# Fast Computation Tools for Adaptive Wavelet Schemes

Von der Mathematisch–Naturwissenschaftlichen Fakultät der
Rheinisch–Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
genehmigte Dissertation

vorgelegt von

Diplom–Mathematiker
Arne Barinka

aus Trier

Berichter: Univ.–Prof. Dr. rer. nat. Wolfgang Dahmen
Univ.–Prof. Dr. rer. nat. Reinhold Schneider

Tag der mündlichen Prüfung: 17.03.2005

**Acknowledgments**

I wish to express my sincere gratitude to Prof. Dr. W. Dahmen, Prof. Dr. R. Schneider, Prof. Dr. St. Dahlke, Dr. M. Konik, Dr. T. Barsch, Prof. Dr. K. Urban, and A. Voß – my collaborators, co-authors and teachers.

Special thanks are devoted to Prof. Wolfgang Dahmen, who provides so much inspiring challenge, Alexander Voß my math- and soul-mate, Nina Gier for being there when it counts, and my parents, for their bullet-proof support. Danke.

**Abstract**

During the past few years, a new algorithmic paradigm for adaptive wavelet schemes was developed. First approaches covered elliptic problems, but meanwhile, the class of feasible problems could be significantly enlarged, including even certain nonlinear problems.

This thesis will present and analyze routines for key tasks arising in connection with those schemes. The central point will be a so called *recovery scheme* that allows to compute arrays of wavelet coefficients efficiently by treating the array as a whole instead of treating each entry separately by quadrature schemes.

The tools and methods we develop are realized in a C++ implementation by the author, which we present in form of a schematic overview. All numerical studies presented in the thesis are based on this implementation.

**Zusammenfassung**

In den letzten Jahren wurde ein neues algorithmisches Paradigma für adaptive Wavelet-Algorithmen entwickelt. Erste Ansätze behandelten elliptische Probleme, doch mittlerweile konnte die Klasse der behandelbaren Probleme stark erweitert werden und deckt mittlerweile sogar bestimmte nichtlineare Probleme ab.

In der vorliegenden Arbeit präsentieren und analysieren wir Lösungsmethoden für zentrale Teil-Aufgaben dieser neuen Schemata. Der zentrale Punkt wird ein sogenanntes *recovery scheme* sein, welches erlaubt, Vektoren von Wavelet-Koeffizienten als Ganzes zu berechnen, anstatt jeden Eintrag einzeln mittels Quadratur zu bestimmen.

Die entwickelten Werkzeuge und Methoden wurden vom Autor in einer C++ Implementierung umgesetzt. Diese wird in schematischer Form vorgestellt. Alle hier präsentierten numerischen Beispiele wurden mit Hilfe dieser Implementierung erstellt.

# Contents

# Introduction

During the past few years, a new algorithmic paradigm for adaptive wavelet schemes was developed, [DHU00, CDD01, CDD00, CDD03a, CDD03c]. First approaches covered elliptic problems such as elliptic boundary integral and boundary value problems, but meanwhile, the class of feasible problems could be significantly enlarged, including even certain nonlinear problems.

This theses will present and analyze routines for key tasks arising in connection with these schemes. The central point will be a so called *recovery scheme* that allows to compute arrays of wavelet coefficients efficiently by treating the array as a whole, instead of treating each entry separately by quadrature schemes.

The new paradigm comes with new challenges concerning computational issues, whence there is a need for new computational tools suited for the new adaptive schemes. This is mainly due to their special requirements concerning a) data-handling and b) efficiency.

a) The data-handling requirements result from the fact that the proposed strategy is an iterative routine using finite approximants that are *dynamically updated* during each step of the iteration.

b) Efficiency is a major issue, because the central points of investigation in the development of the new paradigm were theoretical convergence and complexity estimates. They finally lead to numerical schemes that are capable of solving a given problem with an *asymptotically optimal* expense of computational complexity whenever their computational ingredients satisfy certain now well identified requirements. Optimal means in this context, that the work in terms of *floating point operations* and *storage manipulations* stays proportional to the number of degrees of freedom that are *intrinsically* needed to achieve a desired target accuracy.

For certain model cases, computational ingredients that at least asymptotically satisfy these requirements can be easily realized, however the next crucial step is to develop computational tools that still satisfy these conditions for a wider realistic class of problems. The final goal is of course is

to put at disposal efficient and flexible computational tools that allow to fastly realize the new schemes in form of computer programs that preserve all theoretical achievements.

### Background and Motivation

The general format of the problems we are concerned with can be stated as follows. Let $\mathcal{H}$ be a Hilbert space and $\mathcal{H}'$ its dual. Given $F : \mathcal{H} \to \mathcal{H}'$ and any $f \in \mathcal{H}'$ we wish to find a $u \in \mathcal{H}$ such that

$$\langle v, F(u)\rangle = \langle v, f\rangle \qquad \forall \;\; v \in \mathcal{H}, \tag{0.0.1}$$

where this problem is assumed to be *well-posed* in the sense, cf. [CDD03a], that there exists a unique solution $u \in \mathcal{H}$ and that for $v$ in a neighborhood $\mathcal{U}$ of $u$, there exist constants $c_{F,v}, C_{F,v}$, such that for all $w \in \mathcal{H}$

$$c_{F,v}\|w\|_{\mathcal{H}} \leq \|DF(v)w\|_{\mathcal{H}'} \leq C_{F,v}\|w\|_{\mathcal{H}}, \tag{0.0.2}$$

(see also [CDD03d, CDD03b]). Here $DF(\cdot)$ denotes the *Frechet* derivative, which is defined as mapping from $\mathcal{H}$ to $\mathcal{H}'$ by

$$\langle v, DF(\cdot)w\rangle = \lim_{h\to 0} h^{-1}\langle v, F(\cdot + hw) - F(\cdot)\rangle.$$

It is essential in this context, that $\mathcal{H}$ permits a *wavelet characterization*. A wavelet basis $\Psi = \{\psi_\lambda; \lambda \in \mathcal{J}\}$ is called a *Riesz-basis for* $\mathcal{H}$ and is said to characterize $\mathcal{H}$, if a *norm-equivalence* can be established, i.e., the $\mathcal{H}$-norm of an element is equivalent to the $\ell_2$-norm of its (wavelet-) coefficients.

Under these circumstances it has been shown ([CDD01]) that one can find an equivalent formulation of the original problem in terms of wavelet coefficients in form of an *infinite dimensional* system $\mathbf{F}(\mathbf{u}) = \mathbf{f}$, which is now well-posed in $\ell_2$. Here $\mathbf{u}$ is the unknown array of wavelet coefficients of the solution $u$ and $\mathbf{f}$ are the dual wavelet coefficients of the right-hand side $f$.

One then can formulate an *ideal* iterative scheme for the discrete, *infinite* dimensional problem $\mathbf{Fu} = \mathbf{f}$, e.g.

$$\mathbf{u}^{i+1} = \mathbf{u}^i - \mathbf{C}_i\left(\mathbf{F}(\mathbf{u}^i) - \mathbf{f}\right), i = 0, 1, 2, ..., \tag{0.0.3}$$

where $\mathbf{C}_i$ is a 'preconditioner', chosen such that in each step of the iteration, the error is at least reduced by a fixed factor, which is possible because of the asserted $l_2$-well-posedness ([CDD01]). Each step of the iteration of this idealized scheme is (approximately) executed by approximating the

weighted residuals $\mathbf{C}_i\,(\mathbf{F}(\mathbf{u}^i)-\mathbf{f})$ within certain dynamically updated tolerances. This is done by adaptive evaluation of $\mathbf{F}(\mathbf{u}^i)$, which mainly requires the computation of (0.0.1) and an adaptive application of $\mathbf{C}_i$. The latter task being problem dependent, we focus here on the computation of (0.0.1).

An evaluation scheme as mentioned above, e.g. for $\mathbf{F}$ acting on a finitely supported array $\mathbf{v}$, consists of two steps and takes the following form:

S1) *Prediction step:* Given a set $\Lambda=\Lambda(\varepsilon)$ such that $\|\mathbf{v}-\mathbf{v}|_\Lambda\|_{\ell_2}\le\varepsilon$, for the current accuracy tolerance $\varepsilon>0$, predict a possibly small set $\Gamma\subset\mathcal{J}$ such that

$$\|\mathbf{F}(\mathbf{v})-\mathbf{F}(\mathbf{v})|_\Gamma\|_{\ell_2}\le C\varepsilon, \tag{0.0.4}$$

where $C$ is a fixed constant.

S2) *Recovery step:* Compute an array $\mathbf{w}$ such that

$$\|\mathbf{F}(\mathbf{v})|_\Gamma-\mathbf{w}\|_{\ell_2}\le C'\varepsilon, \tag{0.0.5}$$

with an amount of work that stays proportional to $\#\Gamma$.

This leads to perturbed, but applicable iteration steps, cf. [CDD01, CDD00, BDD03], that produce *finitely* supported $\ell_2$-approximants within controlled tolerance.

### The Main Task

In this thesis, we mainly concentrate on the recovery step S2), and we shall for most of the time assume, that the prediction step S1) and therefor $\Gamma$ is derived from $\Lambda(\varepsilon)$ through suitable background information provided by the individual problem at hand. We refer to [DSX00, CDD03c, CDD03d] in that regard.

The recovery step S2) however leaves us with computing or approximating an array of wavelet coefficients at a prescribed (order of) tolerance, namely we have to cope with the following recovery **Task R**[1]:

Given $g\in\mathcal{H}'$ and some finite subset $\Gamma\subset\mathcal{J}$ of indices, compute an array $\mathbf{w}$ supported in $\Gamma$ that approximates

$$\mathbf{g}(\Gamma):=(g_\lambda)_{\lambda\in\Gamma}:=(\langle\psi_\lambda,g\rangle)_{\lambda\in\Gamma},$$

such that $\|\mathbf{g}(\Gamma)-\mathbf{w}\|_{\ell_2(\mathcal{J})}$ is *comparable* to $\|\mathbf{g}-\mathbf{g}(\Gamma)\|_{\ell_2(\mathcal{J})}$.

[1] We postpone a more precise and technical formulation to Chapter 3.

One of the difficulties we have to face is caused by the multilevel structure of adaptive wavelet schemes, i.e. that already point evaluations are hampered somewhat by the fact that at a given point, increasingly more functions may be involved. Another problem concerning the computation of $(\langle \psi_\lambda, g\rangle)_{\lambda\in\Gamma}$ is that wavelets of low level have a support comparable to the whole domain of interest and that the quadrature has to be executed at an accuracy comparable to the overall error. Therefore, the amount of work for a strategy based on computing each individual entry $\langle \psi_\lambda, \cdot\rangle$ could not be kept within the desired bounds.

For the recovery scheme, we will therefore pursue a different strategy ([BDS04]) in order to treat the array of inner products as a whole by picking up an idea suggested already in [DSX00]. There an analytic framework has been proposed that facilitates complexity estimates also for those situations where the sparseness of wavelet expansions is significant and prohibits transformations to uniform single scale representations at the highest level.

The recovery-scheme presented in this thesis improves on earlier findings from [DSX00] in several respects. First, on the algorithmic side, it has a much simpler structure and offers a significant quantitative improvement of performance and storage demands. The main difference is that in [DSX00] at some point a full, usually pessimistically large prediction index set had to be assembled and allocated. Moreover, approximate coefficients were generated through a coarse-to-fine-to-coarse sweep. This is avoided in the new scheme which generates the significant inner products in a single sweep from fine to coarse. This results in an overall more simply structured algorithm with less subroutines and reduced storage demands.

Second, a different approach to the complexity analysis is pursued which addresses the demands put forward by the above mentioned recent developments of adaptive wavelet methods. The major deficiencies of the approach in [DSX00] can be summarized as follows. The error analysis relied on relating the accuracy of the computed entries to an approximation error of a function approximation in $L_2$. As pointed out above, in the context of adaptive schemes the role of $L_2$ is played by the dual $\mathcal{H}'$ of an energy space, typically a Sobolev space of negative order. We shall explain later in more detail why, in spite of norm equivalences induced by scaling wavelet bases, this poses a serious obstruction.

Moreover, the error analysis in [DSX00] worked under the assumption that local polynomial errors (caused by quadrature) are essentially equibalanced, a fact that can generally not be guaranteed in the context of interest. Finally, the regularity assumptions which the error analysis in [DSX00] was

based upon are not quite compatible with the demands of the above mentioned developments.

When it comes to realizations (in terms of computer programs) within the framework given by the paradigm sketched above, one has to deal with special difficulties that are immanent to the adaptive nature of the method. A major issue is, that the involved index sets have a *lacunary*, *unstructured* form and that they may change *dynamically* in each step of the iteration. This makes the handling of the data extremely tricky and requires not only sophisticated code design, but also a fine tuning of the underlying algorithms to prevent the theoretical achievements from being spoiled by the practical realization.

Nonetheless, first tools and implementations of corresponding adaptive schemes are at hand, see [BBC$^+$02, MV99, Bar01, Jür01], and also the methods described in this thesis are realized in a C++ implementation. All these codes are based on decisions on how to handle the specific obstructions, e.g. concerning the data representation, and – even though this development has just started and much more experience is required – first paradigms on how to efficiently implement adaptive wavelet schemes seem to evolve.

### The Layout

This thesis is divided into four parts, each starting with a brief overview on its contents, hopefully providing some additional orientation.

Part I provides notation and preliminary facts for the formulation and analysis of our algorithms. In Chapter 1, some facts on $N$-term approximation are collected, which is essential for formulating our optimality benchmark and to understand certain thresholding and selection routines in the adaptive context. This is followed by a proposition on how to economically realize these routines using a so-called *binning process*. Chapter 2 presents a short introduction to wavelets and their properties as far as we shall need them. Moreover, so called *cardinal B-spline wavelets* according to [CDF92] are introduced as our model discretization tool in the course of this theses, see also Sections 3.4, 4.1.2 and Part III. To complete the collection of prerequisites, Section 2.3 fixes notation to formulate structural demands on index sets.

We shall then be ready to turn to the development of the tools in Part II. Chapter 3 first concentrates on algorithmic issues of our main scheme RECOVER before outlining the related error- and complexity-analysis in Section 3.3. Here, certain demands on the scheme become obvious, and

corresponding computational tools are treated in Chapter 4, mainly focusing on quadrature (Section 4.1) and structural requirements (Section 4.2). Section 4.3 demonstrates in brief, how the top-down structure can be used to incorporate conceptually different steps of the approximation process necessary to accomplish TASK R in a single level-sweep of RECOVER. Note that a list of the algorithms (presented in form of Nassi-Schneidermann diagrams), can be found on page 171.

Part III is concerned with several applications of the tools developed so far, and presents numerical tests in one and two dimensions, whose general layout is described in Chapter 5. $L_2$-applications of the recovery scheme are studied in Chapter 6. Chapter 7 is concerned with a brief comment on the prediction step S1), namely a discussion of a heuristical strategy based on [DSX00]. Applications of RECOVER within dual norms are treated in Chapter 8.

The issue of realizing the above tools in form of computer routines is addressed in Part IV, where we sketch the way we have implemented the schemes presented so far. After a few conceptual remarks in Chapter 9, Chapter 10 briefly outlines the organization and purposes of the different components of our wavelet library `igpm_w_lib`.

# Part I

# Prerequisites

**Overview:** In this Part, we collect all prerequisites for the development, description and analysis of our central algorithm RECOVER.

Chapter 1 will briefly review some basics on $N$-term approximation and will formulate the benchmark for all schemes presented in this thesis. We also introduce the concept of binning, a tool that helps to turn certain concepts from the theory of $N$-term approximation into efficient algorithms.

In Chapter 2, we collect all facts concerning wavelets needed later on. This includes their construction and main features as well as an introduction to the basic notions concerning structural requirements on (wavelet) index sets.

# Chapter 1

# Nonlinear Approximation: Sorting and Thresholding

In this chapter, we are concerned with the principle idea of constructing an approximant to a target function by selecting only $N$ coordinates. We clarify, what the 'optimal' outcome of such an approximation process can be, and we specify the benchmark for our algorithms. We also briefly investigate a so called *binning-scheme*, a computational tool for realizing certain algorithmic steps derived from theory.

## 1.1 $N$-term Approximation

The basic task in approximation is to replace a given, mostly complicated *target function*, by a more simple function, the *approximant*, satisfying certain accuracy requirements, typically formulated in a specified norm.

For any normed linear space $S$, the norm is denoted by $\|\cdot\|_S$. Important examples are $L_p$ spaces: For $1 \leq p \leq \infty$, and any Lebesques-measure space $\Omega$, the space $L_p(\Omega)$ consist of those measurable functions $v$ such that

$$\|v\|_{L_p(\Omega)} := \Big( \int_\Omega |v(x)|^p dx \Big)^{1/p} < \infty,$$

with the usual sup-norm interpretation for $p = \infty$. The case $p = 2$ occurs most often in this thesis, and in this case $\|\cdot\|^2_{L_2(\Omega)} = \langle \cdot, \cdot \rangle_\Omega$, where

$$\langle u, v \rangle_\Omega := \int_\Omega u(x)\overline{v(x)}dx$$

denotes the standard inner product. In all cases of practical interest we have $\Omega = \mathbb{R}$ or $\Omega \subset \mathbb{R}^d$ being (at least) a Lipschitz domain, bounded, open

and connected. If $r$ is a positive integer, the *Sobolev* space $W^{r,p}(\Omega)$ consists of all functions $v \in L_p(\Omega)$, whose distributional (partial) derivatives $\partial^\nu v, |\nu| = r$ satisfy

$$|v|_{W^{r,p}(\Omega)} := \Big( \sum_{|\nu|=r} \|\partial^\nu v\|^p_{L_p(\Omega)} \Big)^{1/p} < \infty.$$

Here the usual multi-index notation $|\nu| = |\nu_1| + \ldots + |\nu_d|$ in $d$ dimensions is used. Adding this semi-norm for $W^{r,p}(\Omega)$ to the $L_p(\Omega)$-norm of $v$ gives the $W^{r,p}(\Omega)$-norm $\|v\|_{W^{r,p}(\Omega)}$. Again, the most important case is $p = 2$, where one uses the short-hand notation $H^r(\Omega) := W^{r,2}(\Omega)$. Sobolev spaces with non-integer index $s \in R$ can be introduced in several ways, e.g. by interpolation between $L_2(\Omega)$ and $H^r(\Omega), r > s, r \in I\!N$, see e.g. [Ada00, BL76, DP88]. If $s < 0$, one can use duality. For any normed linear space $S$, the dual space of bounded linear functionals on $S$ is denoted by $S'$ and is a Banach space under the norm

$$\|w\|_{S'} := \sup_{\|v\|_S=1} |w(v)|.$$

If $\Omega$ is a closed manifold we have $\big(H^s(\Omega)\big)' = H^{-s}(\Omega)$.

We shall also be concerned with *Besov spaces* $B^s_q(L_p(\Omega)), 0 < p, q < \infty, s > 0$, which arise by interpolating between $L_p(\Omega)$ and $W^{r,p}(\Omega), r \in I\!N$, see [Ada00, DeV98]. One way of defining the corresponding norm is by means of *moduli of continuity*

$$\omega_r(v, t, \Omega)_p := \sup_{|h|\le t} \|\Delta^r_h v\|_{L_p(\Omega_{r,h})},$$

where $|h|$ denotes the Euclidean norm of $h \in I\!R^d$, $\Delta^r_h v$ the $r$-th forward difference of $v$ in direction $h$ and

$$\Omega_{r,h} := \{x \in \Omega : x + lh \in \Omega, l = 0, \ldots, r\}.$$

Defining the semi-norm for $s < r, r$ fixed

$$|v|^q_{B^s_q(L_p(\Omega))} := \sum_{j=0}^{\infty} 2^{qsj} \omega_r(v, 2^{-j}, \Omega)^q_p,$$

the Besov norm is given by

$$\|v\|^q_{B^s_q(L_p(\Omega))} := \|v\|^q_{L_p(\Omega)} + |v|^q_{B^s_q(L_p(\Omega))}.$$

For alternative definitions of Besov spaces, e.g., by means of a certain $K$ functional, and a detailed discussion of their properties, see e.g., the books [DL93, RS96], the articles [DeV98, CDDD01, DSX00, DP88] and the references therein.

Suppose now that $\Psi := \{\psi_\lambda, \lambda \in \mathcal{J}\}$ is a *Riesz*-basis for a normed linear space $S$, which means that each $v \in S$ has a *unique* expansion

$$v = \sum_{\lambda \in \mathcal{J}} v_\lambda \psi_\lambda, \tag{1.1.1}$$

and that there exist constants $c_\Psi, C_\Psi$ such that

$$c_\Psi \|\mathbf{v}\|_{\ell_2(\mathcal{J})} \leq \|v\|_S \leq C_\Psi \|\mathbf{v}\|_{\ell_2(\mathcal{J})}, \tag{1.1.2}$$

where $\mathbf{v} := (v_\lambda)_{\lambda \in \mathcal{J}}$ and $\ell_2$ is the space of sequences s.t. $\|\mathbf{v}\|_{\ell_2}^2 := \sum_{\lambda \in \mathcal{J}} |c_\lambda|^2 < \infty$.

We shall often use the 'formal' scaler product notation $\mathbf{v}^T \Psi$ to abbreviate the right-hand side of (1.1.1). Relation (1.1.2) can also be denoted by $\|\mathbf{v}\|_{\ell_2(\mathcal{J})} \sim \|v\|_S$, using the convention that $a \lesssim b$ means that $a$ can be bounded by a constant multiple of $b$ (independent of any parameters on which $a$ and $b$ may depend) and that $a \sim b$ means $a \lesssim b$ and $b \lesssim a$.

When approximating an element $v \in S$, approximants will be chosen from a subspaces $\hat{S}$ of $S$ spanned by finitely many basis functions $\psi_\lambda$. One aims at increasing the accuracy of an approximation by enlarging the number of degrees of freedom, but that also increases the associated work and storage. We aim at balancing this trade off between complexity and accuracy in a possibly optimal way. Rather than setting

$$\hat{S} = \text{span}\{\psi_{\lambda_k}, 1 \leq k \leq N\}, \quad N < \infty, \tag{1.1.3}$$

which will result in a *linear approximation process*, we will therefore choose the set of linear combinations of at most $N \in I\!N_0, N < \infty$, coordinates

$$\Sigma_N := \bigcup_{\#\Lambda \leq N} \text{span}\{\psi_\lambda, \lambda \in \Lambda \subset \mathcal{J}\},$$

without a-priorily specifying the basis functions in $\Sigma_N$. Let us mention, that $\Sigma_N$ is obviously a nonlinear space, as the sum of two elements in $\Sigma_N$ has in general $2N$ non-zero elements.

For any given $g \in S$, the error of this *N-term approximation* is defined as

$$\sigma_N(g) := \inf_{\hat{g} \in \Sigma_N} \|g - \hat{g}\|_S. \tag{1.1.4}$$

A function that realizes $\sigma_N(g)$ is called a *best $N$-term approximation*. In practice, given a function $g$, it is often easier to construct a *near best* $N$-term approximant, i.e., to find $g_N \in \Sigma_N$, such that

$$\|g - g_N\|_{\mathcal{H}} \leq c\, \sigma_N(g), \tag{1.1.5}$$

for some uniform constant $c \geq 1$.

We are interested in characterizing the quality of approximations with $N$ degrees of freedom in terms of smoothness. In the case where the approximation error is measured in $H^s$, we refer to results in [DeV98] which roughly state:

C1) For the rate of best approximation of a function $v$ by *linear* approximation (on a hierarchy of uniform refinement), it holds: The approximation error decays like $N^{-(\alpha+s)/d}$ if and only if $v$ has smoothness $H^\alpha$, $\alpha > s$.

C2) For the rate of best approximation of a function $v$ by *nonlinear* approximation (on a hierarchy of adaptive refinement), it holds: The approximation error decays like $N^{-(\alpha+s)/d}$ if and only if $v$ has smoothness $B^\alpha_\tau(L_\tau(\Omega))$, $\alpha > s, 1/\tau := (\alpha - s)/d + 1/2)$.

In these results the range of $\alpha$ and $s$ are limited by the order of the method and the smoothness of the approximation space. The advantage in nonlinear approximation, is that C2) requires a much weaker smoothness condition then C1) to ensure the same error decay, i.e., one can compensate the loss of regularity by judiciously placing degrees of freedom so to retain the same asymptotically optimal rate.

One example where nonlinear approximation techniques pay off, are elliptic boundary value problems, as the regularity of the solution could be shown to be significantly higher in the Besov scale than in the Sobolev scale, see [DD97, CDD01, CDD00] for more details and further examples.

In the light of the *norm equivalence* (1.1.2), approximating the function $v \in S$ is equivalent to approximating its array of coefficients $\mathbf{v} \in \ell_2(\mathcal{J})$. In the special case of $\ell_2$, it is in principle easy to obtain a best $N$-term approximation. Given a known sequence $\mathbf{v} \in \ell_2(\mathcal{J})$, a best $N$-term approximation $\mathbf{v}^*_N \in \Sigma_N$ satisfying (1.1.5) is constructed by taking the $N$ largest entries of $\mathbf{v}$ as the elements of $\mathbf{v}^*_N$.

A naive possibility to extract $\mathbf{v}^*_N$ from $\mathbf{v} = (v_i)_{i\in\mathbb{N}} \in \ell_2(\mathcal{J})$ is therefore to rearrange its entries in a non-increasing order of absolute values. This rearrangement process is not necessarily unique, but any outcome will be

denoted by $\mathbf{v}^* = (v_i^*)_{i\in I\!N}$, i.e. we have $|v_i^*| \geq |v_{i+1}^*|$. The corresponding best $N$-term approximation is then given by $(v_i^*)_{i=1}^N$ .

As seen earlier, the quantity of major interest in approximation processes is the so-called *error rate* that is the dependence of the error $\sigma_N$ on $N$ or the question how accurately one can approximate with $N$ degrees of freedom in the space under consideration. We want to identify sequences $\mathbf{v}$ that allow approximations of accuracy $\sigma_N(\mathbf{v}) \lesssim N^{-s}, s > 0$. We therefore are interested in sequences $\mathbf{v}$ where for $s > 0$

$$\|\mathbf{v}\|_{\mathcal{A}^s} := \sup_{N\geq 0}(N+1)^s\sigma_N(\mathbf{v}) < \infty, \tag{1.1.6}$$

with $\sigma_0 := \|\mathbf{v}\|_{\ell_2(\mathcal{J})}$. The corresponding space of sequences is denoted by

$$\mathcal{A}^s := \{\mathbf{v} \in \ell_2(\mathcal{J}), \|\mathbf{v}\|_{\mathcal{A}^s} < \infty\}. \tag{1.1.7}$$

We now drive at an alternate characterization of $\mathcal{A}^s$ to understand its nature. The above rearrangement $\mathbf{v} \to \mathbf{v}^*$ is now used to introduce the so-called *weak* $\ell_\tau$ spaces $\ell_\tau^w$, which are defined by

$$\mathbf{v} = (v_\lambda)_{\lambda\in\mathcal{J}} \in \ell_\tau^w(\mathcal{J}) \iff \#\{\lambda \in \mathcal{J}, |v_\lambda| \geq \varepsilon\} \leq c\,\varepsilon^\tau, \tag{1.1.8}$$

This space is contained in $\ell_2(\mathcal{J})$, whenever $0 < \tau < 2$. Moreover, introducing the quasi-norm

$$|\mathbf{v}|_{\ell_\tau^w(\mathcal{J})} := \sup_{n\geq 1} n^{1/\tau}|v_n^*|, \quad \mathbf{v} \in \ell_2(\mathcal{J}), \tag{1.1.9}$$

the *Lorentz space* $\ell_\tau^w(\mathcal{J})$ can then be written as

$$\ell_\tau^w(\mathcal{J}) := \{\mathbf{v} \in \ell_2(\mathcal{J}), |\mathbf{v}|_{\ell_\tau^w(\mathcal{J})} < \infty\}. \tag{1.1.10}$$

Thus the value of the semi-norm (1.1.12) is just equal to the smallest constant $c$ such that

$$|v_n^*| \leq cn^{-1/\tau}, \quad n \geq 1, \tag{1.1.11}$$

stating that a sequence $\mathbf{v} \in \ell_\tau^w(\mathcal{J})$ indeed has a polynomial decay rate $1/\tau$. Moreover the smallest $c$ which satisfies (1.1.8) is equivalent to $|\mathbf{v}|_{\ell_\tau^w(\mathcal{J})}^\tau$.

We finally set

$$\|\mathbf{v}\|_{\ell_\tau^w(\mathcal{J})} := \|\mathbf{v}\|_{\ell_2(\mathcal{J})} + |\mathbf{v}|_{\ell_\tau^w(\mathcal{J})}, \quad 0 < \tau < 2. \tag{1.1.12}$$

It is easy to see that $\|\mathbf{v}\|_{\ell_\tau^w(\mathcal{J})} \leq 2\|\mathbf{v}\|_{\ell_\tau(\mathcal{J})}$.

Conditions like $\mathbf{v} \in \ell_\tau$ or $\mathbf{v} \in \ell_\tau^w(\mathcal{J})$ are equivalent to smoothness conditions on the function $v$, cf. (1.1.1). For $H^s, s > 0$ and $1/\tau = s + 1/2$ we

have $\mathbf{v} \in \ell_\tau$ if and only if $u \in B_\tau^{sd+t}(L_\tau(\Omega))$ and the condition $\mathbf{v} \in \ell_\tau^w(\mathcal{J})$ is similarly linked to a slightly larger space.

The following Lemma (cf. e.g., [DT96, DeV98]) shows the close connection between the decay property of $\ell_\tau^w(\mathcal{J})$ and the approximation spaces $\mathcal{A}^s$.

**Lemma 1.1.1.** *The spaces $\mathcal{A}^s$ and $\ell_\tau^w(\mathcal{J})$ are the same and have equivalent norms for $s > 0$ and $1/\tau = s + 1/2$. In this case it holds*

$$\|\mathbf{v}\|_{\mathcal{A}^s} \sim \|\mathbf{v}\|_{\ell_\tau^w(\mathcal{J})}.$$

*Especially, $\mathbf{v} \in \ell_\tau^w(\mathcal{J})$ gives $\sigma_N(\mathbf{v}) \leq CN^{-s}\|\mathbf{v}\|_{\ell_\tau^w(\mathcal{J})}, N \in \mathbb{N}$, where the constant $C$ only depends on $\tau$ when $\tau$ tends to zero.*

This states that a sequence in $\ell_\tau^w(\mathcal{J})$, $1/\tau = s+1/2$, can be at least approximated at an error proportional to $N^{-s}$ when using at most $N$ coordinates.

Of course, when approximating a target function, most likely we are not in the ideal situation, that we explicitly know all coordinates of the sequence of coefficients, i.e., we cannot determine the (near) best $N$-term approximation simply by sorting. However, given $\mathbf{v} \in \mathcal{A}^s$, approximation at accuracy $\varepsilon$ requires at most the order of $\varepsilon^{-1/s}$ degrees of freedom. Therefore, a lower bound for the complexity of determining *any* element in the unit ball of $\mathcal{A}^s$ within accuracy $\varepsilon$ is $\varepsilon^{-1/s}$. This suggests to choose the ideal setting of $N$-term approximation in a given space $S$ as benchmark for any numerical scheme operating in $S$. In this thesis, we will therefore use the following terminology, cf. [CDD03a].

**Definition 1.1.2.** *An approximation scheme* Approx *is called* asymptotically optimal *or $s*$-asymptotically optimal, if for any $\mathbf{v}$ in the unit ball of $\mathcal{A}^s, s < s^*$ and tolerance $\varepsilon > 0$, it determines an approximation $\hat{\mathbf{g}}$ such that $\|\mathbf{v} - \hat{\mathbf{v}}\|_{\ell_2(\mathcal{J})} \leq \varepsilon$ and the number of resources (in terms of floating point operations and storage manipulations) needed to do so, stays proportional to $\varepsilon^{-1/s}$.*

Note, that here the term storage manipulations includes storage allocation *and* read/write access, hence in particular sorting operations.

Examples of (up to sorting) asymptotically optimal adaptive schemes can be found in [CDD01, CDD00, CDD03a, CDD03c, DHU00]. These schemes apply routines based on best $N$-term approximation, e.g. thresholding routines to sparsify approximants and reduce their number of degrees of freedom in accordance with the rate of the best $N$-term approximation. This can be arranged in such a way, that the approximation quality measured in the $\ell_2$-norm is not spoiled, as the following has been proven in [CDD01, CDD03a].

**Theorem 1.1.3.** *Assume a possibly infinite dimensional vector* $\mathbf{v}$ *is given and let* $\mathbf{w}$ *be a finite approximant to* $\mathbf{v}$ *satisfying*

$$\|\mathbf{v}-\mathbf{w}\|_{\ell_2(\mathcal{J})} \leq \eta,$$

*with* $0 < \eta \leq \|\mathbf{v}\|_{\ell_2(\mathcal{J})}$. *Let* $N$ *be the smallest integer such that*

$$\|\mathbf{w}-\mathbf{w}_N\|_{\ell_2} \leq \eta.$$

*If* $\mathbf{v} \in \ell^w_\tau(\mathcal{J})$ *for* $\tau = (s+1/2)^{-1}$ *for some* $s > 0$, *then there is a constant* $C > 0$ *which depends only on* $s$ *when* $s \to \infty$ *such that*

$$\|\mathbf{v}-\mathbf{w}_N\|_{\ell_2(\mathcal{J})} \leq \eta \quad \textit{and} \quad \|\mathbf{v}-\mathbf{w}_N\|_{\ell_2(\mathcal{J})} \leq C\|\mathbf{v}\|_{\ell^w_\tau(\mathcal{J})} N^{-s}.$$

*Moreover* $\|\mathbf{w}_N\|_{\ell^w_\tau(\mathcal{J})} \leq C\|\mathbf{v}\|_{\ell^w_\tau(\mathcal{J})}$.

This gives rise to a so-called *coarsening* or *clean-off step* which is frequently applied in the course of various adaptive algorithms. We will next be concerned with its practical realization.

## 1.2 Binning: Approximate Sorting

When executing a coarsening/thresholding following the above reasoning by constructing an N-term approximant $\mathbf{w}_N$ to a (finite) vector $\mathbf{w}$, a natural idea is to construct $\mathbf{w}_N$ by sorting $\mathbf{w}$. This however leads to a suboptimal scheme, as sorting involves $\mathcal{O}(N' \log N')$, $N' := \#\operatorname{supp}\mathbf{w}$ storage manipulations, which is not in accordance with Definition 1.1.2.

The following observations however reveal that this sub-optimality can be removed, as sorting in a strict sense is actually not required. This becomes clear, when writing Definition 1.1.8 of the weak space $\ell^w_\tau(\mathcal{J})$ in the following form

$$\ell^w_\tau(\mathcal{J}) = \{\mathbf{v} \in \ell_2(\mathcal{J}), \#\{\lambda \in \mathcal{J},\ 2^{-j} \geq |v_\lambda| \geq 2^{-j-1}\} \leq c\, 2^{j\tau}, j \in \mathbb{Z}\}. \tag{1.2.1}$$

Note that this last characterization of $\ell^w_\tau(\mathcal{J})$ is based on *orders of magnitude* only and requires no strict sorting.

Instead of performing a strict sorting element by element which is a log-linear operation, one can drive at grouping the coordinates of the sequence according to equivalence classes of orders of magnitude. This can be arranged in such a way that the resulting rearrangement will be monotone on the scale of equivalence classes, or *bins*, as they are sometimes called, but not on the scale of single elements. We will refer to this as an *essentially* monotone rearrangement.

### 1.2.1 Construction and Estimates

The principle of *binning* can be found in various contexts, compare [Met02], e.g., in the method of *quantizing*. Here we will apply it especially to construct near best $N$-term approximants to finite vectors in linear complexity. To describe this, we introduce the following notation.

**Definition 1.2.1.** *Let $\mathbf{v} = (v_\lambda)_{\lambda\in\mathcal{J}}$ be a sequence. For $j \in \mathbb{Z}$ we define*

$$\mathcal{B}_j(\mathbf{v}) := \{\lambda \in \mathcal{J},\ 2^{-j} \geq |v_\lambda| > 2^{-j-1}\}.$$

*$\mathcal{B}_j(\mathbf{v})$ will be called the $j$-th bin of $\mathbf{v}$, $\mathbf{v}^{(*)}$ will denote the rearrangement of $\mathbf{v}$ according to decreasing bins and $\mathbf{v}_j^{(*)} := \{v_\lambda, \lambda \in \mathcal{B}_j(\mathbf{v})\}$.*

The resulting $\mathbf{v}^{(*)}$ is essentially non-increasing, as for any $i, j, j' \in \mathbb{Z}$ and $p \in N$ we have: If $(\mathbf{v}^{(*)})_i \in \mathcal{B}_j(\mathbf{v})$ and $(\mathbf{v}^{(*)})_{i+p} \in \mathcal{B}_{j'}(\mathbf{v})$ then $j' \geq j$. However, the elements of $\mathbf{v}_j^{(*)}, j \in \mathbb{Z}$ are not necessarily ordered in any way.

Recalling (1.2.1), we are able to characterize $\ell_\tau^w(\mathcal{J})$ spaces by the magnitude of the bins of corresponding sequences. A near best $N$-term approximation to a vector $\mathbf{v}$ can therefore be constructed by taking the $N$ first entries of $\mathbf{v}^{(*)}$. The next lemma will show, that the vector $\mathbf{v}_{(N)}$ constructed by this method is also a near best $N$-term approximant to $\mathbf{v} \in \ell_\tau^w(\mathcal{J})$.

**Lemma 1.2.2.** *Let $s > 0$ and $\mathbf{v} \in \ell_\tau^w(\mathcal{J})$, $1/\tau = s + 1/2$. Then there is a constant $C > 0$ such that*

$$\|\mathbf{v} - \mathbf{v}_{(N)}\|_{\ell_2} \leq CN^{-s}|\mathbf{v}|_{\ell_\tau^w(\mathcal{J})}, \quad N \in \mathbb{N}.$$

*Consequently, we also have $\|\mathbf{v} - \mathbf{v}_{(N)}\|_{\ell_2} \leq CN^{-s}\|\mathbf{v}\|_{\ell_\tau^w(\mathcal{J})}$.*

This lemma is analogous to Lemma 1.1.1 and our proof follows [DeV98]. It works also for $1 < p < \infty, s > 0, \tau = s + 1/p$.

*Proof.* Because of (1.2.1) for each $k \in \mathbb{N}$ we have

$$\sum_{-\infty}^{k} \#\mathcal{B}_k(\mathbf{v}) \leq C2^{k\tau}|\mathbf{v}|_{\ell_\tau^w(\mathcal{J})}^\tau.$$

Setting $v_k := (v_\lambda)_{\lambda\in U_k}, U_k := \bigcup_{j=-\infty}^k \mathcal{B}_j$, we know that $v_k \in \Sigma_N$ with $N = \lceil 2^{k\tau}|\mathbf{v}|_{\ell_\tau^w(\mathcal{J})}^\tau \rceil$. We have

$$\|\mathbf{v} - v_k\|_{\ell_2} \leq \sum_{j=k+1}^{\infty} \|\mathbf{v}_j^{(*)}\|_{\ell_2}.$$

Fixing $j > k$, let us estimate $\|\mathbf{v}_j^{(*)}\|_{\ell_2}$. As $|v_\lambda| \leq 2^{-j}$ for all $\lambda \in \mathcal{B}_j(\mathbf{v})$, we get

$$\|\mathbf{v}_j^{(*)}\|_{\ell_2} \leq 2^{-j}(\#\mathcal{B}_j(\mathbf{v}))^{1/2} \leq 2^{j(\tau/2-1)}|\mathbf{v}|^{\tau/2}_{\ell^w_\tau(\mathcal{J})}. \tag{1.2.2}$$

As $\tau = 2/(2s+1)$, i.e., $\tau/2 < 1$, we conclude that

$$\|\mathbf{v} - v_k\|_{\ell_2} \leq |\mathbf{v}|^{\tau/2}_{\ell^w_\tau(\mathcal{J})} \sum_{j=k+1}^{\infty} 2^{j(\tau/2-1)} \leq |\mathbf{v}|_{\ell^w_\tau(\mathcal{J})}(2^k|\mathbf{v}|_{\ell^w_\tau(\mathcal{J})})^{\tau/2-1}. \tag{1.2.3}$$

Hence, for $N = 2^{k\tau}|\mathbf{v}|^\tau_{\ell^w_\tau(\mathcal{J})}$ we obtain with $\tau/2 - 1 = \tau(1/2 - 1/\tau)$

$$\|\mathbf{v} - \mathbf{v}_{(N)}\|_{\ell_2} \leq N^{-s}|\mathbf{v}|_{\ell^w_\tau(\mathcal{J})}.$$

Since $\|\mathbf{v} - \mathbf{v}_{(N)}\|_{\ell_2}$ is monotone in $N$, the last inequality also holds for $N \in \mathbb{N}$.

■

In what follows, we will notationally distinguish (if necessary) an $N$-term approximant $\mathbf{v}_{(N)}$ to a sequence $\mathbf{v}$ obtained by the above method from $\mathbf{v}^*_N$ constructed using strict sorting. In general, $\mathbf{v}_{(N)}$ will differ from $\mathbf{v}^*_N$, yet - apart from the different work count - this has no influence on the asymptotical performance of the various schemes due to the above lemma.

