \frac{1}{\pi\gamma} \left(1 + \left(\frac{x - x_0}{\gamma} \right)^2 \right)^{-1} \quad (4.125)$$

a one–dimensional function on $[0, 1]$ with $x_0 = 0.8$, $\gamma = 1/2$ as in Section 4.5.2. Then, we compute the right–hand side and boundary conditions such that

$$f(x) = \sum_{\substack{u \subseteq \{1, \dots, d\} \\ |u|=q}} \prod_{j \in u} g(x_j) \quad (4.126)$$

solves the partial differential equation (4.120).

The result is shown in Figure 4.20 for $d = 4, 5, 6, 7, 8, 9$: the respective convergence rates appear to be the same or become even better with larger d . Furthermore, the

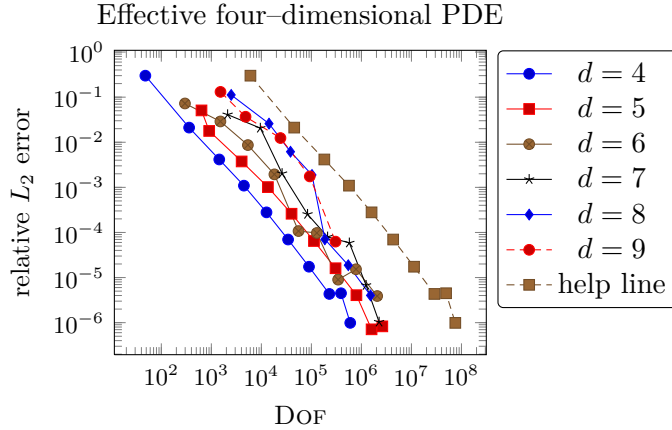


Figure 4.20: Dimension adaptive results for the partial differential equation (4.120) with solution (4.126) solved in dimensions $d = 4, \dots, 9$. The solution is a superposition of four-dimensional functions. The line for $- \blacksquare -$ shows the error of the case $d = 4$ and the degrees of freedom $N = \binom{9}{4}N_{(4)}$ as indicator for the expected cost growth of the combinatorial problem.

increase of degrees of freedom required for a fixed error appears to grow slower than the expected combinatorial factor $\binom{d}{4}$ which can be seen using the help line $(\binom{d}{4} \cdot N_{(4)}, \text{err}_{(4)})$. This help line is displayed for $d = 9$, it takes a multiple of the four-dimensional cost, $N_{(4)}$, and places the marks at the error $\text{err}_{(4)}$ of the four-dimensional case. It is way beyond the curve for $d = 9$. The experiment has been computed using the hierarchical hat basis with basis transforms to apply the preconditioner. Furthermore, we employed the tensor product structure of the right-hand-side using specialized one-dimensional tensor product quadrature formulas.

We conclude that our dimension adaptive procedure of Algorithm 12, combined with the Neumann prewavelet or a different (semi) orthogonal basis, allows to treat partial differential equations as well as the interpolation problem (or even better).

Concerning the overall cost complexity, we find that since an effectively low dimensional solution involves only (sums of) low dimensional Laplacians, we can expect a polynomial growth of the condition number with respect to the nominal dimension d . Furthermore, we discussed fast algorithms to perform the matrix-vector products provided variable coefficients are at most of low-dimensional product type. For a smooth solution, the overall cost complexity can be expected to be $\mathcal{O}(c(q)3^d + c(q)\text{poly}(d) \cdot N)$ since then, the number of refinement iterations becomes a geometric series (see Lemma 2.2.1) and each iteration involves matrix-vector products of complexity $\mathcal{O}(\text{poly}(d)N)$ during an iterative solution of the linear system. The coefficient $c(q)$ indicates the dependence on the effective dimension, it can be exponential in q (since the condition number will grow exponentially in q , see [GO95], unless one uses L_2 orthogonal wavelets, compare [DSS09]). Note that the choice of basis is crucial for the dimension dependent runtime factors. For the hierarchical hat basis, one may need basis transformations to/from the prewavelet

basis in order to reduce the mass matrix multiplication cost.

4.6 Summary of the Dimension Adaptive Method

This chapter elaborates the connection between ANOVA decompositions formulated by means of projectors on the one hand and a multi-level representation on the other hand. We find that both are equivalent formulations, provided the basis is chosen consistently with the projector. Furthermore, the multi-level form allows to employ sparse grids as natural discretized ANOVA decompositions provided the components have bounded second mixed derivatives. We find that the hierarchical hat basis actually *is* the anchor ANOVA decomposition with anchor $a = 0$ and the Neumann prewavelet is the Lebesgue ANOVA. We discussed algorithms to compute all ANOVA components simultaneously and more effectively than integration based approaches. Our analysis algorithms yield the variance analysis for free by computing squared L_2 norms of the approximant in linear time.

For functions with low order ANOVA structure, we derived and analyzed algorithms to employ such structure either automatically using the proposed dimension adaptive approximation algorithm or by means of the weighted sparse grid space introduced in Section 4.2. Our approaches are generalizations of methods known for integration [Hol08, GG03] or machine learning [Gar04]. Besides the detection of low-dimensional structure, our formulation also allows simultaneous detection of local irregularities; it connects the well-known sparse grid space adaptivity with dimension decompositions, for the first time formulated precisely within the ANOVA framework. The ANOVA framework allows to improve the reliability and gains more insight into the dimension adaptive procedure.

5 Conclusion and Outlook

This thesis presents three new aspects of sparse grids: extensions to the sparse grid toolbox, an analysis on inherent limitations of sparse grids along with advantages over conventional methods, and a new dimension approach to approximate functions with axis parallel superposition structure.

The first contribution contains the first (to the knowledge of the author) analysis of sparse grid error bounds for non-homogeneous boundary conditions and an extended sparse grid function algebra supporting addition, multiplication, concatenation, and an a posteriori adaptive collocation approach to approximate kernel transformations with arbitrary kernels. The analysis is generalized to data errors by means of condition numbers rather than elementary estimates compared to [Gri98, MgF07] and respects, for the first time, also the consistency error arising for such operations. We also propose an adaptive method which improves old weaknesses arising due to unbalanced consistency- and data errors to complete the sparse grid algebra.

The second contribution is our investigation in dimension-dependent order coefficients for the relevant application of probability density approximation (with the Fokker-Planck-Equation in mind): our a priori bounds and a posteriori error measurements document exponential increase of the d -dependent order coefficients for the representative Gaussian density. Thus, density approximation with respect to relative L_2 , L_∞ , or energy norm on \mathbb{R}^d is inherently limited to dimensions $d = 5$ or maybe $d = 6$. Our reasoning on the normal distribution identifies the necessity to capture the complete density mass on \mathbb{R}^d as main cause for the limitations: the domain to be covered with grid points grows or shrinks with the density's width (variance) and the relative error effectively rescales the resulting normalized density to a reference density (the standard Gaussian). Nevertheless, the sparse grid complexity grows mainly in their order coefficients (if we consider the log terms as d -dependent coefficients for the moment) whereas alternative full grid methods of similar degree suffer from exponential growth in their degrees of freedom. In our examples, a comparison of degrees of freedom between d and $d + 1$ for fixed $\epsilon \approx 10^{-3}$ revealed a factor of 10 for sparse grids compared to a factor of 100 for full grids. Consequently, sparse grids increase the feasible dimension from about $d = 3$ to $d = 5$ or perhaps $d = 6$. The "curse of dimensionality" is thus lessened in so far as almost twice as many dimensions become computationally feasible.

The third contribution of this thesis is the first dimension adaptive approach which is especially designed for use in approximation problems. It solves early termination issues of previous approaches to generalize dimension adaptive methods known from quadrature to approximation problems (compare [Gar04]). The new approach allows for a systematic reduction of degrees of freedom if functions effectively depend only on few (subsets of) directions. The approach combines the successful space adaptive sparse grid

techniques known for approximation problems with ideas known for dimension adaptive quadrature [GG03, Hol08] and machine learning [Gar04]. To this end, our approach is – for the first time – directly based on a given ANOVA decomposition (like anchor ANOVA or Lebesgue ANOVA) by using the following two–step algorithm: identifying the minimum set of ANOVA components using grid compression of a coarse grid, and then resolving these ANOVA components adaptively. As such, it also provides (for the first time) space adaptive refinement inside of ANOVA components and thus yields dimension- and space adaptive resolution of such input functions. The approach allows for compact representations for functions in weighted spaces or functions with finite order weights [SWW04, NW08], i.e. if a function $f(x_1, \dots, x_d)$ is actually a superposition of few relevant lower dimensional contributions depending on subsets of the $\{x_1, \dots, x_d\}$ (if it is of “low effective dimension”). The superposition is formulated by means of the ANOVA decomposition and uses a power set construction of the directions $\{x_1, \dots, x_d\}$, making it inherently dependent on the choice of axes. This thesis proposes a priori optimized sparse grid spaces which are optimal for spaces weighted with respect to second mixed derivatives among their ANOVA components. Thus, if the weights are known in advance, functions from such spaces can be discretized by means of a superposition of lower–dimensional regular sparse grids with known weight–dependent levels. If the weights are unknown, we propose a new dimension adaptive algorithm to find the optimal decomposition for a given function automatically. For functions where relevant ANOVA components are all low dimensional but which have local singularities, we propose a new dimension– and space adaptive algorithm.

In summary, we find that sparse grid methods are inherently limited if the approximant depends equally on all its variables (like our example of the full–space Gaussian). However, they extend limitations of classical full grid methods from $d = 3$ to about $d = 6$. Higher dimensional problems are possible if the function exhibits more structure: for functions with axis parallel structure in form of decaying ANOVA decompositions, we show constructively by means of a new dimension adaptive tool that higher dimensions become feasible where the cost is determined by the effective dimension and the number of relevant ANOVA components.

Outlook on Further Investigations

The remaining open question is: can we reformulate high dimensional problems like the Fokker–Planck–Equations such that they become computationally feasible with respect to d ? In our case, the nonlinearly graded grids of Section 3.2.3 which fit perfectly to the model density reduce the dependence of d to a certain extent (at the expense of losing approximation orders near the probability tails) and might be appropriate if the solution is simple enough to be almost Gaussian yet difficult enough to require numerical methods. Besides such a specialized approach, an outlook to improve the complexity at least quantitatively might be to formulate the ansatz space directly on \mathbb{R}^d , allowing potentially more control over the degrees of freedom compared to our box truncation and grid adaptation process (see also [GH10] for full space sparse grid methods).

Qualitative improvements, i.e. sub–exponential complexity might be possible if the

problem at hand can be reformulated as low order ANOVA model as in Chapter 4, i.e. using axis-parallel superposition structure. For the density approximation however, the natural representation of a “constant” is the Dirac delta instead of a constant function, so a (possibly nonlinear) transformation appears to be necessary. We provide a discussion of limitations and potential of log density formulations, nonlinearly enriched ansatz spaces and nonlinearly weighted ANOVA decompositions in Section 3.2.3 and Section 4.4, respectively. It might be helpful to employ a general pairwise weight decomposition similar to the Cluster Expansion [LS96], i.e. ANOVA-type decompositions where the role of the constant is played by $\prod_{i < j} g_{ij}(x_i, x_j)$ for some nonlinear pair-coupling $g_{ij}(\cdot, \cdot)$ and remaining components which consist of single component interactions (as discussed in Section 4.4) and pairwise interactions, yet it is difficult to formulate such decompositions in the general case (beyond the Gaussian approach of [LS96]). In the general case, candidates for transformations to axis parallel structure are also linear coordinate transformations like rotations, followed by dimension adaptive ANOVA approaches in transformed coordinates (at the time of this writing, such an approach is subject of the thesis [Oet10]).

It might even prove to be useful to consider ANOVA structure in nonlinear manifolds, perhaps using manifold detection techniques as we presented them in [FG09] or [Hul09], see also the references cited therein for the related unsupervised learning techniques. Such approaches need to be derived for specific applications at hand, provided the problem has some sort of intrinsic effective dimension which is lower than its nominal dimension.

Besides the need for further research with respect to applications and problem formulations, technical improvements concerning prewavelet adaptivity and data transport should be considered as discussed in Appendix A.4.1.

Thus, sparse grid methods provide higher dimensional dynamics if they are complemented with properly chosen analytical tools to reveal low intrinsic dimensionality.