### 1.2.2 Computational Issues

When constructing approximations $\mathbf{v}_{(N)}$ in practice, namely for a finite vector $\mathbf{v}_\Lambda$ on a computer having a finite precision $\eta_C$, we suggest the following routines. Let $\eta > \eta_C > 0$ be a given tolerance. We define

$$l_0 = l_0(\eta) := \lceil -\log_2 \eta \rceil - 1 \quad \text{and} \quad l_1 = l_1(\eta_C) := \lfloor -\log_2 \eta_C \rfloor . \tag{1.2.4}$$

We will often drop $\eta_C$ in the notation, as this quantity is neither method- nor problem-dependent. With this, for a vector $\mathbf{v}$ we define $\mathcal{B}_j(\mathbf{v}, \eta), j = l_0, l_0 + 1, \ldots, l_1$

$$\mathcal{B}_j(\mathbf{v}, \eta) := \begin{cases} \bigcup_{l=-\infty}^{l_0} \mathcal{B}_l(\mathbf{v}), & j = l_0, \\ \mathcal{B}_j(\mathbf{v}), & l_0 < j < l_1, \\ \bigcup_{l=l_1}^{\infty} \mathcal{B}_l(\mathbf{v}), & j = l_1. \end{cases} \tag{1.2.5}$$

Hence, if $\eta = 2^{-i}, \eta_C = 2^{-i'}$ with $i, i' \in \mathbb{N}$, we have that $\mathcal{B}_{l_0}(\mathbf{v}) = \mathcal{B}_{i-1}$ contains all elements $v > \eta$, and $\mathcal{B}_{l_1}(\mathbf{v}) = \mathcal{B}_{i'}$ all elements $v \leq \eta_C$ respectively. If the assembling of the bins $\mathcal{B}_j$, i.e., an *explicite construction* of $\mathbf{v}_{(N)}$ is necessary, one can use the following routine.

BIN-SORT — $(\mathbf{v}, \eta) \to [\mathcal{B}_j(\mathbf{v}, \varepsilon),\ j = l_0, \ldots, l_1]$

| | |
|---|---|
| Determine $l_0, l_1$ according to (1.2.4) | |
| Set $\mathcal{B}_j = \emptyset, j = l_0, \ldots, l_1$ | |
| For each element $v$ of $\mathbf{v}$ | |
| | Compute $j(v) = \log_2(v)$ (e.g. by a bit shift) |
| | Set $\mathcal{B}_{j(v)} \to \mathcal{B}_{j(v)} \cup \{v\}$ |

Algorithm 1.1: Procedure BIN-SORT: Assemble bins

Figure 1.1 shows a small example vector before and after binning.

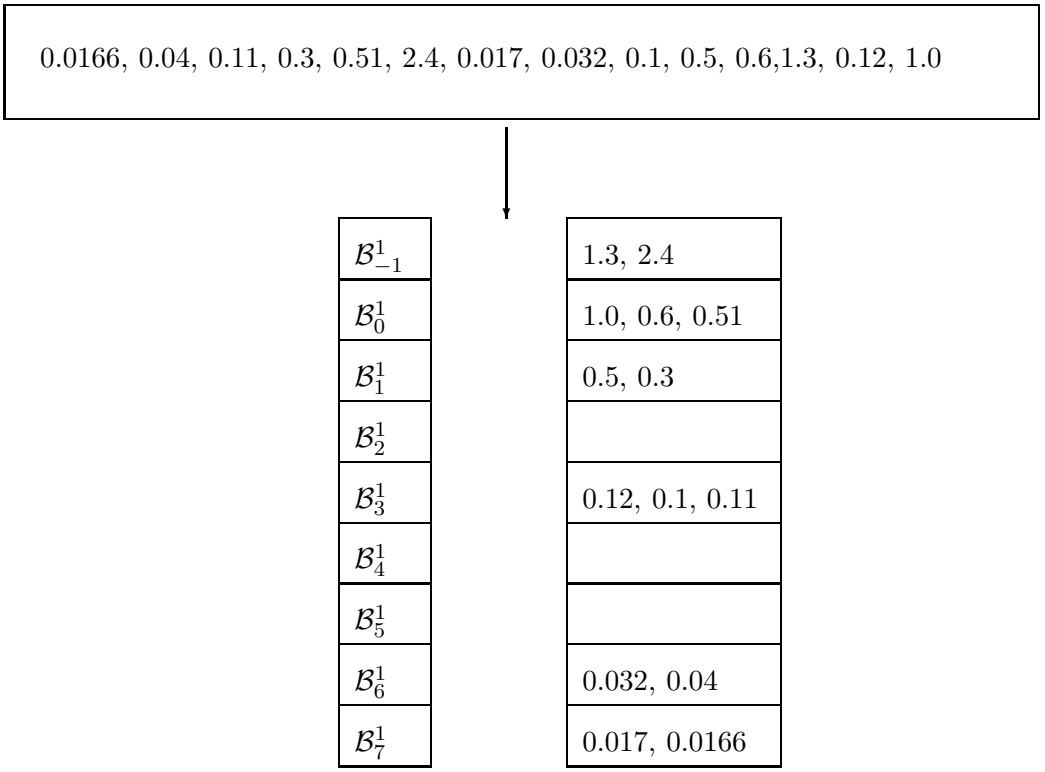



Figure 1.1: An example vector before and after binning with $\eta = 1$

Given $\varepsilon > 0$, the task of finding an $N$-term approximation $\mathbf{v}_{(N)}$ to a finite

vector $\mathbf{v}$ that realizes an error no greater than $\varepsilon$ can be accomplished by applying Bin-Sort and setting successively coordinates in bins $\mathcal{B}_j$ with large $j$, i.e. small entries, to zero until the introduced error reaches the target accuracy $\varepsilon$. This requires $\mathcal{O}(N'), N' := \#\operatorname{supp}\mathbf{v}$ floating point operations and storage in order to construct $\mathbf{v}_{(N)}$ and $\mathcal{O}(N')$ operations to check the error bound each time a coordinate is set to zero.

This operation has optimal asymptotical complexity, i.e., we managed to build a near best $N$-term approximation $\mathbf{v}_{(N)}$ to a finite vector sequence in $\mathbf{v}$ with an *overall* amount of work proportional to $\#\operatorname{supp}\mathbf{v}$. This includes the cost of rearranging, floating point operations and storage.

However, we can avoid to construct the actual bins when thresholding. Reviewing the above proof of Lemma 1.2.2, we suggest the following procedure: At first we will make use of a mechanism to determine the magnitude $B_j := \#\mathcal{B}_j$ of the bins *without constructing* them.

Bin-Count — $(\mathbf{v}, \eta) \to [\#\mathcal{B}_j(\mathbf{v}, \eta),\ j = l_0, \ldots, l_1]$

| |
|---|
| Determine $l_0, l_1$ according to (1.2.4) |
| Set $B_j = 0, j = l_0, \ldots, l_1$ |
| For each element $v$ of $\mathbf{v}$ |
| Compute $j(v) = \log_2(v)$ (e.g. by a bit shift) |
| Set<br>$B_{l_0} \to B_{l_0} + 1$ if $j(v) \le l_0$,<br>$B_{j(v)} \to B_{j(v)} + 1$ if $l_0 < j(v) < l_1$,<br>$B_{l_1} \to B_{l_1} + 1$ if $j(v) \ge l_1$. |
| Finally $B_j = \#\mathcal{B}_j(\mathbf{v}, \eta), j = l_0, \ldots l_1$. |

Algorithm 1.2: Procedure Bin-Count: Determine magnitude of bins

Bin-Count requires $l_1 - l_0 + 1$ integer storage and $\mathcal{O}(\#\operatorname{supp}\mathbf{v})$ floating point operations. Note that neither searching nor operations to rearrange data is needed since the scheme computes the magnitude of the bins without actually assembling them. Another useful application of Bin-Count is a pre-processing to Bin-Sort: If one needs to construct $\mathbf{v}^{(*)}$ by rearranging the entries of $\mathbf{v}$, one can use Bin-Count to determine the amount

of storage for $\mathcal{B}_j, j = l_0, \ldots, l_1$ a-priorily, i.e. before executing BIN-SORT. This allows an optimized (static) memory management.

**Remark 1.2.3.** *Be $N' := \operatorname{supp} \mathbf{v}$. The routine* BIN-COUNT$(\mathbf{v}, \eta)$ *requires $\mathcal{O}(N')$ floating point operations and $l(\eta, \eta_C) := l_1 - l_0 + 1$ integer storage. Again, we will often drop the dependency of $\eta_C$ in notation and write $l(\eta)$. Note that by (1.2.4)*

$$l(\eta) \leq \log_2(\frac{\eta}{\eta_C}).$$

*We will henceforth always assume that $N' \geq l(\eta)$, which is the typical situation. Given this, performing* BIN-SORT$(\mathbf{v}, \eta)$ *requires $\mathcal{O}(N')$ floating point operations and storage.*

We will now use BIN-COUNT to execute a thresholding without constructing the bins.

BIN-THRESH — $(\mathbf{v}, \eta, \varepsilon) \to [\hat{\mathbf{v}}, \hat{\Lambda}]$

| |
|---|
| Determine $l_0, l_1$ according to (1.2.4) |
| Execute BIN-COUNT$(\mathbf{v}, \eta)$ and set<br>$n_j := \#\mathcal{B}_j(\mathbf{v}, \eta), j = l_0, \ldots, l_1$ |
| Set $l = l_1$ and $E := n_{l_1} * 2^{-2l_1}$. |
| If $E \leq \varepsilon^2$ |
| Set $l \to l - 1$ and $E \to E + n_l * 2^{-2l}$ |
| Set $n_j \to 0$ for all $j > l$ |
| Let $n_l := \left\lceil \frac{E-\varepsilon^2}{2^{-2l}} \right\rceil$, $\hat{\mathbf{v}} := \mathbf{0}$, $e = 0$ |
| For each element $v_\lambda$ of $\mathbf{v}$ |
| Compute $j(v_\lambda) = \log_2(v_\lambda)$ |
| If $n_{j(v_\lambda)} > 0$: $\hat{v}_\lambda \to v_\lambda$ and $n_{j(v_\lambda)} \to n_{j(v_\lambda)} - 1$<br>Else: $\hat{v}_\lambda \to 0$, $e \to e + \lvert v_\lambda \rvert^2$ |
| Finally $\hat{\Lambda} := \operatorname{supp} \hat{\mathbf{v}}$. |

Algorithm 1.3: Procedure BIN-THRESH: Thresholding using bins

The idea is the following: As $v_\lambda \in \mathcal{B}_j$ means that $|v_\lambda|^2 \leq 2^{-2j}$, we can bound the error introduced by setting $n$ entries in $\mathcal{B}_j$ to zero by $n2^{-2j}$, and, given $\#\mathcal{B}_j$, we know that $\|\mathbf{v}(\mathcal{B}_j)\|_{\ell_2}^2 \leq \#\mathcal{B}_j 2^{-2j}$, which are bounds for the error if *all* entries in $\mathcal{B}_j$ are neglected. Hence, given an accuracy $\varepsilon > 0$, Bin-Thresh first computes $l, n_l \in \mathbb{N}$, such that we have

$$\sum_{j=l+1}^{l_1} \#\mathcal{B}_j 2^{-2j} + n_l 2^{-2l} \leq \varepsilon^2.$$

We then know that we meet the target accuracy, when keeping all elements in $\mathcal{B}_j, j < l$, $n_l$ entries in $\mathcal{B}_l$ as and no entry in $\mathcal{B}_j, j > l$. The computation of the true truncation error $\sqrt{e}$ is optional and can be used to discard additional elements if $e < \varepsilon^2$. The same reasoning leads to the lower bound

$$\|\mathbf{v} - \hat{\mathbf{v}}\|_{\ell_2(\mathcal{J})} > \sum_{j=l+1}^{l_1} \#\mathcal{B}_j 2^{-2(j-1)} + n_l 2^{-2(l-1)},$$

for the output $\hat{\mathbf{g}}$ of Bin-Thresh, which guarantees that Bin-Thresh is efficient.

**Corollary 1.2.4.** *The routine* Bin-Thresh$(\mathbf{v}, \eta, \varepsilon)$ *can be executed with an amount of* $\mathcal{O}(\#\mathbf{v})$ *floating point operations and an amount of* $l(\eta)$ *integer storage. Furthermore for its output* $\hat{\mathbf{v}}$*, we have*

$$\|\mathbf{v} - \hat{\mathbf{v}}\|_{\ell_2} \leq \varepsilon.$$

*If* $\mathbf{v} \in \ell_\tau^w(\mathcal{J})$ *for some* $s > 0$ *and* $1/\tau = s + 1/2$*, we have*

$$\|\mathbf{v} - \hat{\mathbf{v}}\|_{\ell_2} \lesssim (\#\hat{\Lambda})^{-s} |\mathbf{v}|_{\ell_\tau^w(\mathcal{J})}^\tau.$$

*Proof.* Performing Bin-Count$(\mathbf{v}, \varepsilon)$ requires $\mathcal{O}(\#\mathbf{v})$ floating point operations and $l(\eta)$ integer storage. The determination of $l$ is $\mathcal{O}(l(\eta))$, the determination of $n_l$ of course is $\mathcal{O}(1)$. Finally the computation of $\hat{\mathbf{v}}$ is $\mathcal{O}(\#\mathbf{v})$. As $l(\eta) < \#\mathbf{v}$, the first assertion follows. The error estimates are analogous to the proof of Lemma 1.2.2. Note that

$$\|\mathbf{v} - \hat{\mathbf{v}}\|_{\ell_2} \leq \sum_{j=l}^{l_1} 2^{-j} n_j^{1/2} \leq \sum_{j=l}^{\infty} 2^{-j} \#\mathcal{B}_j(\mathbf{v}, \varepsilon)^{1/2}$$

which can be treated analogously to (1.2.2) and (1.2.3). ∎

If not stated differently we will always assume throughout the rest of this thesis that an $N$-term approximation $\mathbf{v}_N$ is constructed using the above binning routines.

# Chapter 2

# Wavelet Prerequisites

The recovery scheme that we will discuss in the next chapter will depend heavily on the fact, that we are working with wavelets. Before we turn to its description, let us therefore collect the basic facts on wavelets needed later on.

Wavelet bases are usually constructed with the aid of a *multi-resolution analysis* [Mal89], i.e., a sequence of linear, closed, nested spaces with the properties, that the closure of their union is dense in the considered Hilbert space $\mathcal{H}$. If the underlying domain is unbounded one has to ensure that their intersection contains only $\{0\}$.

The space $\mathcal{H}$ plays, for instance, the role of an energy space for a variational problem and one should think of $\mathcal{H}$, e.g., as $L_2(\Omega)$, $\Omega$ a domain or manifold, or as a (closed subspace of a) Sobolev space (defined e.g. by homogeneous boundary conditions) or a product of such spaces. The space of bounded linear functionals on $\mathcal{H}$, the normed dual of $\mathcal{H}$, is denoted by $\mathcal{H}'$. The dual pairing $\langle \cdot, \cdot \rangle$ on $\mathcal{H} \times \mathcal{H}'$ is always tacitly assumed to be induced by the standard $L_2$-inner product on the underlying domain $\Omega$ and defining the dual space $\mathcal{H}'$. As usual we set $\|w\|_{\mathcal{H}'} := \sup_{\|v\|_{\mathcal{H}} \leq 1} \langle v, w \rangle$.

## 2.1 Construction and Essential Features

In the following, we shall rely on the fact that the wavelet basis $\Psi$ is a Riesz basis in $\mathcal{H}$ as introduced in Section 1.1. The Riesz basis property of $\Psi$ implies the existence of a *dual basis* $\tilde{\Psi} \subset \mathcal{H}'$ such that $\langle \tilde{\psi}_\lambda, \psi_\mu \rangle = \delta_{\lambda,\mu}$. This in turn means that

$$v = \sum_{\lambda \in \mathcal{J}} v_\lambda \psi_\lambda = \sum_{\lambda \in \mathcal{J}} \langle \tilde{\psi}_\lambda, v \rangle \psi_\lambda.$$

It will be important, that *rescaling* such a *dual pair* $\Theta = \{\theta_\lambda : \lambda \in \mathcal{J}_j\}$ and $\tilde{\Theta} = \{\tilde{\theta}_\lambda : \lambda \in \mathcal{J}_j\}$ of *biorthogonal* wavelet bases for $L_2(\Omega)$ provides Riesz bases for a whole range of smoothness spaces, see [Dah94, Dah96], whence we sometime call $\Theta, \tilde{\Theta}$ *anchor bases*.

Recall, that a multi-scale basis $\Theta$ is called a *Riesz basis* for $L_2$, if every element $v$ in $L_2$ has a unique expansion $v = \sum_{\lambda\in\mathcal{J}} v_\lambda\theta_\lambda =: \mathbf{v}^{\mathrm{T}}\Theta$, such that the following *norm equivalence* holds

$$\|\mathbf{v}\|_{\ell_2(\mathcal{J})} \sim \|v\|_{L_2}. \tag{2.1.1}$$

This means that there exist constants $c_\Theta, C_\Theta$ such that for every $v \in L_2$

$$\begin{aligned} c_\Theta\|(\langle\tilde{\theta}_\lambda, v\rangle)_{\lambda\in\mathcal{J}}\|_{\ell_2(\mathcal{J})} &\leq \|v\|_{L_2(\Omega)} \leq C_\Theta\|(\langle\tilde{\theta}_\lambda, v\rangle)_{\lambda\in\mathcal{J}}\|_{\ell_2(\mathcal{J})}, \\ C_\Theta^{-1}\|(\langle\theta_\lambda, v\rangle)_{\lambda\in\mathcal{J}}\|_{\ell_2(\mathcal{J})} &\leq \|v\|_{L_2(\Omega)} \leq c_\Theta^{-1}\|(\langle\tilde{\theta}_\lambda, v\rangle)_{\lambda\in\mathcal{J}}\|_{\ell_2(\mathcal{J})}. \end{aligned} \tag{2.1.2}$$

Such anchor bases in $L_2$ are usually constructed with the aid of a *multiresolution analysis*. To describe this mechanism, we start with two sequences of spaces $S_j, \tilde{S}_j \subset L_2(\Omega)$ with

$$\overline{\bigcup_{j\in\mathbb{N}_0} S_j} = \mathcal{H}, \tag{2.1.3}$$

and likewise for the dual spaces $\tilde{S}_j$. The spaces $S_j, \tilde{S}_j$ are usually defined as the span of corresponding *generator bases* $\Phi_j = \{\phi_{j,k} : j \in \mathbb{N}_0; k \in \mathcal{I}_j\}$,

$$S_j = S(\Phi_j) := \overline{\operatorname{span}\Phi_j}$$

and likewise $\tilde{S}_j = S(\tilde{\Phi}_j)$. Focusing first on the *primal scaling functions* $\phi_{j,k}$, we will require that the basis $\Phi_j$ is *uniformly stable*

$$\|\mathbf{c}\|_{\ell_2(\mathcal{I}_j)} \sim \|\mathbf{c}^{\mathrm{T}}\Phi_j\|_{\mathcal{H}}, \quad j \in \mathbb{N}_0.$$

Here we used the compact notation of a (formal) vector product

$$\mathbf{c}^{\mathrm{T}}\Phi_j := \sum_{k\in\mathcal{I}_j} c_{j,k}\phi_{j,k}, \tag{2.1.4}$$

viewing $\Phi_j := \Phi(\mathcal{I}_j)$ as a column vector. By $\ell_2(I)$ we will always denote the subspace of $\ell_2(\mathcal{J})$ containing sequences whose components with index not in $I \subset \mathcal{J}$ are zero. Hence a vector will be interpreted as an infinite

sequence with only finite support and the support of an element $\mathbf{v} \in \ell_2(\mathcal{J})$ will always be understood as

$$\operatorname{supp} \mathbf{v} := \{\lambda \in \mathcal{J}, v_\lambda \neq 0\}.$$

The functions $\phi_{j,k}$ are usually derived from a single refinable function $\phi$ by dilation and integer translation, i.e.,

$$\phi_{j,k}(\cdot) = 2^{jd/2}\phi(2^{j/2} \cdot -k), \quad j \in I\!N_0, k \in \mathcal{I}_j, \tag{2.1.5}$$

$d$ being the spacial dimension. In the so called *shift invariant* univariate case $\mathcal{I}_j = \mathbb{Z}$.

Nestedness of the spaces can then be expressed by a *two-scale* or *refinement relation*, which says that any coarse scale basis function can be written as a linear combination of functions of the finer scales. Using again the shorthand notation (2.1.4), the refinement relation can be written as

$$\Phi_j^{\mathsf{T}} = \Phi_{j+1}^{\mathsf{T}} \mathbf{M}_{j,\Phi}, \tag{2.1.6}$$

where the $k$-th column of $\mathbf{M}_{j,\Phi}$ consists of the mask of $\phi_{j,k}$.

For the *dual scaling functions* $\tilde{\phi}_{j,k}$, we will analogously require compact support and refinability, i.e.,

$$\tilde{\Phi}_j^{\mathsf{T}} = \tilde{\Phi}_{j+1}^{\mathsf{T}} \mathbf{M}_{j,\tilde{\Phi}}.$$

The bases $\Phi, \tilde{\Phi}$ shall be connected by the following duality relation

$$\langle \phi_{j,k}, \tilde{\phi}_{j,k'} \rangle = \delta_{k,k'}, \quad k, k' \in \mathcal{I}_j, \tag{2.1.7}$$

which implies $\mathbf{M}_{j,\Phi}^T \mathbf{M}_{j,\tilde{\Phi}} = \mathbf{I}$. Now we can use this to define projectors onto the spaces $S_j, \tilde{S}_j$

$$P_j := \sum_{k \in \mathcal{I}_j} \langle \cdot, \tilde{\phi}_{j,k} \rangle \tilde{\phi}_{j,k}, \quad \tilde{P}_j := \sum_{k \in \mathcal{I}_j} \langle \cdot, \phi_{j,k} \rangle \phi_{j,k}. \tag{2.1.8}$$

One then looks for complement spaces $W_j, \tilde{W}_j$ such that

$$S_{j+1} = S_j \oplus W_j, \quad \tilde{S}_{j+1} = \tilde{S}_j \oplus \ W_j. \tag{2.1.9}$$

For larger flexibility, we do not enforce orthogonality $W_j \perp S_j$ but *biorthogonality*

$$\tilde{W}_j \perp S_j, \quad W_j \perp \tilde{S}_j.$$

Note that, because of the nestedness of $S_j, \tilde{S}_j$ the differences

$$Q_j := P_j - P_{j-1} \quad \text{and} \quad \tilde{Q}_j := \tilde{P}_j - \tilde{P}_{j-1} \tag{2.1.10}$$

are again projectors and they map onto $W_j, \tilde{W}_j$.
Wavelets form bases for these $L_2$-complement spaces

$$W_j = S(\Theta_j), \quad \tilde{W}_j = S(\tilde{\Theta}_j),$$

where again $\theta_{j,k}$ is obtained by a *mother wavelet* via the relation $\theta_{j,k}(\cdot) = 2^{jd/2}\theta(2^j \cdot -k)$. We will distinguish wavelets and scaling functions in one dimension by a subscript $e \in \{0, 1\}$ and we will sometimes write

$$\phi_{j,k} =: \theta_{j-1,k,0}, \quad \text{and } \theta_{j,k} =: \theta_{j,k,1}, \quad j \in \mathbb{N}_0, k \in \mathbb{Z},$$

which also explains the notation $\mathbf{M}_{j,0}, \mathbf{M}_{j,1}$ and $\tilde{\mathbf{M}}_{j,0}, \tilde{\mathbf{M}}_{j,1}$. For convenience we will condense scale, translation and type in one index

$$\lambda := (j, k, e),$$

with $|\lambda| := j, k(\lambda) := k$ and $e(\lambda) := e$. Concerning notation, also note that we will distinguish the set of wavelet indices $\mathcal{J}$ from the set of scaling function indices $\mathcal{I}$ and that we will identify $(j, k)$ with $\lambda = (j, k, 0)$, which explains the notation $\lambda \in \mathcal{I}$. Using these conventions, the wavelet bases for the $j$-th complement spaces are given by

$$\Theta_j = \{\theta_\lambda : \lambda \in \mathcal{J}_j\} \quad \text{and} \quad \tilde{\Theta}_j = \{\tilde{\theta}_\lambda : \lambda \in \mathcal{J}_j\},$$

and

$$\Theta := \Phi_0 \cup \bigcup_{j \in \mathbb{N}_0} \Theta_j, \quad \tilde{\Theta} := \tilde{\Phi}_0 \cup \bigcup_{j \in \mathbb{N}_0} \tilde{\Theta}_j.$$

Biorthogonality can then also be expressed as

$$\langle \Theta, \tilde{\Theta} \rangle = \mathbf{I}, \tag{2.1.11}$$

where we used the convention to denote the generalized infinite Gramian of two bases (of $L_2$) as

$$\langle \Theta_1, \Theta_2 \rangle := (\langle \theta_1, \theta_2 \rangle)_{\theta_1 \in \mathcal{J}, \theta_2 \in \mathcal{J}}.$$

Note that since $\Theta_j \subset S_{j+1}$ and $\tilde{\Theta}_j \subset \tilde{S}_{j+1}$ there must exist $(\#\mathcal{I}_{j+1}) \times (\#\mathcal{J}_j)$-matrices $\mathbf{M}_{j,\Theta}$, and $\mathbf{M}_{j,\tilde{\Theta}}$ such that

$$\Theta_j^{\mathsf{T}} = \Phi_{j+1}^{\mathsf{T}} \mathbf{M}_{j,\Theta}, \quad \tilde{\Theta}_j^{\mathsf{T}} = \tilde{\Phi}_{j+1}^{\mathsf{T}} \mathbf{M}_{j,\tilde{\Theta}}, \tag{2.1.12}$$

and biorthogonality again gives $\mathbf{M}_{j,\Theta}^T \mathbf{M}_{j,\tilde{\Theta}} = \mathbf{I}$.

As for the scaling functions, we will require *locality* also for the wavelets, i.e. for

$$\operatorname{supp} \phi_\lambda := \sigma_\lambda, \quad \operatorname{supp} \tilde{\phi}_\lambda := \tilde{\sigma}_\lambda, \qquad \operatorname{supp} \theta_\lambda := \Omega_\lambda, \quad \operatorname{supp} \tilde{\theta}_\lambda := \tilde{\Omega}_\lambda,$$

there shall always exist some $\varpi \in I\!N$

$$B_\lambda \subseteq 2^{-|\lambda|}[k - \varpi, k + \varpi], \quad B \in \{\sigma, \tilde{\sigma}, \Omega, \tilde{\Omega}\}, \tag{2.1.13}$$

so that in particular $\operatorname{diam}(B_\lambda) \sim 2^{-|\lambda|}$. As a consequence, the two-scale matrices $\mathbf{M}_{j,e}, \tilde{\mathbf{M}}_{j,e}$, $e \in \{0,1\}$ are *uniformly sparse* which means that the number of non-zero entries per row and column in these matrices remains uniformly bounded in $j$.

The next basic property we require from our bases is that the wavelets $\theta$ of the primal system will be orthogonal to polynomials up to the order $\tilde{m}$

$$\langle x^i, \theta \rangle = 0, \quad i = 0, \ldots, \tilde{m} - 1. \tag{2.1.14}$$

In this case, we will say that $\theta$ has $\tilde{m}$ *vanishing moments*.

The consequence is a so-called *cancellation property*, stating the existence of some $\tilde{m} \in I\!N$ such that

$$\langle v, \theta_\lambda \rangle \lesssim 2^{-|\lambda|(\frac{d}{2}+\tilde{m})} |v|_{W_\infty^{\tilde{m}}(\Omega_\lambda)}. \tag{2.1.15}$$

Clearly $\tilde{m}$ reflects the accuracy of the dual multi-resolution spaces $\tilde{S}_j := S(\tilde{\Phi}_j)$ which are called $\tilde{m}$-th order accurate, if

$$\inf_{\tilde{v}_j \in \tilde{S}_j} \|v - \tilde{v}_j\|_{L_2} \lesssim 2^{-j\tilde{m}} |v|_{H^{\tilde{m}}}. \tag{2.1.16}$$

Due to (2.1.6) and (2.1.12) a change between the two bases $\Phi_{j+1}$ and $\Phi_j \cup \Theta_j$ of $S_{j+1}$ can be described by the matrices

$$\mathbf{M}_j := (\mathbf{M}_{j,\Phi}, \mathbf{M}_{j,\Theta}), \tag{2.1.17}$$

mapping $\ell_2(\mathcal{I}_j) \oplus \ell_2(\mathcal{J}_j)$ onto $\ell_2(\mathcal{I}_{j+1})$. The fact that $S(\Phi_{j+1}) = S(\Phi_j) \oplus S(\Theta_j$ implies that $\mathbf{M}_j$ has to be invertible and it was shown in [CDP96] that the bases $\Phi_j \cup \Theta_j$ is uniformly stable if and only if

$$\|\mathbf{M}_j\|, \ \|\mathbf{M}_j^{-1}\| = \mathcal{O}(1), j \in I\!N, \tag{2.1.18}$$

where $\|\cdot\|$ denotes the spectral norm. The analogous reasoning holds for

$$\tilde{\mathbf{M}}_j := (\mathbf{M}_{j,\tilde{\Phi}}, \mathbf{M}_{j,\tilde{\Theta}}). \tag{2.1.19}$$

The construction of wavelets can therefore also be interpreted as the determination of a *stable completion* for $\mathbf{M}_{j,\Phi}$ ($\mathbf{M}_{j,\tilde{\Phi}}$), i.e., the task to find $\mathbf{M}_{j,\Theta}$ ($\mathbf{M}_{j,\tilde{\Theta}}$) such that $\mathbf{M}_j$ ($\tilde{\mathbf{M}}_j$ respectively) is regular and 2.1.18 holds. For further details, see [Dah97]. It is convenient to block $\mathbf{M}_j^{-1}$, $\tilde{\mathbf{M}}_j^{-1}$ as

$$\mathbf{M}_j^{-1} =: \mathbf{G}_j = \begin{pmatrix} \mathbf{G}_{j,\Phi} \\ \mathbf{G}_{j,\Theta} \end{pmatrix}, \quad \tilde{\mathbf{M}}_j^{-1} =: \tilde{\mathbf{G}}_j = \begin{pmatrix} \mathbf{G}_{j,\tilde{\Phi}} \\ \mathbf{G}_{j,\tilde{\Theta}} \end{pmatrix}, \tag{2.1.20}$$

so that

$$\mathbf{M}_j\mathbf{G}_j = \mathbf{G}_j\mathbf{M}_j = \mathbf{M}_{j,\Phi}\mathbf{G}_{j,\Phi} + \mathbf{M}_{j,\Theta}\mathbf{G}_{j,\Theta} = \mathbf{I},$$

which, in particular, implies because of (2.1.6) and (2.1.12)

$$\Phi_{j+1}^T = \Phi_j^T\mathbf{G}_{j,\Phi} + \Theta_j^T\mathbf{G}_{j,\Theta}. \tag{2.1.21}$$

Also, note that

$$\mathbf{G}_{j,\Phi} := \mathbf{M}_{j,\tilde{\Phi}}{}^T, \quad \mathbf{G}_{j,\Theta} := \mathbf{M}_{j,\tilde{\Theta}}{}^T, \quad \mathbf{G}_{j,\tilde{\Phi}} := \mathbf{M}_{j,\Phi}{}^T, \quad \mathbf{G}_{j,\tilde{\Theta}} := \mathbf{M}_{j,\Theta}{}^T. \tag{2.1.22}$$

Each function $g_J \in S_J$, $J \in \mathbb{N}$, can be decomposed into *coarse scale* information situated in $S_{J-1}$ and the *details* in $W_{J-1}$

$$g_J = \sum_{k\in I_J} c_{J,k}\phi_{J,k} = \sum_{k\in I_{J-1}} c_{J-1,k}\phi_{J-1,k} + \sum_{k\in\mathcal{J}_{J-1}} d_{J-1,k}\theta_{J-1,k}. \tag{2.1.23}$$

This leads to the *multi-scale representation* of $g_J$, given by

$$g_J = \sum_{k\in I_0} c_{0,k}\phi_{0,k} + \sum_{j=0}^{J-1}\sum_{k\in\mathcal{J}_j} d_{j,k}\theta_{j,k} = \sum_{j=-1}^{J-1}\sum_{k\in\mathcal{J}_j} d_{j,k}\theta_{j,k},$$

where $S_0 := W_{-1}$, $\theta_{-1,k} := \phi_{0,k}$ . The representation of $g_J$ in terms of scaling functions on some level $J$ will be referred to as its *single-scale representation*.

Concerning the expansion coefficients, (2.1.23) takes the form

$$\mathbf{c}_J = (\mathbf{M}_{J,0}, \mathbf{M}_{J,1})\begin{pmatrix} \mathbf{c}_{J-1} \\ \mathbf{d}_{J-1} \end{pmatrix}. \tag{2.1.24}$$

To describe the successive *reconstruction* that takes the multi-scale representation into the single-scale representation, one defines

$$\mathbf{T}_J := \mathbf{T}_J^{J-1}\dots\mathbf{T}_1^0, \quad \text{with} \quad \mathbf{T}_j^{j-1} := \operatorname{diag}(\mathbf{M}_j, \mathbf{I}).$$

The *decomposing* inverse transformation $\mathbf{T}_j^{-1}$ is realized by successive application of

$$\Phi_{j+1}^{\mathsf{T}}\mathbf{c}_{j+1} = \Phi_j^{\mathsf{T}}(\mathbf{G}_{j,0}\mathbf{c}_{j+1}) + \Theta_j^{\mathsf{T}}(\mathbf{G}_{j,1}\mathbf{c}_{j+1}) = \Phi_j^{\mathsf{T}}\mathbf{c}_j + \Theta_j^{\mathsf{T}}\mathbf{d}_j. \tag{2.1.25}$$

Hence,

$$\mathbf{T}_J^{-1} = \mathbf{T}_0^1 \dots \mathbf{T}_{J-1}^J, \quad \text{with} \quad \mathbf{T}_{j-1}^j := \mathrm{diag}(\mathbf{G}_{j-1}, \mathbf{I}).$$

The multi-scale transformation $\mathbf{T}_J$ is uniformly well conditioned for $J \to \infty$ in the sense that

$$\|\mathbf{T}_J\|, \|\mathbf{T}_J^{-1}\| = \mathcal{O}(1), J \in I\!N,$$

if and only if the biorthogonal multi-scale bases $\Theta, \tilde{\Theta}$ both are $L_2$-Riesz bases, see [Dah94, Dah96, Dah97].

Given a dual pair of $L_2$-anchor bases one can generate a whole scale of bases for other function spaces reflecting for instance different scales of smoothness. The most prominent example are Sobolev spaces $H^s$ for some range $s \in (-\tilde{\gamma}, \gamma)$ (where for $s < 0$ we define $H^s = (H^{-s})'$ by duality) but other variants of energy spaces such as $\|v\|_{\mathcal{H}}^2 = \nu\|\nabla v\|_{L_2(\Omega)}^2 + \|v\|_{L_2(\Omega)}^2$ are covered as well [Dah03]. Here $H^s$ stands for $H^s(\Omega)$ or for a closed subspace of $H^s(\Omega)$ defined e.g. by homogeneous boundary conditions. In this case a suitable diagonal scaling of an $L_2$ basis yields one for a Sobolev space: Let $\mathbf{D} := \mathrm{diag}(\omega_\lambda, \lambda \in \mathcal{J})$, where typically $\omega_\lambda = 2^{-|\lambda|s}$, then

$$\|\mathbf{D}^{-1}\mathbf{d}\|_{\ell_2(\mathcal{J})} \sim \|\mathbf{d}^{\mathsf{T}}\Theta\|_{H^t(\Omega)} \quad \text{and} \quad \|\mathbf{D}\mathbf{d}\|_{\ell_2(\mathcal{J})} \sim \|\mathbf{d}^{\mathsf{T}}\tilde{\Theta}\|_{H^{-t}(\Omega)}. \tag{2.1.26}$$

Therefore, $\Psi := \mathbf{D}\Theta, \tilde{\Psi} := \mathbf{D}^{-1}\tilde{\Theta}$, i.e.

$$\psi_\lambda := \omega\theta_\lambda \quad \text{and} \quad \tilde{\psi}_\lambda := \omega^{-1}\tilde{\theta}_\lambda, \quad \lambda \in \mathcal{J},$$

are indeed Riesz bases for $H^s, H^{-s}$ respectively.

Note that the scaling can be easily incorporated directly in the two scale relations. We immediately infer from (2.1.12) that

$$\Psi_j^T = \Phi_{j+1}^T\mathbf{M}_{j,\Psi}, \quad \mathbf{M}_{j,\Psi} := \mathbf{M}_{j,\Theta}\mathbf{D}_j. \tag{2.1.27}$$

Moreover, since $\tilde{\Psi}_j^T = \tilde{\Theta}_j^T\mathbf{D}_j^{-1}$, we infer from (2.1.21)

$$\tilde{\Phi}_{j+1}^T = \tilde{\Phi}_j^T\mathbf{G}_{j,\tilde{\Phi}} + \tilde{\Psi}_j^T\mathbf{G}_{j,\tilde{\Psi}}, \quad \text{where} \quad \mathbf{G}_{j,\tilde{\Psi}} := \mathbf{D}_j\mathbf{G}_{j,\tilde{\Theta}}. \tag{2.1.28}$$

In the context of adaptivity, confer Chapter 1, it is important to note, that such *norm equivalences* extend to the Besov spaces of positive regularity

$s > 0$, see e.g. [Dah96, DeV98, Dah97]. If $\Theta, \tilde{\Theta}$ are biorthogonal wavelet bases in $L_2$, one has

$$\|v\|^q_{B^s_q(L_p)} \sim \sum_j \left( \sum_{|\lambda|=j} 2^{jq(s+\frac{d}{2}-\frac{d}{p})} |d_\lambda|^p \right)^{q/p}.$$

There exist elegant constructions of wavelets in arbitrary dimensions satisfying all of the above requirements also for non trivial domain geometries [CTU98, CTU00, DS99a, DS99b, CM00].

However, we will not be restricted to a special construction of wavelets, yet we will always require that the basic assumptions summarized in the following hold:

**Assumption 2.1.1.** *We will always assume that we are given a dual pair $\langle \Psi, \tilde{\Psi} \rangle$ of biorthogonal wavelet bases in some Hilbert space $\mathcal{H}$ and its dual $\mathcal{H}'$ respectively. Each $v \in \mathcal{H}$ shall have a unique wavelet expansion $v = \mathbf{v}^T \Psi$ with $\mathbf{v} = (\langle v, \tilde{\psi}_\lambda \rangle)_{\lambda \in \mathcal{J}}$. Moreover $\Psi$ is*

- *local, i.e.,* $\operatorname{diam} \operatorname{supp} \psi_\lambda, \sim 2^{-|\lambda|}$,
- *stable in $\mathcal{H}$ in the sense that $\|\mathbf{v}\|_{\ell_2} \sim \|\mathbf{v}^T \Psi\|_{\mathcal{H}}$, cf. (2.1.26),*
- *provides a cancellation property, $\langle v, \psi_\lambda \rangle \lesssim 2^{-|\lambda|(\frac{d}{2}+\tilde{m})} \|v\|_{W^{\tilde{m}}_\infty}$, cf. (2.1.15).*

*The dual basis $\tilde{\Psi}$ is required to satisfy the analogous conditions with $\mathcal{H}$ and $\tilde{m}$ canonically replaced by $\tilde{\mathcal{H}}$ and $m$.*

One widespread approach to construct wavelets in higher dimensions is a classical tensor product ansatz

$$\theta_\lambda(\mathbf{x}) = \theta_{j,\mathbf{k},\mathbf{e}}(x_1, x_2, \ldots, x_d) = \prod_{i=1}^d \theta_{j,k_i,e_i}(x_i), \tag{2.1.29}$$

with $j \in I\!N_0, \mathbf{k} = (k_1, \ldots, k_d) \in Z\!\!Z^d, \mathbf{e} = (e_1, \ldots e_d) \in \{0,1\}^d$. In this case, we get $2^d - 1$ types of wavelets ($\mathbf{e} \in \{0,1\}^d \backslash \mathbf{0}$) and one (type of) scaling function ($\mathbf{e} = \mathbf{0}$). The masks of the corresponding functions are obtained by tensor products of the univariate masks. In the tensor product case, we will describe changes of bases analogously to (2.1.17) and (2.1.20) by setting

$$\mathbf{M}_j = (\mathbf{M}_{j,0}, \mathbf{M}_{j,1}, \ldots, \mathbf{M}_{j,2^d-1}), =: (\mathbf{M}_{j,\Phi}, \mathbf{M}_{j,\Theta}) \tag{2.1.30}$$

and likewise

$$\mathbf{G}_j = (\mathbf{G}_{j,0}, \mathbf{G}_{j,1}, \ldots, \mathbf{G}_{j,2^d-1}), =: (\mathbf{G}_{j,\Phi}, \mathbf{G}_{j,\Theta})^{\mathtt{T}}. \tag{2.1.31}$$

Tensor products of the spline wavelet family developed in [CDF92] fit into this setting. As a meanwhile widespread discretization tool, this family is of particular interest in practice and we will treat spline wavelets as our model-type of wavelet basis throughout this thesis. We will therfore review some of their properties in the next section. For further information on wavelet analysis and construction, the reader is referred to one of the textbooks on wavelets [Chu92, Dah97, Dau92, KLR95, Mey92, Woj97].

## 2.2 Cardinal B-spline Wavelets

Cardinal B-spline functions are meanwhile a standard discretization tool for wavelet methods treating e.g., operator equations. We will use the term *cardinal B-spline wavelets* for wavelet systems according to [CDF92] because the underlying refinable function is a cardinal $B$-spline. One of the most important features of these splines at least for our context is that they allow fast point evaluation at arbitrary points.

In general, the cardinal $B$-spline ${}_\kappa\varphi$ of order $\kappa \in \mathbb{N}$ is a piecewise polynomial of degree $\kappa - 1$ which can be defined recursively as a convolution product, i.e.,

$$\begin{aligned} {}_1\varphi(x) &:= \chi_{[0,1)}(x), \\ {}_\kappa\varphi(x) &:= ({}_{\kappa-1}\varphi * {}_1\varphi)(x) = \int_{\mathbb{R}} {}_{\kappa-1}\varphi(x-y)\,{}_1\varphi(y)\,dy. \end{aligned}$$

One can show that cardinal $B$-splines have the following properties: They are non-negative ${}_\kappa\varphi(x) \geq 0$ and normalized $\int_{\mathbb{R}} {}_\kappa\varphi(x)dx = 1$. Their integer shifts form a partition of unity $\sum_{k\in\mathbb{Z}} {}_\kappa\varphi(x-k) = 1$ , their support is contained in $[0,\kappa]$ and they are refinable with

$${}_\kappa\varphi = 2^{1-\kappa} \sum_{k=0}^{\kappa} \binom{\kappa}{k} {}_2\varphi(2\cdot - k).$$

Moreover, ${}_\kappa\varphi \in C^{\kappa-2}$ and the derivatives for $\kappa > 1$ satisfy

$$\frac{d}{dx}{}_\kappa\varphi(x) = {}_{\kappa-1}\varphi(x) - {}_{\kappa-1}\varphi(x-1). \tag{2.2.1}$$

For the evaluation of ${}_\kappa\varphi$ at a given point $x \in \mathbb{R}$, one uses an alternative definition of $B$-splines

$$\kappa = 1 \quad : \quad {}_\kappa\varphi^i(t) = {}_\kappa\varphi^1(t) = \begin{cases} 1 & t_i \le t < t_{i+1} \\ 0 & \text{elsewhere} \end{cases}, \tag{2.2.2}$$

$$\kappa > 1 \quad : \quad {}_\kappa\varphi^i(t) = \frac{t - t_i}{t_{i+k} - t_i} {}_{\kappa-1}\varphi^i(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} {}_{\kappa-1}\varphi^{i+1}(t), \tag{2.2.3}$$

where $\mathcal{N} := (t_0, t_1, \ldots, t_k)$, $t_i < t_{i+1}$, $i = 0, 1, \ldots, k-1$ is a given *knot-vector*. In the case of *cardinal* $B$-splines the knots are chosen to be integer (or cardinal) numbers $t_i = i$. Given ${}_\kappa\varphi(t)$, a repeated application of (2.2.3) finally yields a representation of ${}_\kappa\varphi(t)$ consisting only of splines of order 1, hence (2.2.2) is applicable. This principle leads to the fast and stable evaluation scheme known as the *de Boor algorithm*, see e.g. [Sch39].

In the case $\kappa = 2$, we obtain the classical symmetric hat function by an integer shift

$${}_2\phi(x) := {}_2\varphi(x+1) = \begin{cases} 1 + x, & -1 \le x < 0, \\ 1 - x, & 0 \le x \le 1, \\ 0, & \text{otherwise.} \end{cases} \tag{2.2.4}$$

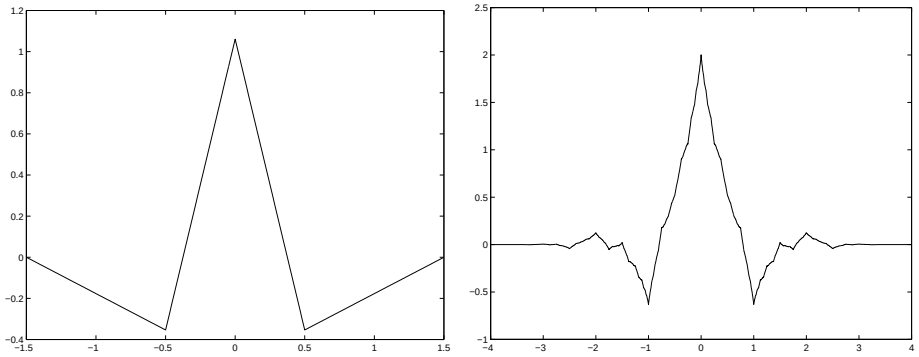
As for the hat function, one often centralizes the spline functions by a shift

$${}_\kappa\phi(x) = {}_\kappa\varphi(x + \left\lfloor \frac{\kappa}{2} \right\rfloor),$$

so that one obtains symmetry around $\kappa^* := (\kappa \bmod 2)/2$, i.e. ${}_\kappa\phi(x - \kappa^*) = {}_\kappa\phi(-x + \kappa^*)$. As a consequence, the function as well as the associated refinement mask is supported in $[-\lfloor \frac{\kappa}{2} \rfloor, \lceil \frac{\kappa}{2} \rceil]$. For further details on splines including computational remarks we refer to [Sch39].

For cardinal $B$-splines, suitable dual generators are known. In fact, in [CDF92], families of refinable functions ${}_{\kappa,\tilde{\kappa}}\tilde{\phi}$, $\kappa + \tilde{\kappa}$ even, were constructed such that a biorthogonality relation of the form (2.1.7) is satisfied.

The next figure shows primal wavelet according to the hat function (2.2.4) and one of its duals, ${}_{2,4}\tilde{\psi}$.



(a) Wavelet according to ${}_{2,2}\phi$ (b) Dual scaling function ${}_{2,4}\tilde{\phi}$

Figure 2.1: Wavelet basis functions

The pair of cardinal $B$-spline wavelet bases according to two parameters $\kappa$ and $\tilde{\kappa}$ will henceforth be denoted by $N(\kappa, \tilde{\kappa})$ and we will use later on, that the mask of the dual function ${}_{\kappa,\tilde{\kappa}}\tilde{\phi}$ is supported in $[-\lfloor \frac{\kappa}{2} \rfloor - \tilde{\kappa} + 1, \lceil \frac{\kappa}{2} \rceil + \tilde{\kappa} - 1]$.

## 2.3 Tree–like Structured Index Sets

In principle, the best possible (in terms of the ratio accuracy/cardinality) set $\Lambda$ for an approximant $u_\Lambda$ to a target function $u$ could be a completely unstructured selection of indices. However, for our algorithm, we will require certain structural features such as 'tree structure', 'completeness' and 'gradedness', to be explained in a moment.

Working with *tree approximations*, i.e., with linear combinations of wavelets whose indices form a 'complete tree', has turned out to be essential in the analysis of adaptive wavelet schemes for *nonlinear* problems of the form (3.1.8), see [CDD03a, CDD03d]. Yet, such structural demands are also imposed for adaptive schemes with different background, e.g. one can find similar conditions for meshes used in adaptive schemes based on cell averages in the context of hyperbolic conservation laws and finite volume methods, see e.g. [Mül03].

**Support cubes:** Recall that $\lambda = (|\lambda|, \mathbf{k}, \mathbf{e})$ where $\mathbf{k} = \mathbf{k}(\lambda)$ is a multi-index encoding the spatial location and $\mathbf{e} = \mathbf{e}(\lambda) \in \{0,1\}^d \setminus \mathbf{0}$ refers to the *type* of the wavelet. The index sets $\mathcal{J}_j$ identify the true wavelets $\psi_\lambda$, $|\lambda| = j, \mathbf{e}(\lambda) \neq \mathbf{0}$, on level $j$. The type $\mathbf{e} = \mathbf{0}$ is always reserved to the

scaling functions and we will therefore identify $(j, \mathbf{k}, \mathbf{0})$ with $\lambda^\circ = (j, \mathbf{k})$, $|\lambda^\circ| = j, \mathbf{k}(\lambda^\circ) = \mathbf{k}$. The set of scaling function indices $(j, \mathbf{k})$ on level $j$ is denoted by $\mathcal{I}_j$ while $\mathcal{I}$ denotes the union of these sets over all levels.

For all constructions mentioned one can associate with $\lambda^\circ = (j, \mathbf{k})$ a *support cell* $\square_{\lambda^\circ}$ with the following properties:

$$\square_{\lambda^\circ} \subseteq \operatorname{supp} \psi_\lambda, \quad |\lambda| = j, \quad \mathbf{k}(\lambda) = \mathbf{k}. \tag{2.3.1}$$

Moreover, $\mathcal{P}_j := \{\square_{\lambda^\circ} : \lambda^\circ \in \mathcal{I}_j\}$ shall form a hierarchy of nested partitions of $\Omega$, i.e. $\bar{\Omega} = \bigcup_{\lambda^\circ \in \mathcal{I}_j} \square_{\lambda^\circ}$ and each $\square_{\lambda^\circ}$ is an essentially disjoint union of support cells of level $|\lambda| + 1$. In the simplest case of translation invariant scaling functions the support cells have the form

$$\square_{j,\mathbf{k}} := 2^{-j}(\mathbf{k} + [0, 1]^d). \tag{2.3.2}$$

In what follows, we will often identify $\lambda \in \mathcal{J}$ with the support cell $\square_{\lambda^\circ}$ and by $\Lambda^\circ$ we will denote the index-set of support cells associated with the set of indices in $\Lambda \subset \mathcal{J}$. We will make use of the fact, that $\Lambda^\circ \subset \mathcal{I}$. Note, that except for the case of Haar-wavelets, $\square_{\lambda^\circ} \subsetneq \operatorname{supp} \psi_\lambda$ and that the difference grows with the order of the wavelets.

Figure 2.2 shows a non-uniform set of wavelet coefficients in one dimension represented by support cubes. The different gray-shades correspond to the size of the associated coefficient value.

**Parent/child:** The $\mathcal{P}_j$ induce a natural hierarchy with respect to set inclusion. Whenever $\square_{\mu^\circ} \subsetneq \square_{\lambda^\circ}, \mu^\circ, \lambda^\circ \in \mathcal{I}$, we say that $\mu^\circ$ is a *descendant* of $\lambda^\circ$ which we express by $\mu^\circ \succ \lambda^\circ$. When equality is permitted we write $\mu^\circ \succeq \lambda^\circ$. Conversely $\lambda^\circ$ is called an *ancestor* of $\mu^\circ$, which will be denoted by $\lambda^\circ \prec \mu^\circ$ (or $\lambda^\circ \preceq \mu^\circ$). When $|\lambda^\circ| = |\mu^\circ| - 1$, $\mu^\circ$ is called a *child* of $\lambda^\circ$ and $\lambda^\circ$ is referred to as the *parent* of $\mu^\circ$, which will be expressed by the notation $\lambda^\circ = \mathcal{P}(\mu^\circ)$. Here every dyadic cube $\square_{\lambda^\circ}, \lambda^\circ \in \mathcal{J}_j^\circ, j > j_0$ has a *unique* parent $\mathcal{P}(\lambda^\circ) = \lfloor k(\lambda^\circ)/2 \rfloor$, yet note that uniqueness is not necessarily required for general constructions of support cubes. If $\mathcal{P}(\lambda^\circ) = \mathcal{P}(\mu^\circ)$, $\mu^\circ$ will be called a *sibling* of $\lambda^\circ$.

**Complete trees:** A subset $\mathcal{T}^\circ \subset \mathcal{I}$ is called a *tree* if for each $\lambda^\circ \in \mathcal{T}^\circ$ with $|\lambda^\circ| > j_0$ also $\mathcal{P}(\lambda^\circ) \in \mathcal{T}^\circ$. It will greatly simplify data structures when a tree is *complete* in the sense that whenever $\lambda^\circ \in \mathcal{T}^\circ$ implies that *all* the siblings of $\lambda^\circ$ also belong to $\mathcal{T}^\circ$. We shall work in what follows exclusively with complete trees.

On account of the above association of a support cube with those indices in $\mathcal{J}$ sharing the same level and spatial index $\mathbf{k}$ the tree structure of $\mathcal{I}$ induces a *tree-like structure* in $\mathcal{J}$ by saying that $\mathcal{T} \subset \mathcal{J}$ has tree structure if $\mathcal{T}^\circ$ is

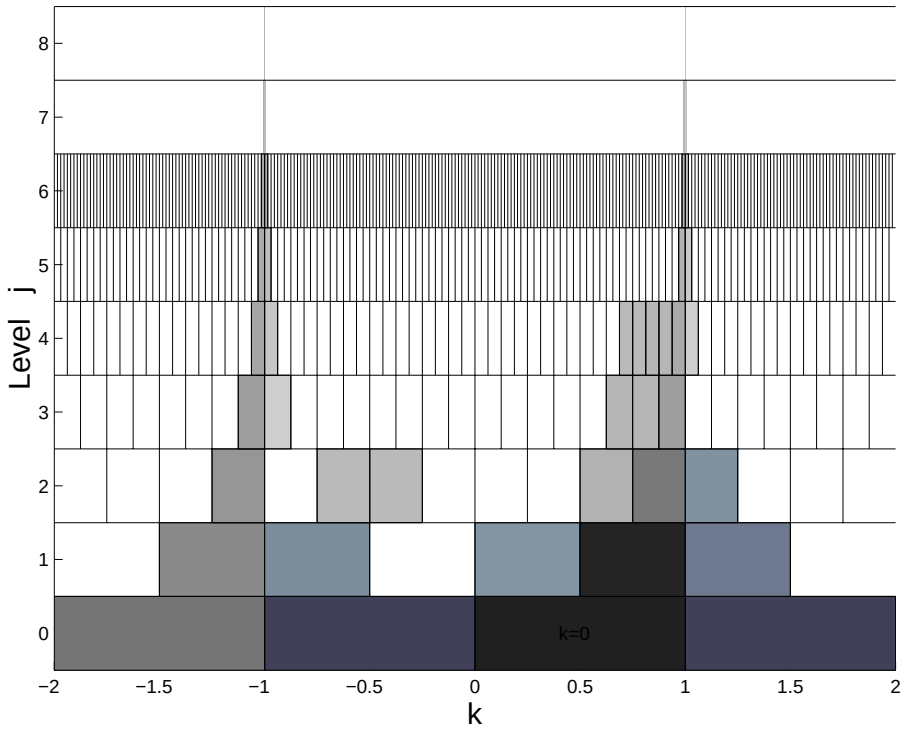



Figure 2.2: Non-uniform index set represented by support cubes

a tree in the above sense. Note that therefore $\lambda \in \mathcal{T}$ implies that *all* $\mu \in \mathcal{T}$ with $|\mu| = |\lambda|$ and $\mathbf{k}(\mu) = \mathbf{k}(\lambda)$. We shall again refer to this property as *completeness* and call also the set $\mathcal{T}$ (in slight abuse of terminology) a *(complete) tree*.

It is worth the following two remarks: Of course, the tree structure imposes a restriction on the selection of active wavelets when compared with unconstrained $N$-term approximation. Yet, permitting only tree-like sets $\mathcal{T}$ in the best $N$-term approximation in $H^t$, cf. Section 1, the desired error rate $N^{-s}$ is obtained for $v \in B_q^{t+sd}(L_p)$ as long as $\frac{1}{p} < \frac{s}{d} + \frac{1}{2}$, cf. C2) in Section 1.1 and see [DeV98, CDDD01]. Compared to the unrestricted case, this does just no longer include Besov spaces where $\frac{1}{p} = \frac{s}{d} + \frac{1}{2}$, and the restriction is in that sense not as strong as one might expect. For a more detailed discussion of this restriction we refer to [CDD03a, CDD03b].

Secondly, the notion of complete trees is quite natural at least regarding the output of a heuristic adaptive strategy. We indeed automatically end up with complete trees, when we compute an initial approximation on a coarse scale and adapt it by incorporating the descendants of indices with 'big' error contributions until a certain stopping criterion is fulfilled.

Yet note that an approximation process consisting in thresholding or a refined prediction strategy may result in an index set not exhibiting the latter features. In that regard however, it is important to mention, that in [CDD03c], thresholding strategies have been investigated that will produce tree shaped index sets. Those 'tree-coarsening' schemes are based on a special penalized thresholding by Binev/DeVore.

**Leaves:** The set of *leaves* $\partial\mathcal{T}$ of a tree $\mathcal{T}^\circ \subset \mathcal{I}$ consists of those $\lambda^\circ \in \mathcal{T}^\circ$ which belong to $\mathcal{T}^\circ$ but none of their children are in $\mathcal{T}^\circ$,i.e., The support cube $\square_{\lambda^\circ}$ and likewise $\lambda^\circ$ is called a leaf if none of the children of $\square_{\lambda^\circ}$ corresponds to an index in $\Lambda^\circ$. One may confirm that $\#\mathcal{T}^\circ \sim \#\partial\mathcal{T}^\circ$. In a complete tree $\mathcal{T}^\circ$, the level $|\lambda|$ is the highest level of resolution on $\square_\lambda$ for $\lambda^\circ \in \partial\mathcal{T}^\circ$.

We shall have to deal also with the set $\partial\mathcal{T}^-$ of *outer leaves* of $\mathcal{T}^\circ$ which consists of those $\lambda^\circ \notin \mathcal{T}^\circ$ whose parent is in $\mathcal{T}^\circ$. It is easy to see that the set of outer leaves $\partial\mathcal{T}^-$ forms a partition of $\Omega$ consisting of support cubes. Hence the span of wavelets from a complete tree $\mathcal{T} \subset \mathcal{J}$ are associated in a natural way with a locally refined mesh, namely $\partial\mathcal{T}^-$. Thus the adaptation potential offered by spans of wavelets whose indices form a complete tree is comparable to trial spaces on locally refined meshes.

We will make use of the fact that the support cubes of the leaves of a complete tree form a disjoint non-uniform partition of $\square$, i.e.,

$$\bigcup_{\lambda^\circ \in \partial\Lambda^\circ} \square_{\lambda^\circ} = \square. \tag{2.3.3}$$

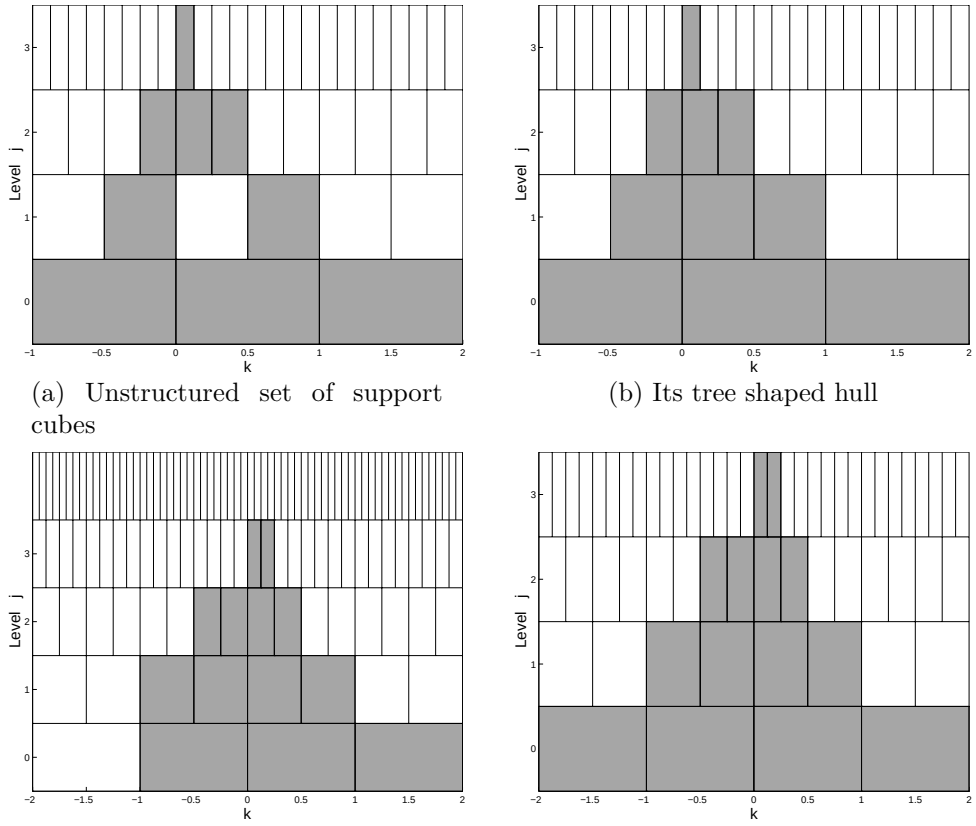
Of course, the set of leaves of the *fully refined sets* $\Lambda(J) := \{\lambda \in \mathcal{J} : |\lambda| \leq J\}$ is just the set of cubes $\{\square_\lambda : \lambda \in \mathcal{J}_J\}$ forming a uniform mesh of mesh size $2^{-J}$. By contrast, the set of leaves $\partial\Lambda^\circ$ of a *sparse* tree $\Lambda$ induces, in view of (2.3.3), a *non-uniform locally refined* mesh composed of the dyadic cubes in $\partial\Lambda^\circ$, see e.g.Figure 2.3 d).

**Gradedness** As we will see later on, our structural demands will insist on a certain *grading*, which is again best explained through the dyadic partitions. 'Graded' means that in the partition $\partial\mathcal{T}^\circ$ any two neighboring cubes, i.e., cubes which share some lower dimensional face, differ at most by *one* dyadic level. Thus the transition between cubes of different size is gradual.

**Remark 2.3.1.** *It has been shown in [Dah82] that every tree $\mathcal{T}^\circ$ can be extended to a graded tree $\hat{\mathcal{T}}^\circ$ such that $\#\hat{\mathcal{T}}^\circ \lesssim \#\mathcal{T}^\circ$ where the constant depends only on the spatial dimension $d$.*

Figure 2.3 below shows an example of an unstructured set of support

cubes and its minimal tree-shaped hull which is then completed and finally graded.




(a) Unstructured set of support cubes

(b) Its tree shaped hull

(c) The completed tree

(d) The completed and graded tree

Figure 2.3: A set of support cubes under various structural demands

We even quantify gradedness to further disentangle the transition between dyadic levels: A tree $\mathcal{T}^\circ \subset \mathcal{J}^\circ$ is called $M$-graded if for all $\lambda^\circ \in \mathcal{T}^\circ$

$$\mathbf{k}(\mathcal{P}(\lambda^\circ)) + [-M, M+1]^d \subseteq \mathcal{T}^\circ. \tag{2.3.4}$$

Note that when $M \geq \frac{5}{2}\varpi$ one has, in view of (2.1.13), that

$$U(\partial\mathcal{T}_j) \cap U(\partial\mathcal{T}_{j-2}) = \emptyset, \tag{2.3.5}$$

where for any finite subset $\Gamma \subset \mathcal{J}$

$$U(\Gamma) := \bigcup_{\lambda\in\Gamma} \tilde{\Omega}_\lambda,$$

denotes the domain covered by the dual scaling functions in $\Gamma$. Thus, only supports of wavelets associated with leaves from at most two consecutive levels overlap, smaller $M$ would permit more levels to interact.

Figure 2.4 shows the 2-graded complete hull of the example set of support cubes from Figure 2.3.

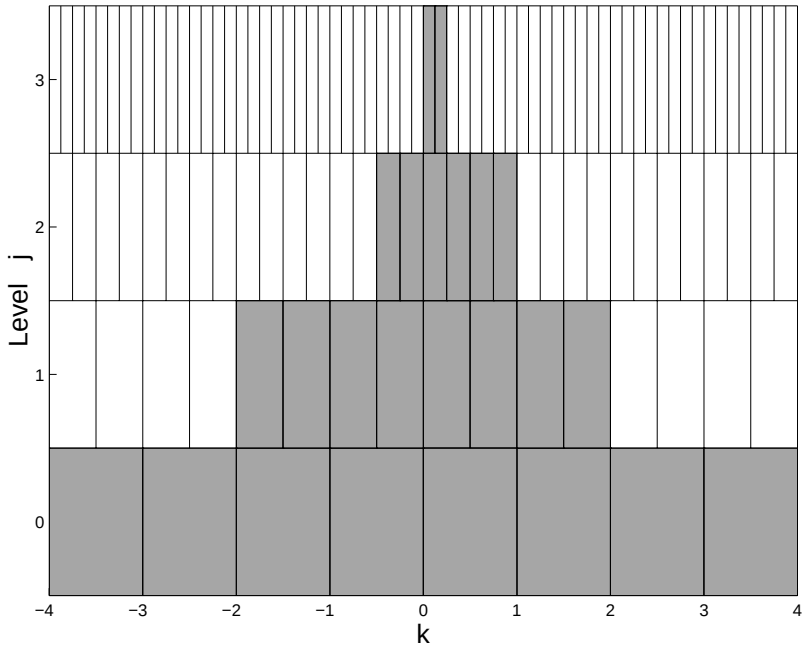



Figure 2.4: A complete, non uniform 2-graded index tree

Obviously one can pass from any graded tree $\mathcal{T}^\circ$ to an $M$-graded tree $\tilde{\mathcal{T}}^\circ$ by $M$ dyadic subdivisions of the leaves. Hence one still has $\#\tilde{\mathcal{T}}^\circ \lesssim \#\mathcal{T}^\circ$ which together with Remark 2.3.1 says that at least asymptotically the restriction to $M$-graded trees is not essential. It may well matter quantitatively as observed earlier in [DSX00] which is one of the main reasons for developing an alternative to the scheme from [DSX00]. In Section 4.2 we will discuss in detail procedures to find the minimal $M$-graded tree containing a given unstructured index set.

# Part II

# Development of the Tools

**Overview:** Part II is the central part of this thesis. Here we present and analyze our main tools, above all the recovery algorithm RECOVER, cf. [BDS04].
In Chapter 3, we first review the background of the recovery task to understand how RECOVER fits in the context of adaptive schemes. We then turn to the layout of the scheme and its analysis for two types of spaces, namely $L_2$ and in duals of $H^t, t > 0$.
The demands of RECOVER lead us to the construction of several supporting tools, described in Chapter 4. The main issues there are quadrature, structural requirements and a thresholding-strategy to minimize overhead due to erroneous prediction.

# Chapter 3

# The Recovery Scheme

The central problem treated in this thesis will be explained now in a sufficiently general format in order to accommodate later several different cases of interest. We will always assume that for the dual pair $(\mathcal{H}, \mathcal{H}')$ of Hilbert-spaces under consideration, we are given a biorthogonal pair of wavelet basis satisfying our basic assumption 2.1.1.

## 3.1 Objectives and Background

The core objective can be formulated as the following *recovery* Task R which has two parts:

**Task R1**: Given $g \in \mathcal{H}'$ and some finite subset $\mathcal{T} \subset \mathcal{J}$ of indices,

compute an array $\mathbf{w}$ supported in $\mathcal{T}$ that approximates

$$\mathbf{g}(\mathcal{T}) := (g_\lambda)_{\lambda\in\mathcal{T}} := (\langle \psi_\lambda, g\rangle)_{\lambda\in\mathcal{T}}.$$

**Task R2**: Moreover, whenever for a given target accuracy

$\varepsilon > 0$ the truncation error satisfies

$$\|\mathbf{g} - \mathbf{g}(\mathcal{T})\|_{\ell_2(\mathcal{J})} \le \varepsilon, \tag{3.1.1}$$

the approximation $\mathbf{w}$ should satisfy for some fixed constant $C$

$$\|\mathbf{g}(\mathcal{T}) - \mathbf{w}\|_{\ell_2(\mathcal{J})} \le C\varepsilon, \tag{3.1.2}$$

I.E. THE APPROXIMATION ERROR SHOULD BE COMPARABLE TO A GIVEN BOUND FOR THE TRUNCATION ERROR.

To understand this task, recall that the Riesz basis property of $\Psi$ implies the existence of a *dual basis* $\tilde{\Psi} \subset \mathcal{H}'$ such that $\langle \psi_\lambda, \tilde{\psi}_\mu \rangle = \delta_{\lambda,\mu}$. Thus, the $g_\lambda = \langle \psi_\lambda, g \rangle$ are the expansion coefficients of $g \in \mathcal{H}'$ with respect to the dual basis

$$g = \sum_{\lambda \in \mathcal{J}} \langle \psi_\lambda, g \rangle \tilde{\psi}_\lambda.$$

Moreover, a duality argument yields (see e.g. [Dah03])

$$C_\Psi^{-1} \|(\langle \psi_\lambda, w \rangle)_{\lambda \in \mathcal{J}}\|_{\ell_2(\mathcal{J})} \le \|w\|_{\mathcal{H}'} \le c_\Psi^{-1} \|(\langle \psi_\lambda, w \rangle)_{\lambda \in \mathcal{J}}\|_{\ell_2(\mathcal{J})}, \quad \forall \ w \in \mathcal{H}'. \tag{3.1.3}$$

Hence

$$\|\mathbf{w} - \mathbf{g}(\mathcal{T})\|_{\ell_2(\mathcal{J})} \le C_\Psi \| \sum_{\lambda \in \mathcal{T}} w_\lambda \tilde{\psi}_\lambda - \sum_{\lambda \in \mathcal{T}} g_\lambda \tilde{\psi}_\lambda \|_{\mathcal{H}'}, \tag{3.1.4}$$

i.e. the accuracy of $\mathbf{w}$ as an approximation to $\mathbf{g}(\mathcal{T})$ is controlled by an *approximation error* for the truncated expansion $g_\mathcal{T} := \sum_{\lambda \in \mathcal{T}} g_\lambda \tilde{\psi}_\lambda$ in $\mathcal{H}'$.

As we will explain later in more detail, the approximation of the finite array $\mathbf{g}(\mathcal{T})$ is often just a vehicle for fabricating an approximation to the *whole infinite* array $\mathbf{g}$. The rationale then is that, from some background information, the set $\mathcal{T}$ is known to contain the most significant terms $\langle \psi_\lambda, g \rangle$ of the whole expansion. Since, due to the norm equivalences, accuracy is measured in $\ell_2(\mathcal{J})$, "most significant" means here the largest in modulus. The relevant error is then controlled by

$$\|\mathbf{w} - \mathbf{g}\|_{\ell_2(\mathcal{J})} \le C_\Psi \| \sum_{\lambda \in \mathcal{T}} w_\lambda \tilde{\psi}_\lambda - \sum_{\lambda \in \mathcal{T}} g_\lambda \tilde{\psi}_\lambda \|_{\mathcal{H}'} + \|\mathbf{g} - \mathbf{g}(\mathcal{T})\|_{\ell_2(\mathcal{J})}, \tag{3.1.5}$$

which explains TASK R2. In fact, ideally the *approximation error* of the finite expansion in the first term should be at most of the same order as (a bound for) the *prediction error* in the second term.

One might think of many situations where the computation of wavelet coefficients is needed. The considerations in the present investigations are guided primarily by the following two scenarios whose brief description may help identifying the particular computational demands.

**$L_2$-approximation:** The first case concerns simply $\mathcal{H} = L_2(\Omega)$. Thus $\mathcal{H}' = \mathcal{H} = L_2(\Omega)$ and the objective is to approximate $g$ in $L_2(\Omega)$ by some finite expansion $\sum_{\lambda \in \mathcal{T}} w_\lambda \tilde{\psi}_\lambda$ (where the role of primal and dual basis does

not matter). By (3.1.4), the accuracy of the coefficient array is comparable to the approximation error in $L_2$ which will later be seen to simplify matters and is essentially the situation considered in [DSX00].

**Dual norms:** The second scenario typically involves topologies where $\mathcal{H}$ is compactly embedded in $L_2(\Omega)$ such as a Sobolev space of positive order. A corresponding model problem may be formulated as follows, see [CDD03a] for more details. Given $\Omega \subset I\!R^d$, consider

$$-\mathrm{div}(a\nabla u) + G(u) = f \quad \text{in} \quad \Omega, \quad u = 0 \quad \text{on} \quad \partial\Omega, \tag{3.1.6}$$

where $G : v \mapsto G \circ v = G(v)$ is a possibly nonlinear composition map and $a$ is a (possibly variable) uniformly positive definite matrix. For this problem to be well-posed, cf. (0.0.2), it helps looking at the weak formulation

$$\langle \nabla v, a\nabla u\rangle + \langle v, G(u)\rangle = \langle v, f\rangle, \quad v \in \mathcal{H}, \tag{3.1.7}$$

where now $\mathcal{H}$ is to be chosen suitably. Clearly the leading second order term suggests $\mathcal{H} = H_0^1(\Omega)$ (the space of $L_2$-functions on $\Omega$ whose first order derivatives are also in $L_2$ and whose trace on $\partial\Omega$ vanishes). Thus we require that $G$ maps $\mathcal{H} = H_0^1(\Omega)$ into $\mathcal{H}' = H^{-1}(\Omega)$ so that (3.1.7) makes sense whenever the data $f$ belong to $\mathcal{H}' = H^{-1}(\Omega)$ as well. The general format of such problems can be stated as follows. Given $F : \mathcal{H} \to \mathcal{H}'$ and any $f \in \mathcal{H}'$ we wish to find a $u \in \mathcal{H}$ such that

$$\langle v, F(u)\rangle = \langle v, f\rangle \qquad \forall \quad v \in \mathcal{H}. \tag{3.1.8}$$

In the above example it is not hard to confirm well-posedness when $G$ is monotone so that (3.1.7) is the Euler-Lagrange equation of a strictly convex minimization problem. For instance,

$$G(v) := v^3 \quad \Longrightarrow \quad G : v \mapsto G(v),\ G : H_0^1(\Omega) \to H^{-1}(\Omega), \quad \text{for} \quad d \le 3. \tag{3.1.9}$$

To explain how Task R fits into this context recall from the introduction that the strategy proposed in [CDD03a], [CDD03b] deviates from conventional approaches in that the problem is first transformed with the aid of the basis $\Psi$ into an equivalent one that lives now on $\ell_2(\mathcal{J})$. In fact, testing both sides of $F(u) = f$ with all basis functions $\psi_\lambda, \lambda \in \mathcal{J}$ generates for $u = \sum_{\lambda\in\mathcal{J}} u_\lambda\psi_\lambda$ the arrays $\mathbf{F}(\mathbf{u}) = (\langle\psi_\lambda, F(u)\rangle)_{\lambda\in\mathcal{J}}$, $\mathbf{f} = (\langle\psi_\lambda, f\rangle)_{\lambda\in\mathcal{J}}$ so that (3.1.8) is equivalent to

$$\mathbf{F}(\mathbf{u}) = \mathbf{f}, \tag{3.1.10}$$

where $\mathbf{u}$ is the array of wavelet coefficients of $u$ with respect to the Riesz basis $\Psi$ for $\mathcal{H}$. One then can formulate an iteration in $\ell_2(\mathcal{J})$, see (0.0.3)

$$\mathbf{u}^{n+1} = \mathbf{u}^n - \mathbf{C}_n(\mathbf{F}(\mathbf{u}^n) - \mathbf{f}), \quad n = 0, 1, 2, \ldots, \tag{3.1.11}$$

so that (for a suitable initial guess $\mathbf{u}_0$ with $u_0$ in some neighborhood $\mathcal{U}$ of $\mathbf{u}$) the error is reduced in each step by at least a fixed factor $\rho < 1$, i.e.

$$\|\mathbf{u} - \mathbf{u}^{n+1}\|_{\ell_2(\mathcal{J})} \leq \rho \|\mathbf{u} - \mathbf{u}^n\|_{\ell_2(\mathcal{J})}, \quad n = 0, 1, 2, \ldots. \tag{3.1.12}$$

Only then one addresses a numerical realization by carrying out such an iteration approximately. The key task is then to evaluate *adaptively* for a given finitely supported input array $\mathbf{v}$ the image $\mathbf{F}(\mathbf{v})$ within some dynamically updated accuracy tolerance. Such an evaluation hinges on two pillars, namely first the *a-posteriori* information on the given input $v$ (i.e. the knowledge of the significant coefficients in $\mathbf{v}$) and, second, the *a-priori* information on the mapping $F$, $\mathbf{F}$ respectively.

Aside from the norm equivalences (2.1.2), at this point the second key feature of wavelet bases namely the *cancellation properties* (2.1.15), comes into play see [CDD03d].

This will allow one then to *predict* the indices hosting the significant coefficients of the image $\mathbf{F}(\mathbf{v})$, cf. (0.0.4). (The prediction set $\mathcal{T}$ will actually in general not be completely arbitrary but will have tree structure, see Section 2.3 and cf. [CDD03d]).

Thus here we have $g = F(v) \in \mathcal{H}'$ and the set of significant coefficients identified by the index set $\mathcal{T}$ is *known* from the prediction in step S1). For the concrete construction of *asymptotically optimal* prediction sets for a certain class of linear and nonlinear operators $F$ (of at most polynomial growth at infinity) we refer to [CDD03d]. In brief [CDD03d] show that, given a target accuracy $\varepsilon > 0$, one can predict an $\varepsilon$-significant tree $\mathcal{T} = \mathcal{T}_\varepsilon$ such that $\|\mathbf{g} - \mathbf{g}(\mathcal{T})\|_{\ell_2(\mathcal{J})} \leq \varepsilon$ while the cardinality of $\mathcal{T}_\varepsilon$ grows with decreasing $\varepsilon$ at some optimal rate.

In [CDD03a] the precise requirements on the approximation of $\mathbf{F}(\mathbf{v})$ have been identified under which the overall adaptive scheme has asymptotically optimal complexity, see the notion of $s^*$-sparsity in [CDD03a]. Together with the above mentioned prediction results these requirements are met whenever Task R is fulfilled.

Therefore we focus in this thesis on the recovery step step S2), cf. (0.0.5), which is exactly Task R. It will be important that the accuracy in the $\ell_2(\mathcal{J})$-error (0.0.4) is here related to the accuracy of the corresponding functions (or distributions) in $\mathcal{H}'$, due to the above mentioned mapping properties of $F$.

Of course, in order to benefit from the ability of the above schemes to optimally track the significant coefficients of the unknown solution, one would like to compute the entries of $\mathbf{w}$ in a possibly efficient way which means at a computational cost (in terms of floating point operations and storage manipulations) that ideally stays proportional to $\#\mathcal{T}$. Moreover, one has to assert the accuracy of such computations e.g. via estimates like (3.1.5).

A first natural idea would be to compute each individual quantity $\langle \psi_\lambda, g\rangle$ by quadrature. However, a quick thought confirms that this would never allow one to keep the above desirable computational budget. In fact, the coarse scale wavelets give rise to integration domains that are comparable to the size of $\Omega$ and each of these entries would require a growing quadrature effort with increasing accuracy. In addition wavelets from different scales still interact.

Instead we shall pursue a different strategy ([BDS04]) and take up an idea suggested already in [DSX00]. As already pointed out in the introduction, the algorithm in this thesis improves earlier findings from [DSX00] in several respects concerning the algorithmic side as well as the complexity analysis to be now suited for modern adaptive wavelet schemes.

## 3.2 The Algorithm

We address now Task R. $\mathcal{T}$ denotes always a complete finite tree and we will assume henceforth, that the biorthogonal Riesz bases $\Psi, \tilde{\Psi}$ for $H$ is obtained through diagonal scaling of a biorthogonal Riesz bases $\Theta, \tilde{\Theta}$ for $L_2$ as described in Section 2.1, i.e., $\Psi := \mathbf{D}\Theta, \tilde{\Psi} := \mathbf{D}^{-1}\tilde{\Theta}$ with $\mathbf{D} := \mathrm{diag}(\omega_\lambda, \lambda \in \mathcal{J})$ or

$$\psi_\lambda := \omega_\lambda \theta_\lambda, \qquad \tilde{\psi}_\lambda := \omega_\lambda^{-1}\tilde{\theta}_\lambda, \quad \lambda \in \mathcal{J}, \tag{3.2.1}$$

Recall that by setting $\mathbf{D}_j := \mathrm{diag}\,(\omega_\lambda : \lambda \in \mathcal{J}_j)$, we can incorporate the scaling in the refinement (2.1.12) in the following form

$$\Psi_j^T = \Phi_{j+1}^T \mathbf{M}_j^\Psi, \qquad \mathbf{M}_j^\Psi := \mathbf{M}_j^\Theta \mathbf{D}_j. \tag{3.2.2}$$

By (3.2.1) $\tilde{\Psi}_j^T = \tilde{\Theta}_j^T \mathbf{D}_j^{-1}$ we have, cf. (2.1.28)

$$\tilde{\Phi}_{j+1}^T = \tilde{\Phi}_j^T \mathbf{G}_j^\Phi + \tilde{\Psi}_j^T \mathbf{G}_j^\Psi, \quad \text{where} \quad \mathbf{G}_j^\Psi := \mathbf{D}_j \mathbf{G}_j^\Theta. \tag{3.2.3}$$

According to Task R, we have to compute a sequence $\mathbf{w} = (w_\lambda)_{\lambda\in\mathcal{T}}$ with $\mathrm{supp}\,\mathbf{w} \subseteq \mathcal{T}$ that approximates the array

$$\mathbf{g}(\mathcal{T}) := (\langle \psi_\lambda, g\rangle)_{\lambda\in\mathcal{T}}.$$

Keep in mind that the entries $\langle\psi_\lambda, g\rangle$ are nothing but the wavelet coefficients of $g = \sum_{\lambda\in\mathcal{J}}\langle\psi_\lambda, g\rangle\tilde{\psi}_\lambda$ with respect to the *dual basis* $\tilde{\Psi}$ whose projection to the span of $\tilde{\Psi}_\mathcal{T} := \{\tilde{\psi}_\lambda : \lambda \in \mathcal{T}\}$ is denoted by $g_\mathcal{T} = \sum_{\lambda\in\mathcal{T}}\langle\psi_\lambda, g\rangle\tilde{\psi}_\lambda$. Of course, since $\langle\psi_\lambda, g\rangle\tilde{\psi}_\lambda = \langle\theta_\lambda, g\rangle\tilde{\theta}_\lambda$, it suffices to compute the array

$$\tilde{\mathbf{d}}(\mathcal{T}) = (\langle\psi_\lambda, g\rangle)_{\lambda\in\mathcal{T}} = (\langle\psi_\lambda, g_\mathcal{T}\rangle)_{\lambda\in\mathcal{J}}, \tag{3.2.4}$$

to obtain

$$\mathbf{g}(\mathcal{T}) = \mathbf{D}_\mathcal{T}\tilde{\mathbf{d}}(\mathcal{T}), \qquad \mathbf{D}_\mathcal{T} := \mathrm{diag}\,(\omega_\lambda : \lambda \in \mathcal{T}). \tag{3.2.5}$$

### 3.2.1 Motivation and Main Idea

The scheme we are going to develop next is based on the coefficients $\langle\psi_\lambda, g\rangle$, $\lambda \notin \mathcal{T}$, being presumably small in a sense that will be made more precise later. Specifically, we shall at this point only make use of the fact (to be established later) that quadrature errors are small on $\mathrm{supp}\,\tilde{\phi}_{j,\mathbf{k}}$ whenever $\square_{j,\mathbf{k}}$ is a support cell whose index belong to $\partial\mathcal{T}^-$, the set of outer leaves of $\mathcal{T}^\circ$. Recall that the support cells associated with $\partial\mathcal{T}^\circ$, form a partition of $\Omega$. On account of these accuracy considerations we shall speak of *safe approximations/quadrature* whenever it applies to the quantities $\langle\phi_{j,\mathbf{k}}, g\rangle$ for $(j,\mathbf{k}) \notin \mathcal{T}^\circ$.

We shall explain next how to use this in order to compute correspondingly accurate approximations of all $\langle\Theta_\lambda, g\rangle$ (resp. $\langle\Psi_\lambda, g\rangle$), $\lambda \in \mathcal{T}$. We deliberately postpone a detailed discussion of accuracy issues and focus first on the computational ingredients.