5 *Conclusion and Outlook*

A Technical Reference

A.1 Recursive Grid Traversal Routines

A.1.1 Visiting Every Grid Point in Linewise Ordering

Sparse grid data structures need methods to visit every grid point, often in a well-defined sequence (compare Section 2.2.3). Usually, every algorithm can be decomposed using the Unidirectional Principle (Section 2.2.2) into a sequence of one-dimensional algorithms operating on lines. The requirement is to provide access to lines, for any direction. Inside of each line, sparse grids are just binary trees (with special handling for the two boundary nodes).

Linewise traversal operations usually accumulate information. For example, a pre-order traversal (top-down) works recursively and communicates stack data from the boundary down to the finest levels, see Algorithm 13. Similarly, a post-order traversal (bottom-up) traversal communicates stack data from bottom to top (starting usually with values of 0 in the leafs), see Algorithm 14. Another important sequence is a breadth-first-search (levelwise) tree traversal which is based on queues, see Algorithm 15. They accumulate levelwise information (for example in arrays). Occasionally, one needs in-order traversals as well (which report from left to right or from right to left).

Algorithm 13 Pre-order tree traversal on one line in direction m (Top-Down)

Input: A point (\mathbf{l}, \mathbf{i}) such that (l_m, i_m) is the tree's root

- 1: compute operation on boundary $(l_m, i_m) := (0, 0)$ and $(l_m, i_m) := (0, 1)$
- 2: fill `StackData`
- 3: invoke `pre-order` $((\mathbf{l}, \mathbf{i}), m, \text{StackData})$ for $(l_m, i_m) = (1, 1)$

`pre-order` $((\mathbf{l}, \mathbf{i}), m, \text{StackData})$:

Input: `StackData` from father

- 1: compute operation for (\mathbf{l}, \mathbf{i})
 - 2: fill `StackDataLeft` and `StackDataRight`
 - 3: invoke `pre-order`(left child of $(\mathbf{l}, \mathbf{i}), m, \text{StackDataLeft}$)
 - 4: invoke `pre-order`(right child of $(\mathbf{l}, \mathbf{i}), m, \text{StackDataRight}$)
-

The task to visit every line in direction m of an adaptive sparse grid can be split into two separate tasks. The first is memory access, for example to find $u_{\mathbf{l}, \mathbf{i}}$ for a given (\mathbf{l}, \mathbf{i}) . The second is to generate a sequence of (\mathbf{l}, \mathbf{i}) such that the traversal task is logically complete. If the data structure supports access to all d fathers of a given node (\mathbf{l}, \mathbf{i}) and all $2d$ sons of (\mathbf{l}, \mathbf{i}) as well, memory access to any $u_{\mathbf{l}, \mathbf{i}}$ can be granted by a hierarchically

Algorithm 14 Post-order tree traversal on one line in direction m (Bottom-Up)

Input: A point (\mathbf{l}, \mathbf{i}) such that (l_m, i_m) is the tree's root

- 1: invoke `StackData := post-order((\mathbf{l}, \mathbf{i}), m)` for $(l_m, i_m) = (1, 1)$
- 2: finish operation on boundary, based on `StackData`

`post-order((\mathbf{l}, \mathbf{i}), m)`

Output: `StackData` for father

- 1: `StackDataLeft := post-order(left child of (\mathbf{l}, \mathbf{i}), m)`
 - 2: `StackDataRight := post-order(right child of (\mathbf{l}, \mathbf{i}), m)`
 - 3: compute operation for (\mathbf{l}, \mathbf{i}) , based on `StackDataLeft` and `StackDataRight`
 - 4: fill `StackData` for father
-

Algorithm 15 Breadth-First-Search traversal on one line in direction m (Levelwise)

Input: A point (\mathbf{l}, \mathbf{i}) such that (l_m, i_m) is the tree's root

- 1: process operation on boundary
 - 2: `queue := (1, 1)` if $(l_m, i_m) = (1, 1)$ exists
 - 3: **while** `queue $\neq \emptyset$` **do**
 - 4: $(l_m, i_m) :=$ pop first of `queue`
 - 5: apply operation on (\mathbf{l}, \mathbf{i})
 - 6: enqueue left son of $(\mathbf{l}, \mathbf{i}), m$ to `queue`
 - 7: enqueue right son of $(\mathbf{l}, \mathbf{i}), m$ to `queue`
 - 8: **end while**
-

working sequence generator. We will focus on such sequence generators, i.e. to report every multi-index (\mathbf{l}, \mathbf{i}) exactly once, in a prescribed sequence.

Algorithm 16 Line Traversal on *each* line in direction m

Input: The grid's d -dimensional root (\mathbf{l}, \mathbf{i})

Input: A direction $0 \leq m < d$ (0-based to simplify implementations.)

Input: A one-dimensional tree traversal algorithm A_m (like Algorithm 13)

- 1: Define a $(d - 1)$ dimensional slice by holding (l_m, i_m) fixed
 - 2: **if** $(\mathbf{l}, \mathbf{i}) = (1, \dots, 1 | 1, \dots, 1)$ **then**
 - 3: Visit all points on the $(d - 1)$ dimensional slice using Algorithm 17.
 - 4: For each reported point (\mathbf{k}, \mathbf{j}) : invoke A_m in direction m
 - 5: **else if** $(\mathbf{l}, \mathbf{i}) = (0, \dots, 0 | 0, \dots, 0)$ **then**
 - 6: Visit all points on the $(d - 1)$ dimensional slice using Algorithm 18.
 - 7: For each reported point (\mathbf{k}, \mathbf{j}) : invoke A_m in direction m
 - 8: **end if**
-

Access to lines can be reduced to the problem of visiting every point on a slice: if we want to access all lines in direction m , fix (l_m, i_m) to the tree's root in direction m and report all grid points on the slice defined by the fixed (l_m, i_m) . Per construction, the tree's root exists in every direction and on every line as part of the hanging nodes conditions; it is the middle point $(l_m, i_m) = (1, 1)$ for space adaptive grids and the left

boundary $(l_m, i_m) = (-1, 0)$ (or, equivalently, $(0, 0)$) for dimension adaptive grids. A slice is nothing but a sparse grid of lower dimensionality. Consequently, we get access to every line once we solve the problem of visiting every point on a d -dimensional (adaptive) sparse grid: visit every point on the root slice and start line traversals in direction m . This is summarized in Algorithm 16.

The adaptive case will necessarily employ the multi-level hierarchy. Since sparse grids are recursively defined in both, dimension and one-dimensional tree structure, the algorithm will contain two nested recursions¹.

If the tree's root is in the middle (standard for a space adaptive grid), $(\mathbf{l}, \mathbf{i}) = (1, \dots, 1 | 1, \dots, 1)$, Algorithm 17 provides a possible implementation to visit every grid point once and only once (compare [Bal94]). Its runtime is linear in the number of points visited. Note that the indexing of Algorithm 17 starts at 0 to allow a simpler implementation. The algorithm does not need any boundary points, it is possible to eliminate all boundary operations and work only in the inner domain. Such a feature is particularly important for partial differential equations or other applications where the expensive boundary nodes are not desired. The resulting grid point sequence is shown in Figure A.1 (left) for a two-dimensional grid.

If the tree's root is at the left boundary $(0, \dots, 0 | 0, \dots, 0)$ as for a dimension adaptive grid, Algorithm 18 describes how to perform the traversal. It relies on the boundary, but it does not need any other point than $(0, \dots, 0 | 0, \dots, 0)$. The resulting grid point sequence is shown in Figure A.1 (right).

A.2 Basis Bestiary – Hierarchical Transformations

If we are given a (sparse, full or adaptive) grid G with elements of the form $(\mathbf{l}, \mathbf{i}) \in G$, we are interested in fast transformations from nodal values $u(x_{\mathbf{l}, \mathbf{i}})$ to hierarchical basis coefficients $u_{\mathbf{l}, \mathbf{i}}$ and from basis coefficients to nodal values. We will present fast algorithms to compute these transformations for a couple of basis sets in the following sections.

All of these transformations make use of the tensor product structure and the unidirectional principle discussed in Section 2.2.2: they are formulated for the one-dimensional case only and the d -dimensional case is performed by applying one-dimensional routines on grid lines. Furthermore, most transformation matrices can be decomposed into upper *or* lower triangular matrices, so only one pass in each direction is necessary and the recursion of the unidirectional principle becomes as simple as possible.

A.2.1 The Multilevel Piecewise Linear Generating System

The multi-level generating system is required as intermediate step for transformations or hierarchical algorithms. It spans the same space as the hierarchical linear basis, but it contains even *and* odd space indices on each level. This is illustrated in Figure A.2: the

¹The case of regular sparse grids allows simplifications since only level indices \mathbf{l} need to be generated. Only one recursion and a simple loop to iterate through all relevant \mathbf{i} are necessary.

Algorithm 17 Visit every point of an adaptive sparse grid, anchored at the middle.

Start: `reportBoundaryAndInner((l, i), $d - 1$)` with $(\mathbf{l}, \mathbf{i}) = (1, \dots, 1 | 1, \dots, 1)$.

`reportBoundaryAndInner((l, i), m):`

Input: Actual direction $-1 \leq m < d$ // Using $(\mathbf{l}, \mathbf{i}) = (l_0, \dots, l_{d-1} | i_0, \dots, i_{d-1})$

Input: Actual point (\mathbf{l}, \mathbf{i}) with $(l_r, i_r) = (1, 1)$ for $0 \leq r \leq m$

```

1: if (l, i) belongs to grid then
2:     report (l, i) and its value  $u_{\mathbf{l}, \mathbf{i}}$ 
3:     for  $r = 0$  to  $m$  do
4:          $(\mathbf{k}, \mathbf{j}) := (\mathbf{l}, \mathbf{i})$  with  $(k_r, j_r) = (0, 0)$ 
5:         reportBoundaryAndInner((k, j),  $r - 1$ ) // outer recursion:  $d$ 
6:          $(\mathbf{k}, \mathbf{j}) := (\mathbf{l}, \mathbf{i})$  with  $(k_r, j_r) = (0, 1)$ 
7:         reportBoundaryAndInner((k, j),  $r - 1$ )
8:          $(\mathbf{k}, \mathbf{j}) := (\mathbf{l}, \mathbf{i})$  with  $(k_r, j_r) = (l_r + 1, 2i_r - 1)$ 
9:         depthFirstSearchInnerInclBoundary((k, j),  $r$ ) // inner recursion: tree
10:         $(\mathbf{k}, \mathbf{j}) := (\mathbf{l}, \mathbf{i})$  with  $(k_r, j_r) = (l_r + 1, 2i_r + 1)$ 
11:        depthFirstSearchInnerInclBoundary((k, j),  $r$ )
12:    end for
13: end if

```

`depthFirstSearchInnerInclBoundary((l, i), m):`

Input: Actual direction $0 \leq m < d$

Input: Actual point (\mathbf{l}, \mathbf{i}) with $(l_r, i_r) = (1, 1)$ for $0 \leq r < m$

```

1: if (l, i) belongs to grid then
2:     report (l, i) and its value  $u_{\mathbf{l}, \mathbf{i}}$ 
3:     for  $r = 0$  to  $m$  do
4:         if  $r < m$  then
5:              $(\mathbf{k}, \mathbf{j}) := (\mathbf{l}, \mathbf{i})$  with  $(k_r, j_r) = (0, 0)$ 
6:             reportBoundaryAndInner((k, j),  $r - 1$ ) // outer recursion:  $d$ 
7:              $(\mathbf{k}, \mathbf{j}) := (\mathbf{l}, \mathbf{i})$  with  $(k_r, j_r) = (0, 1)$ 
8:             reportBoundaryAndInner((k, j),  $r - 1$ )
9:         end if
10:         $(\mathbf{k}, \mathbf{j}) := (\mathbf{l}, \mathbf{i})$  with  $(k_r, j_r) = (l_r + 1, 2i_r - 1)$ 
11:        depthFirstSearchInnerInclBoundary((k, j),  $r$ ) // inner recursion: tree
12:         $(\mathbf{k}, \mathbf{j}) := (\mathbf{l}, \mathbf{i})$  with  $(k_r, j_r) = (l_r + 1, 2i_r + 1)$ 
13:        depthFirstSearchInnerInclBoundary((k, j),  $r$ )
14:    end for
15: end if

```

Algorithm 18 Visit every point of a dimension ($-$ and space) adaptive sparse grid which is anchored at the left boundary.

Start: `reportBoundaryAndInner`((\mathbf{l}, \mathbf{i}), $d - 1$) with $(\mathbf{l}, \mathbf{i}) = (0, \dots, 0 | 0, \dots, 0)$.