We shall abbreviate in the following

$$c_{j,\mathbf{k}} := \langle\phi_{j,\mathbf{k}}, g\rangle, \quad (j,\mathbf{k}) \in \mathcal{I}_j, \quad d_\lambda := \langle\Theta_\lambda, g\rangle, \quad \lambda \in \mathcal{J}, \tag{3.2.6}$$

and write for any subset $\mathcal{G}$ of $\mathcal{I}_j$ or of $\mathcal{J}$ briefly

$$\mathbf{c}_j(\mathcal{G}) := (c_{j,\mathbf{k}} : (j,\mathbf{k}) \in \mathcal{G}), \qquad \mathbf{d}(\mathcal{G}) := (d_\lambda : \lambda \in \mathcal{G}),$$

where we simply set $\mathbf{c}_j = \mathbf{c}_j(\mathcal{I}_j)$, $\mathbf{d}_j := \mathbf{d}(\mathcal{J}_j)$. In terms of these coefficients (3.2.10) takes, upon using the two-scale relations (2.1.6), (2.1.12), the form

$$\tilde{\Theta}_j^T\mathbf{d}_j = \tilde{\Phi}_{j+1}^T\mathbf{M}_{j,\tilde{\Theta}}\mathbf{d}_j = \tilde{\Phi}_{j+1}^T(\mathbf{c}_{j+1} - \mathbf{M}_{j,\tilde{\Phi}}\mathbf{c}_j), \tag{3.2.7}$$

whence we conclude that

$$\mathbf{c}_{j+1} - \mathbf{M}_{j,\tilde{\Phi}}\mathbf{c}_j = \mathbf{M}_{j,\tilde{\Theta}}\mathbf{d}_j. \tag{3.2.8}$$

We shall employ the canonical projectors

$$P_j w = \sum_{\mathbf{k}\in\mathcal{I}_j} \langle \phi_{j,\mathbf{k}}, w\rangle \tilde{\phi}_{j,\mathbf{k}} = \sum_{\mathbf{k}\in\mathcal{I}_j} c_{j,\mathbf{k}} \tilde{\phi}_{j,\mathbf{k}} \tag{3.2.9}$$

and exploit the fact that

$$(P_{j+1} - P_j) w = \sum_{\lambda\in\mathcal{J}_j} \langle \Theta_\lambda, w\rangle \tilde{\Theta}_\lambda. \tag{3.2.10}$$

To point out the significance of (3.2.8), it will be convenient to introduce some further notation. Let $\mathcal{T}_j := \mathcal{T} \cap \mathcal{J}_j$ denote the set of elements in $\mathcal{T}$ of level $j$ and let $J = \max\{j \geq j_0 : \mathcal{T}_j \neq \emptyset\}$ the highest level appearing in $\mathcal{T}$. If $\mathbf{d}_j$ is supported on $\mathcal{T}_j$, (3.2.8) says that the array $\mathbf{c}_{j+1} - \mathbf{M}_{j,\tilde{\Phi}}\mathbf{c}_j$ vanishes outside the union of the columns of $\mathbf{M}_{j,\tilde{\Theta}}$ selected by $\mathcal{T}_j$. We record this observation as follows.

**Remark 3.2.1.** *Denoting for $\lambda \in \mathcal{I} \cup \mathcal{J}_j$ by $\tilde{\mathbf{M}}_j{}^{|\lambda}$, $\overline{\tilde{\mathbf{G}}_j{}^{\lambda}}$ the $\lambda$-th column of $\tilde{\mathbf{M}}_j$, respectively the $\lambda$-th row of $\tilde{\mathbf{G}}_j$, and defining $\mathcal{G}_{j+1} \subset \mathcal{I}_{j+1}$ by*

$$\mathcal{G}_{j+1} := \bigcup_{\lambda\in\mathcal{T}_j^\circ} \operatorname{supp} \tilde{\mathbf{M}}_j{}^{|\lambda} \cup \bigcup_{\lambda\in\mathcal{T}_j^\circ} \operatorname{supp} \overline{\tilde{\mathbf{G}}_j{}^{\lambda}}, \tag{3.2.11}$$

*one concludes from (3.2.8) that, whenever $\operatorname{supp} \mathbf{d}_j \subseteq \mathcal{T}_j$, one has*

$$\big(\mathbf{c}_{j+1} - \mathbf{M}_{j,\tilde{\Phi}}\mathbf{c}_j\big)(\mathcal{I}_{j+1} \setminus \mathcal{G}_{j+1}) = \mathbf{0}. \tag{3.2.12}$$

Concrete identifications of the sets $\mathcal{G}_j$ for the widespread cardinal B-spline wavelets, cf. Section 2.2, will be given in Section 3.4.

Relation (3.2.12) means that if we approximate the values of $\mathbf{c}_{j+1}$ and $\mathbf{c}_j$ only for a subset of indices, such truncated arrays combined via (3.2.8) cause no error outside a certain finite set of indices. The following considerations aim at exploiting this fact systematically.

To this end, one derives from (2.1.21) that

$$\tilde{\Phi}_{j+1}^T \mathbf{c}_{j+1} = \tilde{\Phi}_j^T \mathbf{G}_{j,\tilde{\Phi}} \mathbf{c}_{j+1} + \tilde{\Theta}_j^T \mathbf{G}_{j,\tilde{\Theta}} \mathbf{c}_{j+1} = \tilde{\Phi}_j^T \mathbf{c}_j + \tilde{\Theta}_j^T \mathbf{d}_j, \tag{3.2.13}$$

so that

$$\mathbf{c}_j = \mathbf{G}_{j,\tilde{\Phi}} \mathbf{c}_{j+1}, \quad \mathbf{d}_j = \mathbf{G}_{j,\tilde{\Theta}} \mathbf{c}_{j+1}. \tag{3.2.14}$$

We wish to compute now as few scaling function coefficients as possible in order to generate in a reliable way the arrays $\mathbf{d}(\mathcal{T}_j)$ of non-vanishing

wavelet coefficients. This requires taking the truncation effects in the two-scale relations into careful account. First, note that, by the second relation in (3.2.14), we only need to know $\mathbf{c}_{j+1}(\mathcal{G}_{j+1})$ to determine $\mathbf{d}_j(\mathcal{T}_j)$ because only the rows $\overline{\tilde{\mathbf{G}}_j{}^{\lambda}}$ with $\lambda \in \mathcal{T}_j$ are required and they are all supported in $\mathcal{G}_{j+1}$. Likewise, the first relation in (3.2.14) says by the same reasoning that $\mathbf{c}_j(\mathcal{T}_j^\circ)$ is accurately determined from $\mathbf{c}_{j+1}(\mathcal{G}_{j+1})$. So we can think of $\mathbf{c}_j(\mathcal{T}_j^\circ)$ being obtained from accurate data on level $j+1$ through (3.2.14). The remaining coefficients $\mathbf{c}_j(\mathcal{I}_j \setminus \mathcal{T}_j^\circ)$ need to be computed independently. So, splitting $\mathbf{c}_j$ into those two parts in (3.2.13), provides, upon making use of (2.1.6),

$$\tilde{\Phi}_{j+1}^T\Big(\mathbf{c}_{j+1} - \mathbf{M}_{j,\tilde{\Phi}}\mathbf{c}_j(\mathcal{I}_j \setminus \mathcal{T}_j^\circ)\Big) = \tilde{\Phi}_j^T(\mathbf{G}_{j,\tilde{\Phi}}\mathbf{c}_{j+1})(\mathcal{T}_j^\circ) + \tilde{\Theta}_j^T\mathbf{d}_j(\mathcal{T}_j). \quad (3.2.15)$$

The main point of this relation can be formulated as follows.

**Remark 3.2.2.** *Relation (3.2.15) remains valid when the full array* $\mathbf{c}_{j+1}$ *and the array* $\mathbf{M}_{j,\tilde{\phi}}\mathbf{c}_j(\mathcal{I}_j \setminus \mathcal{T}_j^\circ)$ *are restricted to* $\mathcal{G}_{j+1}$*, i.e.*

$$\tilde{\Phi}_{j+1}^T\Big(\mathbf{c}_{j+1}(\mathcal{G}_{j+1}) - \big(\mathbf{M}_{j,\tilde{\Phi}}\mathbf{c}_j(\mathcal{I}_j\setminus\mathcal{T}_j^\circ)\big)(\mathcal{G}_{j+1})\Big) = \tilde{\Phi}_j^T(\mathbf{G}_{j,\tilde{\Phi}}\mathbf{c}_{j+1})(\mathcal{T}_j^\circ) + \tilde{\Theta}^T\mathbf{d}_j(\mathcal{T}_j). \quad (3.2.16)$$

**Proof:** This is an immediate consequence of (3.2.12). ∎

Applying now $\mathbf{G}_{j,\tilde{\Phi}}$ to the array $\mathbf{c}_{j+1}(\mathcal{G}_{j+1}) - \big(\mathbf{M}_{j,\tilde{\Phi}}\mathbf{c}_j(\mathcal{I}_j \setminus \mathcal{T}_j^\circ)\big)(\mathcal{G}_{j+1})$ yields by (3.2.14) the safe coefficients $(\mathbf{G}_{j,\tilde{\Phi}}\mathbf{c}_{j+1})(\mathcal{T}_j^\circ)$ and applying $\mathbf{G}_{j,\tilde{\Theta}}$ yields $\mathbf{d}_j(\mathcal{T}_j)$, provided the partial array $\mathbf{c}_{j+1}(\mathcal{G}_{j+1})$ is safe. Moreover, we do not need all of $\mathbf{c}_j(\mathcal{I}_j \setminus \mathcal{T}_j^\circ)$ but only $\mathbf{c}_j(\mathcal{G}_j \setminus \mathcal{T}_j^\circ)$ because, by the above reasoning these are the only relevant coefficients for the next lower level.

The main point conveyed by these observations is that, computing only the entries $\mathbf{c}_{j+1}(\mathcal{G}_{j+1})$ and also a suitable truncation of $\mathbf{c}_j$ on the next lower level, these truncated arrays suffice to represent $\mathbf{c}_{j+1} - \mathbf{M}_{j,\tilde{\Phi}}\mathbf{c}_j$ exactly on all of $\mathcal{I}_{j+1}$ whenever $\mathbf{d}_j$ is supported in $\mathcal{T}_j$. This motivates us to express expansions in terms of quantities like $\mathbf{c}_{j+1} - \mathbf{M}_{j,\tilde{\Phi}}\mathbf{c}_j$. In fact, we can write in view of (3.2.15),

$$\begin{aligned} P_{\mathcal{T}}g &:= P_{j_0}g + \sum_{j=j_0}^{J}(P_{j+1} - P_j)g \\ &= \sum_{j=j_0+1}^{J+1}\tilde{\Phi}_j^T\Big(\mathbf{c}_j(\mathcal{I}_j \setminus \mathcal{T}_j^\circ) - \mathbf{M}_{j-1,\tilde{\Phi}}\mathbf{c}_{j-1}(\mathcal{I}_{j-1} \setminus \mathcal{T}_{j-1}^\circ)\Big) \quad (3.2.17) \end{aligned}$$

In order to derive all necessary information within suitable accuracy tolerances, in view of the above localization considerations, we only have to take into account

$$\mathcal{G}_j^- := \mathcal{G}_j \setminus \mathcal{T}_j^\circ,\ j \le J, \quad \mathcal{G}_{J+1}^- = \mathcal{G}_{J+1}. \tag{3.2.18}$$

**Main idea:** The idea now is to compute the arrays $\mathbf{c}_j(\mathcal{G}_j^-)$ by *quadrature*. Since quadrature is therefore applied only at places where local truncation errors are small one can hope that a fixed quadrature order, independent of the scale, suffices. Note that the above reasoning remains unchanged when the sets $\mathcal{G}_j$ are replaced by somewhat larger sets. We shall point out in Section 3.4 that such supersets are efficiently and easily obtained for an important class of wavelet constructions. So employing for the purpose of analysis the exact format of the sets $\mathcal{G}$ from (3.2.11) does not interfere with practical realizations using slightly larger sets.

In view of the above localization considerations, the idea is to compute only the arrays $\mathbf{c}_j(\mathcal{G}_j^-)$ by quadrature in order to derive all necessary information within suitable accuracy tolerances. This gives rise to the scheme described below in the next section.

### 3.2.2 A Top–To–Bottom Scheme

We shall compute an approximation to $\mathbf{d}(\mathcal{T})$ working from top to bottom, guided by the considerations of the previous section. One main point will be that the vanishing of the coefficients $d_\lambda$ for $\lambda \notin \mathcal{T}$ implies that the scaling function coefficients $c_{j,\mathbf{k}} := \langle \phi_{j,\mathbf{k}}, g_\mathcal{T} \rangle$, $(j,\mathbf{k}) \notin \mathcal{T}^\circ$ can be computed by quadrature with high accuracy. At this point it does not matter yet which quadrature rule is used. Since its choice will be intertwined with a subsequent error analysis we postpone its specification.

As mentioned before, the important point is that quadrature will only be used to approximate the arrays $\mathbf{c}_j(\mathcal{G}_j^-)$. We shall denote these approximations by $\mathbf{q}_j$ which are always understood to be supported on $\mathcal{G}_j^-$. The key is then relation (3.2.16) which is used to successively decompose the representation (3.2.17) from top to bottom. The scaling function coefficients produced on the next lower level are then those obtained by quadrature on the sets $\mathcal{G}_j^-$ complemented by those living on $\mathcal{T}_j^\circ$ obtained from transforming safe coefficients from higher levels.

This leads to the scheme RECOVER that determines for a given tree $\mathcal{T}$ an approximation $\mathbf{d}^{\mathrm{R}}(\mathcal{T})$ supported on $\mathcal{T}$ to the wavelet coefficients of $g_\mathcal{T}$ with respect to the dual basis $\tilde{\Theta}$ as follows:

RECOVER — $(g, \mathcal{T}) \to [\mathbf{d}^{\mathrm{R}}(\mathcal{T})]$

| |
|---|
| Find minimal $J \in I\!N$ such that $\mathcal{T}_j = \emptyset$, $j > J$ |
| Determine $\mathcal{G}_{J+1}^- = \mathcal{G}_{J+1}$, compute $\mathbf{q}_{J+1}(\mathcal{G}_{J+1}^-)$ and set $\check{\mathbf{c}}_{J+1} := \mathbf{0}$. |
| For $j = J, J-1, \ldots, j_0$ do |
| Determine $\mathcal{G}_j^- = \mathcal{G}_j \backslash \mathcal{T}_j$ and $\mathbf{q}_j(\mathcal{G}_j^-)$ |
| Set $\bar{\mathbf{c}}_{j+1} := \check{\mathbf{c}}_{j+1} + \mathbf{q}_{j+1}(\mathcal{G}_{j+1}^-) - \left(\mathbf{M}_{j,\tilde{\Phi}}\mathbf{q}_j(\mathcal{G}_j^-)\right)\left(\mathcal{G}_{j+1}^-\right)$ (3.2.19) |
| Compute $\check{\mathbf{c}}_j := \mathbf{G}_{j,\tilde{\Phi}}\bar{\mathbf{c}}_{j+1}$, and $\mathbf{d}_j^{\mathrm{R}} := \mathbf{G}_{j,\tilde{\Theta}}\bar{\mathbf{c}}_{j+1}$ (3.2.20) |
| Set $\mathbf{d}_{j_0-1}^{\mathrm{R}} := \mathbf{c}_{j_0}^{\mathrm{R}} := \check{\mathbf{c}}_{j_0}$ |

Algorithm 3.1: Procedure RECOVER: Basic algorithm of the recovery scheme

From now on we define for the sake of convenience $\mathbf{d}_{j_0-1} := \mathbf{c}_{j_0}$ and $\mathbf{d}_{j_0-1}^{\mathrm{R}} := \mathbf{c}_{j_0}^{\mathrm{R}}$.

**Remark 3.2.3.** *Note that the support of $\mathbf{d}_j^R$ is always contained in $\mathcal{T}_j$. The scheme requires computing the quantities $q_{j,\mathbf{k}}$ as approximations to $c_{j,\mathbf{k}} = \langle \phi_{j,\mathbf{k}}, g \rangle$ only on the set*

$$\mathcal{G}^- := \mathcal{G}^-(\mathcal{T}) = \bigcup_{j=j_0}^{J+1} \mathcal{G}_j^-. \tag{3.2.21}$$

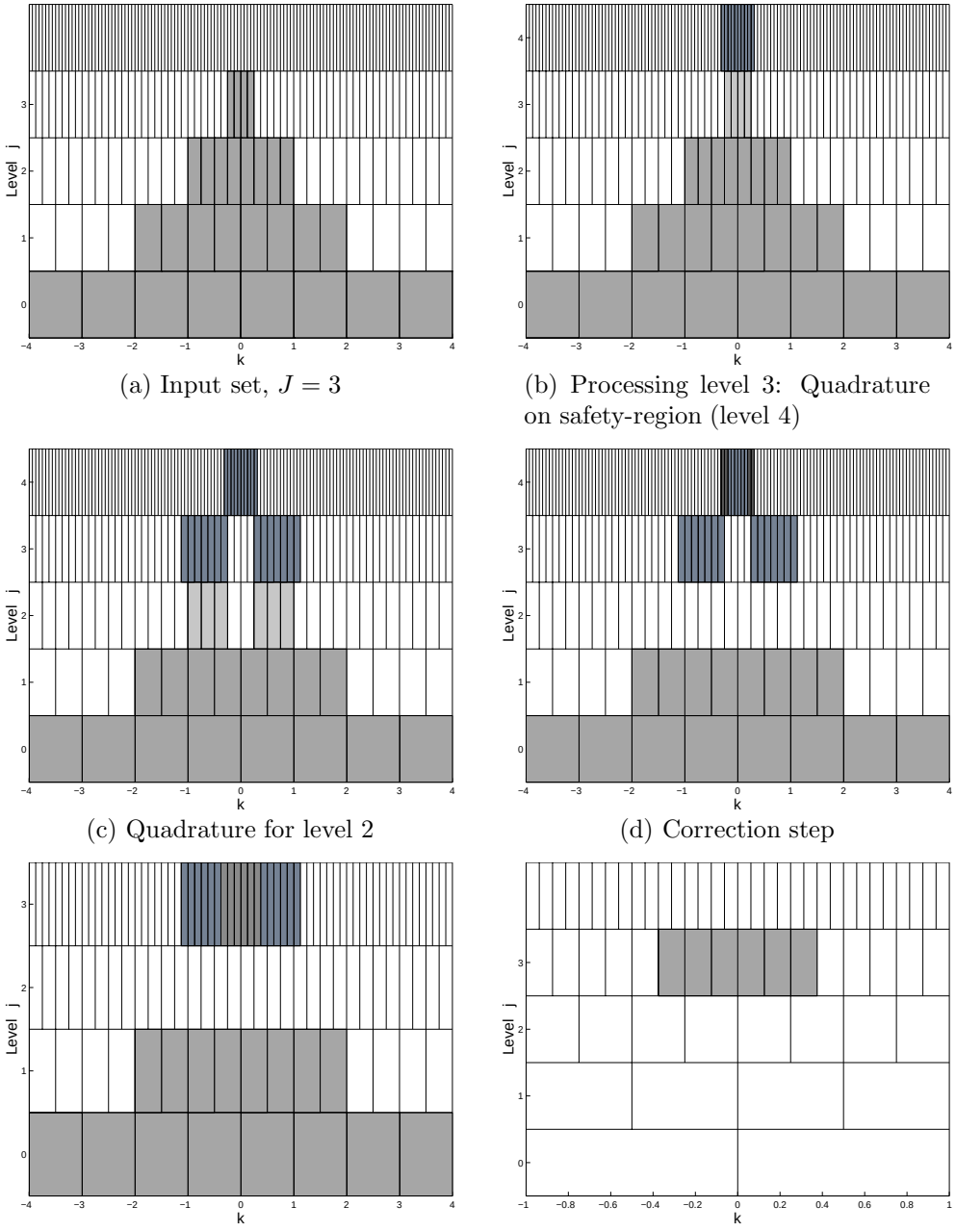
*Moreover, one has*

$$\#\mathcal{G}^- \lesssim \#\partial\mathcal{T} \lesssim \#\mathcal{T}. \tag{3.2.22}$$

*Hence, whenever the quadrature requires at most a constant cost per entry, the number of flops needed in* RECOVER *remains uniformly proportional to $\#\mathcal{T}$.*

Thus from the computational complexity point of view the above scheme satisfies all the requirements put forward in the adaptive solution context described earlier in Chapter 1.

Figure 3.1 illustrates the first steps of the recovery scheme according to the above algorithm on a very simple one dimensional example.




(a) Input set, $J = 3$

(b) Processing level 3: Quadrature on safety-region (level 4)

(c) Quadrature for level 2

(d) Correction step

(e) Decomposition: Scaling functions

(f) Decomposition: Wavelet output (only support cubes plotted)

Figure 3.1: First steps of the recovery scheme

Subplot 3.1 (a) shows the 2-graded input set of support cubes. Starting on the top level $J = 3$, we compute the (approximated) scaling-coefficients for $\mathcal{G}_4$ marked in medium gray in Subplot 3.1 (b). Proceeding with level 2, we also determine the corresponding approximated scaling function coefficients (see 3.1 (c)) in order to use them to perform a first correction step, as illustrated in 3.1 (d). We are then ready to decompose the highest level in order to obtain the desired wavelet coefficients (the associated support cubes are plotted in 3.1 (f)). The scaling function portion will merge with the coefficients on level 3. We would continue by computing the quasi-interpolants associated to level 1, perform a correction step for the coefficients on level 2 and decompose the result.

Recall that in the context of TASK R2 the objective of RECOVER is to produce an array $\mathbf{w}$, that satisfies $\|\mathbf{g}(\mathcal{T}) - \mathbf{w}\|_{\ell_2(\mathcal{J})} \leq C\varepsilon$ whenever $\|\mathbf{g} - \mathbf{g}(\mathcal{T})\|_{\ell_2(\mathcal{J})} < \varepsilon$. However, when $\mathbf{g} = \mathbf{F}(\mathbf{v})$ is a composition the prediction set $\mathcal{T}$ depends on a-posteriori information about $\mathbf{v}$ and priori knowledge of $\mathbf{F}$. The latter typically refers to a whole class of nonlinearities and may lead in concrete cases to rather pessimistic predictions rendering $\mathcal{T}$ unnecessarily large. The top-to-bottom structure of RECOVER easily accommodates individual adaptive adjustments on the fly, see also Section 4.3. In fact, let $J$ be again the highest level appearing in $\mathcal{T}$ and chose for a given overall target accuracy $\varepsilon$ any sequence $\varepsilon_j$ such that $\sum_{j=j_0}^{J} \varepsilon_j = \varepsilon$. As soon as the array $\mathbf{d}^{\mathrm{R}}(\mathcal{T}_j)$ is computed in (3.2.20) we can replace it by an array $\mathbf{d}_j^{\mathrm{R}}$ with possibly small support in $\mathcal{T}_j$ such that $\|\mathbf{d}^{\mathrm{R}}(\mathcal{T}_j) - \mathbf{d}_j^{\mathrm{R}}\|_{\ell_2(\mathcal{T}_j)} \leq \varepsilon_j$. This is a typical coarsening step as described in [CDD03a] based on quasi-sorting, cf. Section 1.2, and thresholding coefficients with smallest modulus. In this way the complete tree $\mathcal{T}$ may never have to be completely assembled which is a crucial distinction from earlier versions in [DSX00].

## 3.3 Error Estimates

The scheme RECOVER introduced above produces for a tree $\mathcal{T}$ an array $\mathbf{d}^{\mathrm{R}}(\mathcal{T})$ supported on $\mathcal{T}$ to approximate the exact sequence $\mathbf{d}$ of wavelet coefficients of $g$ with respect to $\tilde{\Theta}$. Likewise the scaled array $\mathbf{g}^{\mathrm{R}}(\mathcal{T}) = \mathbf{D}_{\mathcal{T}}\mathbf{d}^{\mathrm{R}}(\mathcal{T})$ should approximate the sequence $\mathbf{g}$ of wavelet coefficients of $g$ with respect to $\tilde{\Psi}$.

Recall from (3.1.5) that in either case the total error of such an approximation has two sources. First, there is the *truncation error* $\|\mathbf{d} - \mathbf{d}(\mathcal{T})\|_{\ell_2(\mathcal{J})}$, respectively $\|\mathbf{g} - \mathbf{g}(\mathcal{T})\|_{\ell_2(\mathcal{J})}$, which depends on the prediction set $\mathcal{T}$. Second, there is the error incurred by quadrature in the scheme RECOVER when

recovering the truncated array $\mathbf{d}(\mathcal{T})$.

### 3.3.1 Exact Quadrature

In a number of relevant cases the goal in TASK R2, namely that the quadrature error is dominated by (a bound for) the truncation error, can be achieved trivially because, in principle, the quadrature can be made exact. To explain this, recall the example in (3.1.6) with $F(v) = -\mathrm{div}\, a\nabla v + G(v)$ and suppose first that $G \equiv 0$, i.e. the problem is linear. In this case one has to approximate the entries $c_{j,\mathbf{k}} = \langle \phi_{j,\mathbf{k}}, g\rangle$ with $g = F(v)$. In the context of adaptive schemes $v$ is a finite wavelet expansion $v = \sum_{\lambda\in\mathcal{T}_v} v_\lambda \psi_\lambda$, with $\mathcal{T}_v$ the tree of significant coefficients of $v$. Just as done in (3.2.17) we can rewrite $v$ in local scaling function representation $v = \sum_{(j,\mathbf{k})\in\mathcal{G}_v^-} \bar{c}_{j,\mathbf{k}}\phi_{j,\mathbf{k}}$. Assuming that $\mathcal{T}_v$ is well-graded, which means that the transition between scales is sufficiently gradual (see (3.3.7) below for a precise definition), only a uniformly finite number of scaling functions in this expansion overlap a given point. Thus

$$\begin{aligned} c_{j,\mathbf{k}} &= \langle \phi_{j,\mathbf{k}}, F(v)\rangle = \sum_{(j,\mathbf{k}')\in\mathcal{G}_v^-} \bar{c}_{j,\mathbf{k}'} \langle \phi_{j,\mathbf{k}}, -\mathrm{div}(a\nabla\phi_{j,\mathbf{k}'})\rangle \\ &= \sum_{\substack{(j,\mathbf{k}')\in\mathcal{G}_v^-,\\ \mathrm{supp}\,\phi_{j,\mathbf{k}'}\cap\mathrm{supp}\,\phi_{j,\mathbf{k}}\neq\emptyset}} \bar{c}_{j,\mathbf{k}'} \langle \nabla\phi_{j,\mathbf{k}}, a\nabla\phi_{j,\mathbf{k}'}\rangle. \end{aligned}$$

By the previous remarks, the sets $\{(j,\mathbf{k}') : \mathrm{supp}\,\phi_{j,\mathbf{k}'} \cap \mathrm{supp}\,\phi_{j,\mathbf{k}} \neq \emptyset\}$ have uniformly bounded cardinality. Thus, when using piecewise polynomial scaling functions and whenever the diffusion matrix $a$ is piecewise polynomial, the quantities $\langle \nabla\phi_{j,\mathbf{k}}, a\nabla\phi_{j,\mathbf{k}'}\rangle$ can be computed exactly, so that in this case $q_{j,\mathbf{k}} = c_{j,\mathbf{k}}, (j,\mathbf{k}) \in \mathcal{G}^-$, and no quadrature error occurs. By the above reasoning, the computational cost stays proportional to $\#\mathcal{G}^-$ so that Remark 3.2.3 applies.

This extends to simple nonlinearities. For instance, in the case (3.1.9) we have $g = G(v) = v^3$ so that

$$c_{j,\mathbf{k}} = \sum_{\substack{(j_i\mathbf{k}_i)\in\mathcal{G}_v^-, i=1,2,3,\\ \mathrm{supp}\,\phi_{(j_i,\mathbf{k}_i)}\cap\mathrm{supp}\,\phi_{j,k}\neq\emptyset}} \left\langle \phi_{j,\mathbf{k}}, \prod_{i=1}^{3} \phi_{j_i,\mathbf{k}_i} \right\rangle.$$

Again for each $(j,\mathbf{k})$ there is a uniformly bounded number of integrals involving still only piecewise polynomials that can be calculated exactly.

Thus in such cases one can, in principle, make the scheme RECOVER exact so that, in view of the relation $\|\mathbf{d}^{\mathrm{R}}(\mathcal{T}) - \mathbf{d}(\mathcal{T})\|_{\ell_2(\mathcal{J})} \lesssim \|\mathbf{d} - \mathbf{d}(\mathcal{T})\|_{\ell_2(\mathcal{J})}$, and likewise $\|\mathbf{D}_{\mathcal{T}}(\mathbf{d}^{\mathrm{R}}(\mathcal{T}) - \mathbf{d}(\mathcal{T}))\|_{\ell_2(\mathcal{J})} \lesssim \|\mathbf{g} - \mathbf{g}(\mathcal{T})\|_{\ell_2(\mathcal{J})}$, TASK R2 is trivially fulfilled.

In general, however, one may have to accept a quadrature error or prefers to accept a quadrature error, because even in the above cases the necessary exactness order of quadrature may be fixed but rather high. Therefore we need to estimate this error part and devise ways of decreasing this error if necessary. The subsequent sections are devoted to both issues. We present ways of estimating this error, out of necessity in an essentially different way from [DSX00], and derive strategies for controlling it relative to the truncation or prediction error.

### 3.3.2 The $L_2$-Case

We shall address first the case $\mathcal{H} = L_2(\Omega)$ which corresponds to the first scenario indicated in Section 3.1. Let for $(j, \mathbf{k}) \in \mathcal{G}^-$

$$q_{j,\mathbf{k}} - c_{j,\mathbf{k}} = e_{j,\mathbf{k}} \tag{3.3.1}$$

denote the deviation of the computed value $q_{j,\mathbf{k}}$ from the true one $c_{j,\mathbf{k}} = \langle \phi_{j,\mathbf{k}}, g\rangle$. The corresponding arrays $\mathbf{e}_j = (e_{j,\mathbf{k}})_{(j,\mathbf{k})\in\mathcal{G}_j^-}$ are always supported on $\mathcal{G}_j^-$ without further mentioning. Recall that $\mathcal{G}_{J+1}^- = \mathcal{G}_{J+1}$.
Since (3.2.17) involves only coefficients that will be approximated by quadrature we can think of RECOVER to produce the approximation

$$P_{\mathcal{T}}^{\mathrm{R}} g = \sum_{j=j_0}^{J+1} \tilde{\Phi}_j^T \Big( \mathbf{q}_j - \mathbf{M}_{j-1,\tilde{\Phi}} \mathbf{q}_{j-1}(\mathcal{G}_{j-1}^-) \Big)(\mathcal{G}_j^-) \tag{3.3.2}$$

to $P_{\mathcal{T}} g$, which, by (3.3.1), immediately yields the error representation

$$E_{\mathcal{T}} := P_{\mathcal{T}}^{\mathrm{R}} g - P_{\mathcal{T}} g = \sum_{j=j_0}^{J+1} \tilde{\Phi}_j^T \Big( \mathbf{e}_j - \mathbf{M}_{j-1,\tilde{\Phi}} \mathbf{e}_{j-1}(\mathcal{G}_{j-1}^-) \Big)(\mathcal{G}_j^-). \tag{3.3.3}$$

Of course, the approximate coefficients in $\mathbf{d}^{\mathrm{R}}(\mathcal{T})$ produced by RECOVER arise by decomposing $P_{\mathcal{T}}^{\mathrm{R}} g$ which generates as byproducts the auxiliary arrays $\check{\mathbf{c}}_j, \bar{\mathbf{c}}_j$, cf. (3.2.19), (3.2.20). It will be therefore important to understand how the quadrature errors related to $\mathcal{G}^-$ effects those arrays and

thereby $\mathbf{d}^{\mathrm{R}}(\mathcal{T})$. Inserting (2.1.28) into (3.3.3), proceeding from top to bottom, provides

$$E_{\mathcal{T}} = \tilde{\Phi}_{j_0}^T \mathbf{E}_{j_0-1} + \sum_{j=j_0}^{J} \tilde{\Theta}_j^T \Big( \mathbf{G}_{j,\tilde{\Theta}} \mathbf{E}_{j+1} \Big)(\mathcal{T}_j), \tag{3.3.4}$$

where for $j = J, J-1, \ldots, j_0$

$$\begin{aligned} \mathbf{E}_{j+1} \;=\; & \Big[ \mathbf{G}_{j+1,\tilde{\Phi}}(\ldots \mathbf{G}_{J-2,\tilde{\Phi}}([\mathbf{G}_{J-1,\tilde{\Phi}}([\mathbf{G}_{J,\tilde{\Phi}} \mathbf{R}_{J+1}](\mathcal{T}_J^\circ) + \mathbf{R}_J)](\mathcal{T}_{J-1}^\circ) \\ & + \mathbf{R}_{J-1})(\mathcal{T}_{J-2}^\circ) + \ldots + \mathbf{R}_{j+1}) \Big](\mathcal{G}_{j+1}), \end{aligned}$$

and

$$\mathbf{R}_j := \mathbf{R}_j(\mathcal{G}_j^-) := \Big( \mathbf{e}_j - \mathbf{M}_{j-1,\tilde{\Phi}} \mathbf{e}_{j-1}(\mathcal{G}_{j-1}^-) \Big)(\mathcal{G}_j^-). \tag{3.3.5}$$

Thus the error on level $j+1$ prior to decomposing is obtained by successively decomposing error contributions from higher levels. Here and in what follows we always assume that the multiscale decompositions are applied exactly. Alternatively, one can describe $\mathbf{E}_j$ recursively as follows

$$\mathbf{E}_j = \Big[ \mathbf{G}_{j,\tilde{\Phi}}(\mathbf{E}_{j+1}) \Big](\mathcal{T}_j^\circ) + \mathbf{R}_j. \tag{3.3.6}$$

Thus the error on level $j$ consists of a part living inside the tree obtained from decomposing higher level contributions plus a remainder situated on $\mathcal{G}_j^-$ that is newly introduced on that level through quadrature.

The overall $\ell_2(\mathcal{J})$-error in $\mathbf{d}^{\mathrm{R}}(\mathcal{T})$ caused by the quadrature in RECOVER can be easily estimated by exploiting the norm equivalence (2.1.11). To simplify matters technically we shall specify next somewhat our requirements on the tree $\mathcal{T}$. The theoretical foundation of good predictions for significant trees $\mathcal{T}$ developed in [CDD03d] required the trees to satisfy the so called *expansion property* which, roughly speaking means that the leaves of subsequent levels do not overlap too much, which is familiar in this context, see also [DSX00]. Here, it will be convenient to quantify this as follows. The tree $\mathcal{T}$ is called *well-graded* if one has

$$\Big( \bigcup_{\lambda : \lambda^\circ \in \mathcal{G}_j^-} \mathbf{M}_{j,\tilde{\Phi}}{}^{|\lambda} \Big) \cap \mathcal{T}_{j+1}^\circ = \emptyset. \tag{3.3.7}$$

In other words, as soon as one leaves the tree on level $j$ those scaling functions will not overlap the safe area on level $j+1$. It has been shown

in [CDD03d] that the expansion property does not affect the *asymptotic* growth of the $\varepsilon$-significant tree $\mathcal{T} = \mathcal{T}_\varepsilon$ when $\varepsilon \to 0$. Since condition (3.3.7) can be realized by at most a finite uniformly bounded number of subdivisions of the leaves of a tree with expansion property, the same is true for well-graded trees. We shall always assume this property in what follows, see Section 3.4.2 for an easy to check criterion ensuring well-gradedness. The main result of this section can now be stated as follows.

**Theorem 3.3.1.** *There exists a constant $C$ depending only on the wavelet bases $\Theta, \tilde{\Theta}$ and the well-gradedness parameters so that the computed array $\mathbf{d}^R(\mathcal{T})$ satisfies*

$$\|\mathbf{c}_{j_0} - \mathbf{d}^R_{j_0-1}\|^2_{\ell_2(\mathcal{J})} + \|\mathbf{d}(\mathcal{T}) - \mathbf{d}^R(\mathcal{T})\|^2_{\ell_2(\mathcal{J})} \le C \sum_{j=j_0+1}^{J+1} \|\mathbf{e}_j(\mathcal{G}_j^-)\|^2_{\ell_2(\mathcal{J})}. \quad (3.3.8)$$

**Proof:** Recall that $\partial\mathcal{T}^-$ denotes the set of *outer leaves* of the tree $\mathcal{T}^\circ$, i.e. $\lambda^\circ \in \partial\mathcal{T}^-$ means that $\lambda^\circ$ does not belong to $\mathcal{T}^\circ$, yet its parent does. As the support cubes associated with $\partial\mathcal{T}^-$ form a partition of $\Omega$, by (2.1.11) we have with $\mathbf{c}^{\mathrm{R}}_{j_0} = \mathbf{d}^{\mathrm{R}}_{j_0-1}$

$$\begin{aligned} \|\mathbf{c}_{j_0} - \mathbf{c}^{\mathrm{R}}_{j_0}\|^2_{\ell_2(\mathcal{J})} + \|\mathbf{d}(\mathcal{T}) - \mathbf{d}^{\mathrm{R}}(\mathcal{T})\|^2_{\ell_2(\mathcal{J})} &\le c_\Theta^{-1} \|E_\mathcal{T}\|^2_{L_2(\Omega)} \\ &= c_\Theta^{-1} \sum_{\lambda^\circ \in \partial\mathcal{T}^-} \|E_\mathcal{T}\|^2_{L_2(\square_{\lambda^\circ})}. \end{aligned}$$

The well-gradedness of $\mathcal{T}$ implies that for every $\lambda^\circ \in \partial\mathcal{T}_j^-$ one has that $(\tilde{\Phi}_l^T \mathbf{R}_l)|_{\square_{\lambda^\circ}} \not\equiv 0$ holds only for a uniformly bounded number $K$ of levels $|l - j| \le K$. Since the two scale matrices define uniformly bounded mappings on $\ell_2(\mathcal{J})$, the assertion follows. ∎

Note that the main difference from earlier results in [DSX00] lies in the fact that the estimate involves no assumptions e.g. concerning the equilibration of local polynomial approximation errors of $g$, or in the case $g = G \circ v$, of $v$. However, the results in [DSX00] can be recovered under such assumptions. In fact, suppose the $q_{j,k}$ have the form $q_{j,k} = \langle Q_{j,k}(g), \phi_{j,k} \rangle$ where $Q_{j,k}(g)$ is a polynomial approximation of $g$ on the support $S_{j,k}$ of $\phi_{j,k}$ of some order $m$, say. Suppose that e.g. a discrete least squares fit to $g$ on a fixed sufficiently dense sample set in $S_{j,k}$ even assures (for the give $g$) that $\|g - Q_{j,k}(g)\|_{L_2(S_{j,k})} \le C \inf_{P \in \mathbb{P}_m} \|g - P\|_{L_2(S_{j,k})} =: \eta_{j,k}(g)$. Moreover suppose that the tree $\mathcal{T}$ is *essentially balanced* with respect to the $\eta_{j,k}$. This means that for some $\delta > 0$ and all $(j,k) \in \partial\mathcal{T}^-$ we have $\eta_{i,k}(g) \le \delta$ while

for a subset $\Lambda$ of $\partial\mathcal{T}$ with $\#\Lambda \geq c\#\partial\mathcal{T}$ one has $\eta_{j,k}(g) \geq \delta$, $(j,k) \in \Lambda$. Then, whenever $g \in B^s_q(L_p)$ with $1/p < s/d + 1/2$ for some $s \leq m$ one has (see [DSX00])

$$\left( \sum_{(j,k)\in\partial\mathcal{T}^-} \eta^2_{j,k} \right)^{1/2} \leq C(\#\mathcal{T})^{-s/d} |g|_{B^s_q(L_p)}, \tag{3.3.9}$$

see Section 1 and, e.g., [DeV98] for the definition of the Besov spaces $B^s_q(L_p)$ and their semi-norms. One immediately derives from (3.3.8) and (3.3.9)

$$\left( \|\mathbf{c}_{j_0} - \mathbf{c}^{\mathrm{R}}_{j_0}\|^2_{\ell_2(\mathcal{J})} + \|\mathbf{d}(\mathcal{T}) - \mathbf{d}^{\mathrm{R}}(\mathcal{T})\|^2_{\ell_2(\mathcal{J})} \right)^{1/2} \leq C(\#\mathcal{T})^{-s/d} |g|_{B^s_q(L_p)}. \tag{3.3.10}$$

The interest in estimates of this type lies in the following points. The right hand side of (3.3.10) involves measuring smoothness of $g$ in $L_p$ which for the above admissible range of $p$ imposes much weaker conditions than measuring smoothness in $L_2$. Thus a proper choice of a tree can make up for the lack of smoothness so as to retain an optimal order relating accuracy to the number of degrees of freedom even in the presence of singularities. Moreover, under the above assumptions the truncation error can be shown to be also bounded by the right hand side of (3.3.10) (although it could actually happen to be smaller), so that at least for the *class* of functions $g \in B^s_q(L_p)$ one obtains optimal error estimates.

However, this line of argument works under certain assumptions on the distribution of local errors and does not quite meet the demands arising in the context of adaptive solution schemes, see Section 3.1 and the notion of $s^*$-sparsity in [CDD03a], see Section 1.1. Recall from Task R2 that ideally one would like to relate the errors created by Recover directly to the truncation error $\|g - P_{\mathcal{T}} g\|_{L_2(\Omega)}$ or equivalently to $\|\mathbf{d} - \mathbf{d}(\mathcal{T})\|_{\ell_2(\mathcal{J})}$ (or bounds for those terms). Note that one has

$$\|\mathbf{d} - \mathbf{d}(\mathcal{T})\|^2_{\ell_2(\mathcal{J})} = \sum_{\lambda^\circ\in\partial\mathcal{T}} \sum_{\mu:\mu^\circ\succ\lambda^\circ} |d_\mu|^2. \tag{3.3.11}$$

Recall that $\mu^\circ \succ \lambda^\circ$ means that $\mu^\circ$ is a descendant of (not equal to) $\lambda^\circ$. Clearly, the "local remainders"

$$r_{\lambda^\circ} = r_{\lambda^\circ}(\mathbf{d}, \mathcal{T}) := \left( \sum_{\mu:\mu^\circ\succ\lambda^\circ} |d_\mu|^2 \right)^{1/2}, \quad \lambda^\circ \in \partial\mathcal{T}, \tag{3.3.12}$$

are just local $L_2$-errors of the approximation $P_{\mathcal{T}} g$ to $g$. Thus it would be ideal if the quadrature errors $e_{\lambda^\circ}$ (see (3.3.1)) could be estimated in terms

of $r_{\lambda^\circ}$, or at most by a finite number of nearby local remainders, which would entail

$$\|\mathbf{d} - \mathbf{d}^{\mathrm{R}}(\mathcal{T})\|_{\ell_2(\mathcal{J})} \lesssim \|\mathbf{d} - \mathbf{d}(\mathcal{T})\|_{\ell_2(\mathcal{J})}. \tag{3.3.13}$$

We shall return to this issue in the more general case discussed below.

### 3.3.3 Dual Norms

As indicated before, when dealing with adaptive schemes for operator equations such as PDE's the relevant function space is most often not $L_2$ but some other Hilbert space $\mathcal{H}'$, typically a Sobolev space of positive order, see Section 3.1 and [CDD03a]. What will matter here is that the primal norms can be localized, i.e. restrictions of $f \in \mathcal{H}$ to sub-domains $\Omega' \subset \Omega$ (of suitable regularity) belong to a localized version $\mathcal{H}(\Omega')$ and that one has for any partition $\mathcal{P}$ of $\Omega$ into cells $\Delta$

$$\sum_{\Delta\in\mathcal{P}} \|f\|^2_{\mathcal{H}(\Delta)} \lesssim \|f\|^2_{\mathcal{H}}, \quad f \in \mathcal{H}. \tag{3.3.14}$$

This is, for instance, the case for $\mathcal{H} = H^t$ with $t \geq 0$. Under the assumption (3.2.1) that a Riesz basis $\Psi$ is obtained from an $L_2$-Riesz basis $\Theta$ through scaling we can still apply the above scheme RECOVER to obtain

$$\mathbf{g}^{\mathrm{R}}(\mathcal{T}) := \mathbf{D}_{\mathcal{T}}\mathbf{d}^{\mathrm{R}}(\mathcal{T}) \tag{3.3.15}$$

as an approximation to the desired array $\mathbf{g}(\mathcal{T}) = (\langle\psi_\lambda, g\rangle)_{\lambda\in\mathcal{T}}$. Of course, this scaling should be incorporated in the loop of RECOVER, namely 3.2.20 will be replaced by $\mathbf{d}^{\mathrm{R}}_j := \mathbf{D}_{\mathcal{T}_j}\mathbf{G}_{j,\tilde{\Theta}}\bar{\mathbf{c}}_{j+1}$. Again our objective is to estimate $\|\mathbf{g}(\mathcal{T}) - \mathbf{g}^{\mathrm{R}}(\mathcal{T})\|_{\ell_2(\mathcal{J})}$ and ultimately $\|\mathbf{g} - \mathbf{g}^{\mathrm{R}}(\mathcal{T})\|_{\ell_2(\mathcal{J})}$.

#### Main obstruction and inherent problems

Estimating $\|\mathbf{g}(\mathcal{T}) - \mathbf{g}^{\mathrm{R}}(\mathcal{T})\|_{\ell_2(\mathcal{J})}$ poses a certain principal difficulty which is perhaps worth pointing out first. The above estimate in Theorem 3.3.1 provides a bound in terms of local $L_2$-errors resulting from approximating the function $g$ in $L_2$. The relevant function space here, however, is $\mathcal{H}'$. So in order to estimate the accuracy of the scaled array in (3.3.15) one would have to relate it to the error in $\mathcal{H}'$. This is not straightforward as Sobolev norms of negative order do not localize in a simple way.

At the first glance an easy remedy is offered by the norm equivalences (2.1.11). In fact, we recall that one can deduce from (2.1.11) and (2.1.2) that for some constants $\hat{c}, \hat{C}$ one has

$$\begin{aligned} \hat{c}\|\sum_{\lambda\in\mathcal{J}}\langle\psi_\lambda, w\rangle\tilde{\psi}_\lambda\|_{\mathcal{H}'} &\leq \|\sum_{\lambda\in\mathcal{J}}\langle\psi_\lambda, w\rangle\tilde{\theta}_\lambda\|_{L_2} \\ &= \|\sum_{\lambda\in\mathcal{J}}\omega_\lambda\langle\theta_\lambda, w\rangle\tilde{\theta}_\lambda\|_{L_2} \leq \hat{C}\|\sum_{\lambda\in\mathcal{J}}\langle\psi_\lambda, w\rangle\tilde{\psi}_\lambda\|_{\mathcal{H}'}. \end{aligned}$$

This tells us that approximation of some expansion $w = \sum_{\lambda\in\mathcal{J}}\langle\psi_\lambda, w\rangle\tilde{\psi}_\lambda$ in $\mathcal{H}'$ is equivalent to approximating the scaled expansion

$$\Sigma(w) := \sum_{\lambda\in\mathcal{J}}\omega_\lambda\langle\theta_\lambda, w\rangle\tilde{\theta}_\lambda$$

in $L_2$, which by (2.1.11) yields an estimate for the approximations to $\langle\psi_\lambda, w\rangle$ in terms of an $L_2$ error for the approximant. For simplicity we shall assume that the scaling weights $\omega_\lambda$ depend only on the scale $|\lambda|$, $\omega_\lambda = \omega_{|\lambda|}$. Accordingly, it is suggested in [XZ03] to approximate

$$\Sigma(g)_{\mathcal{T}} := \sum_{\lambda\in\mathcal{T}}\omega_\lambda\langle\theta_\lambda, g\rangle\tilde{\theta}_\lambda = \omega_{j_0}P_{j_0}g + \sum_{j=j_0}^{J}\omega_j(P_{j+1} - P_j)g$$

in $L_2$ and then use the above error estimation. However, that would require approximating the inner products $\langle\Sigma(g), \phi_{j,k}\rangle$. This, in turn, would require knowing the coefficients $\langle\Sigma(g), \phi_{j',k'}\rangle$ for all $(j', k')$ such that the support of $\tilde{\phi}_{j',k'}$ intersects that of $\phi_{j,k}$, as the level dependent scaling factor $\omega_j$ prevents cancellation inside the tree, cf. (3.3.16). This difficulty could be seen as a 'lacking locality' which, due to the negative norm, is inherent to the problem. Thus the approximate scaling function coefficients $q_{j,k}$ cannot be computed by just sampling $g$ locally on the support of $\phi_{j,\mathbf{k}}$ and a related approach (see [XZ03]) therefore does *not* seem to be computationally feasible.

In order to get around this difficulty and use only computationally feasible approximations to $\langle g, \phi_{j,k}\rangle$ for indices in $\mathcal{G}^-$, one could attempt to employ the same decomposition strategy as before trying to "hollow" the tree. To

this end, note first that one has

$$\begin{aligned} g_{\mathcal{T}} &= \omega_{j_0} P_{j_0} g + \sum_{j=j_0}^{J} \omega_j (P_{j+1} - P_j) g_{\mathcal{T}} \\ &= \sum_{j=j_0+1}^{J+1} \tilde{\Phi}_j^T \Big( [\omega_{j-1} - \omega_j] \mathbf{c}_j(\mathcal{T}_j^\circ) + \omega_{j-1} (\mathbf{c}_j(\mathcal{I}_j \setminus \mathcal{T}_j^\circ) - \\ &\qquad \mathbf{M}_{j-1,\tilde{\Phi}} \mathbf{c}_{j-1}(\mathcal{I}_{j-1} \setminus \mathcal{T}_{j-1}^\circ) \Big) \\ &= \sum_{j=j_0+1}^{J+1} \tilde{\Phi}_j^T \Big( [\omega_{j-1} - \omega_j] \mathbf{c}_j(\mathcal{T}_j^\circ) + \omega_{j-1} \Big( \mathbf{c}_j - \mathbf{M}_{j-1,\tilde{\Phi}} \mathbf{c}_{j-1}(\mathcal{G}_{j-1}^-) \Big) (\mathcal{G}_j^-) \Big), \end{aligned} \tag{3.3.16}$$

where we have used (3.2.12), see Section 3.2.1. In order to turn this into an approximation one needs approximations to the arrays $\mathbf{c}_j(\mathcal{T}_j^\circ)$ and $\mathbf{c}_j(\mathcal{G}_j^-)$. These approximations are already produced by the scheme RECOVER. Thus, we are led to define

$$\begin{aligned} P_{\mathcal{T}} g &:= \sum_{j=j_0+1}^{J+1} \tilde{\Phi}_j^T \Big( [\omega_{j-1} - \omega_j] \check{\mathbf{c}}_j(\mathcal{T}_j^\circ) + \omega_{j-1} \Big( \mathbf{q}_j - \mathbf{M}_{j-1,\tilde{\Phi}} \mathbf{q}_{j-1}(\mathcal{G}_{j-1}^-) \Big) (\mathcal{G}_j^-) \Big) \\ &= \sum_{j=j_0+1}^{J+1} \tilde{\Phi}_j^T \Big( \omega_{j-1} \bar{\mathbf{c}}_j(\mathcal{G}_j) - \omega_j \check{\mathbf{c}}_j(\mathcal{T}_j^\circ) \Big). \end{aligned}$$

The main difference from the previous situation, reflected by (3.3.2) and (3.3.3), lies now in the additional term $[\omega_{j-1} - \omega_j] \check{\mathbf{c}}_j(\mathcal{T}_j^\circ)$ due to the new scaling in front of the telescoping summands. This means that the error terms will no longer live only on $\mathcal{G}^-$ but that multiscale decompositions of higher level error components are transported into the interior of $\mathcal{T}$.

It does not seem to be clear how this will effect the overall error. These observations seem to indicate that error estimation for $L_2$ does not simply carry over to more general norms. Thus, in summary, the computational complexity of the approach proposed in [XZ03] is therefore not clear to us.

### A general estimate

On account of the remarks brought up in the previous section, we prefer to stay here with the original order of operation, namely to apply RECOVER to $g$ and then scale the resulting coefficients, see (3.3.15), as opposed to scaling $g$ first into $\Sigma(g)$ and then applying RECOVER. It is now

less obvious though how to estimate the resulting array in a proper way and, based on such estimates, how to realize TASK R2. One probably cannot expect a complete analysis that works in the generality considered so far. Therefore the objective of this section is to further explore some basic aspects of TASK R2 and to derive from these considerations suitable algorithmic ingredients. This is to identify more specific requirements on the quadrature that may lead to the desired asymptotically optimal computational performance. In particular, it will be seen that, in a strict sense, this requires generally more than just accuracy bounds for the individual coefficients $q_{j,\mathbf{k}}$.

For technical simplicity we shall continue assuming in what follows that the scaling weights in (3.2.1) depend only on the scales, i.e. $\omega_\lambda = \omega_{|\lambda|}$. The following reasoning, however, can be extended to more general situations.

One easily concludes from (2.1.2) that then for (well behaved sub-domains $\Omega' \subset \Omega$)

$$\inf_{v_j \in S_j} \|v - v_j\|_{L_2(\Omega')} \lesssim \omega_j \|v\|_{\mathcal{H}(\Omega')}, \quad v \in \mathcal{H}. \tag{3.3.17}$$

Our goal is again to estimate the scaled arrays $\|\mathbf{g}(\mathcal{T}) - \mathbf{g}^{\mathrm{R}}(\mathcal{T})\|_{\ell_2(\mathcal{J})}$ in terms of the quadrature errors $e_{\lambda^\circ}$, $\lambda^\circ \in \mathcal{G}^-$. One might think that the ideal estimate would be

$$\|\mathbf{g}(\mathcal{T}) - \mathbf{g}^{\mathrm{R}}(\mathcal{T})\|^2_{\ell_2(\mathcal{J})} \le C \sum_{j=j_0+1}^{J+1} \omega_j^2 \|\mathbf{e}_j(\mathcal{G}_j^-)\|^2_{\ell_2(\mathcal{J})}, \tag{3.3.18}$$

i.e., the error contributions scale like wavelet coefficients. This would indeed be the case if the error components were scaling function coefficients of some underlying error function, i.e., $e_{\lambda^\circ} = \langle \phi_{\lambda^\circ}, E_{\mathcal{T}} \rangle$ which would entail the relations

$$\mathbf{G}_{j,\tilde{\Phi}} \mathbf{e}_{j+1} = \mathbf{e}_j. \tag{3.3.19}$$

This, in turn, is equivalent to

$$\mathbf{G}_{j,\tilde{\Phi}} \mathbf{q}_{j+1} = \mathbf{q}_j, \tag{3.3.20}$$

which, of course, will generally not be true.

So it seems that all we can hope for is an estimate that involves the deviation from (3.3.19) which then hopefully is relatively small. In order to make this more precise, we introduce the following notation. Let

$$\Omega_j := \bigcup_{(j,\mathbf{k}) \in \mathcal{G}_j^-} \operatorname{supp} \tilde{\phi}_{j,\mathbf{k}}, \quad P_j(w|\mathcal{G}_j^-) := \sum_{(j,\mathbf{k}) \in \mathcal{G}_j^-} \langle \phi_{j,\mathbf{k}}, w \rangle \tilde{\phi}_{j,\mathbf{k}}. \tag{3.3.21}$$

We can use (2.1.2) to conclude that

$$\|\mathbf{g}(\mathcal{T}) - \mathbf{g}^{\mathrm{R}}(\mathcal{T})\|_{\ell_2(\mathcal{J})} \leq C_\Psi \|E_{\mathcal{T}}\|_{\mathcal{H}'}, \tag{3.3.22}$$

where $E_{\mathcal{T}}$ is given by (3.3.3). Furthermore let $P_j^*(\cdot|\mathcal{G}_j^-)$ denote the adjoint of $P_j(\cdot|\mathcal{G}_j^-)$ and

$$\|v\|_{(\mathcal{H}(\Omega'))'} := \sup_{w \in \tilde{\mathcal{H}}(\Omega')} \frac{\langle w, v\rangle_{\Omega'}}{\|w\|_{\mathcal{H}(\Omega')}}$$

where $\tilde{\mathcal{H}}(\Omega')$ consists for $\Omega' \subset \Omega$ of those $w \in \mathcal{H}(\Omega')$ whose extension by zero to all of $\Omega$ is still in $\mathcal{H}$. In order to develop suitable quadrature techniques, it will be convenient to employ the mappings

$$L_j g := \sum_{(j,\mathbf{k}) \in \mathcal{G}_j^-} q_{j,\mathbf{k}}(g) \tilde{\phi}_{j,\mathbf{k}}, \tag{3.3.23}$$

which always involve only indices from $\mathcal{G}_j^-$. As before, the coefficients $q_{j,\mathbf{k}}(g)$ forming the arrays $\mathbf{q}_j$ on $\mathcal{G}_j^-$ are to approximate the exact scaling function coefficients $c_{j,\mathbf{k}} = \langle \phi_{j,\mathbf{k}}, g\rangle$ for $(j,\mathbf{k}) \in \mathcal{G}_j^-$. We shall now present some general estimates for the error in the dual norm.

**Proposition 3.3.2.** *Under the above assumptions we have the following estimates of the quadrature error $E_{\mathcal{T}}$ defined in (3.3.3).*

*a) One has*

$$\|E_{\mathcal{T}}\|_{\mathcal{H}'} \lesssim \left( \sum_{j=j_0}^{J+1} \|\tilde{\Phi}_j^T \mathbf{e}_j\|^2_{\mathcal{H}(\Omega_j)'} \right)^{1/2}. \tag{3.3.24}$$

*b) Recalling the definition of $P_j$ (2.1.8) we have for any $L_j$ given by (3.3.23)*

$$\|E_{\mathcal{T}}\|_{\mathcal{H}'} \lesssim \left( \sum_{j=j_0}^{J+1} \|L_j g - P_j g\|^2_{\mathcal{H}(\Omega_j)'} \right)^{1/2}. \tag{3.3.25}$$

**Proof:** As for a), we recall the definition (3.3.3) and obtain, upon using

(2.1.6) and biorthogonality of $\Phi_j, \tilde{\Phi}_j$ several times,

$$\begin{aligned}
\|E_{\mathcal{T}}\|_{\mathcal{H}'} &= \sup_w \frac{\langle E_{\mathcal{T}}, w\rangle}{\|w\|_{\mathcal{H}}} = \sup_w \frac{\sum_{j=j_0+1}^{J+1}\langle \tilde{\Phi}_j^T\Big(\mathbf{e}_j - \mathbf{M}_{j-1,\tilde{\Phi}}\mathbf{e}_{j-1}(\mathcal{G}_{j-1}^-)\Big)(\mathcal{G}_j^-), w\rangle_{\Omega_j}}{\|w\|_{\mathcal{H}}} \\
&= \sup_w \frac{\sum_{j=j_0+1}^{J+1}\langle \tilde{\Phi}_j^T\Big(\mathbf{e}_j - \mathbf{M}_{j-1,\tilde{\Phi}}\mathbf{e}_{j-1}(\mathcal{G}_{j-1}^-)\Big)(\mathcal{G}_j^-), P_j^*(w|\mathcal{G}_j^-)\rangle_{\Omega_j}}{\|w\|_{\mathcal{H}}} \\
&= \sup_w \frac{\sum_{j=j_0+1}^{J+1}\langle \tilde{\Phi}_j^T\Big(\mathbf{e}_j - \mathbf{M}_{j-1,\tilde{\Phi}}\mathbf{e}_{j-1}(\mathcal{G}_{j-1}^-)\Big), P_j^*(w|\mathcal{G}_j^-)\rangle_{\Omega_j}}{\|w\|_{\mathcal{H}}} \\
&= \sup_w \frac{\sum_{j=j_0+1}^{J+1}\langle \tilde{\Phi}_j^T\mathbf{e}_j - \tilde{\Phi}_{j-1}^T\mathbf{e}_{j-1}, P_j^*(w|\mathcal{G}_j^-)\rangle_{\Omega_j}}{\|w\|_{\mathcal{H}}} \\
&\le \sup_w \frac{\sum_{j=j_0+1}^{J+1}\|\tilde{\Phi}_j^T\mathbf{e}_j - \tilde{\Phi}_{j-1}\mathbf{e}_{j-1}\|_{(\mathcal{H}(\Omega_j))'}\|P_j^*(w|\mathcal{G}_j^-)\|_{\mathcal{H}(\Omega_j)}}{\|w\|_{\mathcal{H}}} \\
&\lesssim \sup_w \frac{\sum_{j=j_0+1}^{J+1}\|\tilde{\Phi}_j^T\mathbf{e}_j - \tilde{\Phi}_{j-1}\mathbf{e}_{j-1}\|_{(\mathcal{H}(\Omega_j))'}\|w\|_{\mathcal{H}(\Omega_j)}}{\|w\|_{\mathcal{H}}},
\end{aligned}$$

where we have also used that the $P_j^*(\cdot|\mathcal{G}_j^-)$ are bounded in $\mathcal{H}$. Since $\mathbf{e}_{j-1}$ is supported in $\mathcal{G}_{j-1}^-$, we have that

$$\|\tilde{\Phi}_{j-1}\mathbf{e}_{j-1}\|_{(\mathcal{H}(\Omega_j))'} = \|\tilde{\Phi}_{j-1}\mathbf{e}_{j-1}\|_{(\mathcal{H}(\Omega_j\cap\Omega_{j-1}))'} \le \|\tilde{\Phi}_{j-1}\mathbf{e}_{j-1}\|_{(\mathcal{H}(\Omega_{j-1}))'},$$

since the dual norm is monotone in the domain.

Using this, applying Cauchy-Schwarz, bearing in mind that, by well-gradedness, only a uniformly bounded finite number of $\Omega_j$ overlap at any given point and employing (3.3.14), yields the assertion a).

Concerning b), we have the representation, cf. (3.3.1)

$$e_{j,\mathbf{k}} = q_{j,\mathbf{k}} - c_{j,\mathbf{k}} = \langle \phi_{j,\mathbf{k}}, L_j g - P_j g\rangle, \text{ i.e } \mathbf{e}_j = \langle \Phi_j, L_j g - P_j(g|\mathcal{G}_j^-)\rangle. \quad (3.3.26)$$

Clearly, (3.3.26) says that

$$\tilde{\Phi}_j^T\mathbf{e}_j = P_j(L_j g|\mathcal{G}_j^-) - P_j(g|\mathcal{G}_j^-)) = L_j g - P_j(g|\mathcal{G}_j^-).$$

Hence, using the biorthogonality of $\Phi_j, \tilde{\Phi}_j$, we can rewrite the terms in (3.3.24) as follows.

$$\begin{aligned}
\|\tilde{\Phi}_j^T\mathbf{e}_j\|_{\mathcal{H}(\Omega_j)'} &= \sup_{w\in\mathcal{H}(\Omega_j)} \frac{\langle \tilde{\Phi}_j^T\mathbf{e}_j, P_j^*(w|\mathcal{G}_j^*)\rangle_{\Omega_j}}{\|w\|_{\mathcal{H}(\Omega_j)}} \\
&= \sup_{w\in\mathcal{H}(\Omega_j)} \frac{\langle L_j(g|\mathcal{G}_j^-) - P_j(g|\mathcal{G}_j^-), P_j^*(w|\mathcal{G}_j^*)\rangle_{\Omega_j}}{\|w\|_{\mathcal{H}(\Omega_j)}} \\
&= \sup_{w\in\mathcal{H}(\Omega_j)} \frac{\langle L_j g - P_j g, P_j^*(w|\mathcal{G}_j^*)\rangle_{\Omega_j}}{\|w\|_{\mathcal{H}(\Omega_j)}},
\end{aligned}$$

which confirms the claim b) by the same arguments as used in a). ■

### 3.3.4 A Wishlist for the Quadrature Mappings $L_j$

So far we have not specified how to compute in RECOVER the approximate scaling function coefficients $q_{j,k}$. So the objective of this section is to find relevant requirements on quadrature. We will start by identifying desirable properties of the mappings $L_j$ from (3.3.23).

In what follows we shall make frequent use of the following fact.

**Remark 3.3.3.** *For any sub-domain $\Omega'$ of $\Omega$ one has*

$$\|g\|^2_{(\mathcal{H}(\Omega'))'} \lesssim \sum_{\lambda: S_\lambda \cap \Omega' \neq \emptyset} |\langle \psi_\lambda, g\rangle|^2, \tag{3.3.27}$$

*where $S_\lambda = \operatorname{supp} \tilde{\psi}_\lambda$.*

**Proof:** For $w \in \tilde{\mathcal{H}}(\Omega')$ one has

$$\begin{aligned} |\langle w, g\rangle| &= \left| \left\langle w, \sum_{\lambda \in \mathcal{J}} \langle \psi_\lambda, g\rangle \tilde{\psi}_\lambda \right\rangle \right| = \left| \sum_{\lambda: S_\lambda \cap \Omega' \neq \emptyset} \langle w, \tilde{\psi}_\lambda\rangle \langle \psi_\lambda, g\rangle \right| \\ &\leq \left( \sum_{\lambda: S_\lambda \cap \Omega' \neq \emptyset} |\langle w, \tilde{\psi}_\lambda\rangle|^2 \right)^{1/2} \left( \sum_{\lambda: S_\lambda \cap \Omega' \neq \emptyset} |\langle \psi_\lambda, g\rangle|^2 \right)^{1/2} \\ &\lesssim \|w\|_{\mathcal{H}(\Omega')} \left( \sum_{\lambda: S_\lambda \cap \Omega' \neq \emptyset} |\langle \psi_\lambda, g\rangle|^2 \right)^{1/2}, \end{aligned}$$

whence the assertion follows. ■

Recalling that

$$P_j g = \sum_{(j,\mathbf{k}) \in \mathcal{I}_j} \langle \phi_{j,\mathbf{k}}, g\rangle \tilde{\phi}_{j,\mathbf{k}} = \sum_{(j,\mathbf{k}) \in \mathcal{I}_j} c_{j,\mathbf{k}} \tilde{\phi}_{j,\mathbf{k}}, \tag{3.3.28}$$

the subsequent discussion will be guided by the following simple observations based on Proposition 3.3.2 b). They show what the $L_j$ should ideally satisfy.

**Proposition 3.3.4.** *a) If*

$$\|L_j g - P_j g\|_{(\mathcal{H}(\Omega_j))'} \le C\|g - P_j g\|_{(\mathcal{H}(\Omega_j))'} \tag{3.3.29}$$

*uniformly in $j$, then the approximate array $\mathbf{g}^R(\mathcal{T})$ on the well-graded tree $\mathcal{T}$ produced by* RECOVER *satisfies*

$$\|\mathbf{g}(\mathcal{T}) - \mathbf{g}^R(\mathcal{T})\|_{\ell_2(\mathcal{J})} \le C^*\|\mathbf{g} - \mathbf{g}(\mathcal{T})\|_{\ell_2(\mathcal{J})}, \tag{3.3.30}$$

*where $C^*$ is independent of $\mathcal{T}$ but depends on $C$ from (3.3.29) and the constant in (3.3.25).*
*b) Suppose that $\mathcal{T}$ satisfies*

$$\|\mathbf{g} - \mathbf{g}(\mathcal{T})\|_{\ell_2(\mathcal{J})} \le \varepsilon \tag{3.3.31}$$

*and that $J$ is the highest level appearing in $\mathcal{T}$. If for $j_0 \le j \le J+1$*

$$\|L_j g - P_j g\|_{(\mathcal{H}(\Omega_j))'} \le C\varepsilon\sqrt{\frac{\#\mathcal{G}_j^-}{\#\mathcal{G}^-}} \tag{3.3.32}$$

*uniformly in $j \le J$, one has*

$$\|\mathbf{g} - \mathbf{g}^R(\mathcal{T})\|_{\ell_2(\mathcal{J})} \le (1 + C^*)\varepsilon, \tag{3.3.33}$$

*where $C^*$ depends on the constants in (3.2.22) and (3.3.25), but is independent of $\mathcal{T}$.*

**Proof:** Applying (3.3.27) to $g - P_j g$, (3.3.29) says that for $g_\lambda = \langle \psi_\lambda, g\rangle$

$$\|L_j g - P_j g\|^2_{(\mathcal{H}(\Omega_j))'} \lesssim \sum_{\substack{|\lambda|>j \\ \operatorname{supp}\tilde{\psi}_\lambda \cap \Omega_j \neq \emptyset}} |g_\lambda|^2. \tag{3.3.34}$$

Since by well-gradedness of $\mathcal{T}$ only finitely many of the $\Omega_j$ overlap at a given point and since by (3.2.22) $\#\mathcal{G}^- \lesssim \#\mathcal{T} \lesssim \#\partial\mathcal{T}^-$, we conclude that the right hand side of (3.3.25) can be bounded in the following way

$$\begin{aligned}
\|E_{\mathcal{T}}\|_{\mathcal{H}'} &\lesssim \Big(\sum_{j=j_0}^{J+1} \|L_j g - P_j g\|^2_{\mathcal{H}(\Omega_j)'}\Big)^{1/2} \lesssim \Big(\sum_{j=j_0}^{J+1} \sum_{\substack{|\lambda|>j \\ \operatorname{supp}\tilde{\psi}_\lambda \cap \Omega_j \neq \emptyset}} |g_\lambda|^2\Big)^{1/2} \\
&\lesssim \Big(\sum_{\lambda \notin \mathcal{T}} |g_\lambda|^2\Big)^{1/2} = \|\mathbf{g} - \mathbf{g}(\mathcal{T})\|_{\ell_2(\mathcal{J})},
\end{aligned}$$

which confirms a). The proof of the second claim is analogous. ∎

Obviously, (3.3.29) is the stronger assumption which would yield the best possible result, namely that the quadrature error is dominated by the prediction error. Recall, however, that the scenario given by part b) in Proposition 3.3.4 is sufficient for TASK R2 which warrants optimal complexity estimates in the adaptive context described in Section 3.1. This leads us to the following definition.

**Definition 3.3.5.** *We shall call a quadrature* reliable for*, respectively* $\varepsilon$-reliable for $g$ *when the corresponding mappings* $L_j$ *satisfy (3.3.29), respectively, when (3.3.32) holds whenever (3.3.31) is provided.*

Before we turn to discuss possibilities to *construct* reliable quadrature formulas, we will first point out some details concerning RECOVER for our model wavelet-basis from Section (2.2).

## 3.4 RECOVER for Cardinal B-Spline Wavelets

In the following, we will specify in more detail computational issues concerning the *working sets* $\mathcal{G}_{j+1}, j = j_0, \dots, J$ and the property *well-gradedness* in the case of cardinal B-Spline wavelets [CDF92], see Section 2.2. We again will confine the discussion to the one dimensional case on $\mathbb{R}$, as adaptions to cardinal B-spline based wavelets on intervals or in the periodic case depend on the construction at hand. In any case, a tensor product extention to higher dimensions is straight forward.

### 3.4.1 Determination of $\mathcal{G}_j$

According to (3.2.11), $\mathcal{G}_{j+1}$ is defined as follows,

$$\mathcal{G}_{j+1} = \bigcup_{\lambda \in \mathcal{T}_j^\circ \cup \mathcal{T}_j} \operatorname{supp} \tilde{\mathbf{M}}_j^{|\lambda} \cup \bigcup_{\lambda \in \mathcal{T}_j^\circ \cup \mathcal{T}_j} \operatorname{supp} \tilde{\mathbf{G}}_j^{\overline{\lambda}}.$$

Recall from Section 2, that the entries of the matrices $\tilde{\mathbf{M}}_j, \tilde{\mathbf{G}}_j$ are mainly determined by the *masks* of the generator functions $\mathbf{a}, \tilde{\mathbf{a}}$ and that $\tilde{\mathbf{M}}_j, \tilde{\mathbf{G}}_j$ do depend on the scale $j$ only concerning their size, [DKU99]. We will therefore restrict the following discussion to $j = j_0$.

We are always concerned with compactly supported masks, i.e., there exist numbers $m_1, m_2, \tilde{m}_1, \tilde{m}_2 \in \mathbb{Z}$ such that

$$\operatorname{supp} \mathbf{a} = [m_1, m_2] \cap \mathbb{Z} \qquad \text{and} \qquad \operatorname{supp} \tilde{\mathbf{a}} = [\tilde{m}_1, \tilde{m}_2] \cap \mathbb{Z}.$$

With a slight abuse of notation, in the following, we will always write $\operatorname{supp} \mathbf{a} = [m_1, m_2]$.

As introduced in Remark 3.2.1, the symbol $\mathbf{X}^{\overline{l}}$ will denote the $l$-th row and $\mathbf{X}^{|k}$ the $k$-th column of a matrix $\mathbf{X}$. Using this notation, it can be confirmed, that

$$\begin{aligned} \operatorname{supp} \tilde{\mathbf{M}}^{|k} &= ([\tilde{m}_1, \tilde{m}_2] \cup [1 - m_2, 1 - m_1]) + 2k \quad \text{and} \\ \operatorname{supp} \tilde{\mathbf{G}}^{\overline{k}} &= ([m_1, m_2] \cup [1 - \tilde{m}_2, 1 - \tilde{m}_1]) + 2k. \end{aligned} \tag{3.4.1}$$

Note, that $\alpha \in \operatorname{supp} \tilde{\mathbf{M}}^{|k}$ implies that $(1 - \alpha) \in \operatorname{supp} \overline{\tilde{\mathbf{G}}}^{k}$.

Recall from Section 2.2, that for the cardinal $B$-spline System $N(\kappa, \tilde{\kappa})$, we have

$$[m_1, m_2] = [-\lfloor \frac{\kappa}{2} \rfloor, \lceil \frac{\kappa}{2} \rceil] \quad \text{and} \quad [\tilde{m}_1, \tilde{m}_2] = [-\lfloor \frac{\kappa}{2} \rfloor - \tilde{\kappa} + 1, \lceil \frac{\kappa}{2} \rceil + \tilde{\kappa} - 1], \tag{3.4.2}$$

with $(\kappa + \tilde{\kappa}) \bmod 2 = 0$, and $\kappa, \tilde{\kappa}$ denoting the order of $\phi, \tilde{\phi}$ respectively.

Hence, for spline wavelets, we arrive at

$$\begin{aligned} \operatorname{supp} \tilde{\mathbf{M}}^{|k} &= \left[ -\frac{\kappa}{2} - \tilde{\kappa} + \kappa^* + 1, \frac{\kappa}{2} + \tilde{\kappa} + \kappa^* - 1 \right] + 2k, =: [l_1^G, l_2^G] + 2k \quad \text{and} \\ \operatorname{supp} \tilde{\mathbf{G}}^{\overline{k}} &= \left[ -\frac{\kappa}{2} - \tilde{\kappa} + 2 - \kappa^*, \frac{\kappa}{2} + \tilde{\kappa} - \kappa^* \right] + 2k =: [l_1^M, l_2^M] + 2k, \end{aligned}$$

where $\kappa^* := \frac{\kappa \bmod 2}{2}$. Note that $\operatorname{supp} \tilde{\mathbf{M}}^{|k} = \operatorname{supp} \tilde{\mathbf{G}}^{\overline{k}}$ if $\kappa$ is odd, while $\operatorname{supp} \tilde{\mathbf{M}}^{|k} = [l_1^G, l_2^G] + 2k + 1$ if $\kappa$ is even, which leads to

$$\operatorname{supp} \tilde{\mathbf{M}}^{|k} \cup \operatorname{supp} \tilde{\mathbf{G}}^{\overline{k}} = [l_1^M, l_2^G + (\kappa + 1) \bmod 2] + 2k.$$

Therefore, by (3.4.1)

$$\mathcal{G}_{j+1} = \bigcup_{k(\lambda), \lambda \in \mathcal{T}_j^\circ \cup \mathcal{T}_j} \left[ -\lceil \frac{\kappa}{2} \rceil - \tilde{\kappa} + 2, \lceil \frac{\kappa}{2} \rceil + \tilde{\kappa} - \kappa^* \right] + 2k, j = j_0, \ldots, J.$$

For any realization, it is essential to construct all $\mathcal{G}_j$ in an efficient way, and minimal cardinality of $\mathcal{G}_j$ might not be the main issue. So one might consider to work with the minimal *symmetric* superset set of $\mathcal{G}_j$. Therefore it is in order to remark that with

$$\mathbf{L}_k := [-m, m] + 2k \quad \text{with} \quad m := \frac{\kappa}{2} + \tilde{\kappa} - \kappa^* = \lfloor \frac{\kappa}{2} \rfloor + \tilde{\kappa}, \tag{3.4.3}$$

we can define the symmetric set $\mathcal{L}_{j+1} \supset \mathcal{G}_{j+1}$

$$\mathcal{L}_{j+1} := \bigcup_{k(\lambda), \lambda \in \mathcal{T}_j^\circ \cup \mathcal{T}_j} \mathbf{L}_k.$$

In general, $\mathcal{L}_{j+1}$ is somewhat larger then $\mathcal{G}_{j+1}$, but its symmetry will substantially ease efficient programming especially in the case of multi-dimensional tensor product wavelets.

### 3.4.2 Well-Gradedness

Concerning well-gradedness, we record the following observation.

**Remark 3.4.1.** *For the cardinal B-spline system $N(\kappa, \tilde{\kappa})$, every $M$-graded tree, with $M \geq l/2, l := \lceil \frac{\kappa}{2} \rceil + \tilde{\kappa} - 1$, is well-graded.*

**Proof:** Let $j > j_0$ and $\mu^\circ \in \mathcal{T}^\circ_{j+1}$ be arbitrary. If $\mathcal{T}^\circ$ is $M$-graded, $M \in \mathbb{N}$, by Definition (2.3.4) of $M$-gradedness we know that

$$\mathcal{T}^\circ_j \supset \lfloor \frac{k(\mu)}{2} \rfloor + [-M, M+1].$$

As $\mathcal{G}^-_j = \mathcal{G}_j \backslash \mathcal{T}^\circ_j \subset \mathcal{I}_j$, we conclude

$$\left\{ [-M, M] + \frac{k(\mu^\circ)}{2} \right\} \cap \mathcal{G}^-_j = \emptyset. \tag{3.4.4}$$

For any $\lambda^\circ \in \mathcal{I}_j$ we have that

$$k(\lambda^\circ) \in [-M, M] + \frac{k(\mu^\circ)}{2} \quad \Leftrightarrow \quad k(\mu^\circ) \in 2\,[-M, M] + 2k(\lambda^\circ),$$

which by (3.4.4) leads to

$$k(\mu^\circ) \notin \left( \bigcup_{\lambda : \lambda^\circ \in \mathcal{G}^-_j} 2[-M, M] + 2k(\lambda^\circ) \right). \tag{3.4.5}$$

On the other hand, we have by (3.4.2), again preferring to work with symmetric sets, that

$$\begin{aligned} \operatorname{supp} \mathbf{M}_j^{\tilde{\Phi}\,|\lambda} &= \left[ -\lfloor \frac{\kappa}{2} \rfloor - \tilde{\kappa} + 1, \lceil \frac{\kappa}{2} \rceil + \tilde{\kappa} - 1 \right] + 2k(\lambda) \\ &\subset \left[ -\lceil \frac{\kappa}{2} \rceil - \tilde{\kappa} + 1, \lceil \frac{\kappa}{2} \rceil + \tilde{\kappa} - 1 \right] + 2k(\lambda). \end{aligned}$$

Therefore, with $l := \frac{1}{2}\left( \lceil \frac{\kappa}{2} \rceil + \tilde{\kappa} - 1 \right)$, we have

$$\bigcup_{\lambda : \lambda^\circ \in \mathcal{G}^-_j} \operatorname{supp} \mathbf{M}_j^{\tilde{\Phi}\,|\lambda} \subset \left( \bigcup_{\lambda^\circ \in \mathcal{G}^-_j} [-l, l] + 2k(\lambda^\circ) \right). \tag{3.4.6}$$

Combining (3.4.6) and (3.4.5) with $M = l/2$, we conclude as $\mu^\circ \in \mathcal{T}^\circ_{j+1}$ was arbitrary, that

$$\mathcal{T}^\circ_{j+1} \cap \bigcup_{\lambda^\circ \in \mathcal{G}^-_j} \operatorname{supp} \mathbf{M}_j^{\tilde{\Phi}\,|\lambda^\circ} = \emptyset,$$

hence that $\mathcal{T}$ is well-graded according to Definition (3.3.7). ■

Similar results can of course be formulated for other wavelet constructions. Note that the notion of $M$-gradedness does not involve the refinement matrix, cf. Definition (3.3.7), and is therfore well suited for fast computational schemes.

# Chapter 4

# Supporting Tools

In the previous chapter, we identified essential requirements on two key ingredients of RECOVER, namely the employed quadrature method and and the processed index set. In this chapter, we will investigate on how to ensure these properties and we will propose corresponding computational tools.

## 4.1 Quadrature

We still have to detail on how to set up a reliable quadrature routine, cf. Definition 3.3.5. Before going into this, we comment briefly on the fact that we are actually dealing with approximating $g$ in spaces on which point evaluations are generally not continuous. One should keep in mind that the main context where RECOVER applies has been outlined in Section 3.1. In this case we have $g = F(v)$ where $v$ is a *finite* wavelet expansion and where $F$ is a local operator. Moreover, $v$ itself has usually the form $v = \sum_{\lambda \in \mathcal{T}_v} v_\lambda \psi_\lambda$ where $\mathcal{T}_v$ is a tree that is contained in the prediction tree $\mathcal{T}$. Since quadrature affects only terms $\langle \phi_{j,\mathbf{k}}, F(v) \rangle$ where $(j, \mathbf{k}) \in \mathcal{G}^-$ and since $\mathcal{T}$ is a well graded tree $\phi_{j,\mathbf{k}}$ "sees" only the image of lower scale wavelets in the expansion of $v$ under the mapping $F$. Thus, on the support of $\phi_{j,\mathbf{k}}$, $F(v)$ is roughly speaking "finite dimensional" and can only oscillate with a frequency comparable at most with the diameter of $\phi_{j,\mathbf{k}}$. Moreover, the composition actually has some pointwise smoothness even of some positive Hölder degree. (Of course, when $\varepsilon \to 0$ these norms may eventually grow unboundedly because $F(v)$ is only stable in $\mathcal{H}'$). Thus it is justified to assume always that point evaluations are well defined and that, for any fixed $\mathcal{T} = \mathcal{T}_\varepsilon$, refining quadrature locally, does provide increasing accuracy

of approximations to quantities like $\langle \phi_{j,\mathbf{k}}, g\rangle$.

The approximations $q_{j,\mathbf{k}}(g)$ approximating the exact scaling function coefficients $c_{j,\mathbf{k}} = \langle \phi_{j,\mathbf{k}}, g\rangle$ on $\mathcal{G}_j^-$ can be obtained, for instance, by making use of the fact that

$$\int_\Omega \phi_{j,k}(x) f(x) dx$$

can often be computed *exactly* for certain functions $f$ such as arbitrary other *refinable* functions or polynomials of any degree, see [DM93] and Section 4.1.2. A simple way then is to determine some local approximation $Q_{j,\mathbf{k}}(g)$ on the support of $\phi_{j,\mathbf{k}}$, either by interpolation, cf. Section 4.1.2, or by a least squares fit using proper oversampling. Then $\langle Q_{j,\mathbf{k}}(g), \phi_{j,\mathbf{k}}\rangle$ can be computed exactly providing

$$q_{j,\mathbf{k}} := \int_\Omega Q_{j,\mathbf{k}}(g)(x)\phi_{j,\mathbf{k}}(x)dx, \qquad (j,\mathbf{k}) \in \mathcal{G}_j^-. \tag{4.1.1}$$

**Exactness in $\tilde{S}_j$:** By the above remarks, $Q_{j,\mathbf{k}}(g)$ might be a polynomial of possibly high fixed order, or a linear combination of high order B-splines. However, recall from (3.3.29) that the main issue is to make $L_j g$ close to $P_j g$ in $(\mathcal{H}(\Omega_j))'$. This suggests to make $L_j$ map into $\tilde{S}_j = \operatorname{span}\tilde{\Phi}_j$, cf. Section 3.1. Especially, for fixed but sufficiently dense sampling sets $Y_{j,\mathbf{k}}$ one could define $Q_{j,\mathbf{k}}(g)$ by the *least squares* fit

$$\begin{aligned} Q_{j,\mathbf{k}}(g) &:= \sum_{(j,\mathbf{k}'):\operatorname{supp}\tilde{\phi}_{j,\mathbf{k}'}\cap\operatorname{supp}\phi_{j,\mathbf{k}}\neq\emptyset} q^*_{j,\mathbf{k}'}\tilde{\phi}_{j,\mathbf{k}'}, \quad \text{where} \\ \mathbf{q}^* &= \operatorname{argmin}_{\tilde{\mathbf{q}}} \sum_{y\in Y_{j,\mathbf{k}}} \left( g(y) - \sum_{(j,\mathbf{k}')} \tilde{q}_{j,\mathbf{k}'}\tilde{\phi}_{j,\mathbf{k}'}(y)\right)^2, \end{aligned} \tag{4.1.2}$$

so that, by biorthogonality, $q_{j,\mathbf{k}} = q^*_{j,\mathbf{k}}$. In this case the mapping $L_j$ would be indeed exact on $\tilde{S}_j$.