`reportBoundaryAndInner`((\mathbf{l}, \mathbf{i}), m):

Input: Actual direction $-1 \leq m < d$ // Using $(\mathbf{l}, \mathbf{i}) = (l_0, \dots, l_{d-1} | i_0, \dots, i_{d-1})$

Input: Actual point (\mathbf{l}, \mathbf{i}) with $(l_r, i_r) = (0, 0)$ for $0 \leq r \leq m$

```

1: if ( $\mathbf{l}, \mathbf{i}$ ) belongs to grid then
2:   report ( $\mathbf{l}, \mathbf{i}$ ) and its value  $u_{\mathbf{l}, \mathbf{i}}$ 
3:   for  $r = 0$  to  $m$  do
4:      $(\mathbf{k}, \mathbf{j}) := (\mathbf{l}, \mathbf{i})$  with  $(k_r, j_r) = (0, 1)$ 
5:     reportBoundaryAndInner(( $\mathbf{k}, \mathbf{j}$ ),  $r - 1$ ) // outer recursion:  $d$ 
6:     depthFirstSearchInnerInclBoundary(( $\mathbf{l}, \mathbf{i}$ ),  $r$ ) // inner recursion: tree
7:   end for
8: end if

```

`depthFirstSearchInnerInclBoundary`((\mathbf{l}, \mathbf{i}), m):

Input: Actual direction $0 \leq m < d$

Input: Actual point (\mathbf{l}, \mathbf{i}) with $(l_r, i_r) = (0, 0)$ for $0 \leq r < m$

```

1: if ( $\mathbf{l}, \mathbf{i}$ ) belongs to grid then
2:   report ( $\mathbf{l}, \mathbf{i}$ ) and its value  $u_{\mathbf{l}, \mathbf{i}}$ 
3:   for  $r = 0; r < m; r = r + 1$  do
4:      $(\mathbf{k}, \mathbf{j}) := (\mathbf{l}, \mathbf{i})$  with  $(k_r, j_r) = (0, 1)$ 
5:     reportBoundaryAndInner(( $\mathbf{k}, \mathbf{j}$ ),  $r - 1$ ) // outer recursion:  $d$ 
6:      $(\mathbf{k}, \mathbf{j}) := (\mathbf{l}, \mathbf{i})$  with  $(k_r, j_r) = (1, 1)$ 
7:     depthFirstSearchInnerInclBoundary(( $\mathbf{k}, \mathbf{j}$ ),  $r$ )
8:   end for
9:    $(\mathbf{k}, \mathbf{j}) := (\mathbf{l}, \mathbf{i})$  with  $(k_m, j_m) = (l_r + 1, 2i_r - 1)$ 
10:  depthFirstSearchInnerInclBoundary(( $\mathbf{k}, \mathbf{j}$ ),  $m$ )
11:   $(\mathbf{k}, \mathbf{j}) := (\mathbf{l}, \mathbf{i})$  with  $(k_m, j_m) = (l_r + 1, 2i_r + 1)$ 
12:  depthFirstSearchInnerInclBoundary(( $\mathbf{k}, \mathbf{j}$ ),  $m$ )
13: end if

```

basis functions are displayed in black whereas the additional elements of the generating system are shown in gray.

The transformation from generating system to hierarchical basis employs the two-scale relation for generating system functions (even i), $\phi_{l,i} = -\frac{1}{2}\phi_{l,i-1} + \phi_{l-1,i/2} - \frac{1}{2}\phi_{l,i+1}$. Given a one-dimensional representation in the generating system $u = \sum_{(l,i)} \tilde{u}_{l,i} \phi_{l,i}$, the transformation to the hat basis can be described as recursive procedure using temporary stack values

$$t_{l,i} := \tilde{u}_{l,i} + t_{l+1,2i} \quad (\text{A.1})$$

by

$$u_{l,i} = t_{l,i} - \frac{1}{2}t_{l,i+1} - \frac{1}{2}t_{l,i-1} = \tilde{u}_{l,i} + t_{l+1,2i} - \frac{1}{2}\tilde{u}_{l,i\pm 1} - \frac{1}{2}t_{l+1,2(i\pm 1)}. \quad (\text{A.2})$$

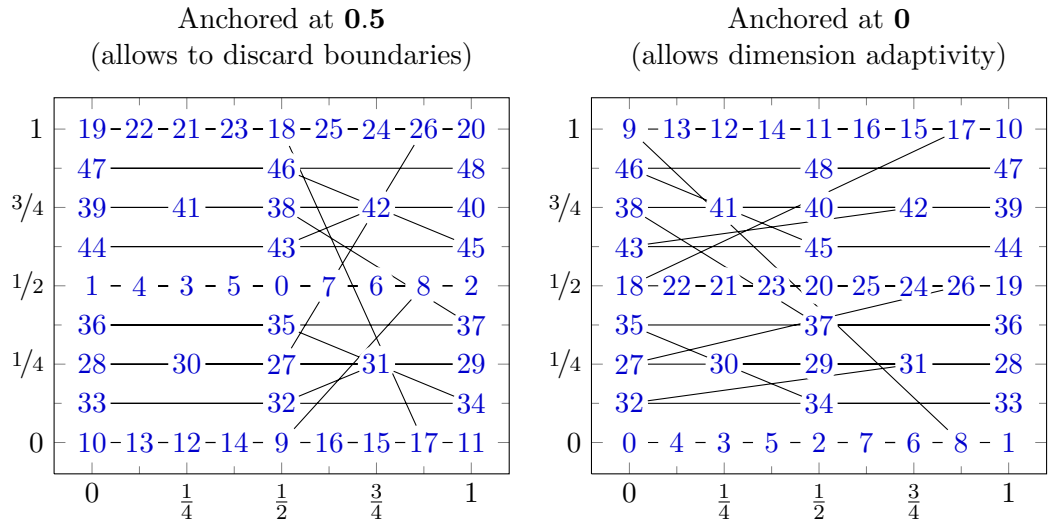


Figure A.1: Grid points in the visited sequence for sparse grids anchored at $(0.5, \dots, 0.5)$ by means of Algorithm 17 (left) and those anchored at $(0, \dots, 0)$ by means Algorithm 18 (right) for a two-dimensional sparse grid of level $n = 2$. The x axis is direction $m = 0$ and the y axis direction $m = 1$.

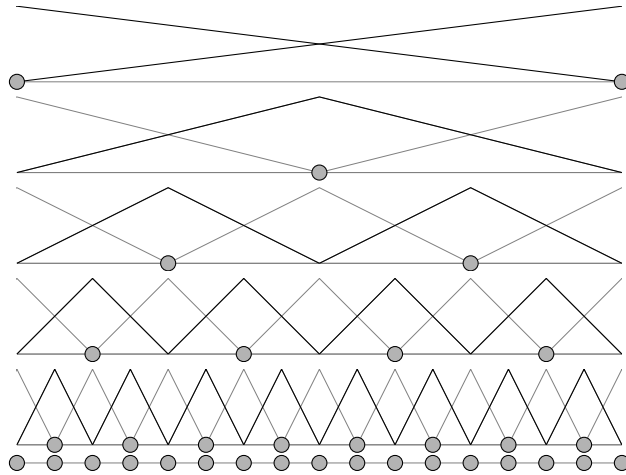


Figure A.2: The one-dimensional linear hierarchical basis and its generating system on four levels (black: basis function elements, gray: generating system).

The values $t_{l,i}$ are defined for every generating system node and can be accumulated during a grid traversal.

A.2.2 The Linear Hierarchical Basis

The transformation from nodal values $n_{l,i} = u(x_{l,i})$ to basis coefficients $u_{l,i}$ of the hierarchical hat basis can be realized in linear time during d traversals through the grid. Since each hierarchical coefficient can be represented as product stencil (2.49) and the information flow is from top to bottom, we apply pre-order tree traversals along each line in direction $m = 1$, followed by the same operation along each line in direction $m = 2$ and so on. Thus, it suffices to provide the one-dimensional transformation only. We assume

$$u: [0, 1] \rightarrow \mathbb{R}. \quad (\text{A.3})$$

We start to deal with the highest hierarchical level, the boundary $l = 0$ and we mention two different linear hierarchical bases which differ only in the two boundary points:

The hierarchical boundary basis uses a constant function attached to the left boundary,

$$\phi_{0,0} = 1, \quad (\text{A.4})$$

often denoted by the equivalent index $(-1, 0)$, and a linear function attached to the right boundary,

$$\phi_{0,1} = x. \quad (\text{A.5})$$

Thus, the coefficient $u_{0,0}$ is actual a nodal value,

$$u_{0,0} = u(x_{0,0}), \quad (\text{A.6})$$

while the right boundary coefficient is a hierarchical surplus. It has the value

$$u_{0,1} = u(x_{0,1}) - u_{0,0}. \quad (\text{A.7})$$

The nodal boundary basis simply uses the nodal basis for the boundary,

$$\phi_{0,0} = x \text{ and } \phi_{0,1} = 1 - x. \quad (\text{A.8})$$

Consequently, the associated basis coefficients are just nodal values,

$$u_{0,0} = u(x_{0,0}) \text{ and } u_{0,1} = u(x_{0,1}). \quad (\text{A.9})$$

We continue with the inner points $l \geq 1$. Due to the disjoint supports of the basis functions on one level, a coefficient $u_{l,i}$ is the difference

$$u_{l,i} = u(x_{l,i}) - I_{G_{l-1}}u(x_{l,i}) \quad (\text{A.10})$$

where $G_{l-1} := \{(k, j) \in G \mid k \leq l - 1\}$ denotes the parts in G which are hierarchically higher than (l, i) . But $I_{G_{l-1}}u$ is piecewise linear on $\text{supp } \phi_{l,i}$ and it interpolates u on G_{l-1} , so it is nothing but the mean of the two end point values of $\phi_{l,i}$:

$$u_{l,i} = u(x_{l,i}) - \frac{1}{2} \left(u(x_{l,i-1}) + u(x_{l,i+1}) \right). \quad (\text{A.11})$$

The relation can be computed efficiently on adaptive sparse grids during the pre-order traversal Algorithm 13 if `StackData` is chosen as left and right nodal value. It is shown in Algorithm 19.

Algorithm 19 Hierarchical Transformation hat $u_{\mathbf{l},i} \leftrightarrow n_{\mathbf{l},i}$ nodal values in direction m , to be used as operation inside of pre-order (Algorithm 13).

Operation on boundary:

- 1: compute the target value from either (A.6) and (A.7) or from (A.9)
- 2: `StackData.left` := $n_{\mathbf{k},\mathbf{j}}$ where $(\mathbf{k},\mathbf{j}) = (\mathbf{l},\mathbf{i})$ and $(k_m, j_m) = (0, 0)$
- 3: `StackData.right` := $n_{\mathbf{k},\mathbf{j}}$ where $(\mathbf{k},\mathbf{j}) = (\mathbf{l},\mathbf{i})$ and $(k_m, j_m) = (0, 1)$

Operation on inner nodes:

Input: `StackData` from father

Output: `StackDataLeft` and `StackDataRight`

- 1: $t := 1/2(\text{StackData.left} + \text{StackData.right})$
 - 2: **if** transform to nodal values $n_{\mathbf{l},i}$ **then**
 - 3: $n_{\mathbf{l},i} := u_{\mathbf{l},i} + t$
 - 4: **else**
 - 5: $u_{\mathbf{l},i} := n_{\mathbf{l},i} - t$
 - 6: **end if**
 - 7: `StackDataLeft` := [`StackData.left`, $n_{\mathbf{l},i}$]
 - 8: `StackDataRight` := [$n_{\mathbf{l},i}$, `StackData.left`]
-

A Prewavelet basis function

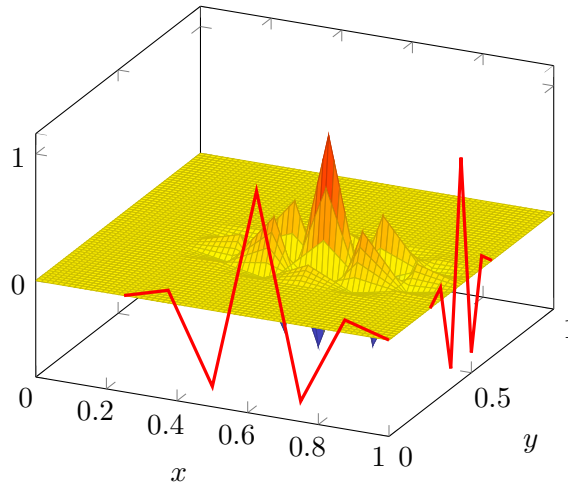


Figure A.3: The prewavelet $\psi_{3,4|5,7} = \psi_{3,5} \cdot \psi_{4,7}$.