**Boundedness in $(\mathcal{H}(\Omega_j))'$:** Moreover, "ideally" the mapping $L_j$ would be a linear projector onto $\tilde{S}_j(\mathcal{G}_j^-) := \operatorname{span}\{\tilde{\phi}_{j,\mathbf{k}} : (j,\mathbf{k}) \in \mathcal{G}_j^-\}$ which is bounded in $(\mathcal{H}(\Omega_j))'$. In fact, one could then write $L_j g - P_j(g|\mathcal{G}_j^-) = L_j(g - P_j g)$ so that the boundedness of $L_j$ in $(\mathcal{H}(\Omega_j))'$ would immediately give (3.3.29) hence Proposition (3.3.4) a) would apply and provide the desired estimate (3.3.30). Such a requirement will generally not be feasible. Without any further assumptions on $g$ this cannot be guaranteed by any fixed rule for forming the $q_{j,\mathbf{k}}$. In fact, $L_j$ can ultimately be based only on discrete information on some finest resolution level. On the other hand,

in the above mentioned "locally finite dimensional case" one can expect to contrive a mechanism for improving the quality of the $q_{j,\mathbf{k}}$ if necessary. This means, one can realize the necessary closeness of $L_j g$ to the particular projector $P_j g$ in $(\mathcal{H}(\Omega_j))'$ required by Proposition 3.3.4. We shall describe next some ingredients of such a mechanism.

### 4.1.1 Reliable Quadrature

Let us denote by $\mathcal{G}_{j,r}^-$ the set of those indices in $\mathcal{I}_{j+r}$ which arise from an $r$-fold subdivision of the $\tilde{\phi}_{j,\mathbf{k}}, (j,\mathbf{k}) \in \mathcal{G}_j^-$. These indices enter local refinements of $\tilde{S}_j$ over $\Omega_j$. Let for $r \geq 0$

$$L_j^r g := \sum_{(j+r,\mathbf{k})\in\mathcal{G}_{j,r}^-} q_{j+r,\mathbf{k}}^r(g)\tilde{\phi}_{j+r,\mathbf{k}}$$

be a mapping into $\tilde{S}_{j+r}(\mathcal{G}_{j,r}^-)$. The $L_j^r$ could be defined by any of the above ways but with respect to higher resolution level $j+r$. If $L_j^r g$ provides better approximations to $g$ with increasing $r$, then its projection back to $\tilde{S}_j(\mathcal{G}_j^-)$ can be expected to yield better approximations to $P_j g$, provided these approximations are stable in $(\mathcal{H}(\Omega_j))'$. Thus, given $L_j^r g$ we then set

$$L_j g := P_j(L_j^r g). \tag{4.1.3}$$

Writing $L_j g = (\mathbf{q}^r)_j^T \tilde{\Phi}_j$ in the form (3.3.23), a repeated application of (2.1.28) yields that the array $\mathbf{q}_j$ is given by

$$\mathbf{q}_j = \mathbf{q}_j^r := \mathbf{G}_{j,\tilde{\Phi}} \cdots \mathbf{G}_{j+r-1,\tilde{\Phi}} \mathbf{q}_{j+r}^r. \tag{4.1.4}$$

As indicated above, the rationale is that the closer $L_j^r g$ gets to $g$, due to the better resolution offered by projecting into $\tilde{S}_{j+r}$, the closer should its projection by $P_j$ be to $P_j g$.

A possible obstruction that has to be taken into account is that high frequency components of $g$ might be picked up through the quadrature so as to render $L_j^r$ unstable in $(\mathcal{H}(\Omega_j))'$. We shall discuss possible remedies under the general assumption that quadrature is feasible. More precisely, we shall always make the following working assumption.

**Assumption 4.1.1.** *Let $\mathcal{T} = \mathcal{T}_\varepsilon$ be the prediction tree for which $\|\mathbf{g} - \mathbf{g}(\mathcal{T})\|_{\ell_2(\mathcal{J})} \leq \varepsilon$ and let $J = J(\mathcal{T})$ be the highest scale occurring in $\mathcal{T}$. For*

*any $\rho > 0$ there exists an $R \in \mathbb{N}$ such that one has for all $j \leq J(\mathcal{T}_\varepsilon)$ uniformly in $\varepsilon$*

$$\inf_{v \in \tilde{S}_{j+R}} \|g - v\|_{(\mathcal{H}(\Omega_j))'} \leq \rho \|g - P_j g\|_{(\mathcal{H}(\Omega_j))'}, \tag{4.1.5}$$

*i.e. the wavelet coefficients of $g$ with respect to $\tilde{\Psi}$ have some fixed scale-wise decay and best tree approximation is a near best $N$-term approximation.*

This assumption can be verified for many operators $F$ in the second scenario in Section 3.1 when $g = F(v)$ and $v$ is a finite tree expansion where the tree for $v$ is contained in the prediction tree $\mathcal{T}$ for $F(v)$. We shall refer to this situation, in a slight abuse of terminology, as the "locally finite dimensional case" on every support of $\phi_{j,\mathbf{k}}$ with $(j,\mathbf{k}) \in \mathcal{G}^-$. This follows from the construction of the prediction trees in [CDD03d] and the corresponding error analysis which shows how expanding the tree decreases the error. Since

$$\|L_j g - P_j g\|_{(\mathcal{H}(\Omega_j))'} = \|P_j(L_j^r g - g)\|_{(\mathcal{H}(\Omega_j))'} \lesssim \|L_j^r g - g\|_{(\mathcal{H}(\Omega_j))'}$$

the condition (3.3.29) would be satisfied for $r \leq R$ provided that the mapping $L_j^r$ is *exact* on $\tilde{S}_{j+r}$ and *stable* in $(\mathcal{H}(\Omega_j))'$. Here are some ingredients towards this aim. Any linear combination $v_{j+r} := \sum_{(j+r,\mathbf{k}') \in \mathcal{G}^-_{j,r}} \mathbf{q}_{j+r,\mathbf{k}'} \tilde{\phi}_{j+r,\mathbf{k}'}$ can be written as

$$v_{j+r} = P_j v_{j+r} + (I - P_j) v_{j+r} = \mathbf{p}_j^T \tilde{\Phi}_j + \sum_{l=j}^{j+r-1} \bar{\mathbf{d}}_l^T \Theta_l,$$

where, by (4.1.4)

$$\mathbf{p}_j = \mathbf{G}_{j,\tilde{\Phi}} \cdots \mathbf{G}_{j+r-1,\tilde{\Phi}} \mathbf{q}_{j+r}, \quad \bar{\mathbf{d}}_l = \mathbf{G}_{l,\tilde{\Theta}} \mathbf{G}_{l+1,\tilde{\Phi}} \cdots \mathbf{G}_{j+r-1,\tilde{\Phi}} \mathbf{q}_{j+r}. \tag{4.1.6}$$

Now let $Y_{j+r}(\mathcal{G}^-_{j,r})$ be a set of sampling points in $\Omega_j$ whose cardinality is larger than $\#\mathcal{G}^-_{j,r}$ but of the same order. Fix a positive weight $\alpha$ and consider the least squares problem

$$\begin{aligned}
\mathbf{q}^*_{j+r} &:= \operatorname{argmin}_{\mathbf{q}_{j+r}} \left\{ \alpha \sum_{y \in Y_{j+r}(\mathcal{G}^-_{j,r})} h^d \left( g(y) - \sum_{(j+r,\mathbf{k}') \in \mathcal{G}^-_{j,r}} q_{j+r,\mathbf{k}'} \tilde{\phi}_{j+r,\mathbf{k}'}(y) \right)^2 \right. \\
&\quad \left. + \sum_{l=j}^{j+r-1} \omega_j^2 \|\mathbf{G}_{l,\tilde{\Theta}} \mathbf{G}_{l+1,\tilde{\Phi}} \cdots \mathbf{G}_{j+r-1,\tilde{\Phi}} \mathbf{q}_{j+r}\|^2_{\ell_2(\mathcal{J})} \right\} \\
&=: \operatorname{argmin}_{\mathbf{q}_{j+r}} \{K_0(\mathbf{q}_{j+r}) + K_1(\mathbf{q}_{j+r})\},
\end{aligned} \tag{4.1.7}$$

where $h$ measures the spacing between the sampling points. First of all, for suitable $Y_{j+r}(\mathcal{G}_{j,r}^-)$ (4.1.7) has a unique solution. Moreover, if $g \in \tilde{S}_j(\mathcal{G}_j^-)$ the minimum of the quadratic functional is zero and $P_j(L_j^r g) = g$ on $\Omega_j$. In fact the first part $K_0(\mathbf{q}_{j+r})$ of the functional can be made zero through the proper fit and the second part vanishes since the wavelet coefficients of $g$ vanish. In general, we have $K_1(\mathbf{q}_{j+r}) \sim \|(I - P_j)v_{j+r}\|^2_{(\mathcal{H}(\Omega_j))'}$ controls the dual norm of the fit whose quality is ensured by $K_0(\mathbf{q}_{j+r})$. In fact, $K_0(\mathbf{q}_{j+r})$ can be viewed as a weighted $L_2$-approximation, especially in the above mentioned locally finite dimensional case. For instance, with $\alpha \sim \omega_j^2$, $K_0(\mathbf{q}_{j+r})^{1/2}$ becomes then a good upper bound for $\|g - v_{j+r}\|_{(\mathcal{H}(\Omega_j))'}$.
In practical terms it is not essential to solve (4.1.7) exactly. What matters is that $K_1(\mathbf{q}_{j+r})$ is controlled thereby stabilizing the $(\mathcal{H}(\Omega_j))'$ approximation. Therefore one can proceed as follows.

(i) For $(j, \mathbf{k}) \in \mathcal{G}_j^-$ determine $q_{j,\mathbf{k}}$ according to (4.1.2).

(ii) Set $\mathbf{q}_{j+r} := \mathbf{M}_{j+r-1,\tilde{\Phi}} \cdots \mathbf{M}_{j,\tilde{\Phi}} \mathbf{q}_j$ so that the corresponding $\bar{\mathbf{d}}_l$, $l = j, \ldots, j + r - 1$, vanish, see (4.1.6).

(iii) With this $\mathbf{q}_{j+r}$ as an initial guess one carries out a few gradient descent steps to drive $K_0(\mathbf{q}_{j+r}) + K_1(\mathbf{q}_{j+r})$ towards the minimum.

Again, when $g \in \tilde{S}_j(\mathcal{G}_j^-)$ one already has (up to round off) $K_0(\mathbf{q}_{j+r}) + K_1(\mathbf{q}_{j+r}) = 0$, i.e. exactness is ensured. Step (iii) provides the desired $(\mathcal{H}(\Omega_j))'$-stability where one could vary the weight $\alpha$ in the course of the iteration. Clearly, for fixed $r$ the overall computational work remains proportional to $\#\mathcal{G}_j^-$.
In order to guarantee the desired accuracy of the approximation one could solve (4.1.7) for several increasing values of $r$ (always using the result of the previous step as initial guess). This procedure is stopped when either some fixed upper bound for $r$ is exceeded or when the following criteria are fulfilled for some $r' > r$ and some fixed constant $c < 1$

$$\begin{aligned} \|\mathbf{q}_j^{r'} - \mathbf{q}_j^r\|_{\ell_2(\mathcal{J})} &\leq c\varepsilon(\#\mathcal{G}_j^-/\#\mathcal{G}^-), \\ \left(\sum_{l=j+r}^{j+r'-1} \omega_j^2 \|\mathbf{G}_{l,\tilde{\Theta}} \mathbf{G}_{l+1,\tilde{\Phi}} \cdots \mathbf{G}_{j+r-1,\tilde{\Phi}} \mathbf{q}_{j+r'}\|^2_{\ell_2(\mathcal{J})}\right)^{1/2} &\leq c\varepsilon(\#\mathcal{G}_j^-/\#\mathcal{G}^-). \end{aligned} \tag{4.1.8}$$

An alternative criterion is to increase $r$ until

$$\left(K_0(\mathbf{q}_{j+r}^*) + K_1(\mathbf{q}_{j+r}^*)\right)^{1/2} \leq C\varepsilon(\#\mathcal{G}_j^-/\#\mathcal{G}^-), \tag{4.1.9}$$

where now $C$ could be a fixed (larger) constant. To explain these criteria note that, under Assumption 4.1.1, $\|g - L_j^r g\|_{(\mathcal{H}(\Omega_j))'}$ gets small compared with $\|g - P_j g\|_{(\mathcal{H}(\Omega_j))'}$ for increasing $r$ so that the $\mathbf{q}_j^r$ become stationary for increasing $r$ while the higher order wavelet coefficients become negligible. Thus the stopping criteria (4.1.8) are met after a finite number of steps depending only on the choice of the constants $c$ in (4.1.8), (4.1.9). As for (4.1.9), a prediction error $\|\mathbf{g} - \mathbf{g}(\mathcal{T})\|_{\ell_2(\mathcal{J})} \leq \varepsilon$ suggests to expect a portion like $\varepsilon(\#\mathcal{G}_j^-/\mathcal{G}^-)$ to be attributed to $\mathcal{G}_j^-$. On account of the norm equivalence (3.1.3) one expects that $\|g - P_j(g|\mathcal{G}_j^-)\|_{(\mathcal{H}(\Omega_j))'} \lesssim \varepsilon(\#\mathcal{G}_j^-/\mathcal{G}^-)$. Thus the value of $\big(K_0(\mathbf{q}_{j+r}^*) + K_1(\mathbf{q}_{j+r}^*)\big)^{1/2}$ would be of the order $\varepsilon(\#\mathcal{G}_j^-/\mathcal{G}^-)$ for $\mathbf{q}_{j+r}^* := \mathbf{M}_{j+r-1,\tilde{\Phi}} \cdots \mathbf{M}_{j,\tilde{\Phi}} \mathbf{c}_j$. Hence, when (4.1.9) is met one has an $\mathcal{H}'$-stable sufficiently good approximation to $P_j(\cdot|\mathcal{G}_j^-)$.

**Remark 4.1.2.** *Recall that* RECOVER *works from top to bottom, i.e., $\mathcal{G}^-$ is not known beforehand and is in fact never assembled. However, on account of (3.2.22) we can replace $\#\mathcal{G}^-$ in (4.1.8), (4.1.9) by the known quantity $\#\mathcal{T}^\circ$.*

Thus, in principle, for a wide range of cases an asymptotically optimal work count, as required by Remark 3.2.3, can be achieved. Nevertheless, the quantitative performance of a strategy outlined above may still be unacceptable. It is therefore important to see whether the full scope of such a strategy is generally necessary. The numerical experiments in Part II are to shed some light on this issue.

### 4.1.2 Gauss-Quadrature for Refinable Functions

Assuming that the given problem permits the application of a quadrature method based on interpolation, the use of one of the well–known quadrature formulas which can be found in any book on numerical analysis, usually still leads to serious trouble. This is due to the fact that the error-estimates for knot-based quadrature rules which reproduce polynomials such as, e.g., Simpson's rule, depend on derivatives of the integrand, that is on the smoothness of the scaling function. As the refinable function is not necessarily very smooth these quadrature rules may not perform satisfactorily.

Another problem that may occur is that in many cases, the scaling functions are not known explicitly but only via certain functional equations from which the function values have to be computed or approximated. This is possible in principle, however, these function evaluations may be expensive and/or inaccurate.

In the last few years, several approaches to this problem have been suggested. Dahmen and Micchelli [DM93] developed a method which is based on the iteration of a specific operator derived from a related subdivision scheme. It works in any dimension and under very low regularity assumptions on the function $g$, but it might be somewhat expensive. A different approach is presented in [PS94b, PS94a], where quadrature formulas for refinable functions are considered. Several types of Newton–Cotes quadratures are discussed, which are determined using Tschebyscheff polynomials.

In this section, we follow the approach in [BBDK01, BBD$^+$02], which is based on Gauss quadrature. The key ingredient here is, that one can use the iteration scheme of Dahmen and Micchelli [DM93] to compute the moments of refinable functions up to round-off. This leads to quadrature formulas, that are independent of the regularity of the refinable function.

The following method can be used for the computation of integrals involving scaling functions or wavelets. In view of the recovery scheme we will put a clear focus on quadrature rules for scaling functions as they are needed in the recovery scheme and because quadrature rules for wavelets can be reduced to the former ones.

### General Setting

A univariate Gauss quadrature rule replaces an integral of the form

$$\int_{[a,b]} g(x)w(x)dx, \quad [a,b] \subset \mathbb{R},$$

by a weighted sum of point evaluations of $g$, i.e.,

$$\int_{[a,b]} g(x)w(x)dx \approx I_w^n(g) := \sum_{i=1}^{n} \gamma_i g(x_i), \qquad [a,b] \subset \mathbb{R}, \quad n \in \mathbb{N}, \tag{4.1.10}$$

with *knots* $x_i$ and *weights* $\gamma_i$, $i = 1, \dots n$. Here $w$ is called the *weight function*, which in the most classical case is chosen to be $w \equiv 1$. However, the theory can be developed for a certain class of functions. In fact, $w$ only has to satisfy the following conditions, cf. [Sto93].

**Definition 4.1.3.** *A function $w$ on the (finite or infinite) interval $(a,b)$ is called a* weight function *if it is non-negative and measurable and if all its moments*

$$\mu_i(w) := \int_a^b x^i w(x)\, dx \quad i = 0, 1, \dots \tag{4.1.11}$$

*exist. Moreover, we always require that $\mu_0(w) > 0$.*

Defining the inner product corresponding to $w$ as

$$\langle h, g\rangle_w := \int_{[a,b]} h(x)g(x)w(x)dx, \tag{4.1.12}$$

the knots $x_i$ of the Gauss rule are the zeros of the $n$–th orthogonal polynomial with respect to this scalar product. A quadrature rule is said to be of *degree* $D$ if it is exact for all polynomials with order up to $D$. Gauss quadrature rules with $n$ points are of degree $D = 2n$. This is optimal, because it uses $2n$ degrees of freedom, $n$ knots and $n$ weights. For each weight function, knots and weights have to be determined only once and for all, since they do not depend on the integrand.

The computation of the Gauss quadrature rule according to a specific weight function $w$ can be done in several ways, see e.g., [DR75, GW69], but in this context we suggest the following well known eigenvector/eigenvalue ansatz, see [BBDK01, GGP00].

Determine the orthogonal polynomials according to $(\cdot,\cdot)_w$, which are given by the following three-term recursion formula

$$\pi_{k+1}(x) = (x - \alpha_k)\pi_k(x) - \beta_k\pi_{k-1}(x),\ k = 0, \ldots, n-1, \tag{4.1.13}$$

$$\pi_{-1}(x) = 0,\ \pi_0(x) = 1. \tag{4.1.14}$$

The coefficients $\alpha_k$, $\beta_k$ can be computed as

$$\alpha_k = \frac{\langle x\,\pi_k, \pi_k\rangle_w}{\langle \pi_k, \pi_k\rangle_w}, \quad \beta_k = \frac{\langle \pi_k, \pi_k\rangle_w}{\langle \pi_{k-1}, \pi_{k-1}\rangle_w}. \tag{4.1.15}$$

Note that all $\alpha_k$ vanish if the weight function is symmetric to the origin. Once $\alpha_k, \beta_k$ are determined, one sets up the following $n \times n$ symmetric tri-diagonal matrix,

$$\mathbf{W}_n := \begin{pmatrix} \alpha_0 & \sqrt{\beta_1} & & \\ \sqrt{\beta_1} & \ddots & \ddots & \\ & \ddots & & \sqrt{\beta_{n-1}} \\ & & \sqrt{\beta_{n-1}} & \alpha_{n-1} \end{pmatrix}. \tag{4.1.16}$$

If $\omega_k, \upsilon^k$ are the eigenvalues and eigenvectors of $\mathbf{W}_n$

$$\mathbf{W}_n\upsilon^k = \omega_k\upsilon^k, \quad k = 1, \ldots, n,$$

with $\|\upsilon^k\|_2^2 = \langle \phi_0, \pi_0\rangle_w = \int_{[a,b]} w(x)dx$, then, following [GW69], the knots and weights of the Gauss formula with weight $w$ are determined by

$$\gamma_k = (\upsilon_1^k)^2, \quad x_k = \omega_k.$$

Note that the weight function only contributes to this process by its moments. If $w$ is a refinable function or a wavelet, this can be done up to round off at unit cost, cf. Section 4.1.2.

Once a Gauss quadrature rule of the form (4.1.10) is established, one is clearly interested in estimating the *quadrature error* which is defined by

$$E_w^n(g) := \int_{[a,b]} g(x)w(x)dx - I_w^n(g). \tag{4.1.17}$$

A classical result states that if $g \in C^{2n}(\mathbb{R})$

$$E_w^n(f) = \frac{g^{(2n)}(\xi)}{(2n)!k_{n,0}^2}, \quad \text{for some } a < \xi < b, \tag{4.1.18}$$

see again [DR75] for details. Here $k_{n,0}$ denotes the leading coefficient of the $n$–th orthonormal polynomial with respect to the scalar product $\langle \cdot, \cdot \rangle_w$.

One could also consider to construct different quadrature formulas like, e.g., Newton-Cotes rules taking $w$ as the weight function. Yet, not only that one will loose the optimal order $2n$ for an $n$-point rule, also the error bounds are less favorable, cf. e.g., [Kon01].

If the weight function and the degree is given by (4.1.18), it is possible to control the error for any specific Gauss quadrature rule. However, we would like to estimate $k_{n,0}$ for $n \to \infty$ in order to be able to quantify the asymptotical behavior of $E_w^n(f)$. This is possible, cf. [BBD$^+$02], at least for weight functions satisfying some additional conditions as we shall see in the following.

**Definition 4.1.4.** *Let $w$ be a weight function according to Definition* 4.1.3 *on the interval $[-1, 1]$. $w$ is said to be in the function class $\mathcal{W}$ if*

$$\int_{-\pi}^{\pi} |\ln w^*(\zeta)| d\zeta < \infty, \tag{4.1.19}$$

*for $w^* := w(\cos(\zeta)) \, |\sin(\zeta)|$.*

For this special class $\mathcal{W}$, the following theorem holds, see e.g., [Sze39] for details.

**Theorem 4.1.5.** *Let $w \in \mathcal{W}$ and*

$$p_n(x) = \sum_{i=0}^{n} k_{n,i} x^{n-i}, \quad n = 0, 1, 2, \ldots \tag{4.1.20}$$

*be the system of orthonormal polynomials associated with the weight function $w$. Then, as $n$ tends to infinity,*

$$k_{n,0} \sim 2^n C_w, \qquad \textit{with } C_w := \pi^{-1/2} \exp\left(\frac{-1}{2\pi} \int_{-1}^{1} \frac{\log(w(x))}{\sqrt{1-x^2}}\, dx\right). \quad (4.1.21)$$

The constant $C_w$ can be approximated using quadrature. See Table 4.1 for a comparison between the true and the estimated $k_{n,0}$ in the B-spline case $w = {}_i\phi, i = 2, 3, 4$.

| | m=2 | | m=3 | | m=4 | |
|---|---|---|---|---|---|---|
| n | $k_{n,0}$ | $2^n C_{1\phi}$ | $k_{n,0}$ | $2^n C_{3\phi}$ | $k_{n,0}$ | $2^n C_{4\phi}$ |
| 1 | 2.45 | 2.84 | 3.67 | 6.11 | 4.90 | 15.96 |
| 2 | 5.07 | 5.68 | 8.71 | 12.22 | 13.02 | 31.92 |
| 3 | 10.51 | 11.36 | 19.09 | 24.44 | 30.82 | 63.84 |
| 4 | 21.26 | 22.72 | 40.79 | 48.88 | 69.12 | 127.68 |
| 5 | 86.80 | 90.24 | 85.14 | 97.75 | 150.62 | 255.36 |

Table 4.1: Computed and approximated leading coefficient of the orthogonal polynomial

Theorem 4.1.5 can now be used to derive the following result for spline functions.

**Corollary 4.1.6.** *Let $w$ be a non-negative spline function on $[-1, 1]$ having only a finite number of zeros and let $k_{n,0}$ be defined according to (4.1.20). Then as $n$ tends to infinity*

$$k_{n,0} \sim 2^n C_w, \qquad \textit{with } C_w := \pi^{-1/2} \exp\left(\frac{-1}{2\pi} \int_{-1}^{1} \frac{\log(w(x))}{\sqrt{1-x^2}}\, dx\right). \quad (4.1.22)$$

*Furthermore, if $g \in C^{2n}, n \in \mathbb{N}$, the error of the corresponding Gauss quadrature satisfies, for $n$ tending to infinity*

$$|E_w^n(g)| = \left| \int_{-1}^{1} g(x)w(x)dx - I_w^n(g) \right| \sim \left| \frac{g^{(2n)}(\xi)}{(2n)!2^{2n}C_w^2} \right|, \quad (4.1.23)$$

*for some $-1 < \xi < 1$.*

Gauss quadrature formulas also exist for multivariate integrals. For certain integration domains such as rectangles Gauss quadrature rules can be derived by a tensor product formula made up of univariate Gauss formulas. Here the weight function has the form

$$\mathcal{V}(x_1, \ldots, x_d) = \vartheta_1(x_1)\vartheta_2(x_2)\cdots\vartheta_d(x_d), \quad (4.1.24)$$

where $\vartheta_i,\ i = 1, \ldots, d$, denote univariate weight functions. We search for a quadrature formula of degree $D, D \in \mathbb{N}$, of the form

$$\int_{\mathrm{supp}(\vartheta_0)} \cdots \int_{\mathrm{supp}(\vartheta_d)} f(x_1, \ldots, x_d)\vartheta_1(x_1) \cdots \vartheta_d(x_d)\, dx_1 \cdots dx_d \sim \sum_{i=1}^{n(D)} \lambda_i f(x_{i,1}, \ldots, x_{i,d}). \tag{4.1.25}$$

It is easy to derive such a formula from the univariate rule by using the following general result shown in [Str71]. Let $\mathbb{R}^d \supset I = I_1 \times I_2,\ I_1 \subset \mathbb{R}^{d_1},\ I_2 \subset \mathbb{R}^{d_2}$ be a cube. Suppose that we are given quadrature rules on $I_1$ and $I_2$ corresponding to weight functions $g_1$ and $g_2$, respectively, and with points and weights

$$(x_{i,1}, \ldots, x_{i,d_1}),\ \lambda_i,\ i = 1, \ldots, n_1, \quad \text{and} \quad (x_{j,d_1+1}, \ldots, x_{j,d}),\ \mu_j,\ j = 1, \ldots, n_2, \tag{4.1.26}$$

which are of degree $D_1$ and $D_2$, respectively. Let $g$ be defined by

$$g(x_1, \ldots, x_d) = g_1(x_1, \ldots, x_{d_1})g_2(x_{d_1+1}, \ldots, x_d).$$

Then the following theorem holds.

**Theorem 4.1.7.** *The $n = n_1 n_2$ points*

$$(x_{i,1}, \ldots, x_{i,d_1}, x_{j,d_1+1}, \ldots, x_{j,d}), \tag{4.1.27}$$

*and weights*

$$\lambda_i \mu_j,\ i = 1, \ldots, n_1,\ j = 1, \ldots, n_2, \tag{4.1.28}$$

*are an integration formula for the weight function $g$ on $I$ which is of degree $D = \min(D_1, D_2)$.*

Applying Theorem 4.1.7 $(d-1)$ times provides a product quadrature formula for a function $\mathcal{V}$ of the form (4.1.24). Error formulas for these product formulas can also be easily deduced from the one dimensional estimates, see e.g. [SS66].

In certain cases, it is also possible to construct non–product formulas. For details in the construction, please see e.g. [BBDK01, BBD$^+$02]. For further and more general information on quadrature, the reader is referred, e.g. to [DR75, Kry62, Str71] and the references therein.

### Moments of Refinable Functions and Wavelets

Let $\vartheta$ be a refinable non-negative weight function. In order to determine the knots and weights of the Gauss formula according to Section 4.1.2,

we have to determine the moments of $\vartheta$. In some special cases they can be computed by simple integration, e.g. in the instance of cardinal B-spline wavelets. However, in general, this is not possible. Using a usual quadrature rule like Simpson's formula, cf. [GGP00] will limit the approach to 'smooth' functions. This would lead to the same problems that we wanted to circumvent.

The computation of integrals of monomials and scaling functions can be done efficiently and – up to round off – exactly by using recursion relations, see [DM93]. The same is true for integrals involving wavelets as they are linear combinations of scaling functions. Let us briefly recall the basic ideas. We start with the case of an arbitrary refinable compactly supported function $\vartheta \in I\!R^d$. Let us define

$$\mu_\beta := \int_{I\!R^d} x^\beta \vartheta(x) dx, \quad \beta = (\beta_1, \ldots, \beta_d), \quad x^\beta = x_1^{\beta_1} \ldots x_d^{\beta_d}. \tag{4.1.29}$$

The normalization

$$\int_{\mathrm{supp}(\phi)} \vartheta(x) dx = 1 \tag{4.1.30}$$

leads to $\mu_0 = 1$. Defining

$$c_\alpha^\beta := 2^{-|\beta|-d} \binom{\beta}{\alpha} \sum_{k \in Z\!\!Z^d} a_k k^\alpha, \quad |\beta| := \sum_{l=1}^d \beta_l, \tag{4.1.31}$$

we obtain

$$\mu_\beta = \sum_{0 \le \alpha \le \beta} c_{\beta-\alpha}^\beta \mu_\alpha = c_0^\beta \mu_\beta + \sum_{0 \le \alpha < \beta} c_{\beta-\alpha}^\beta \mu_\alpha. \tag{4.1.32}$$

Here $\alpha < \beta$ means $\alpha_j \le \beta_j$, $j = 1, \ldots, d$, and $\alpha_i < \beta_i$ for at least one $i \in \{1, \ldots, d\}$. This leads to the recursion

$$\mu_\beta = (1 - 2^{-|\beta|}) \sum_{0 \le \alpha < \beta} c_{\beta-\alpha}^\beta \mu_\alpha. \tag{4.1.33}$$

Let us mention that the above scheme to compute the moments does no longer work for boundary adapted function, compare Section 2. One can still determine the corresponding moments, at least if the wavelet bases are according to [DKU99]. In that case however, a modified ansatz has to be pursued that we will not describe here since it requires details from the construction in [DKU99], see [BBD$^+$02] for further details.

### Gauss Formulas for the Wavelet Setting

The integrals we have to deal with in the (univariate) wavelet setting are of the following form

$$\langle g, \vartheta_{j,k} \rangle \;=\; \int_{\mathrm{supp}(\vartheta_{j,k})} g(x)\vartheta_{j,k}(x)\; dx. \tag{4.1.34}$$

Herein $g$ is a given function and $\vartheta$ is either a refinable function $\phi$ or a wavelet $\psi$. In view of the recovery scheme, we will mainly be concerned with the situation $\vartheta = \phi$. By means of substitution, one can write (4.1.34) as

$$\int_{\mathrm{supp}(\vartheta_{j,k})} g(x)\vartheta_{j,k}(x)\; dx \;=\; 2^{-j/2} \int_{\mathrm{supp}(\vartheta)} g(2^{-j}(u+k))\; \vartheta(u)\; du.$$

Hence, it suffices to find a quadrature rule for integrals of the form

$$\int_{\mathrm{supp}(\vartheta)} g(x)\vartheta(x)dx, \tag{4.1.35}$$

to be able to compute all integrals of the form (4.1.34), i.e. we have to set up only one quadrature rule for $\vartheta$.

The general idea is now, to construct Gaussian quadrature rules using $\vartheta$ as weight function. No smoothness is required for $\vartheta$, we do not even have to know it explicitely as long as we can determine its moments and $\vartheta$ as a weight function according to Definition 4.1.11. This however requires that $\vartheta$ should be non–negative which might not be the case for general scaling functions. Yet, this problem can be fixed using the following *lifting trick*. Suppose that for $\vartheta \ngeq 0, \vartheta \in \{\phi, \psi\}$ there exists an appropriate constant $c > 0$ such that on $\mathrm{supp}(\vartheta)$

$$\vartheta^c(x) := \vartheta(x) + c\chi_{[l_1,l_2]}(x) \geq 0, \tag{4.1.36}$$

where $\chi_{[l_1,l_2]}$ denotes the characteristic function of $[l_1, l_2] \supseteq \mathrm{supp}(\vartheta)$. Then one can set up a quadrature rule with weights $\lambda_i^c$ and knots $x_i^c$ corresponding to the non–negative function $\vartheta^c(x)$. After that, a second Gauss rule with weights $\lambda_i^\chi$ and knots $x_i^\chi$ for the characteristic function $\chi_{[l_1,l_2]}$ has to be determined, which is also no problem, since the moments can easily be computed. In this case, we loose the optimality of the Gauss rules as we now need $2n$ knots for a formula of degree $2n$.

In this context, its important to note that in the case of cardinal B-spline wavelet bases, our main discretization tool, the primal generator functions

$\phi = {}_\kappa\phi$ are always non-negative. Hence in that case, we do not have to rely on the shifting device in our recovery algorithm, as there inner products of the form $\langle g, ({}_\kappa\phi)_\lambda \rangle$ are required.

| n | $w_i^{2\phi}$ | $x_i^{2\phi}$ |
|---|---|---|
| 1 | 1.00000000000000000000000000000000 | 0.00000000000000000000000000000000 |
| 2 | 0.50000000000000000000000000000000 | 0.40824829046386301636621401245 1 |
| 3 | 0.20833333333333333333333333333334 | 0.63245553203367586639977870888 7 |
| | 0.58333333333333333333333333333332 | 0.00000000000000000000000000000000 |
| 4 | 0.09886630457987562678585274594 7 | 0.75092514330444738454981533603 3 |
| | 0.40113369542012437321414725405 3 | 0.26222984265274796382920272100 1 |
| 5 | 0.05165825776549062179091399243 1 | 0.82144059973838152787200028633 2 |
| | 0.23947324070545739044150190970 7 | 0.44992035245984196334944729220 7 |
| | 0.41773700305810397553516819572 5 | 0.00000000000000000000000000000000 |

Table 4.2: Weights and knots for the hat function ${}_2\phi$

For an intensive comparison of various quadrature methods including the presented Gauss type method, in the context of wavelet methods (for boundary integral equations), see [Kon01].

## 4.2 Index-Structure

When introducing the recovery scheme RECOVER in the previous chapter, we have imposed certain structural features on the underlying index set, such as an well-gradedness or $M$-gradedness, completeness etc.

It remains to clarify, how to actually find the minimal $M$-graded tree of support cubes $(\mathcal{T}^M)^\circ \supseteq \Lambda^\circ$ for an arbitrary, i.e., unstructured index set $\Lambda \subset \mathcal{J}$.

Asymptotically, we have for a tree $\mathcal{T} \in \mathcal{J}$ that $\#\mathcal{T}^\circ \sim \#(\mathcal{T}^M)^\circ$, but the constants interrelating $\#\mathcal{T}^\circ$ and $\#(\mathcal{T}^M)^\circ$, and the amount of work necessary to construct $(\mathcal{T}^M)^\circ$ should not be underestimated. Note e.g., that grading will have to be increasingly demanding if higher order wavelets are needed and that the effect of the grading as well as of the enforcing of the completeness property depends heavily on the dimension.

There are three basic *index-related* tasks to accomplish: Construct the $M$-graded hull, enforce completeness and find the leaves of a set. Note that all index manipulations will be formulated in terms of support cubes.

**Grading:** Let $M \in \mathbb{N}_0$ and $\Lambda \subset \mathcal{J}$. The grading routine has to ensure that for reach index, the $M$-cube of indices centered around its parent-cell is also contained in the $\Lambda$. Therefore let us define the set

$$\mathcal{P}^M(\Lambda^\circ) := \bigcup_{\lambda^\circ \in \Lambda^\circ} \left\{ (|\lambda^\circ| - 1, \mathbf{k}'), \mathbf{k}' \in k(\mathcal{P}(\lambda^\circ)) + \{-M, -M+1, \ldots, M\}^d \right\}. \tag{4.2.1}$$

Recall that here for $d = 1$ we have $\mathcal{P}^M(\lambda^\circ) = \lfloor k(\lambda^\circ)/2 \rfloor$. The following simple procedure transforms an unstructured set into an $M$-graded tree on $\square = [0,1]^d$.

GRADING — $(\Lambda^\circ, M) \to [(\mathcal{T}^M)^\circ]$

| |
|---|
| Set $(\mathcal{T}_J^M)^\circ := \Lambda_J^\circ$ |
| For $j = J-1, \ldots, j_0$ do |
| Set $(\mathcal{T}_j^M)^\circ := \mathcal{P}^M((\mathcal{T}_{j+1}^M)^\circ) \cup \Lambda_j^\circ$ |

Algorithm 4.1: Procedure GRADING: Grading a set of support cubes

**Completion:** In addition to gradedness, we also have to complete the tree of support cubes by inserting all children of all coordinates. For $\mu \in \mathcal{J}$ with $k(\mu) = (k_1, \ldots, k_d)$, the cube $[2k(\mu), 2k(\mu)+1] := \{2k_1, 2k_1+1\} \times \ldots \times \{2k_d, 2k_d+1\} \subset \mathbb{Z}^d$ is just the set of the locations of the children of $\square_\mu$ obtained by dyadic subdivision. Let $\overline{\Lambda^\circ}$ denote the complete hull of $\Lambda^\circ$.

COMPLETION — $(\Lambda^\circ) \to \overline{\Lambda^\circ}$

| |
|---|
| For $j = J, \ldots, j_0 + 1$ do |
| Set $\overline{\Lambda^\circ}_j = \mathcal{C}(\mathcal{P}(\Lambda^\circ_j)) := \{(j, \mathbf{k}), \mathbf{k} \in \bigcup_{\mu \in \mathcal{P}(\Lambda^\circ_j)} [2k(\mu), 2k(\mu)+1]\}$ (4.2.2) |
| Set $\overline{\Lambda^\circ}_{j_0} := {\Lambda^\circ}_{j_0}$ |

Algorithm 4.2: Procedure COMPLETION: Completing a set of support cubes

**Leaves:** In view of RECOVER we are actually only interested in the leaves $\partial\overline{\mathcal{T}^{M\circ}}$. Recall that the leaves are those indices having no children in the set, i.e., for $\Gamma \subset \mathcal{J}$ we have

$$\partial\Gamma_j = \Gamma_j \backslash \mathcal{P}(\Gamma_{j+1}), \quad j_0 \le j \le J.$$

For $j_0 < l \le J$ we will combine the two simple routines above in such a way that we compute the leaves of an $M$- graded and complete set $\overline{\mathcal{T}^{M\circ}} \supseteq \Lambda^\circ$ in one sweep.

LEAF — $(\Lambda^\circ, M) \to [\partial\overline{\mathcal{T}^{M\circ}}]$

| | |
|---|---|
| Determine $J = J(\Lambda^\circ)$ such that ${\Lambda^\circ}_j = \emptyset, j > J$ | |
| Set $\Gamma_{J+1} := \emptyset$ | |
| For $j = J, J-1, \ldots, j_0$ | |
| | $(\mathcal{T}_j^M)^\circ := \Gamma_{j+1} \cup {\Lambda^\circ}_j$ |
| | If $j > j_0$: $\Gamma_j := \mathcal{P}((\mathcal{T}_j^M)^\circ)$ and $\partial\overline{(\mathcal{T}_j^M)^\circ} := \mathcal{C}(\Gamma_j)\backslash\Gamma_{j+1}$<br>Else: $\partial\overline{(\mathcal{T}_j^M)^\circ} := (\mathcal{T}_j^M)^\circ \backslash \Gamma_{j+1}$ |

Algorithm 4.3: Procedure LEAF: Computing leaves of $M$-graded complete hull

**Lemma 4.2.1.** *For a given index set* $\Lambda \subset \mathcal{J}$ *and* $M \in \mathbb{N}_0$, GRADING$(\Lambda^\circ, M)$ *produces the minimal* $M$*-graded tree* $(\mathcal{T}^M)^\circ \supseteq \Lambda^\circ$.
*Likewise* LEAF$(\Lambda^\circ, M)$ *determines the leaves of the minimal* $M$*-graded and complete tree* $\overline{\mathcal{T}^{M\circ}} \supseteq \Lambda^\circ$

**Proof:** Let $\hat{\Lambda}^\circ$ with $\Lambda^\circ \subseteq \hat{\Lambda}^\circ \subset \mathcal{J}$ be an $M$-graded tree. We will show that $\hat{\Lambda}^\circ \supseteq (\mathcal{T}^M)^\circ$. Because of $\Lambda^\circ \subseteq \hat{\Lambda}^\circ$ and $(\mathcal{T}_J^M)^\circ = \Lambda_J^\circ$ we have that $\hat{\Lambda}_J^\circ \supseteq (\mathcal{T}_J^M)^\circ$. Concerning level $J-1$, let $\lambda^\circ \in (\mathcal{T}_J^M)^\circ$. Due to (2.3.4) and (2.3.1) we have

$$k(\mathcal{P}(\lambda^\circ))+2^{-J+1}[-M,M+1]^d \subseteq \bigcup_{\mu_\circ \in \hat{\Lambda}_{J-1}^\circ} \square_{\mu_\circ} = \bigcup_{\mu_\circ \in \hat{\Lambda}_{J-1}^\circ} 2^{-J+1}\left(k(\mu_\circ)+[0,1]^d\right).$$

Note that

$$k(\mathcal{P}(\lambda^\circ)) + 2^{-J+1}[-M,M+1]^d = \bigcup_{\mathbf{k}\in\{k(\mathcal{P}(\lambda^\circ))+\{-M,\dots,M\}^d\}} 2^{-J+1}(\mathbf{k}+[0,1]^d),$$

and hence

$$\bigcup_{\mathbf{k}\in\{k(\mathcal{P}(\lambda^\circ))+\{-M,\dots,M\}^d\}} 2^{-J+1}(\mathbf{k}+[0,1]^d) \subseteq \bigcup_{\mu_\circ \in \hat{\Lambda}_{J-1}^\circ} 2^{-J+1}\left(k(\mu_\circ)+[0,1]^d\right).$$

Hence $\mathcal{P}^M(\lambda^\circ) \subseteq \hat{\Lambda}_{J-1}^\circ$ and consequently $\hat{\Lambda}_{J-1}^\circ \supseteq \mathcal{P}^M((\mathcal{T}_J^M)^\circ)$. Therefore, as $\hat{\Lambda}_{J-1}^\circ \supseteq {\Lambda^\circ}_{J-1}$, we have $\hat{\Lambda}_{J-1}^\circ = \hat{\Lambda}_{J-1}^\circ \cup {\Lambda^\circ}_{J-1} \supseteq \mathcal{P}^M((\mathcal{T}_J^M)^\circ) \cup {\Lambda^\circ}_{J-1} = (\mathcal{T}_{J-1}^M)^\circ$. Analogously, one concludes that $(\mathcal{T}_{j-1}^M)^\circ = \mathcal{P}^M((\mathcal{T}_j^M)^\circ) \subseteq \hat{\Lambda}_{j-1}^\circ$ for $J > j > j_0$, i.e., $(\mathcal{T}^M)^\circ \subseteq \hat{\Lambda}^\circ$.
Let us now assume, that the $M$-graded tree $\hat{\Lambda}^\circ \supseteq \Lambda^\circ$ is also complete. As $\hat{\Lambda}^\circ \supseteq (\mathcal{T}^M)^\circ$, for each $\lambda^\circ \in (\mathcal{T}^M)^\circ, |\lambda^\circ| > j_0$ we have that $\mathcal{P}(\lambda^\circ)$ in $\hat{\Lambda}^\circ$. Completeness means that all children of $\mathcal{P}(\lambda^\circ)$ are contained in $\hat{\Lambda}^\circ$, hence $\mathcal{C}(\mathcal{P}(\lambda^\circ)) \subseteq \hat{\Lambda}^\circ$, i.e. $\overline{\mathcal{T}^{M\circ}} \subseteq \hat{\Lambda}^\circ$.
■

One can use LEAF as a preprocessing step to generate an input set suitable for the recovery scheme, but for a realization, we suggest the following version that intertwines the index manipulative and the recovery part.

## 4.3 RECOVER*: Everything in one Sweep

We will now present a second version of the recovery scheme, that combines RECOVER, the level-wise clean off-step mentioned already in Section

3.2.2 and the index manipulations that are necessary if the input set $\Lambda$ is (completely) unstructured.

The main advantage is, that performing the index manipulations in a preprocessing step for RECOVER requires to assemble and to store the entire set including all structural demands a-priorily, while the intertwined version RECOVER$^*$ avoids this.

As a valuable side-effect, the (approximated) coefficients of the structured prediction set can be discarded as soon as the wavelet coefficients $\mathbf{d}_j^{\mathrm{R}}$ of the corresponding level $j$ have been computed during the iteration of the recovery scheme. Recall from Sections 3.1 and 3.2.2 that the input set for the recovery scheme is not necessarily minimal. The above clean-off 'on the fly' frees storage at a possibly early stage and also contributes to reduce the problem that an erroneous entry in the prediction set generates other unsignificant entries due to structural demands.

One of the simplest possible strategies for this clean-off step is to equally distribute the level-dependent thresholding parameter $\varepsilon_j$, i.e. to set $\varepsilon_j := \varepsilon/(J - j_0 + 1)$ and to execute BIN-THRESH$(\mathbf{d}_j^{\mathrm{R}}, \varepsilon_j) \to \overline{\mathbf{d}}_j$. Of course then

$$\|\mathbf{d}^{\mathrm{R}} - \overline{\mathbf{d}}\|_{\ell_2} \leq \sum_{j_0}^{J} \|\mathbf{d}_j^{\mathrm{R}} - \overline{\mathbf{d}}_j\|_{\ell_2} \leq \sum_{j_0}^{J} \varepsilon_j \leq \varepsilon.$$

Another possible choice for $\varepsilon_j$ could be $\varepsilon_j := \varepsilon - \sum_{j+1}^{J} \|\mathbf{d}_j^{\mathrm{R}} - \overline{\mathbf{d}}_j\|_{\ell_2}$, so that

$$\|\mathbf{d}^{\mathrm{R}} - \overline{\mathbf{d}}\|_{\ell_2} \leq \|\mathbf{d}_{j_0}^{\mathrm{R}} - \overline{\mathbf{d}}_{j_0}\|_{\ell_2} + \sum_{j_0+1}^{J} \|\mathbf{d}_j^{\mathrm{R}} - \overline{\mathbf{d}}_j\|_{\ell_2} \leq \varepsilon.$$

The latter strategy of course tends to discard more coefficients at higher level.

For the formulation of RECOVER$^*$, it will be convenient to formulate the core step of the above scheme as an individual routine.

LEAFONE — $(\Lambda_j^\circ, \Gamma^{j+1}, M) \to [\partial\overline{(\Lambda_j^M)^\circ}, \Gamma^j]$

| |
|---|
| $\Lambda_j^{M\circ} := \Gamma_{j+1} \cup \Lambda^\circ{}_j$ |
| If $j > j_0$: <br> $\Gamma_j := \mathcal{P}(\Lambda_j^{M\circ})$ and $\partial\overline{(\Lambda_j^M)^\circ} := \mathcal{C}(\Gamma_j)\backslash\Gamma_{j+1}$ <br> Else: <br> $\partial\overline{(\Lambda_j^M)^\circ} := \Lambda_j^{M\circ}\backslash\Gamma_{j+1}$ |

Algorithm 4.4: Procedure LEAFONE: Core step of LEAF

RECOVER* then reads as follows:

RECOVER* — $(g, \Lambda, M, \varepsilon) \to [\overline{\mathbf{d}}(\mathcal{T})]$

| | |
|---|---|
| Find minimal $J \in \mathbb{N}$ such that $\mathcal{T}_j = \emptyset$, $j > J$ | |
| Perform LEAFONE$(\Lambda_J^\circ, \Gamma^{J+1}, M) \to [\partial\mathcal{T}_J, \Gamma^J]$ | |
| Determine $\mathcal{G}_{J+1}^- = \mathcal{G}_{J+1}$, $\mathbf{q}_{j+1}(\mathcal{G}_{J+1}^-)$ and set $\check{\mathbf{c}}_{J+1} := \mathbf{0}$. | |
| For $j = J, J-1, \ldots, j_0$ do | |
| | Execute LEAFONE$(\Lambda_{j-1}^\circ, \Gamma^j, M) \to (\partial\mathcal{T}_{j-1}, \Gamma^{j-1})$ |
| | Determine $\mathcal{G}_j^- = \mathcal{G}_j\backslash\mathcal{T}_j$ and $\mathbf{q}_j(\mathcal{G}_j^-)$ |
| | Set $\overline{\mathbf{c}}_{j+1} := \check{\mathbf{c}}_{j+1} + \mathbf{q}_{j+1}(\mathcal{G}_{j+1}^-) - \left(\mathbf{M}_j^{\tilde{\Phi}}\mathbf{q}_j(\mathcal{G}_j^-)\right)(\mathcal{G}_{j+1}^-)$ (4.3.1) |
| | Compute $\check{\mathbf{c}}_j := \mathbf{G}_{j,\tilde{\Phi}}\overline{\mathbf{c}}_{j+1}$, and $\mathbf{d}_j^{\mathrm{R}} := \mathbf{G}_{j,\tilde{\Theta}}\overline{\mathbf{c}}_{j+1}$ (4.3.2) |
| | Let (e.g.) $\varepsilon_j := \varepsilon/(J - j_0 + 1)$, do <br> BIN-THRESH$(\mathbf{d}_j^{\mathrm{R}}, \varepsilon_j) \to \overline{\mathbf{d}}_j$ |
| Set $\overline{\mathbf{c}}_{j_0} = \overline{\mathbf{d}}_{j_0-1} := \check{\mathbf{c}}_{j_0}$ | |

Algorithm 4.5: A recovery scheme for arbitrary index sets: Simultaneous index manipulation and computation of coefficients including clean-off

# Part III

# Applications and Numerical Results

**Overview:** Part III is concerned with applications and numerical tests for the schemes described earlier.
In Chapter 5, we first present the general layout of our numerical tests which is basically identical for all following chapters.
Chapter 6 is concerned with approximation in $L_2$. We present 'direct' applications of RECOVER, i.e., $g = u$, and applications to compositions $g = y \circ u$. We also demonstrate the significance of the correction step and a proper grading in quantitative terms.
Concerning the application of RECOVER for compositions, we briefly review the prediction strategy of [DSX00] in Chapter 7, and propose a heuristic strategy for an efficient application in practice.
Chapter 8 is devoted to applications in the case that the recovery takes place in a space with dual norm. There, we also try to shed some light on the question under which circumstances one can avoid sophisticated quadrature strategies, e.g. based on the least squares ansatz presented earlier.

# Chapter 5

# Layout of the Tests

The choice of the test cases presented in this part is not motivated by a special application, as here we are interested in a quantitative validation of the theoretical investigations in Part II and of our realization described in Part IV.

The main question we want to clarify are:

- What is the quantitative relation between approximation error and the error realized by the recovery scheme?
- How strong is the effect of the structural requirements on the index sets in quantitative terms?
- Does the realization de-facto exhibits a linear growth of CPU-time with respect to the size of the problem?

We will apply RECOVER to functions $g$, which may take the form $g = v$ or $g = y \circ v$, where $v$ is a finite wavelet expansion approximating some function $u$, called the *original function*. In the latter case of a composition, $y$ is a smooth nonlinear function. In order to be able to access true errors, the role of the original function $u$ will actually be played by some fixed highly accurate approximation $u_J$ of $u$, namely its (approximate) projection to $S(\Phi_J)$. This is done by first computing the scaling function coefficients by (highly accurate) quadrature on a uniform grid. The standard wavelet transform yields then he finite array of wavelet coefficients $\mathbf{u}_J$ of $u_J$ with maximal level $J$. In the case that $g = y \circ v$ we likewise compute a highly accurate approximation $g_{J'}$ to $y \circ u_J$ for some $J' \geq J$. The arrays $u_J$ and $g_{J'}$ will serve as fully known reference quantities. The test routine reads then as follows:

- For a sequence of decreasing parameters $\varepsilon \in \{\varepsilon_i, i = 1, 2, \ldots\}$ determine approximations $v = u^\varepsilon$ to $u_J$ by thresholding the wavelet coefficients in $\mathbf{u}_J$ yielding $\mathbf{u}^\varepsilon = \mathbf{u}(\Lambda^\varepsilon)$ to $\mathbf{u}_J$.
- In the case of $g = y \circ v$, construct the prediction set $\mathcal{T}^\varepsilon = \mathcal{T}(\Lambda^\varepsilon)$ for $g$ based on $\Lambda^\varepsilon$ according to the methods described in [DSX00, CDD03d]. For $g = v$, set $\mathcal{T}^\varepsilon = \Lambda^\varepsilon$.
- Execute RECOVER (including thresholding, cf. Sect. 4.3) and validate the results $\mathbf{g}^R$ ($\mathbf{g}_t^R$) by comparison with the reference $\mathbf{g}^\varepsilon := \mathbf{g}(\mathcal{T}^\varepsilon)$.

The results are mainly reported on in form of log-log plots showing the approximation error $\|\mathbf{g} - \mathbf{g}^A\|_{\ell_2(\mathcal{J})}, \mathbf{g}^A \in \{\mathbf{g}^\varepsilon, \mathbf{g}^R, \mathbf{g}_t^R\}$ versus the cardinality $\#\mathbf{g}^A$ of the corresponding approximant. The straight lines in the figures are obtained by a least square fit and the numbers $s$ displayed in the legends indicate the corresponding slopes. In some cases, we also include $\mathbf{g}^M := \mathbf{g}((\mathcal{T}^\varepsilon))^M$, i.e., the array of exact coefficients on the M-graded tree, in the plots to show the effect of the index manipulations.

We will apply this test routine in $L_2$ and in spaces with dual norms, yet to unify the representation we shall denote the arrays of wavelet coefficients always by $\mathbf{g}, \mathbf{g}(\mathcal{T})$, etc. where it is understood that in the $L_2$-case the diagonal matrix in (3.3.15) is just the identity.

All computations were carried out on an *AMD Athlon(Tm)XP 2000+* processor with 1.26GHz and 512MB of memory, and we always used biorthogonal wavelet bases from [CDF92], see Section 2.2.

# Chapter 6

# Approximation in $L_2$

This Chapter is devoted to numerical tests in one and two dimensions concerning the first of the two model situation described in Section 3.1, namely approximation in $L_2$. The layout of the tests follows Chapter 5, and in this Chapter, we will use exclusively Gauss-formulas as described in Section 4.1.2.

## 6.1 Direct Applications: $g = v$

### 6.1.1 Applying RECOVER

We will at first consider the following two one-dimensional test functions $g_1, g_2$, see Figure 6.1,

$$g_1 := \begin{cases} exp(x) & x \in [-1,1] \\ 0 & x \in (-2,2) \setminus (-1,1) \end{cases} \quad \text{and} \quad g_2 := exp(-100 * x^2). \tag{6.1.1}$$

Note that both functions are (numerically) zero for $|x| > 1$, $g_2$ is smooth except for jumps at $-1, 1$, while $g_2$ is smooth everywhere, but exhibits a strong gradient in the vicinity of the origin, cf. Figure 6.1.

We are interested in the approximation order for these functions and will therefore compute the $\mathcal{A}^s$-norm, using spline-wavelets of different order. Recall that

$$\|\mathbf{g}\|_{\mathcal{A}^s} = \sup_{N \geq 0} (N+1)^s \sigma_N(\mathbf{g}), s > 0, 1/\tau = s + 1/2,$$

and that $\|\mathbf{v}\|_{\mathcal{A}^s} \sim \|\mathbf{v}\|_{\ell_\tau^w(\mathcal{J})}$. Moreover, $\mathbf{v} \in \ell_\tau^w(\mathcal{J})$ if and only if $\sigma_N(\mathbf{v}) \leq CN^{-s}\|\mathbf{v}\|_{\ell_\tau^w(\mathcal{J})}$, $N \in I\!N$, cf. Definition 1.1.6 and Lemma 1.1.1.

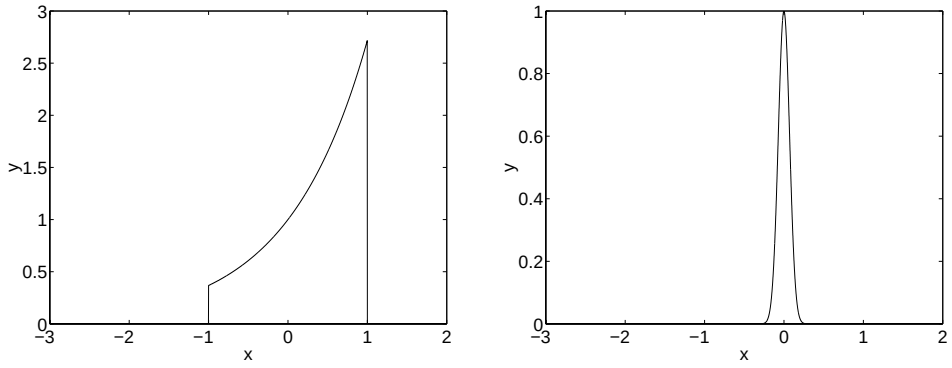



Figure 6.1: One dimensional test functions $g_1$ (left), $g_2$ (right)

To determine $\|g_i\|_{\mathcal{A}^s}, i = 1, 2$ numerically, we will compute the wavelet coefficients up to level 15. From this array, we will calculate $(N+1)^s\sigma_N(g_i)$ for different values of $N$ and $s$. For the wavelet system $N(2,2)$, we obtain for $g_1$:

| $N\backslash s$ | 0.0 | 1.0 | 2.0 | 2.2 | 3.0 |
|---|---|---|---|---|---|
| 250 | 0.64 | 1.38 | 8.58 | 10.60 | 73.50 |
| 1000 | 0.64 | 1.38 | 8.58 | 13.22 | 290.59 |
| 4100 | 0.64 | 1.38 | 8.62 | 14.52 | 1153.9 |
| 16400 | 0.64 | 1.38 | 8.64 | 15.1 | 4632.39 |
| 32800 | 0.64 | 1.38 | 8.71 | 15.8 | 9263.64 |

Table 6.1: Approximated $\mathcal{A}^s$-norm for a one-dimensional test example using $N(2,2)$

We observe, that the values stay (relatively) constant with growing $N$ for $s = 0, 1, 2$, but increase for $s = 2.2, 3.0$, which leads to the conclusion, that $g_1 \in \mathcal{A}^s, s \leq 2$. For a more detailed investigation, we computed $(N+1)^s\sigma_N(g_i)$ for larger variety of $N$ and $s$ and computed the best fit straight line. The following table shows the absolute value of its slope for $g_1$ and $g_1$, using different spline wavelet systems $N(m, \tilde{m})$. We highlight the first value $\geq 0.5$ in each column.

| s | $g_1$ | | $g_2$ | | |
|---|---|---|---|---|---|
| | N(2,2) | N(3,3) | N(2,2) | N(3,3) | N(44) |
| 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 1.5 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

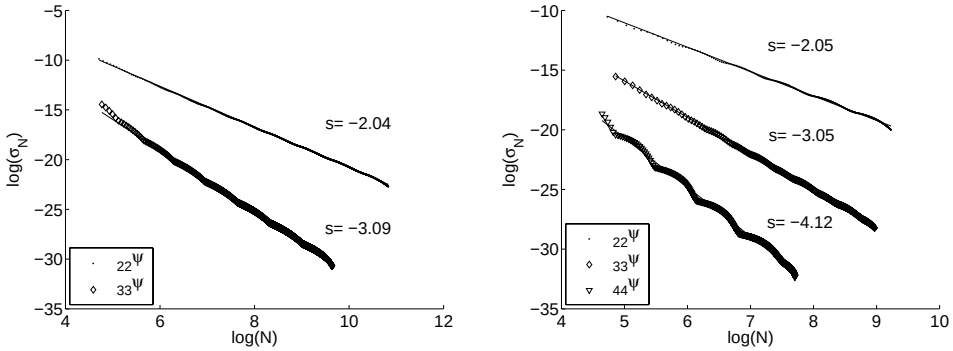
continued on next page

continued from previous page

| s | $g_1$ | | $g_2$ | | |
|---|---|---|---|---|---|
| | N(2,2) | N(3,3) | N(2,2) | N(3,3) | N(44) |
| 1.8 | 0.02 | 0.00 | 0.02 | 0.00 | 0.00 |
| 1.9 | 0.04 | 0.00 | 0.05 | 0.00 | 0.00 |
| 2.0 | 0.06 | 0.00 | 0.12 | 0.00 | 0.00 |
| 2.1 | 0.06 | 0.00 | 0.20 | 0.00 | 0.00 |
| 2.2 | 0.09 | 0.00 | **0.5** | 0.00 | 0.00 |
| 2.3 | **0.84** | 0.00 | 1.05 | 0.00 | 0.00 |
| 2.5 | 14.01 | 0.00 | 4.58 | 0.00 | 0.00 |
| 2.8 | 497.26 | 0.01 | 39.94 | 0.02 | 0.00 |
| 2.9 | 1549.2 | 0.02 | 81.49 | 0.04 | 0.00 |
| 3.0 | 46755.12 | 0.03 | 164.00 | 0.05 | 0.00 |
| 3.1 | | 0.01 | | 0.06 | 0.00 |
| 3.2 | | 0.12 | | 0.01 | 0.00 |
| 3.3 | | **0.61** | | **2.09** | 0.00 |
| 3.5 | | 6.59 | | 4.01 | 0.00 |
| 3.9 | | | | | 0.06 |
| 4.0 | 2.77 e8 | 1050 | 5.5e5 | 584.4 | 0.05 |
| 4.1 | | | | | 0.01 |
| 4.2 | | | | | **1.15** |
| 5.0 | 1.45e13 | 1.52e7 | 1.5e10 | 5.4e6 | 189.3 |

Table 6.2: Variation of the approximated $\mathcal{A}^s$-norm for two examples

The slopes of the log-log diagrams of $\sigma_N$ versus $N$ below confirm these rates in accordance with Lemma 1.1.1.




(a) Example $g_1$ (b) Example $g_2$

Figure 6.2: Error rates of best $N$-term approximation

These results let us expect, that the wavelet coefficients of $g_1$ with respect to $N(2,2)$ ($N(3,3)$) can be approximated with an approximated rate of 2.2

(3.1), while $g_2$ allows approximation rates of about $2.1, 3.2$ and $4.1$ using $N(2,2), N(3,3)$ and $N(4,4)$ respectively.

We will now turn to the recovery scheme. For $g = g_i, i = 1, 2$, we will choose $J = 15$ and the approximation interval $[-2, 2]$.

Concerning corresponding absolute CPU-times, the following numbers may serve as orientation: In order to compute the reference solutions, we have to determine 131073 coefficients $\langle g, \phi_{J,k} \rangle$, which takes about 0.23s using $N(2,2)$, $N(3,3)$ and a two-point quadrature formula. Decomposing these arrays consumes 0.54s ($N(2,2)$) and 0.66s ($N(3,3)$) of time for $g = g_1$. The time difference is due to the different sizes of the masks. Because of the fact that the index set for $g = g_2$ contains less non-zero entries, the decomposition takes 0.28s and 0.32s for $N(2,2)$, $N(3,3)$ respectively. The computation for $g = g_2$ and $N(4,4)$ is executed using a four-point quadrature formula, resulting in 1.04s to compute $\mathbf{q}_{15}$ and 0.39s for the decomposition procedure.

Observe, that the error rates in the log-log plots are in accordance with the expected rate, and that the error of the result of the recovery scheme $\mathbf{g}^R$ and the approximant obtained by thresholding $\mathbf{g}^\varepsilon$ are very similar also in absolute size. It should be noted, that the behavior of the $M$-graded version differs significantly from $\mathbf{g}^\varepsilon$ when $N$ is relatively small. This is due to the fact, that the completion and the grading process add indices to the approximant without taking into account their contribution to the error, i.e., regardless of the resulting error rate. With growing cardinality of the index set $\Lambda_\varepsilon$, the perturbing effect of the index manipulations grows weaker and finally $\mathbf{g}^M$ shows the same error behavior than $\mathbf{g}^\varepsilon$.

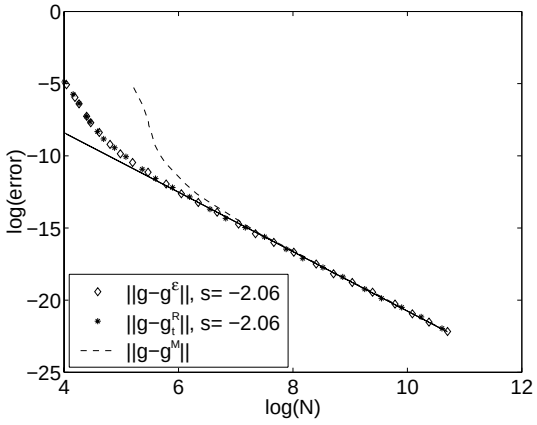



Figure 6.3: Error rate: Recovery scheme and a thresholding of $g_1$ using $N(2,2)$

Concerning the CPU time for the recovery scheme, the graph below shows a linear growth with regard to the number of coefficients.

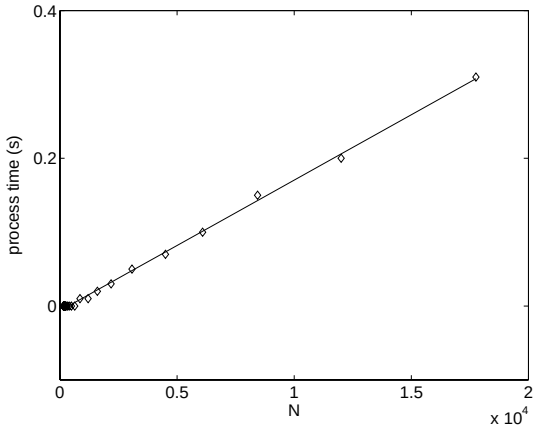
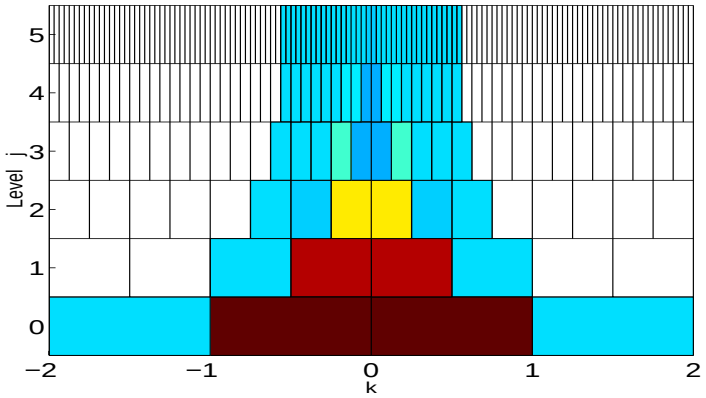
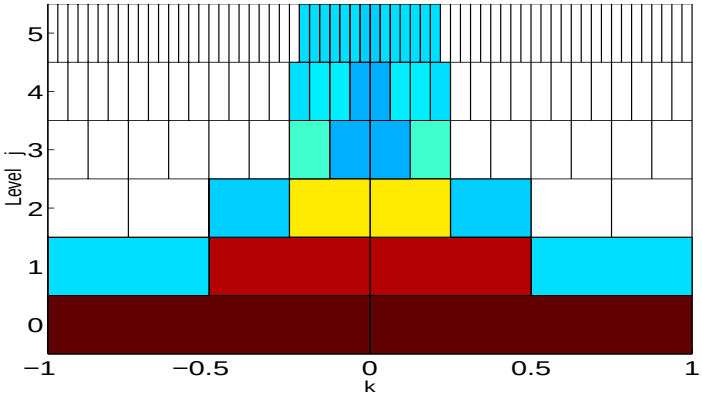



Figure 6.4: CPU time: Recovery scheme for $g_1$ using $N(2, 2)$

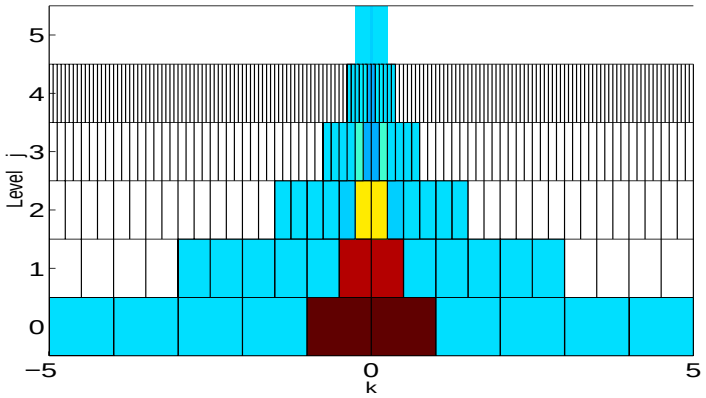
The diagrams below show the index sets corresponding to the major steps in a computation using with $g = g_2$ and $M(2, 2)$.



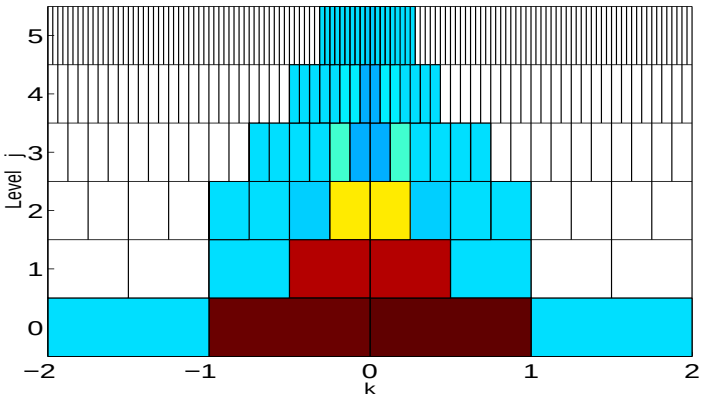

(a) decomp




(b) thresh



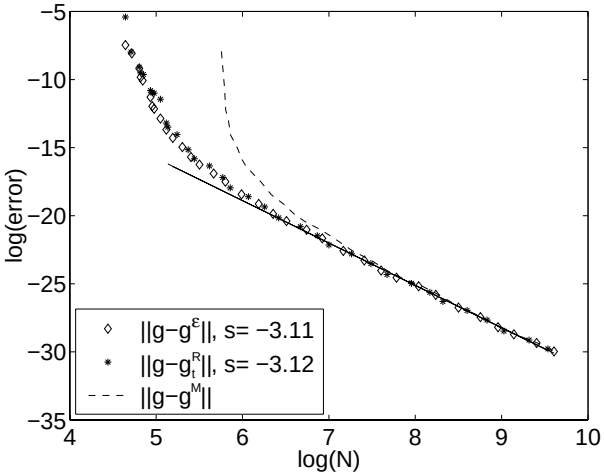

(c) grad



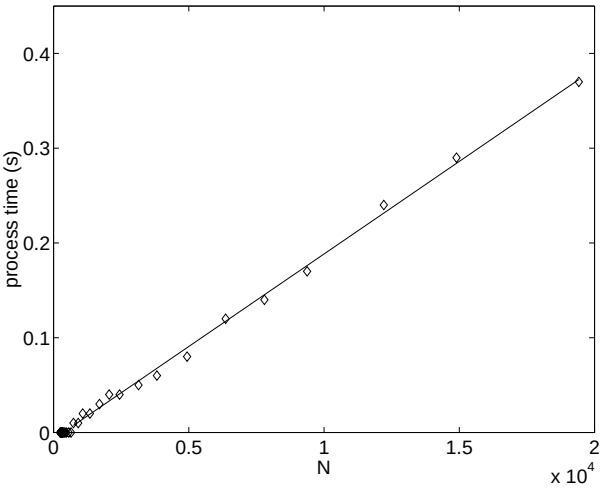

(d) rec

Figure 6.5: Index sets for recovery scheme for $g_2$ using $N(2,3)$

For the wavelet system $N(3,3)$, the tests show similar results, in particular, the error rate is again very close to the expected value and the CPU time grows linearly with the number of coefficients



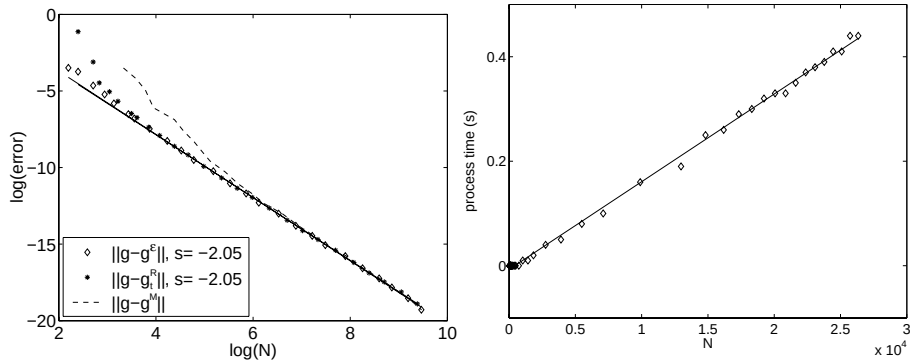

(a) Error rates



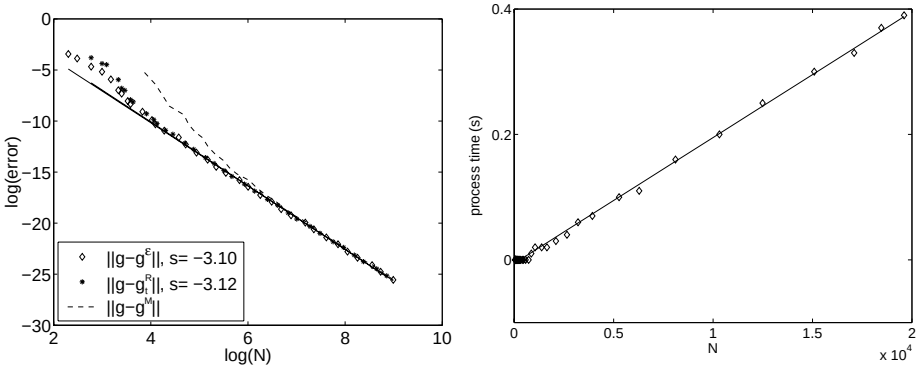

(b) CPU times for recovery

Figure 6.6: Recovery scheme for $g_1$ using $N(3,3)$

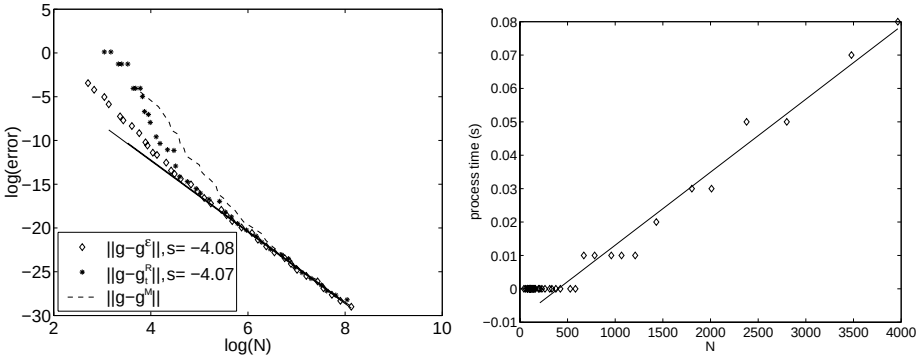
The following diagrams show the results for $g_2$. Again, the approximation interval was chosen to be $[-2,2]$.




(a) Error rates, $N(2,2)$ (b) CPU time for recovery, $N(2,2)$




(c) Error rates, $N(3,3)$ (d) CPU time for recovery, $N(3,3)$




(e) Error rates, $N(4,4)$ (f) CPU time for recovery, $N(4,4)$

Figure 6.7: Comparison: Recovery scheme vs. thresholded version of $g_2$

As a two dimensional test case, we will be concerned with $g_3$, which is

defined in analogy to $g_2$ as follows

$$g_3 := exp(-200 * (x^2 + y^2)). \tag{6.1.2}$$

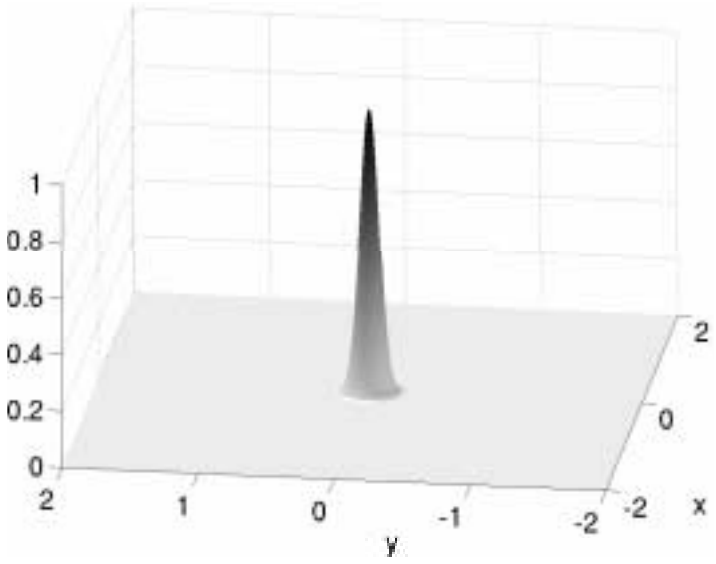



Figure 6.8: Two dimensional test functions $g_3$

Again, we are interested in the possible approximation rate of $g_3$. Using the same methods as in the one-dimensional case, the approximated $\mathcal{A}^s$ norm lead to the following results:

| s | N(2,2) | N(3,3) | N(4,4) |
|---|---|---|---|
| 0 | 0.00 | 0.00 | 0.00 |
| 0.6 | 0.00 | 0.00 | 0.00 |
| 0.7 | 0.00 | 0.00 | 0.00 |
| 0.8 | 0.05 | 0.00 | 0.00 |
| 0.9 | 0.11 | 0.00 | 0.00 |
| 1 | **0.55** | 0.00 | 0.00 |
| 1.1 | 1.98 | 0.00 | 0.00 |
| 1.2 | 6.57 | 0.00 | 0.00 |
| 1.3 | 21.91 | 0.02 | 0.00 |
| 1.4 | 73.31 | 0.09 | 0.00 |
| 1.5 | 246.26 | 0.30 | 0.00 |
| 1.6 | 830.51 | **1.02** | 0.01 |
| 1.8 | 9556.62 | 11.59 | 0.21 |

continued on next page

| continued from previous page | | | |
|---|---|---|---|
| s | N(2,2) | N(3,3) | N(4,4) |
| 2.0 | 1.16e4 | 135 | **0.68** |
| 2.2 | - | - | 34.71 |
| 3.0 | 2.99e8 | 3.5e8 | 5.3e6 |

Table 6.3: Variation of the approximated $\mathcal{A}^s$-norm for a two dimensional function

We therefore expect, that the array of wavelet coefficients of $g_3$ with respect to $N(2,2)$, $N(3,3)$ and $N(4,4)$ can be approximated with a rate of 1, 1.5 and 2.0 respectively. Again, these numbers match the slopes of the log-log diagrams of $\sigma_N$ versus $N$ below.

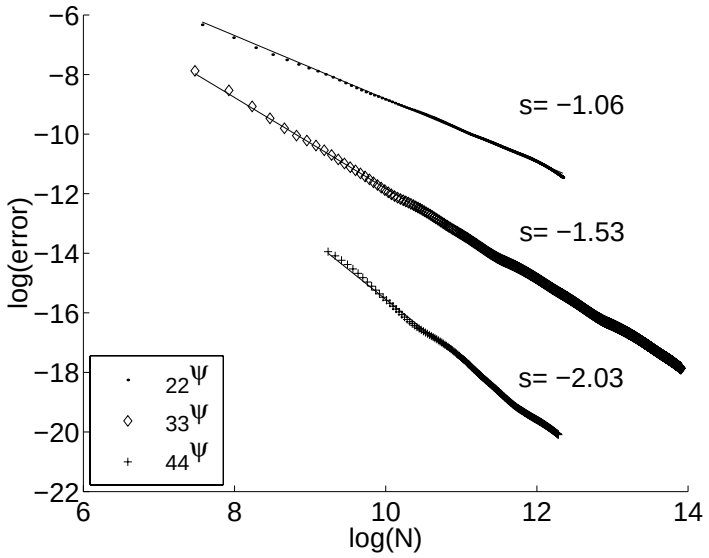


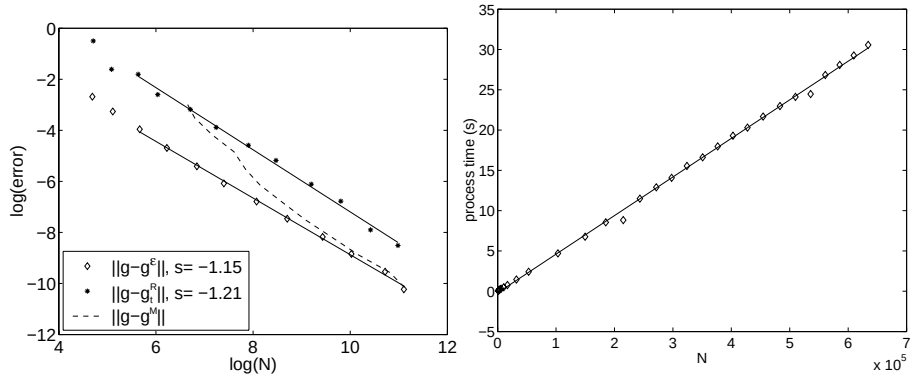

Figure 6.9: Error rates of best $N$-term approximation

The setup for the test of the recovery scheme in two dimensions is in principle identical to the one-dimensional case: For $g = g_3$, we will chose the level $J = 12$ and compute $\mathbf{q}_J \equiv \langle g, \Phi_J \rangle$ on the uniform grid for the interval $[-2, 2]^2$. This results in $8293^2 = 67125249$ coefficients to compute. For a two-point quadrature formula, this takes about 14min (849.05s), while for a formula with four points, as used for $N(4,4)$, this takes 25min (1518.35s). Decomposing the array of single scale coefficients on level $J$ takes about 3min (177.3s) in the case of $N(3,3)$, while 5min (292.2s) for $N(4,4)$.

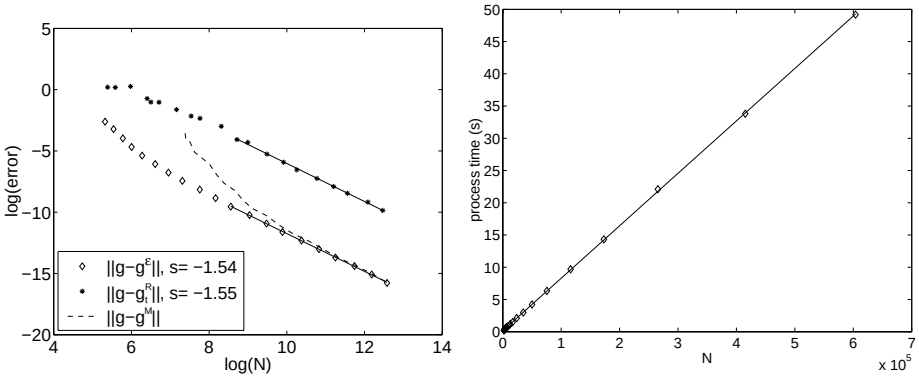
The log-log plots below show again the typical behavior already observed in the 1D case, yet the constants are somewhat larger, The CPU-time grows linearly with the number of coefficients. Keep in mind, that time differences smaller then $0.01s$ are below measurement accuracy.