A.2.3 The Prewavelet Bases

We use two different prewavelet bases: the Dirichlet prewavelet basis which vanishes on the boundary, $\psi_{l,i}(0) = \psi_{l,i}(1) = 0$ for $l > 0$, and the Neumann basis which yields good condition numbers for the Neumann case and discretizes the Lebesgue ANOVA. Each of them consists of linear combinations of hat functions, which makes it considerably simpler to perform the transformation from prewavelets to the hierarchical hat basis

(and backwards) than to the nodal basis. Since fast algorithms between hierarchical hat basis and nodal basis are known [GO95], we follow this approach.

Let $\phi_{l,i}$ for $l \geq 0$, $i = 0, \dots, 2^l + 1$ denote elements of the hat generating system. Then, each of our prewavelet basis functions is defined to be a linear combination of the $\phi_{l,i}$. Denoting the Dirichlet prewavelet basis functions by $\psi_{l,i}^D$ we get the relations

$$\psi_{0,0}^D = \phi_{0,0} \tag{A.12}$$

$$\psi_{0,1}^D = \phi_{0,1} \tag{A.13}$$

$$\psi_{1,1}^D = \phi_{1,1} \tag{A.14}$$

$$\psi_{l,1}^D = \frac{9}{10}\phi_{l,1} - \frac{6}{10}\phi_{l,2} + \frac{1}{10}\phi_{l,3} \tag{A.15}$$

$$\psi_{l,2^{l-1}}^D = \frac{1}{10}\phi_{l,2^{l-3}} - \frac{6}{10}\phi_{l,2^{l-2}} + \frac{9}{10}\phi_{l,2^{l-1}} \tag{A.16}$$

$$\psi_{l,i}^D = \frac{1}{10}\phi_{l,i-2} - \frac{6}{10}\phi_{l,i-1} + \phi_{l,i} - \frac{6}{10}\phi_{l,i+1} + \frac{1}{10}\phi_{l,i+2} \tag{A.17}$$

and for the Neumann prewavelets $\psi_{l,i}^N$

$$\psi_{0,0}^N = \phi_{0,0} + \phi_{0,1} \equiv 1 \tag{A.18}$$

$$\psi_{0,1}^N = -\phi_{0,0} + \phi_{0,1} \tag{A.19}$$

$$\psi_{1,1}^N = -\phi_{1,0} + \phi_{1,1} - \phi_{1,2} \tag{A.20}$$

$$\psi_{l,1}^N = -\frac{12}{10}\phi_{l,0} + \frac{11}{10}\phi_{l,1} - \frac{6}{10}\phi_{l,2} + \frac{1}{10}\phi_{l,3} \tag{A.21}$$

$$\psi_{l,2^{l-1}}^N = \frac{1}{10}\phi_{l,2^{l-3}} - \frac{6}{10}\phi_{l,2^{l-2}} + \frac{11}{10}\phi_{l,2^{l-1}} - \frac{12}{10}\phi_{l,2^l} \tag{A.22}$$

$$\psi_{l,i}^N = \psi_{l,i}^D. \tag{A.23}$$

Some Neumann prewavelet functions are shown in Figure 4.2 on page 110, see also Figure A.3.

Transformation From Prewavelet To Hat Basis

The basic idea for the transformation to the hat basis is: we provide the transformations from prewavelets to hat generating system, $\{\psi_{l,i}^{D,N}\} \rightarrow \{\phi_{l,i}\}$ and from there to the hierarchical hat basis. The backwards transformation follows by inversion.

The first step of the transformation, from prewavelet to hat generating system follows immediately from (A.12) – (A.23): simply apply the prewavelet masks and distribute basis coefficients onto the adjacent generating system nodes with the prewavelet mask as weights. Afterwards, we need to eliminate the generating system using rule (A.2).

The actual realization has to be done carefully since the overlapping basis functions and the boundary modifications yields many case distinctions. However, we can formulate the problem using general prewavelet masks as follows. We drop the superscripts D and N and work only with prewavelet masks in the following. Thus, $\psi_{l,i}$ is a prewavelet

A Technical Reference

basis function defined by real coefficients $a_j^{l,i}$, $j = -2, -1, 0, 1, 2$ and

$$\psi_{l,i} = a_{-2}^{l,i} \phi_{l,i-2} + a_{-1}^{l,i-1} \phi_{l,i-1} + a_0^{l,i} \phi_{l,i} + a_1^{l,i+1} \phi_{l,i+1} + a_2^{l,i} \phi_{l,i+2}. \quad (\text{A.24})$$

For reasons of simplicity, we assume $a_j^{l,i} = 0$ whenever $(l, i + j)$ does not belong to the generating system. Furthermore, $u = \sum_{(l,i)} u_{l,i} \psi_{l,i}$ is a given function in prewavelet basis representation. We search for $u = \sum \hat{u}_{l,i} \phi_{l,i}$, the representation in the hierarchical linear basis. As already motivated, a representation in the generating system E can be written down using the prewavelet mask as $u = \sum_{(l,i) \in E} \tilde{u}_{l,i} \phi_{l,i}$ where $(l, i) \in E$ now takes both, even and odd space indices. We get for *odd* i

$$\tilde{u}_{l,i} = a_0^{l,i} u_{l,i} + a_{\pm 2}^{l,i \pm 2} u_{l,i \pm 2} \quad (\text{A.25})$$

and for *even* i

$$\tilde{u}_{l,i} = a_{-1}^{l,i+1} u_{l,i+1} + a_1^{l,i-1} u_{l,i-1}. \quad (\text{A.26})$$

The generating system transformation rule (A.2) yields hat basis coefficients $\hat{u}_{l,i}$ for odd i ,

$$\hat{u}_{l,i} = \tilde{u}_{l,i} - \frac{1}{2} \tilde{u}_{l,i \pm 1} + t_{l+1,2i} - \frac{1}{2} t_{l+1,2(i \pm 1)} \quad (\text{A.27})$$

with temporary values $t_{l,i} := \tilde{u}_{l,i} + t_{l+1,2i}$. Inserting our expressions yields

$$\begin{aligned} \hat{u}_{l,i} &= a_0^{l,i} u_{l,i} + a_{\pm 2}^{l,i \pm 2} u_{l,i \pm 2} - \frac{1}{2} (a_{-1}^{l,i+2} u_{l,i+2} + a_1^{l,i} u_{l,i}) \\ &\quad - \frac{1}{2} (a_{-1}^{l,i} u_{l,i} + a_1^{l,i-2} u_{l,i-2}) + t_{l+1,2i} - \frac{1}{2} t_{l+1,2(i \pm 1)}, \end{aligned} \quad (\text{A.28})$$

$$\begin{aligned} \Rightarrow \hat{u}_{l,i} &= (a_0^{l,i} - \frac{1}{2} a_1^{l,i} - \frac{1}{2} a_{-1}^{l,i}) u_{l,i} + (a_2^{l,i+2} - \frac{1}{2} a_{-1}^{l,i+2}) u_{l,i+2} + \\ &\quad (a_{-2}^{l,i-2} - \frac{1}{2} a_1^{l,i-2}) u_{l,i-2} + t_{l+1,2i} - \frac{1}{2} t_{l+1,2(i \pm 1)}. \end{aligned} \quad (\text{A.29})$$

Now, we have expressed the hat basis coefficient $\hat{u}_{l,i}$ by means of three prewavelet coefficients, $u_{l,i}$ and $u_{l,i \pm 2}$, on the same level l and several contributions from higher levels, the $t_{l+1,2,*}$. The factor 2 indicates that we need only temporary values of even space index, i.e. only contributions $t_{l,i} = \tilde{u}_{l,i} + t_{l+1,2i}$ together with (A.26).

A possible implementation of (A.26) is to perform a loop from largest level down to lowest level. For each level, the values $t_{l+1,*}$ of the previous (larger) level need to be accessed and the linear combinations (A.29) need to be performed. This involves several case distinctions near the boundary and for small levels. Furthermore, the temporary values $t_{l,*}$ need to be computed for the next iteration. Technically, one can apply a breadth-first-search along each line with Algorithm 15 and store the points in the order of appearance (which uses level wise ordering). Then, one applies loops backwards through the structure, handling temporary arrays on the way.

The implementation needs special cases when it comes to the generating system on level $l = 0$, i.e. for the boundaries. It holds

$$\tilde{u}_{0,0} = a_0^{0,0} u_{0,0} + a_{-1}^{0,1} u_{0,1}, \quad (\text{A.30})$$

$$\tilde{u}_{0,1} = a_0^{0,1} u_{0,1} + a_1^{0,0} u_{0,0} \quad (\text{A.31})$$

as before. However, the Dirichlet prewavelet has $a_0 = 1$ and $a_{\pm 1} = 0$ on level 0, so the resulting values are just nodal values; nothing special has to be done. The Neumann prewavelet with $\psi_{0,0}^N \equiv 1$ can be implemented with

$$\hat{u}_{0,0} = \tilde{u}_{0,0} + t_{1,0}, \quad (\text{A.32})$$

$$\hat{u}_{0,1} = \tilde{u}_{0,1} + t_{1,2} - t_{1,0}, \quad (\text{A.33})$$

and (A.30) and (A.31).

Transformation From Hat Basis To Prewavelets

The inverse prewavelet transformation is more involved: we cannot simply reorder (A.29) to get an expression for $u_{l,i}$. Instead, we need to resolve all couplings between the $u_{l,i}$ on one level l . Since always three adjacent $u_{l,i}$ are coupled by (A.29), we can formulate this as a tri-diagonal linear equation system $\mathbf{A}\mathbf{u}_l = \mathbf{f}_l$ where \mathbf{u}_l is a vector of all prewavelet coefficients on level l and the right hand side \mathbf{f}_l is defined by

$$f_{l,i} := \hat{u}_{l,i} - t_{l+1,2i} + \frac{1}{2}t_{l+1,2(i-1)} + \frac{1}{2}t_{l+1,2(i+1)}. \quad (\text{A.34})$$

Algorithmically, this can be carried out during one level wise grid traversal from highest level down to lowest level: for every level, the right-hand-side \mathbf{f}_l needs to be assembled and the linear equation system needs to be solved for $u_{l,i}$. Then, the temporary values $t_{l,i}$ need to be computed for the next iteration. Since solving a tridiagonal system is possible in linear time (compare [PFTV92]), the complete transformation can be performed in linear time with respect to the grid size.

We are now ready to switch between nodal values and prewavelet coefficients in one dimension by applying the different transformations successively. The d -dimensional case can be implemented using a loop over the dimension: $2d$ grid traversals are sufficient to perform the transformations. We want to stress however, that the transformations do not commute arbitrarily: one should apply all d hat from/to nodal transformations separated from the d prewavelet from/to hat transformations (the direct transformation nodal from/to prewavelets needs the Unidirectional Principle with total runtime $\mathcal{O}(2^d N)$, compare Section 2.2.2).

Remarks About The Adaptive Case

While adaptive grid compression based on prewavelet coefficients produces good results, there is one major complication for the hierarchical transformations: the spaces spanned by the hierarchical hat basis and the prewavelets are no longer the same. Thus, it

might still be possible to compute nodal values at grid points, but there is no 1 : 1 transformation between the two basis sets. One possibility is to introduce intermediate nodes into the grid: for every grid point (l, i) , introduce $(l, i+j)$ for $j \in \{-4, -2, +2, +4\}$ unless the point exists already or does not belong to the basis and set their hierarchical coefficients to 0. Then, apply the transformations to existing points only. This allows to compute nodal values at every grid point out of prewavelet coefficients. However, the backwards transformation from nodal basis (or hat basis) to prewavelet coefficients is different from what one would expect: it does *not* yield the same as if one would apply it to a regular grid followed by a grid compression: the prewavelet compression contains information of finer mesh widths as well.

This approach with intermediate nodes has been suggested in [Feu05] as extension for the Unidirectional Principle to implement matrix-vector-products on adaptive index sets with prewavelets. However, the approach might still introduce an error as pointed out by Andreas Zeiser (private communication): it only yields an approximate matrix-vector-product for adaptive index sets. The issue of intermediate nodes is discussed in Section A.4.1.

A.2.4 The Haar Wavelet

The Haar wavelet basis is one of the most simple wavelets. It is a piecewise constant wavelet and defined as linear combination of adjacent piecewise constant scaling functions $\phi_{l,i}$. We denote the Haar basis functions by $\psi_{l,i}$ and the scaling functions by $\phi_{l,i}$,

$$\phi_{l,i}(x) = \begin{cases} 0 & x < x_{l,i}, \\ 1 & x \in [x_{l,i}, x_{l,i+1}) \\ 0 & x \geq x_{l,i+1}. \end{cases} \quad (\text{A.35})$$

Note that every function vanishes on the right boundary point, $\phi_{l,i}(1) = 0$ and there is no usable scaling function attached to the right boundary (in other words: the right boundary does not belong to the basis points of our piecewise constant bases and is ignored throughout this section).

The Haar basis functions are then defined by

$$\psi_{l,i} := -\phi_{l,i} + \phi_{l,i-1} = -2\phi_{l,i} + \phi_{l-1, \frac{i-1}{2}}, \quad (\text{A.36})$$

$$\psi_{0,0} := \phi_{0,0} \equiv 1. \quad (\text{A.37})$$

It constitutes an L_2 orthogonal wavelet for which classical wavelet transformations exist, see [Haa10] and [Chu92]. However, it can also be formulated efficiently in terms of the sparse grid Unidirectional Principle as follows: we decompose the transformation into one transformation from Haar wavelet to a simple multiscale piecewise constant basis (bottom-up) and one transformation from multiscale constant to nodal values (top-down). We provide these two algorithms and their inverse transformations separately to get optimal runtime $\mathcal{O}(dN)$ in the Unidirectional Principle for sparse grids (Section 2.2.2).

The transformation Haar Basis to Constant Multilevel

To eliminate the redundancies of the generating system, we employ the relation

$$\phi_{l,i} = \phi_{l-1,i/2} - \phi_{l,i+1} \quad (i \text{ even}). \quad (\text{A.38})$$

Taking (A.36) and (A.38) together provides the necessary ingredients for the transformation to the multi-level piecewise constant basis. Assuming we are given $u = \sum u_{l,i} \psi_{l,i}$ in the Haar basis, we are searching for a representation $u = \sum \bar{u}_{l,i} \phi_{l,i}$ in the multi-level piecewise constant basis.

We define local stack variables $t_{l,i} := t_{l+1,2i-1} + u_{l,i}$ where the first contribution is the $+\phi_{l-1,i/2}$ of (A.38) whereas the second term is the $+\phi_{l-1,(i-1)/2}$ summand from (A.36). This allows to formulate

$$\bar{u}_{l,i} = -2u_{l,i} + t_{l,2i+1} - t_{l,2i-1} \quad (\text{A.39})$$

where the -2 comes from the Haar wavelet and the $-t_{l,i-1}$ from the generating system elimination. The weight -2 must be changed to $+1$ for the left boundary $(0, 0)$.

The final procedure is summarized in Algorithm 20.

Algorithm 20 Hierarchical Transformation Haar Basis $u_{\mathbf{l},\mathbf{i}} \leftrightarrow \bar{u}_{\mathbf{l},\mathbf{i}}$ multilevel piecewise constant in direction m , to be used as operation inside of bottom-up (Algorithm 14).

Operation on boundary:

Input: StackData from $(1, 1)$

1: $(\mathbf{k}, \mathbf{j}) := (\mathbf{l}, \mathbf{i}), (k_m, j_m) := (0, 0)$

2: $u_{\mathbf{k},\mathbf{j}} := \bar{u}_{\mathbf{k},\mathbf{j}} + \text{StackData}$ or, for backwards transformation, $\bar{u}_{\mathbf{k},\mathbf{j}} := u_{\mathbf{k},\mathbf{j}} + \text{StackData}$

3: $(\mathbf{k}, \mathbf{j}) := (\mathbf{l}, \mathbf{i}), (k_m, j_m) := (0, 1), u_{\mathbf{k},\mathbf{j}} := \bar{u}_{\mathbf{k},\mathbf{j}} := 0$ // unused.

Operation on inner nodes (\mathbf{l}, \mathbf{i}) :

Input: StackDataLeft and StackDataRight

Output: StackData for father

1: **if** transform Haar to piecewise constant, $u_{\mathbf{l},\mathbf{i}} \rightarrow \bar{u}_{\mathbf{l},\mathbf{i}}$ **then**

2: $\bar{u}_{\mathbf{l},\mathbf{i}} := -2u_{\mathbf{l},\mathbf{i}} + \text{StackDataRight} - \text{StackDataLeft}$

3: $\text{StackData} := \text{StackDataLeft} + u_{\mathbf{l},\mathbf{i}}$

4: **else** // inverse $\bar{u}_{\mathbf{l},\mathbf{i}} \rightarrow u_{\mathbf{l},\mathbf{i}}$

5: $u_{\mathbf{l},\mathbf{i}} := \frac{1}{2}(-\bar{u}_{\mathbf{l},\mathbf{i}} + \text{StackDataLeft} - \text{StackDataRight})$

6: $\text{StackData} := \frac{1}{2}(\text{StackDataLeft} + \bar{u}_{\mathbf{l},\mathbf{i}} + \text{StackDataRight})$

7: **end if**

The backwards transformation from Constant Multilevel to Haar Basis

Now, we suppose we are given a representation in the piecewise constant multi-level basis, $u = \sum \bar{u}_{l,i} \phi_{l,i}$, and search for a representation in the Haar basis, $u = \sum u_{l,i} \psi_{l,i}$.

A Technical Reference

Reordering (A.36) for odd i yields

$$\phi_{l,i} = -\frac{1}{2}\psi_{l,i} + \frac{1}{2}\phi_{l-1,\frac{i-1}{2}} \quad (\text{A.40})$$

whereas the equation for even i is

$$\phi_{l,i} = \phi_{l-1,i/2} - \phi_{l,i+1} = \phi_{l-1,i/2} + \frac{1}{2}(\psi_{l,i+1} - \phi_{l-1,i/2}) = \frac{1}{2}\phi_{l-1,i/2} + \frac{1}{2}\psi_{l,i+1}. \quad (\text{A.41})$$

The idea is perform a bottom-up traversal and apply these equations until every $\phi_{l,i}$ has been expressed by the correct $\psi_{l,i}$, until we finally arrive at $\phi_{00} = \psi_{00} = 1$. Formulating this as recursive procedure, we get with temporary stack variables for even i ,

$$t_{l,i} := \frac{1}{2}\bar{u}_{l,i+1} + \frac{1}{2}t_{l+1,2i} + \frac{1}{2}t_{l+1,2i+1} \quad (\text{A.42})$$

and for odd i ,

$$t_{l,i} := t_{l+1,2i}, \quad (\text{A.43})$$

an expression for the desired Haar basis coefficients:

$$u_{l,i} = -\frac{1}{2}\bar{u}_{l,i} + \frac{1}{2}t_{l+1,2(i-1)} - \frac{1}{2}t_{l+1,2i}. \quad (\text{A.44})$$

The origin of the diagonal weight $-\frac{1}{2}\bar{u}_{l,i}$ is quite clear while the other two terms handle the $\frac{1}{2}\phi_{l-1,\frac{i-1}{2}}$ part communicated from the direct sons. Remember that each of them has to be expressed as linear combination of ψ_{kj} by applying the simplification rules as often as necessary. For $(l,i) = (0,0)$, we get $u_{0,0} = \bar{u}_{0,0} + t_{1,1}$.

The final procedure is summarized in Algorithm 20.

The Transformation between Nodal Values and Multilevel Constant Basis

We continue to provide transformations from nodal values $n_{l,i} = u(x_{l,i})$ on all grid points, $(l,i) \in G$, to a piecewise constant multi-level basis representation, $u = \sum \bar{u}_{l,i}\phi_{l,i}$. As for the piecewise hat multi-level basis, every basis coefficient $u_{l,i}$ is the difference between the nodal value $u(x_{l,i})$ and the (piecewise constant) interpolated value on previous levels,

$$\bar{u}_{l,i} = u(x_{l,i}) - I_{G_{l-1}}u(x_{l,i}) \quad (\text{A.45})$$

where $G_{l-1} := \{(k,j) \in G \mid k \leq l-1\}$ denotes the parts in G which are hierarchically higher than (l,i) . Since in this case, $I_{G_{l-1}}u$ is piecewise constant on $\text{supp } \phi_{l,i} = [x_{l,i-1}, x_{l,i+1})$ and it interpolates u on G_{l-1} , we only need the nodal value $x_{l,i-1}$ to formulate

$$\bar{u}_{l,i} = u(x_{l,i}) - u(x_{l,i-1}). \quad (\text{A.46})$$

The required nodal value can be transported from father to sons during a grid traversal from top ($l=0$) to bottom. To this end, we introduce temporary stack variables

$$t_{l,i} := \begin{cases} n_{l,i} & \text{if } (l,i) \text{ belongs to the basis (i.e. } l=0, i=0 \text{ or } i \text{ odd),} \\ t_{l-1,i/2} & \text{if } (l,i) \text{ is a generating system node} \end{cases} \quad (\text{A.47})$$

and write

$$\bar{u}_{l,i} = u(x_{l,i}) - t_{l-1,i/2}. \quad (\text{A.48})$$

Consequently, the inverse transformation from piecewise constant multi-level to nodal values is

$$u(x_{l,i}) = n_{l,i} = \bar{u}_{l,i} + t_{l-1,i/2}. \quad (\text{A.49})$$

Both transformations can be carried out during a top–down traversal through the binary grid structure, where at every step, one value $t_{l,i}$ is communicated from father to son (similarly to the hat basis transformation, compare Algorithm 19).

Please keep in mind that this transformation does *not* yield the nodal value $u(x_{0,1}) = u(1)$. If required, the right boundary point needs to be extrapolated. Unfortunately, this extrapolation step might become more involved for higher dimensions.

Finally, a transformation from nodal values to the Haar wavelet is a composition of a transformation from nodal to piecewise constant multi-level (top–down), and from there to the Haar basis (bottom–up). For the multi–dimensional case, this can be carried out during $2d$ separate grid traversals, two for every direction. However, the order of the operation matters: at first, all directions have to be transformed to piecewise constant multi-level and *afterwards*, all directions have to be transformed to the Haar basis. Similar statements hold for the backwards transformations. This speciality is due to the fact the top–down and bottom–up correspond to decompositions of the complete transformation matrix into upper triangular and lower triangular matrices – and the Unidirectional principle states that $\mathcal{O}(2^d)$ grid traversals are necessary to allow data transport *and* commutation between the upper– and lower triangular parts. Our approach needs only $\mathcal{O}(dN)$ time.

A.2.5 Higher Order Hierarchical Polynomial Bases

While piecewise constant and piecewise linear basis functions require only few degrees of freedom (grid points), higher order (spline) polynomials have to be defined using additional information. One major advantage of piecewise constant and piecewise linear hierarchical basis sets is the property of nested supports: information is local and the operations can be performed in linear time with respect to the degrees of freedom. The hierarchical polynomial bases proposed by Bungartz [Bun98] have both, high order *and* local support. The additional degrees of freedom to define higher order polynomials are extracted from the *hierarchy*.

Basis Function Definition

We will summarize the definition of hierarchical polynomial bases according to [Ach03] and the hierarchical transformations by following [Bun98].

The idea for the definition of a polynomial basis function $\phi_{l,i}^{(p)}$ of degree $p \geq 2$ is to get a set of $p + 1$ points $P_{l,i}^{(p)} := \{x_k^{l,i}, k = 0, \dots, p\}$, chosen from $x_{l,i}$ and its hierarchical ancestors on lower levels, such that

$$\phi_{l,i}^{(p)}(x_{l,i}) = 1 \text{ and } \phi_{l,i}^{(p)}(x_k^{l,i}) = 0 \text{ for } x_k^{l,i} \neq x_{l,i}. \quad (\text{A.50})$$

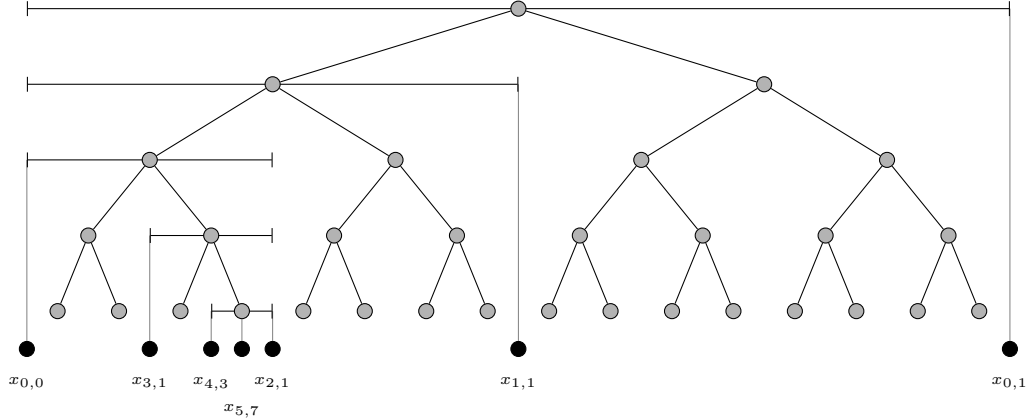


Figure A.4: The defining points $P_{l,i}^{(6)}$, shown as ‘●’, which are used to create the hierarchical polynomial basis function $\phi_{5,7}^{(6)}$.

These $(p + 1)$ conditions uniquely define a polynomial of degree p . To get compact supports, the basis function is truncated to zero outside of

$$J_{l,i} := [x_{l,i-1}, x_{l,i+1}]. \quad (\text{A.51})$$

Before we go into details, we note several aspects. At first, it is not possible to provide arbitrary degrees at every grid point. For example the point $l = 1, i = 1$ has two further points on lower levels, namely $(0, 0)$ and $(0, 1)$. Thus, at most a polynomial of degree $p = 2$ can be found for $(1, 1)$. We will see later that a polynomial of degree p requires $l \geq p - 1$. Furthermore, the truncation to 0 outside of $J_{l,i}$ is certainly not differentiable. The basis proposed in [Bun98] is, indeed, locally of degree (up to) p and only continuous at grid points.

The defining set of points $P_{l,i}^{(p)} = \{x_k^{l,i}\}$ consists of $x_{l,i}$, its two direct neighbors $x_{l,i-1}$ and $x_{l,i+1}$, and the direct neighbors of each of the $(p - 2)$ following ancestors. Each ancestor has two direct neighbors, but, one of them is already contained in $P_{l,i}^{(p)}$ due to the larger levels. This is illustrated in Figure A.4: the basis polynomial of degree $p = 6$, $\phi_{5,7}^{(6)}$ requires the points

$$P_{5,7}^{(6)} := \{x_{0,0}, x_{3,1}, x_{4,3}, x_{5,7}, x_{2,1}, x_{1,1}, x_{0,1}\} \quad (\text{A.52})$$

$$= \{x_{5,0}, x_{5,4}, x_{5,6}, x_{5,7}, x_{5,8}, x_{5,16}, x_{5,32}\}. \quad (\text{A.53})$$

Besides $x_{l,i}$ and its two neighbors, there is always one point per ancestor. To get a total of $p + 1$ points, $p - 2$ ancestors need to be visited. This limits the polynomial degree, based on the level l . Since $l = 0$ does not contribute further points (the points on level $l = 0$ are neighbors of $(1, 1)$), there are exactly $l - 1$ ancestors for every point (l, i) . Thus, for fixed l , we have $p - 2 \leq l - 1 \Rightarrow p \leq l + 1$. It is possible to collect these points iteratively using Algorithm 21.

Algorithm 21 Compute the set of required points $P_{l,i}^{(p)}$ for given polynomial degree p and point (l, i) .

Require: Degree $p \geq 2$

Require: A point (l, i) with $l \geq p - 1$

```

1:  $P^{(p)} = \{x_{l,i}, x_{l,i-1}, x_{l,i+1}\}$ 
2: while  $|P^{(p)}| < p + 1$  do
3:    $(\bar{l}, \bar{i}) := (l, i)$ 
4:   set  $(l, i)$  to its father
5:   if  $(\bar{l}, \bar{i})$  is a left son then //  $\Leftrightarrow \bar{i}/2$  is even (or we have  $(0, 0)$ )
6:      $P^{(p)} := P^{(p)} \cup \{x_{l,i+1}\}$  // We came from left, take right neighbor
7:   else
8:      $P^{(p)} := P^{(p)} \cup \{x_{l,i-1}\}$  // We came from right, take left neighbor
9:   end if
10: end while
11: return  $P^{(p)}$ 

```

The polynomial basis function $\phi_{l,i}$ is then defined by

$$\phi_{l,i}(x) := \begin{cases} \prod_{\substack{x_k \in P_{l,i}^{(p)} \\ x_k \neq x_{l,i}}} \frac{(x - x_k)}{(x_{l,i} - x_k)} & x \in [x_{l,i-1}, x_{l,i+1}], \\ 0 & \text{else,} \end{cases} \quad (\text{A.54})$$

compare [Ach03]. The basis functions are uniformly bounded over all (l, i) and do not suffer from oscillations which occur typically for high order polynomial interpolation since the support restriction uses only the “tame” part of the polynomial, see [Bun98] for details.

Hierarchical Transformations For Hierarchical Polynomial Bases

As for all other hierarchical basis constructions, the coefficients for hierarchical basis polynomials of degree p are the difference between nodal values and the interpolated value of the ancestors,

$$u_{l,i}^{(p)} = u(x_{l,i}) - I_{G_{l-1}}(x_{l,i}). \quad (\text{A.55})$$

The interpolated value up to and including level $(l - 1)$ is now a piecewise polynomial spline which interpolates u at G_{l-1} . Here enters the restriction of polynomial degrees discussed in the previous section: it holds $p \leq l + 1$ in general and thus, $I_{G_{l-1}}$ has at most polynomial degree $\tilde{p} = l$. We conclude that

$$u_{l,i}^{(p)} = u_{l,i}^{(\tilde{p})} = u_{l,i}^{(l)} \text{ for } p > l. \quad (\text{A.56})$$

For example, the quadratic coefficient on level $l = 1$, $(1, 1)$ is identical to the linear coefficient,

$$u_{1,1}^{(2)} = u_{1,1}^{(1)}. \quad (\text{A.57})$$

A Technical Reference

The key observation to actually compute the coefficients is a result due to [Bun98]: the hierarchical coefficients of order p at one point can be computed out of two coefficients of order $(p - 1)$. By iteration, this allows to compute order p hierarchical coefficients based on *linear* hierarchical coefficients on different levels. We denote the hierarchical coefficient for polynomial bases of order p by $u_{l,i}^{(p)}$. Then it holds [Bun98, Theorem 4.3] for $l \geq p \geq 2$ that

$$u_{l,i}^{(p)} = u_{l,i}^{(p-1)} - \alpha_{l,i}^{(p)} \cdot u_{\mathcal{F}(l,i)}^{(p-1)}. \quad (\text{A.58})$$

Here, $\mathcal{F}(l, i)$ denotes the multi-index of the father of (l, i) . Furthermore, $\alpha_{l,i}^{(p)}$ is a weight which depends on the set $P_{l,i}^{(p)}$ and one further hierarchical ancestor, but not on the basis coefficients. For the case $l < p$, a combination of (A.56) and (A.58) allows to compute the transformation and the special case $p = 1$ is just the linear hierarchical basis.

The weight is given by

$$\alpha_{l,i}^{(p)} := \frac{x_{\mathcal{F}(l,i)} - x_{l,i}}{x_{p+1} - x_{\mathcal{F}(l,i)}} \cdot \prod_{x_k \in P_{l,i}^{(p)} \setminus \{x_{l,i}, x_{\mathcal{F}(l,i)}\}} \frac{x_{l,i} - x_k}{x_{\mathcal{F}(l,i)} - x_k} \quad (\text{A.59})$$

where the additional point x_{p+1} can be obtained by Algorithm 21 as well: the algorithm needs one further loop iteration to yield x_{p+1} . It should be noted that $\alpha_{l,i}^{(p)}$ can be computed with integer arithmetics if we write all involved coordinates $x_k \in P_{l,i}^{(p)}$, $k = 0, \dots, p$ and x_{p+1} as $x_k = j_k 2^{-l}$ for adequately chosen space indices j_k . This has already been suggested in (A.53) where we provided all elements of $P_{5,7}^{(6)}$ on level $l = 5$. If furthermore $x_{l,i} = m 2^{-l}$ and $x_{\mathcal{F}(l,i)} = n 2^{-l}$, the weight (A.59) simplifies to

$$\alpha_{l,i}^{(p)} = \frac{n - m}{j_{p+1} - n} \prod_{\substack{k=0 \\ k \notin \{n,m\}}}^p \frac{m - j_k}{n - j_k}; \quad (\text{A.60})$$

only the final division needs to be done in floating point arithmetics. The simplest case is given for $p = 2$, the piecewise quadratic hierarchical basis: the basis functions are defined by $x_{l,i}$ and $x_{l,i \pm 1}$ and the additional point x_{p+1} is nothing but $x_{\mathcal{F}(l,i)}$. There are just two possible configurations: either (l, i) is a left son or it is a right son and all j_k are known a priori. In both cases, (A.60) simplifies to $\alpha_{l,i}^{(p)} \equiv \frac{1}{4}$ for every (l, i) , $l \geq 2$. Starting with $p = 3$, the weight depends on (l, i) .

Now that we have computed the weights $\alpha_{l,i}^{(p)}$, the transformation rules (A.56) and (A.58) can be computed during one top-down traversal through the data structure. The transformation from nodal values $n_{l,i}$ to hierarchical polynomial coefficients $u_{l,i}^{(p)}$ consists of the following steps: for every visited node, check if coefficients of degree p are possible and use $\tilde{p} = l$ instead if $p > l$ according to (A.56). Then, compute all hierarchical coefficients $u_{l,i}^{(j)}$ for $j = 1, 2, \dots, \tilde{p}$ where $u_{l,i}^{(1)}$ is the linear hierarchical coefficient defined by

$$u_{l,i}^{(1)} = n_{l,i} - \frac{1}{2} (n_{l,i-1} + n_{l,i+1}), \quad (\text{A.61})$$

see also (A.11). Higher degrees $\tilde{p} > 1$ follow by iterating (A.58) using the weights (A.60). This is summarized in Algorithm (22), together with the required values from the father and the communication of values to the direct ancestors.

To get boundary conditions right, the basis needs to be complemented with two further *linear* basis functions as done for the hat basis in Section A.2.2. The boundary modifications impose *no* changes on Algorithm 22 since it holds $\tilde{p} = 1$ on level $l = 1$ so only linear hierarchical coefficients are involved anyway. Note that Algorithm 22 can be formulated by means of the generic pre-order Algorithm 13 with `StackData` containing all required input values.

Algorithm 22 Recursive top-down procedure to compute hierarchical polynomial basis coefficients of order (up to) p , $u_{l,i}^{(p)}$ out of nodal values $n_{l,i}$ or backwards.

Require: (l, i) , the current index

Require: a reference to the current grid value $\tilde{u}_{l,i}$

Require: From father: nodal values $n_{l,i-1}$ and $n_{l,i+1}$

Require: From father: $u_{\mathcal{F}(l,i)}^{(j)}$ for $j = 1, \dots, \tilde{p} - 1$ (with $\tilde{p} := \min\{p, l\}$).

```

1: if transform to hierarchical coefficients then
2:    $n_{l,i} := \tilde{u}_{l,i}$ 
3:    $u_{l,i}^{(1)} := n_{l,i} - \frac{1}{2}(n_{l,i-1} + n_{l,i+1})$ 
4:   for  $j = 2$  to  $\tilde{p}$  do
5:      $u_{l,i}^{(j)} := u_{l,i}^{(j-1)} - \alpha_{l,i}^{(j)} \cdot u_{\mathcal{F}(l,i)}^{(j-1)}$ 
6:   end for
7:   set output value  $\tilde{u}_{l,i} := u_{l,i}^{(\tilde{p})}$ 
8: else
9:    $u_{l,i}^{(\tilde{p})} := \tilde{u}_{l,i}$ 
10:  for  $j = \tilde{p}$  to 2 do
11:     $u_{l,i}^{(j-1)} := u_{l,i}^{(j)} + \alpha_{l,i}^{(j)} \cdot u_{\mathcal{F}(l,i)}^{(j-1)}$ 
12:  end for
13:   $n_{l,i} := u_{l,i}^{(1)} + \frac{1}{2}(n_{l,i-1} + n_{l,i+1})$ 
14:  set output value  $\tilde{u}_{l,i} := n_{l,i}$ 
15: end if
16: Recurse into left son  $(l + 1, 2i - 1)$  with
     $n_{l+1,2i-2} := n_{l,i-1}$ ,  $n_{l+1,2i} := n_{l,i}$  and  $u_{l,i}^{(j)}$ ,  $j = 1, \dots, \tilde{p}$ .
17: Recurse into right son  $(l + 1, 2i + 1)$  with
     $n_{l+1,2i} := n_{l,i}$ ,  $n_{l+1,2i+2} := n_{l,i+1}$  and  $u_{l,i}^{(j)}$ ,  $j = 1, \dots, \tilde{p}$ .

```

Note that the weight $\alpha_{l,i}^{(p)}$ can be precomputed by Algorithm 21 according to (A.59) and (A.60) by collecting one more point.

A.3 One-Dimensional Matrix Vector Products Algorithms

A.3.1 The Mass Matrix for the Hierarchical Hat Basis

Computation of the mass matrix for the hierarchical hat basis on sparse grids requires the Unidirectional Principle and its splitting into top-down and bottom-up parts. The required one-dimensional algorithms are summarized in Algorithm 24 (top-down) and Algorithm 23 (bottom-up) as matrix-vector-products. The algorithms are due to [Bal94].

Algorithm 23 One-dimensional matrix vector product bottom-up accumulation of $v = Bu$ for the mass matrix in hierarchical hat basis representation. The algorithm needs to be invoked during the post-order Algorithm 14.

Part I: inner nodes:

Input: inner node (l, i) and $u_{l,i}$

Input: Stack data of left son $S_l = [S_l.\text{left}, S_l.\text{right}]$

Input: Stack data of right son $S_r = [S_r.\text{left}, S_r.\text{right}]$

Output: Stack data to be stored for (l, i) , S

1: $v_{l,i} := -(S_l.\text{right} + S_r.\text{left})$

2: $m := \frac{1}{2} \cdot (h_l \cdot u_{l,i} + v_{l,i})$

3: $S.\text{left} := S_l.\text{left} - m$

4: $S.\text{right} := S_r.\text{right} - m$

5: **return** S

Part II: boundary nodes:

Input: boundary (l, i) and $u_{l,i}$

Input: Stack data of point $(1, 1)$ $S = [S.\text{left}, S.\text{right}]$

1: **if** $\phi_{0,0} = x$ and $\phi_{0,1} = 1 - x$ **then**

2: $v_{0,0} := -S.\text{left}$

3: $v_{0,1} := -S.\text{right}$

4: **else**

5: $v_{0,0} := -(S.\text{left} + S.\text{right}) + \frac{1}{2}u_{0,1}$

6: $v_{0,1} := -S.\text{right}$

7: **end if**

// $\phi_{0,0} = 1$ and $\phi_{0,1} = x$

A.3.2 The Mass Matrix for the Prewavelet Basis

Due to the semi-orthogonality of the prewavelet basis with respect to the L_2 inner product, $(\psi_{l,i}, \psi_{k,j}) = 0$ for $l \neq k$ and $l, k > 0$, only a handful of non-vanishing matrix entries need to be computed. We write $m_{(l,i),(k,j)}$ for an entry.

The matrix entries for the Dirichlet prewavelets are summarized in the following list.

Algorithm 24 One-dimensional matrix vector product top-down accumulation of $v = Tu$ for the mass matrix in hierarchical hat basis representation. The algorithm needs to be invoked during a pre-order Algorithm 13.

Part I: inner nodes:

Input: inner node (l, i) and $u_{l,i}$

Input: Stack data of father, $S = [S.\text{left } S.\text{right}]$

Output: Stack data to be communicated to left son (S_l) and right son (S_r)

- 1: $n := \frac{1}{2}(S.\text{left} + S.\text{right})$
- 2: $v_{l,i} := \frac{2}{3} \cdot h_l \cdot u_{l,i} + h_l \cdot n$
- 3: $t := u_{l,i} + n$ // nodal value on its father
- 4: $S_l.\text{left} := S.\text{left}$
- 5: $S_l.\text{right} := t$
- 6: $S_r.\text{left} := t$
- 7: $S_r.\text{right} := S.\text{right}$

Part II: boundary nodes:

Input: boundary (l, i) and $u_{l,i}$

Output: Stack data to be stored for $(1, 1)$, $S = [S.\text{left}, S.\text{right}]$

- 1: $S.\text{left} = u_{0,0}$
- 2: **if** $\phi_{0,0} = x$ and $\phi_{0,1} = 1 - x$ **then**
- 3: $v_{0,0} := \frac{1}{3} \cdot u_{0,0} + \frac{1}{6} \cdot u_{0,1}$
- 4: $v_{0,1} := \frac{1}{3} \cdot u_{0,1} + \frac{1}{6} \cdot u_{0,0}$
- 5: $S.\text{left} = u_{0,0}$ and $S.\text{right} = u_{0,1}$
- 6: **else** // $\phi_{0,0} = 1$ and $\phi_{0,1} = x$
- 7: $v_{0,0} := u_{0,0}$
- 8: $v_{0,1} := \frac{1}{3} \cdot u_{0,1} + \frac{1}{2} \cdot u_{0,0}$
- 9: $S.\text{left} = u_{0,0}$ and $S.\text{right} = u_{0,0} + u_{0,1}$

10: **end if**

Note that it has non-vanishing terms between level 0 and $l > 0$:

$$\begin{array}{ll}
 m_{(0,0),(0,0)} = \frac{1}{3}, & m_{(0,0),(0,1)} = \frac{1}{6}, \\
 m_{(1,1),(1,1)} = \frac{1}{3}, & m_{(1,1),(0,i)} = \frac{1}{4}, \\
 m_{(2,1),(2,1)} = m_{(2,3),(2,3)} = \frac{44}{75}h_2, & m_{(2,1),(2,3)} = \frac{1}{25}, \\
 m_{(2,1),(0,i)} = \frac{4}{10}h_2, & m_{(l,1),(l,1)} = \frac{44}{75}h_l, \\
 m_{(l,1),(l,3)} = \frac{11}{75}h_l, & m_{(l,1),(l,5)} = -\frac{1}{75}h_l, \\
 m_{(l,1),(0,0)} = \frac{4}{10}h_l, & m_{(l,i),(l,i)} = \frac{18}{25}h_l, \\
 m_{(l,i),(l,i\pm 2)} = \frac{2}{15}h_l, & m_{(l,i),(l,i\pm 4)} = -\frac{1}{75}h_l,
 \end{array}$$

A Technical Reference

and $m_{(l,2^{l-1}),*}$ is symmetrically to $m_{(l,1),*}$. The remaining entries also follow due to symmetry.

Similarly, we collect the matrix entries for the Neumann prewavelet, which is also orthogonal between level 0 and $l > 0$. The non-vanishing entries are, up to symmetry,

$$\begin{aligned}
 m_{(0,0),(0,0)} &= 1, & m_{(0,0),(0,1)} &= \frac{1}{3}, \\
 m_{(1,1),(1,1)} &= \frac{1}{3}, & m_{(2,1),(2,1)} &= m_{(2,3),(2,3)} = \frac{16}{75}, \\
 m_{(2,1),(2,3)} &= \frac{2}{75}, & m_{(l,1),(l,1)} &= \frac{64}{75}h_l, \\
 m_{(l,1),(l,3)} &= \frac{3}{25}h_l, & m_{(l,1),(l,5)} &= -\frac{1}{75}h_l, \\
 m_{(l,i),(l,i)} &= \frac{18}{25}h_l, & m_{(l,i),(l,i\pm 2)} &= \frac{2}{15}h_l, \\
 m_{(l,i),(l,i\pm 4)} &= -\frac{1}{75}h_l, & m_{(l,2^{l-1}),*} &= \text{symmetrically to } m_{(l,1),*}.
 \end{aligned}$$

A.3.3 More on One-Dimensional Matrix-Vector-Products

Besides the already mentioned algorithms, there are general attempts to compute matrix multiplications with stiffness matrices for constant coefficients (i.e. second order and convective terms). For the hat basis, the second order terms (Laplacian) are trivial since the hat basis is orthogonal with respect to the corresponding one-dimensional inner product and results simply in a diagonal scaling (compare [Bal94]). Implementations for second order terms and convective terms for constant coefficients have been elaborated in [Feu05]. Implementations for (separable) variable coefficients have been presented in [Ach03], as well as other constant coefficient implementations with small basis support. The implementation for separable variable coefficients in this thesis is based on the hat basis: the matrix is assembled for the one-scale nodal basis and the hierarchical hat basis multiplication is realized by means of multi grid prolongations and restrictions, separated into top-down and bottom-up according to the discussion in Section 2.2.2. Since the one-scale basis also includes the generating system of Section A.2.1, the approach also eliminates generating system nodes. The resulting runtime for both, top-down and bottom-up is $\mathcal{O}(N)$ as is common for multigrid methods (note that the final matrix has finger structure with more than $\mathcal{O}(N)$ non-vanishing matrix elements, so a direct assembly leads to sub-optimal runtime requirements). Details about multi grid prolongations and restrictions are beyond the scope of this thesis, we refer the reader to standard text books.

A.4 Adaptive Refinement: Algorithmic Details

Algorithm 25 provides details about one adaptive refinement step for reference, and Algorithm 26 provides details about one grid compression step.

Algorithm 25 A single grid refinement step

Input: Grid G

Input: Target threshold $\epsilon = \epsilon^{(\text{abs})}$ or $\epsilon = \epsilon^{(\text{rel})}$ and function norm $\|\cdot\|$

Input: A discrete function $f \in \text{span}\{\phi_{\mathbf{l},\mathbf{i}} \mid (\mathbf{l}, \mathbf{i}) \in G\}$

Input: A boolean `bDimAdaptive` to support dimension adaptive refinement (see Algorithm 12)

Note that dimension adaptive grids use level components $l_m = -1, 0, 1, 2, \dots$

Output: The refined grid G^{refine} and the refined function f^{refine}

```

1: In case of  $\epsilon = \epsilon^{(\text{rel})}$ , compute  $\|f\|$  and use  $\epsilon := \epsilon^{(\text{rel})} \cdot \|f\|$ 
2:  $G^{\text{refine}} := G$ 
3: for  $(\mathbf{l}, \mathbf{i}) \in G$  do
4:     if  $|f_{\mathbf{l},\mathbf{i}}| \cdot \|\phi_{\mathbf{l},\mathbf{i}}\| \geq \epsilon$  then
5:         for  $m = 1$  to  $d$  do
6:             if bDimAdaptive=false or  $l_m \neq -1$  then
7:                  $(\mathbf{k}, \mathbf{j}) := (\mathbf{l}, \mathbf{i}), (k_m, j_m) := (l_m + 1, 2i_m - 1)$ .
8:                 Insert  $(\mathbf{k}, \mathbf{j})$  and all its ancestors into  $G^{\text{refine}}$ .
9:                  $(\mathbf{k}, \mathbf{j}) := (\mathbf{l}, \mathbf{i}), (k_m, j_m) := (l_m + 1, 2i_m + 1)$ .
10:                Insert  $(\mathbf{k}, \mathbf{j})$  and all its ancestors into  $G^{\text{refine}}$ .
11:            end if
12:        end for
13:    end if
14: end for
15:  $f^{\text{refine}} := f$  together with  $f_{\mathbf{l},\mathbf{i}} := 0$  for new nodes  $(\mathbf{l}, \mathbf{i}) \in G^{\text{refine}} \setminus G$ 
16: return  $G^{\text{refine}}$  and  $f^{\text{refine}}$ 

```

A.4.1 Remarks on Prewavelet Adaptivity

In principle, the prewavelet bases (see Section A.2.3) allow the same sort of adaptive refinement as for the hat basis. They even provide better error indicators due to their stability, [GO95]. However, the Unidirectional Principle of Section 2.2.2 which allows fast matrix-vector-product algorithms relies on small supports to provide storage of intermediate results on sparse grid points. It was suggested in [Feu05] that additional, intermediate transport nodes could be inserted on neighbor nodes to improve the data transport, i.e. to insert up to $4 \cdot d$ neighbor nodes $(l_m, i_m \pm 2)$ and $(l_m, i_m \pm 4)$ and their ancestors for every grid point (\mathbf{l}, \mathbf{i}) and every direction m . The resulting grid is thus still adaptive, but it will contain nodes which are not degrees of freedom, increasing the grid size from $\mathcal{O}(N)$ to about $\mathcal{O}(dN)$. This improves storage of intermediate nodes between successive stages of the Unidirectional Principle, compare [Feu05].

However, the approach might still introduce an error as pointed out by Andreas Zeiser (private communication): it only yields an approximate matrix-vector-product for adaptive index sets. Neighbors along the diagonals also need to be inserted to ensure a correct storage of intermediate results. The grid size will never exceed the regular sparse grid size, but the additional blow-up constant now grows exponentially with d .

Bibliography

- [Ach03] S. Achatz. *Adaptive finite Dünngitter-Elemente höherer Ordnung für elliptische partielle Differentialgleichungen mit variablen Koeffizienten*. Dissertation, Technische Universität München, 2003.
- [AS65] M. Abramowitz and I. A. Stegun, editors. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover Publications, New York: Dover, 1965.
- [BAH87] B. Bird, R. C. Armstrong, and O. Hassager. *Dynamics of Polymeric Liquids Volume 1 – Fluid Mechanics*. John Wiley & Sons, 2 edition, 1987.
- [Bal94] R. Balder. *Adaptive Verfahren für elliptische und parabolische Differentialgleichungen auf dünnen Gittern*. Dissertation, Technische Universität München, 1994.
- [Bel61] R. Bellmann. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.
- [BG99] H.-J. Bungartz and M. Griebel. A note on the complexity of solving Poisson’s equation for spaces of bounded mixed derivatives. *J. Complexity*, 15:167–199, 1999. Also as Report No 524, SFB 256, Univ. Bonn, 1997.
- [BG04] H.J. Bungartz and M. Griebel. Sparse grids. *Acta Numerica*, pp. 1-123, Cambridge University Press, 2004.
- [Bun92] H.-J. Bungartz. *Dünne Gitter und deren Anwendung bei der adaptiven Lösung der dreidimensionalen Poisson-Gleichung*. Dissertation, Technische Universität München, 1992.
- [Bun96] H.J. Bungartz. A unidirectional approach for d-dimensional finite element methods of higher order on sparse grids. *Copper Mountain Conference on Iterative Methods, Copper Mountain, CO, 8.-13. 4., 1996*.
- [Bun98] H. J. Bungartz. Finite elements of higher order on sparse grids. Habilitation, Technische Universität München, 1998.
- [Chu92] C. Chui. *An Introduction to Wavelets*. Academic Press, San Diego, 1992.
- [DLO07] Pascal Delaunay, Alexei Lozinski, and Robert G. Owens. Sparse tensor-product fokker-planck-based methods for nonlinear bead-spring chain models of dilute polymer solutions. In A. Bandrauk, M. C. Delfour, and C. Le

Bibliography

- Bris, editors, *CRM Proceedings and Lecture notes - High-Dimensional Partial Differential Equations in Science and Engineering*, volume 41. AMS, 2007.
- [DSS09] T. J. Dijkema, C. Schwab, and R. Stevenson. An adaptive wavelet method for solving high-dimensional elliptic pdes. *Journal Constructive Approximation*, 30(3):425–455, December 2009.
- [DSWW04] J. Dick, I. Sloan, X. Wang, and H. Woźniakowski. Liberating the weights. *Journal of Complexity*, 20(5):593–623, 2004.
- [ES81] B. Efron and C. Stein. The jackknife estimate of variance. *The Annals of Statistics*, 9(3):586–596, 1981.
- [Feu05] C. Feuersänger. Dünngitterverfahren für hochdimensionale elliptische partielle Differentialgleichungen. Diplomarbeit, Institut für Numerische Simulation, Universität Bonn, 2005.
- [FG09] C. Feuersänger and M. Griebel. Principal manifold learning by sparse grids. *Computing*, 85(4), August 2009. Also available as INS Preprint no 0801.
- [Gar04] J. Garcke. *Maschinelles Lernen durch Funktionsrekonstruktion mit verallgemeinerten dünnen Gittern*. Dissertation, Institut für Numerische Simulation, Universität Bonn, 2004.
- [GG03] T. Gerstner and M. Griebel. Dimension-Adaptive Tensor-Product Quadrature. *Computing*, 71(1):65–87, 2003.
- [GH10] M. Griebel and J. Hamaekers. Tensor product multiscale many-particle spaces with finite-order weights for the electronic Schrödinger equation. *Zeitschrift für Physikalische Chemie*, 224:527–543, 2010. Also available as INS Preprint no 0911.
- [Gla04] P. Glassermann. *Monte Carlo Methods in Financial Engineering*. Applications of Mathematics, Stochastic Modelling and Applied Probability. Springer, 2004.
- [GO95] M. Griebel and P. Oswald. Tensor product type subspace splitting and multilevel iterative methods for anisotropic problems. *Adv. Comput. Math.*, 4:171–206, 1995. Also as SFB-Bericht 342/15/94A, Institut für Informatik, TU München, 1994.
- [Gri98] M. Griebel. Adaptive sparse grid multilevel methods for elliptic PDEs based on finite differences. *Computing*, 61(2):151–179, 1998. Also as Proceedings Large-Scale Scientific Computations of Engineering and Environmental Problems, 7. June - 11. June, 1997, Varna, Bulgaria, Notes on Numerical Fluid Mechanics 62, Vieweg-Verlag, Braunschweig, M. Griebel, O. Iliev, S. Margenov and P. Vassilevski (editors).

- [Gri06] M. Griebel. Sparse grids and related approximation schemes for higher dimensional problems. In L. Pardo, A. Pinkus, E. Suli, and M.J. Todd, editors, *Foundations of Computational Mathematics (FoCM05)*, Santander, pages 106–161. Cambridge University Press, 2006.
- [Haa10] Alfred Haar. Zur Theorie der orthogonalen Funktionensysteme. *Mathematische Annalen*, 3:331–371, 1910.
- [Har10] H. Harbrecht. A finite element method for elliptic problems with stochastic input data. *Appl. Numer. Math.*, 60(3):227–244, 2010.
- [Heg03] M. Hegland. Adaptive sparse grids. *ANZIAM J.*, 44(E):C335–C353, 2003.
- [Hoe48] W. Hoeffding. A class of statistics with asymptotically normal distribution. *Annals of Math. Statist.*, 19:293–325, 1948.
- [Hol08] M. Holtz. *Sparse Grid Quadrature in High Dimensions with Applications in Finance and Insurance*. Dissertation, Institut für Numerische Simulation, Universität Bonn, 2008.
- [HSS08] H. Harbrecht, R. Schneider, and C. Schwab. Sparse second moment analysis for elliptic problems in stochastic domains. *Numer. Math.*, 109(3):385–414, 2008.
- [Hul09] A. Hullmann. Schnelle Varianten des Generative Topographic Mapping. Diplomarbeit, Institut für Numerische Simulation, Universität Bonn, December 2009.
- [Kna00] S. Knapek. *Approximation und Kompression mit Tensorprodukt-Multiskalen-Approximationsräumen*. Dissertation, Universität Bonn, Institut für Angewandte Mathematik, 2000.
- [Kne08] D. Knezevic. *Analysis and Implementation of Numerical Methods for Simulating Dilute Polymeric Fluids*. Phd thesis, University of Oxford, 2008.
- [Kos01] F. Koster. *Multiskalen-basierte Finite-Differenzen-Verfahren auf adaptiven dünnen Gittern*. Dissertation, Rheinische Friedrich-Wilhelms-Universität Bonn, 2001.
- [KSWW09] F. Y. Kuo, I. H. Sloan, G. W. Wasilkowski, and H. Woźniakowki. On Decompositions of Multivariate Functions. *Mathematics of Computation*, 79(270):953–966, Apr 2009.
- [LS96] J. D. Lafferty and B. Suhm. Cluster Expansions and Iterative Scaling for Maximum Entropy Language Models. In K. Hanson and R. Silver, editors, *Maximum Entropy and Bayesian Methods*. Kluwer Academic Publishers, 1996.

Bibliography

- [MB05] M. J. Mohlenkamp and G. Beylkin. Algorithms for numerical analysis in high dimensions. *SIAM J. Sci. Comp.*, 26(6):2133–2159, 2005.
- [Mer05] T. Mertens. Optionspreisbewertung mit dünnen Gittern. Diplomarbeit, Institut für Numerische Simulation, Universität Bonn, 2005.
- [MgF07] A. Meyer (geb. Fischer). Interpolation von vektorwertigen Funktionen mit adaptiven dünnen Gittern. Diplomarbeit, Institut für Numerische Simulation, Universität Bonn, December 2007.
- [Mor95a] B. Moro. Fast computation of cumulative normal distribution function. working paper, TMG Financial Products, Greenwich, Connecticut, 1995.
- [Mor95b] B. Moro. *The Full Monte*. RISK, 1995.
- [MST05] Heping Ma, Weiwei Sun, and Tao Tang. Hermite spectral methods with a time-dependent scaling for parabolic equations in unbounded domains. *SIAM J. Numer. Anal.*, 43(1):58–75, 2005.
- [Nah05] T. Nahm. Error estimation and index refinement for dimension-adaptive sparse grid quadrature with applications to the computation of path integrals. Diplomarbeit, Institut für Numerische Simulation, Universität Bonn, 2005.
- [Nie98] A. Niedermeier. Lösung von Differentialgleichungen durch Prewavelets. Diplomarbeit, Technische Universität München, 1998.
- [NW08] E. Novak and H. Woźniakowski. *Tractability of Multivariate Problems, Volume I: Linear Information*, volume 6 of *EMS Tracts in Mathematics*. Eur. Math. Soc., Zürich, 2008.
- [Oet10] J. Oettershagen. Minimierung der effektiven Dimension und ihre Anwendung auf Dünngitter-Methoden. Diplomarbeit, Rheinische Friedrich-Wilhelms-Universität Bonn, 2010. (To appear).
- [Ött96] H.C. Öttinger. *Stochastic Processes in Polymeric Fluids*. Springer, 1996.
- [Owe03] A.B. Owen. The dimension distribution and quadrature test functions. *Statistica Sinica*, pages 1–17, 2003.
- [PFTV92] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical recipes in C*. Cambridge University Press, see also <http://www.library.cornell.edu/nr/bookcpdf.html>, 1992.
- [PPB10] Dirk Pflüger, Benjamin Peherstorfer, and Hans-Joachim Bungartz. Spatially adaptive sparse grids for high-dimensional data-driven problems. *Journal of Complexity*, April 2010. In press.

- [PT95] S. H. Paskov and J. F. Traub. Faster evaluation of financial derivatives. *J. Portfolio Management*, 22(1):113–120, 1995.
- [QSS01] A. Quarteroni, R. Sacco, and F. Saleri. *Numerische Mathematik 1*. Springer, 2001.
- [RA99] H. Rabitz and O. Alis. General foundations of high-dimensional model representations. *Journal of Mathematical Chemistry*, 25:197–233, 1999.
- [Rei04] C. Reisinger. *Numerische Methoden für hochdimensionale parabolische Gleichungen am Beispiel von Optionspreisaufgaben*. Dissertation, Universität Heidelberg, 2004.
- [Sch99] T. Schiekofer. *Die Methode der Finiten Differenzen auf dünnen Gittern zur Lösung elliptischer und parabolischer partieller Differentialgleichungen*. Dissertation, Universität Bonn, 1999.
- [SST08] C. Schwab, E. Süli, and R. A. Todor. Sparse finite element approximation of high-dimensional transport-dominated diffusion problems. *ESAIM: M2AN*, 42(5):777 – 819, 2008.
- [SW98] I. Sloan and H. Woźniakowski. When are quasi-Monte Carlo algorithms efficient for high-dimensional integrals? *J. Complexity*, 14:1–33, 1998.
- [SWW04] I. Sloan, X. Wang, and H. Woźniakowski. Finite-order weights imply tractability of multivariate integration. *Journal of Complexity*, 20:46–74, 2004.
- [TW99] J.F. Traub and A. G. Werschulz. *Complexity and Information*. Accademia Nazionale Dei Lincei, 1999.
- [Wah90] G. Wahba. Spline models for observational data. In *CBMS-NSF Regional Conference Series in Applied Mathematics*, volume 59, Philadelphia, 1990. SIAM.
- [WB00] S. F. Wojtkiewicz and L. A. Bergman. Numerical solution of high dimensional fokker–planck equations. In *8th ASCE Specialty Conference on Probabilistic Mechanics and Structural Reliability*, 2000.
- [Zen91] C. Zenger. Sparse grids. In W. Hackbusch, editor, *Parallel Algorithms for Partial Differential Equations*, Notes on Numerical Fluid Mechanics 31, pages 241–251. Vieweg, Braunschweig, 1991.