On pages 113 and following, we show the major principal steps of the above test computations. The plots 6.11 and 6.12 show the result of the initial thresholding step, i.e., $\mathcal{T}^\varepsilon$. In Figures 6.13 and 6.14, we depict the 2-graded input set $(\mathcal{T}^\varepsilon)^M$ of the recovery scheme for the above test function. For clarity reasons, we show the case where the maximal level is chosen to be 5 as already here, the support cubes on higher level are hardly distinguishable on a picture fitting these pages. Note that due to gradedness the tree is quite wide. On pages 117 and 118 we depict the output of the recovery scheme including a thresholding with $\varepsilon = 2.5E - 4$. The index set now is much slimmer, but despite the thresholding very much tree-shaped.

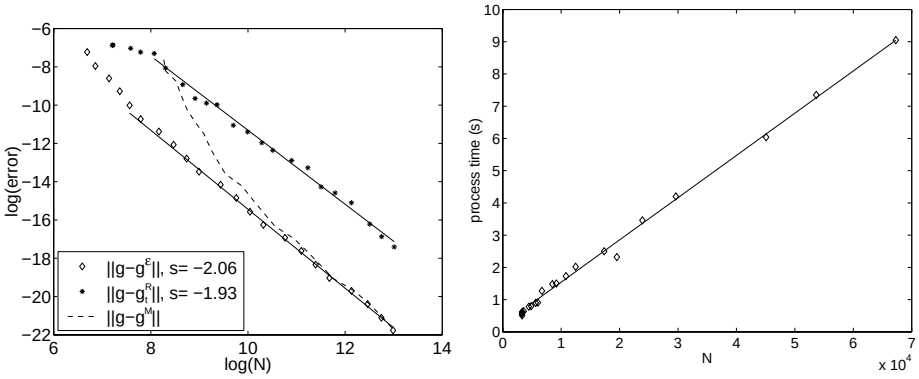
Note, that on each level a square indicates the position of the unit cube $[0, 1]^2$.




(a) Error rates, $N(2,2)$ (b) CPU time for recovery, $N(2,2)$

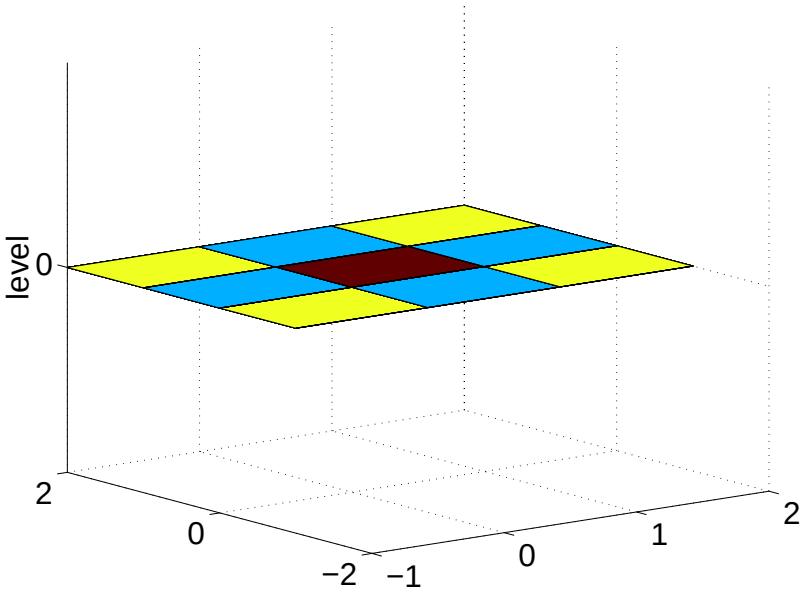



(c) Error rates, $N(3,3)$ (d) CPU time for recovery, $N(3,3)$

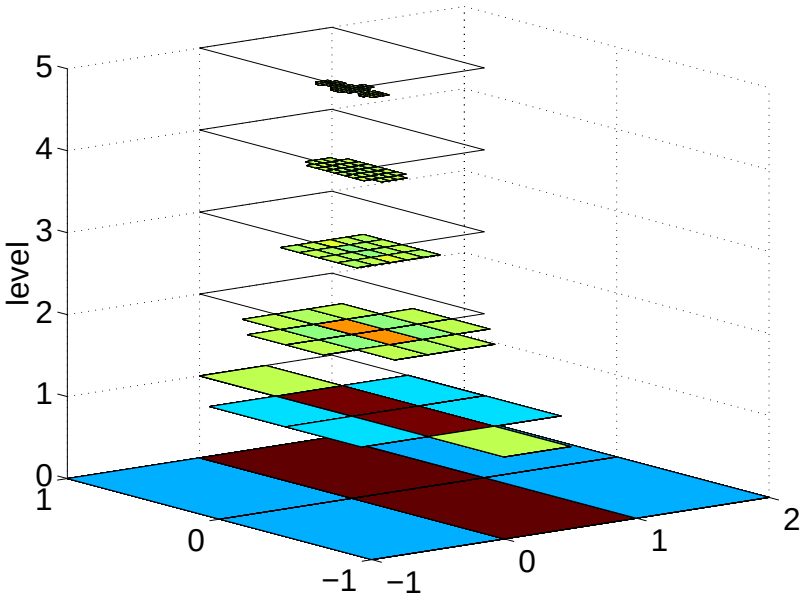



(e) Error rates, $N(4,4)$ (f) CPU time for recovery, $N(4,4)$

Figure 6.10: Comparison: Recovery scheme vs. thresholded version of $g_3$

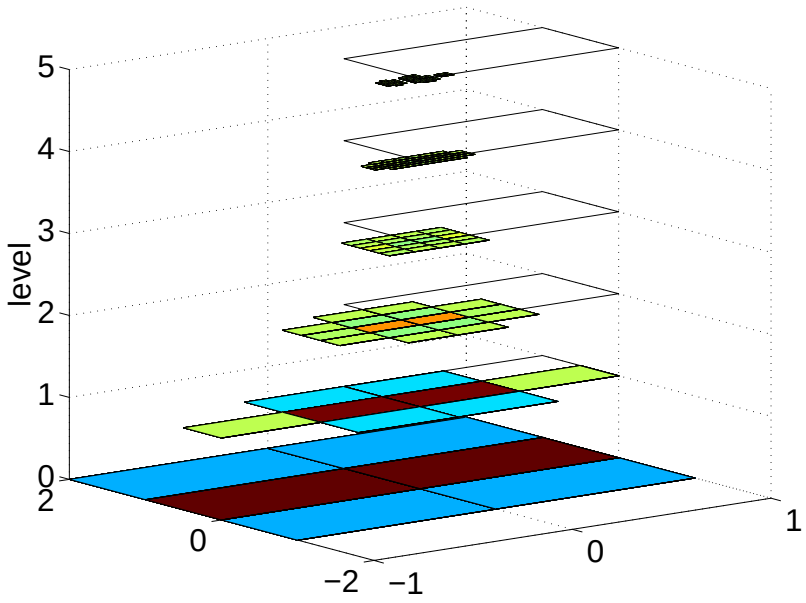



(a) Scaling functions, i.e. wavelets type $e = (00)$

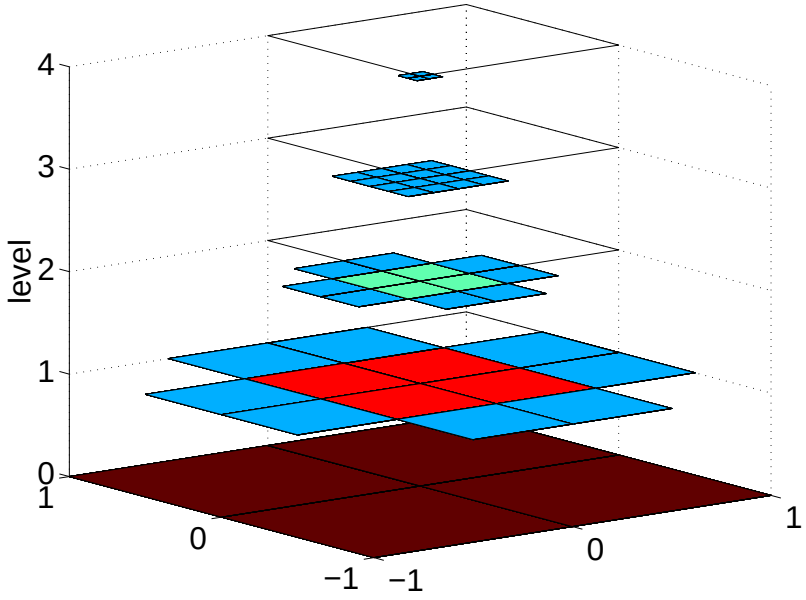



(b) Wavelets, type $e = (0, 1)$

Figure 6.11: $\mathcal{T}^e$ for $g_3$ according to $N(2, 2)$ in 2d

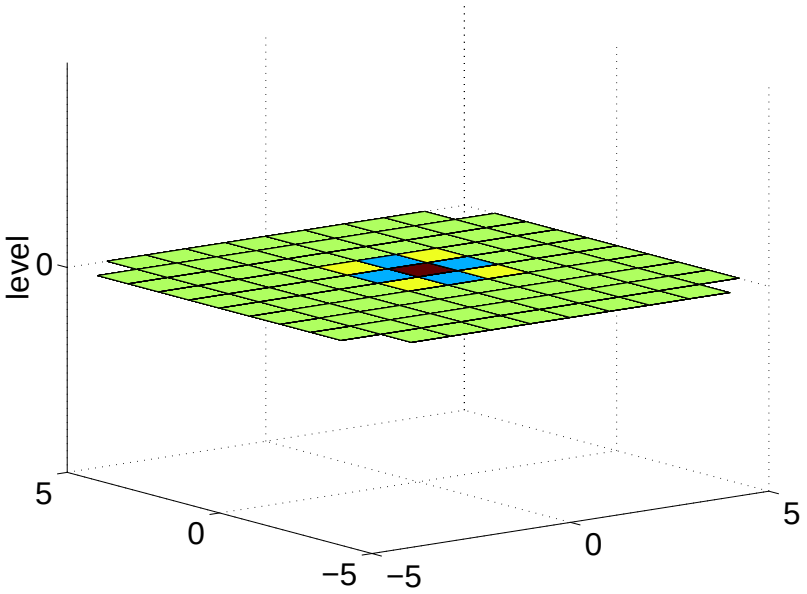



(a) Wavelets, type $e = (10)$

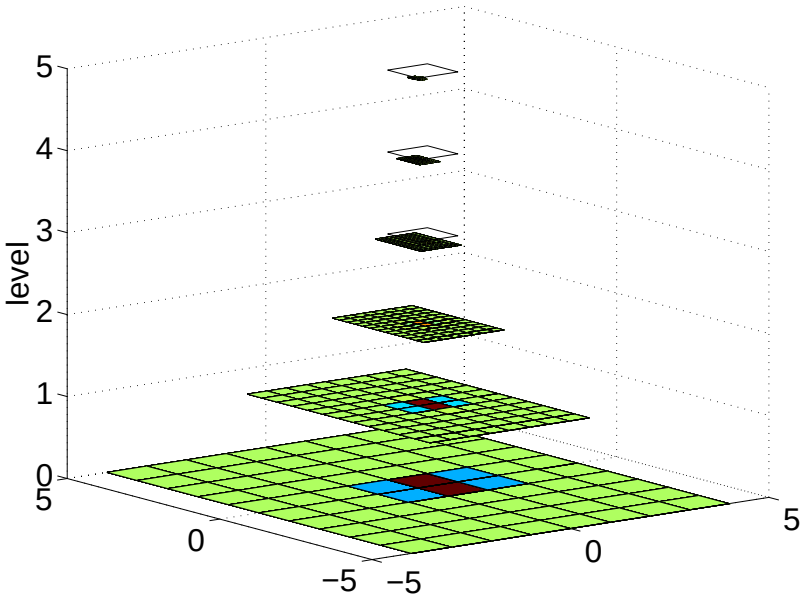



(b) Wavelets, type $e = (11)$

Figure 6.12: $\mathcal{T}^e$ for $g_3$ according to $N(2,2)$ in 2d (continued)

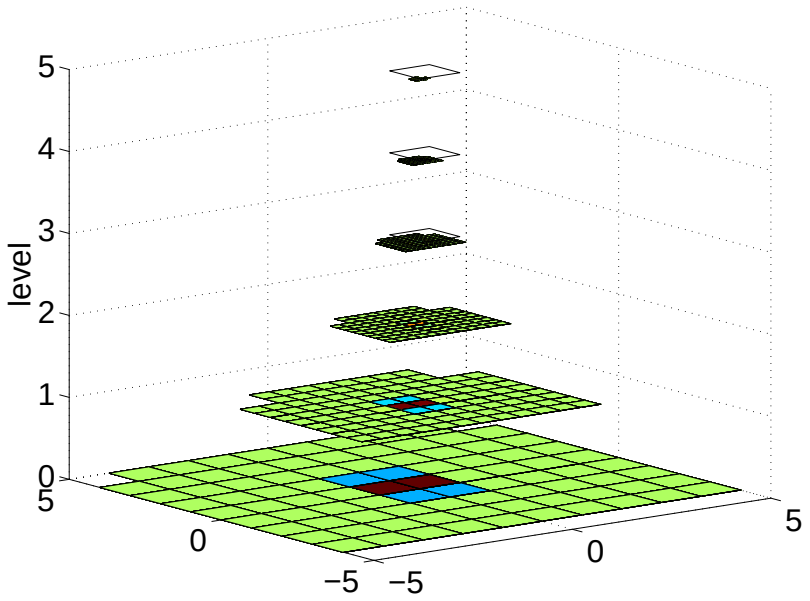



(a) Scaling functions, i.e. wavelets type $e = (00)$

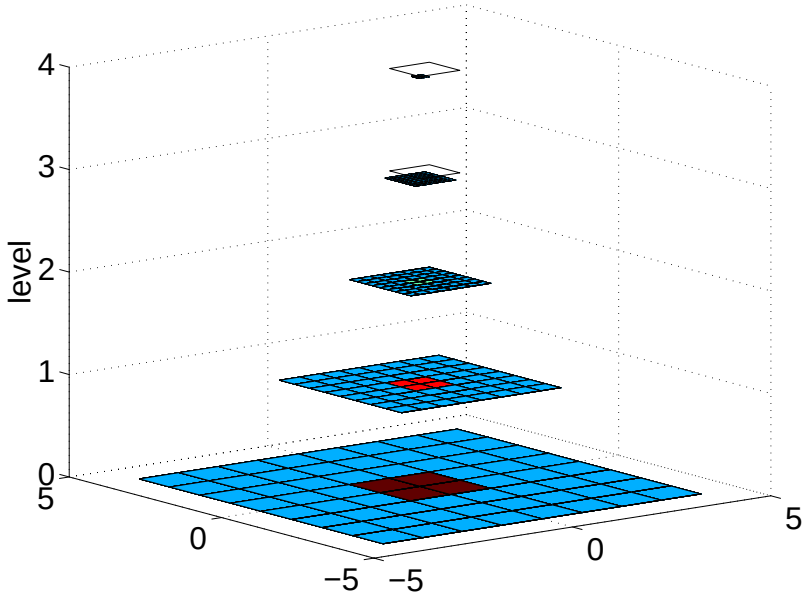



(b) Wavelets, type $e = (0, 1)$

Figure 6.13: $(\mathcal{T}^{\varepsilon})^M$ for $g_3$ according to $N(2, 2)$ in 2d

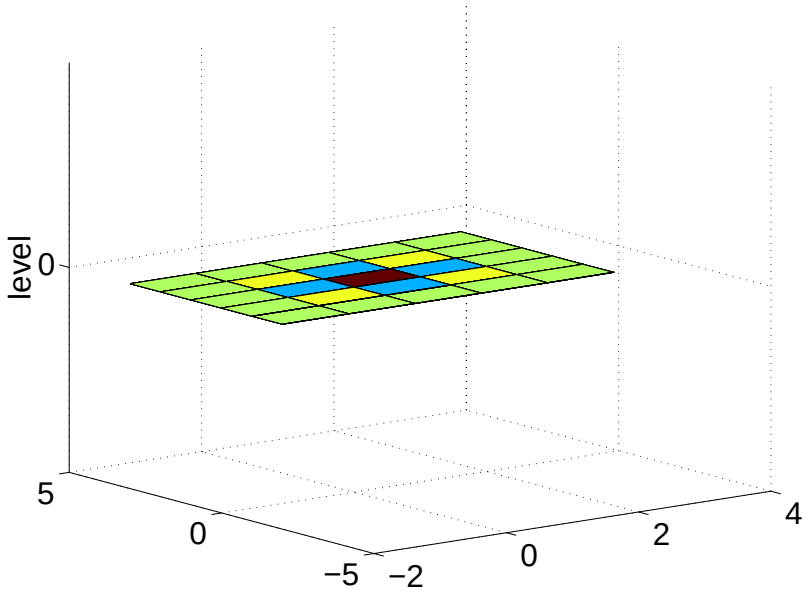



(a) Wavelets, type $e = (10)$

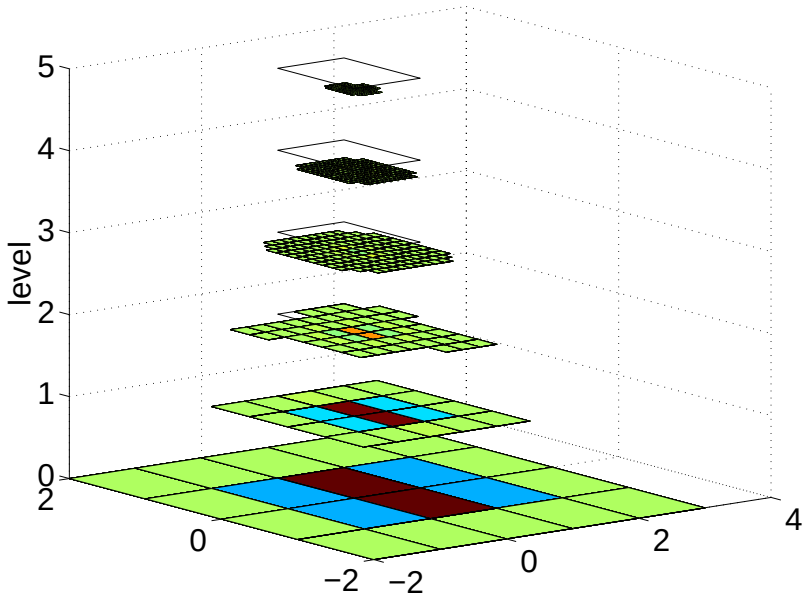



(b) Wavelets, type $e = (11)$

Figure 6.14: $(\mathcal{T}^e)^M$ for $g_3$ according to $N(2,2)$ in 2d (continued)

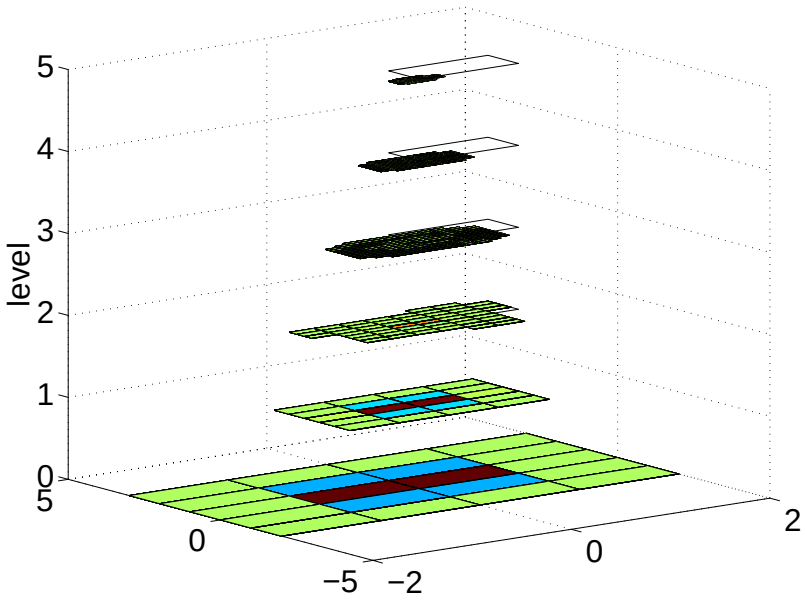



(a) Scaling functions, i.e. wavelets of type $e = (0, 0)$

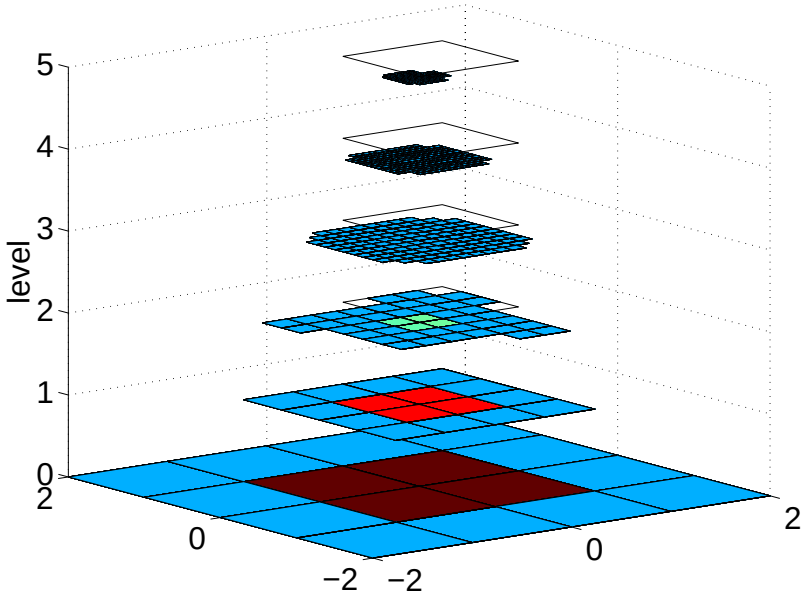



(b) Wavelets, type $e = (0, 1)$

Figure 6.15: Output of recovery scheme for $g_3$ ($N(2, 2)$) in 2d




(a) Wavelets, type $e = (1, 0)$




(b) Wavelets, type $e = (1, 1)$

Figure 6.16: Output of the recovery scheme for $g_3$ ($N(2, 2)$) in 2d (continued)

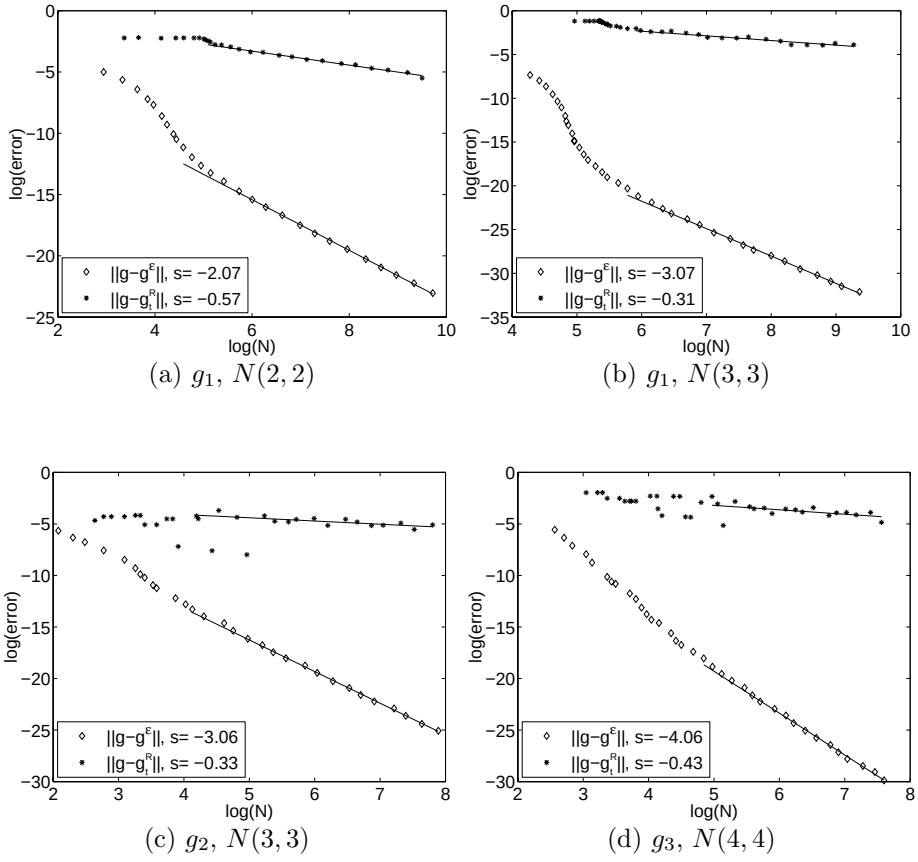
### 6.1.2 Failures: Significance of the Safety Region and the Correction Step

#### The Correction Step

In the following tests, we will again use the one dimensional function $g_1, g_2$ and proceed according to the usual test routine, described at the beginning of this chapter. Yet, for a qualitative classification of the significance of the correction step, we will omit it here. This means, *for test reasons* we replace the line (3.2.19) by

$$\bar{\mathbf{c}}_{j+1} := \check{\mathbf{c}}_{j+1} + \mathbf{q}_{j+1}(\mathcal{G}_{j+1}^{-})$$

in the recovery scheme RECOVER.




(a) $g_1$, $N(2,2)$ (b) $g_1$, $N(3,3)$

(c) $g_2$, $N(3,3)$ (d) $g_3$, $N(4,4)$

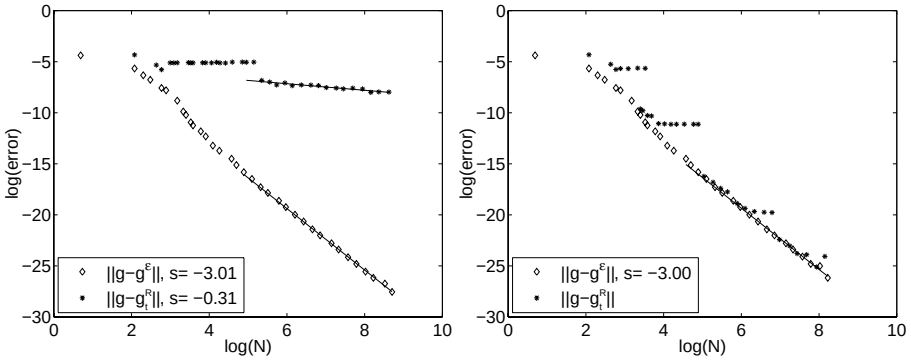
Figure 6.17: Computation without correction step

The graphs of Figure 6.17 show hardly an improvement of accuracy of

the recovery scheme when increasing the number of coefficients, i.e,. the introduced error by the lacking correction dominates the overall accuracy.

### Gradedness

The following example will demonstrate the effect when choosing the grading parameter $M$ too small, namely $< 2$, cf. Remark 3.4.1. This yields an index set that is no longer guaranteed to be well-graded.




(a) $g_1$, $N(3,3)$. $M = 0$ (b) $g_1$, $N(3,3)$. $M = 1$

Figure 6.18: Computation, inadequate grading parameter

Note that this will result in perturbations of the error reduction, In the case $M = 0$, the quality it totally unsatisfactorily, however the principle behavior in the case $M = 1$ is still according to the expected rate. Perturbations are still visible, however their influence seems to decrease with growing number of coefficients.

## 6.2 Approximating Compositions: $g = y \circ u$

In this chapter, we use the recovery scheme to compute terms of the form

$$\langle \psi_\lambda, y(u_\Lambda) \rangle, \ \lambda \in \Lambda' \subset \mathcal{J}, \tag{6.2.1}$$

where $y$ is a nonlinear function and $v_\Lambda$ is a given finite wavelet approximation with support $\Lambda$ to the so called original function $u$. This is exactly the context that motivated the development of the original recovery technique in [DSX00].

Terms of the form (6.2.1) may occur, when solving an operator equation in the variational form

$$\langle \nu, L(u) \rangle = \langle f, \nu \rangle, \quad \text{for all } \nu \in \Xi \subset \mathcal{H},$$

by a wavelet (Petrov-) Galerkin scheme. Consider, e.g., the problem

$$\begin{aligned} -\nabla u + y(u) &= f \qquad \text{in } \Omega, \\ u &= 0 \qquad \text{on } \partial\Omega, \end{aligned}$$

for some domain $\Omega$ and a nonlinear function $y(\cdot)$, e.g., $y(\cdot) = \cdot^2$.

In many applications, the nonlinear term looks actually different. For instance, in flow simulations one encounters expressions like $\frac{\partial}{\partial x_i} y(u(x))$. These terms however can be recast into a derivative-free form by using the fact, that derivatives of spline wavelets can be expressed as simple linear combinations of wavelets of lower order, see [LR92, Urb95]. Therefore, we will stick to the above model.

When we want to approximate a composition in the context of an adaptive wavelet scheme as described above, cf. Sections 3.1, 6.2, we have to cope with the following task.

**Task R3**: Given $u \in \mathcal{H}$, an approximant $u_\Lambda$ and $y : [a, b] \to I\!R$. Assuming that the approximant $u_\Lambda, , |\Lambda| < \infty$, satisfies

$$\|u - u_\Lambda\|_{\mathcal{H}} < \varepsilon, \tag{6.2.2}$$

for some $\varepsilon > 0$, find a finite $\hat{\Lambda} \supset \mathcal{J}$ and determine $\mathbf{d}_{\hat{\Lambda}}$ such that

$$\|\langle \Psi_{\mathcal{J}}, y(u_\Lambda) \rangle - \mathbf{d}_{\hat{\Lambda}}\|_{\ell_2} \lesssim \varepsilon, \tag{6.2.3}$$

spending at most $\mathcal{O}(|\Lambda|)$ work.

Note that the above evaluation task again consists of the typical two steps, i.e. prediction and recovery, however, in this special case, based on suitable knowledge on the nonlinear function $y$, one aims at predicting $\hat{\Lambda}$ from $\Lambda$, hence at predicting the consequences (in terms of local resolution) of the application of $y$.

However, deliberately postponing discussions on a suitable prediction strategy, we shall first treat the example

$$u_1 = \begin{cases} exp(x) & x \in (-1, 1) \\ 0 & x \in (-2, 2) \setminus (-1, 1), \end{cases} \qquad y(\cdot) = 4sin(2\cdot), \tag{6.2.4}$$

relying on the strategy from [DSX00]. Concerning the determination of prediction sets for compositions we refer to Section 7 and [DSX00, CDD03c].

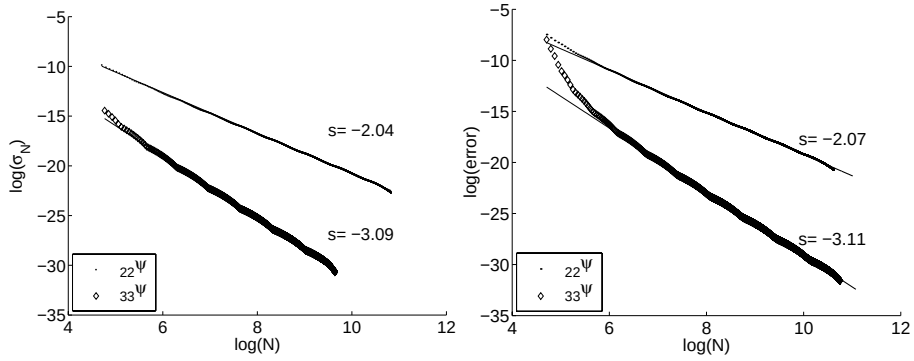
Hence, we will apply the test routine outlined in Chapter 5 to $g = y(u_1)$ assuming that an index set matching the above task has been found. Recall that $\Lambda^\varepsilon$ denotes the support of $\mathbf{u}^\varepsilon$, the approximation to $\mathbf{u}$, and that the prediction set is given by $\hat{\Lambda} = \mathcal{T}^\varepsilon = \mathcal{T}(\Lambda^\varepsilon)$. We again employ the Gauss-type formulas according to [BBDK01], cf. Section 4.1.2, for the computation of a uniform scaling function approximation to $u_1$ on level $J = 13$ (32769 coefficients). The composition is approximated on $(-2, 2)$ in this case on level $J' = 15$ (131073 coefficients). Due to the jumps of $u_1$ at $-1, 1$ we expect significant wavelet coefficients with spatial indices near these points in $\mathbf{u}_J$, and therefore also in $\Lambda^\varepsilon$ and $\mathcal{T}^\varepsilon$.

In the next table we exemplarily display cardinalities and errors, where we used the biorthogonal cardinal B-spline systems $N(\kappa, \tilde{\kappa})$. Concerning the cardinalities of the involved sets, note that as expected, the prediction set $\#\mathcal{T}^\varepsilon$ is significantly larger then the set of actually relevant coefficients and hence the set resulting from RECOVER (including top-to-bottom thresholding as described in Section 3.2.2) $\#\Lambda^R$, yet the difference decreases with growing size of $\#\Lambda$. Recall that, according to **Task R**, the reference accuracy is given by $\|g - g^\varepsilon\|$. where here and below in the table we write briefly $\|\cdot\| := \|\cdot\|_{\ell_2(\mathcal{J})}$.

| $\#\mathcal{T}^\varepsilon$ | 104 | 332 | 3664 | 13908 | 23256 | 27040 |
|---|---|---|---|---|---|---|
| $\|\mathbf{g} - \mathbf{g}^\varepsilon\|$ | 2.90e-3 | 7.41e-5 | 2.93e-7 | 4.40e-9 | 1.57e-11 | 2.60e-13 |
| $\|\mathbf{g} - \mathbf{g}^R\|$ | 2.90e-3 | 7.92e-5 | 3.21e-7 | 4.51e-9 | 1.60e-11 | 2.61e-13 |
| $\|\mathbf{g} - \mathbf{g}_t^R\|$ | 2.92e-3 | 7.92e-5 | 3.3e-7 | 4.56e-9 | 1.71e-11 | 2.61e-13 |
| $\#\Lambda^R$ | 25 | 112 | 1585 | 8771 | 15711 | 18527 |
| $\frac{\|\mathbf{g}-\mathbf{g}^R\|}{\|\mathbf{g}-\mathbf{g}^\varepsilon\|}$ | 1.0 | 1.06 | 1.1 | 1.02 | 1.1 | 1.0 |
| $\frac{\#\mathcal{T}^\varepsilon}{\#\Lambda^R}$ | 4.16 | 2.96 | 2.31 | 1.58 | 1.48 | 1.45 |

Table 6.4: Recovery of $g = 4\sin(2u_1)$: Parameter studies

The numerical determination of the approximation rates for $y \circ u_1$ using the same methods as before yield the following log-log plots comparing the near best N-term approximation for $u_1$ $y \circ u_1$.



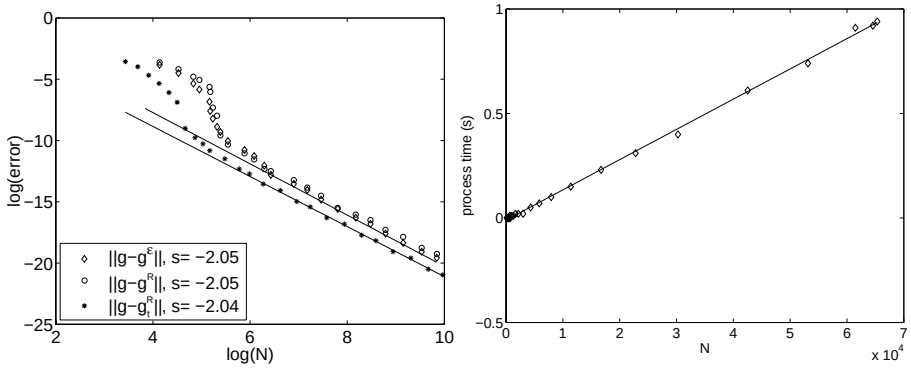

(a) $u_1$ (b) $y \circ u_1$

Figure 6.19: Error rates of best $N$-term approximation for $u_1$ and compositions

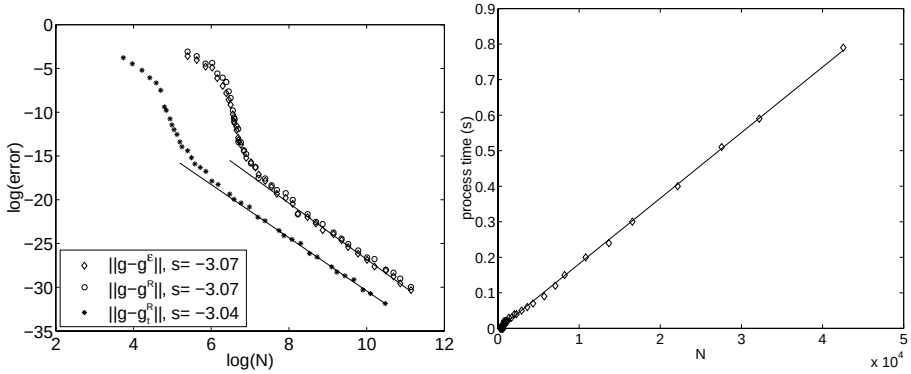
The computation of the (approximated) $\mathcal{A}^s$-norms confirm these results, hence the expected error rate for these tests, i.e. the theoretical slope in the log-log graph error vs. number of coefficients is $\kappa$, cf. Section 6.1.1, which is well reflected by our experiments as soon as the problem reaches a certain size. The graphs below show the log-log plots of the approximation $\mathbf{g}^\varepsilon = \mathbf{g}(\mathcal{T}^\varepsilon)$ to $\mathbf{g}$ in comparison with the result of the recovery of $\mathbf{g}^R$ (on the left) along with the computation time (on the right) for the bases according to $\kappa = 2, \tilde{\kappa} = 2$ and $\kappa = 3, \tilde{\kappa} = 3$ respectively.

For smaller problems, we observe a *tune-in behavior*, however the asymptotic (optimal order) regime is entered at a relatively early stage.

The plots (b), (d) in Figure 6.20 confirm that the CPU-time scales linearly with the size of the problem.




(a) Error rates , $N(2,2)$ (b) CPU time for recovery, $N(2,2)$



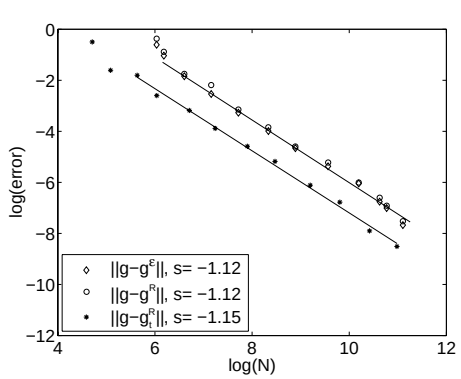

(c) Error rates , $N(3,3)$ (d) CPU time for recovery, $N(3,3)$

Figure 6.20: 1D-Tests of the recovery scheme, $u = u_1$ and $y \circ u = 4sin(2u_1)$

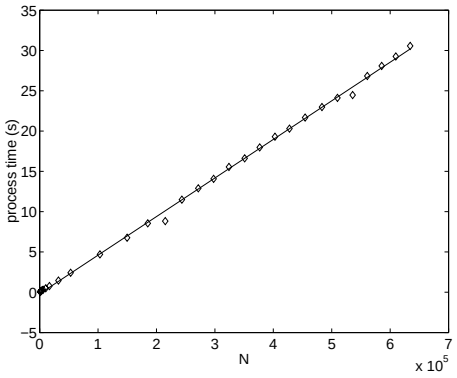
As a two dimensional test example, we choose $u_2$ defined by

$$u_2(x_1, x_2) := exp(-100 * (x_1^2 + x_2^2)), \tag{6.2.5}$$

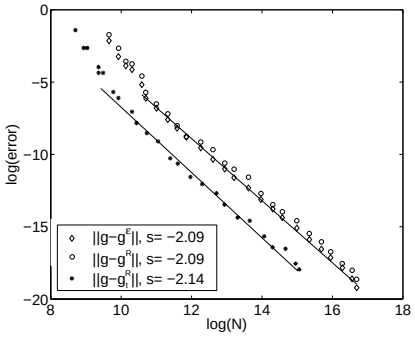
and retain the nonlinearity $y = 4sin(2\cdot)$. The results concerning RECOVER are similar to the one dimensional case including the constant $C$. The expected asymptotical error rate is $s = \kappa^{-d/2}$, which is matched fairly well. The CPU-times again scale linearly.



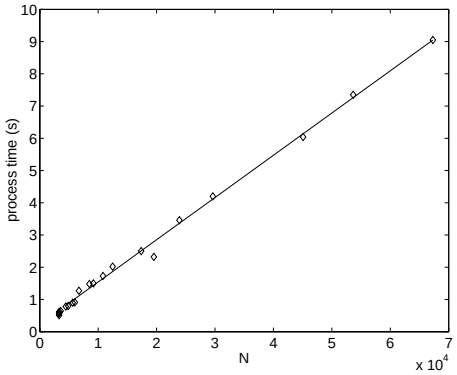

(a) Error rates, $N(2,2)$




(b) CPU time for recovery, $N(2,2)$




(c) Error rates, $N(4,4)$




(d) CPU time for recovery, $N(4,4)$

Figure 6.21: 2D-Tests of the recovery scheme; $u = u_2$, $y \circ u = 4sin(2u_2)$

# Chapter 7

# Prediction Sets for Compositions

We will now briefly present a heuristic strategy based on ideas from [DSX00] to predict a suitable index set for $y \circ u$. Recall that we want to find $\hat{\Lambda} \subset \mathcal{J}$ such that

$$\|\langle \Psi_{\mathcal{J}}, y(u_\Lambda)\rangle - \langle \Psi_{\hat{\Lambda}}, y(u_\Lambda)\rangle\|_{\ell_2} \lesssim \varepsilon, \tag{7.0.1}$$

where we assume for the rest of this chapter that $\mathbf{u}_\Lambda$ is $\varepsilon$-accurate, i.e. $\|\mathbf{u} - \mathbf{u}_\Lambda\|_{\ell_2(\mathcal{J})} \leq \varepsilon$. Recall from Section 1.1 that $\#\Lambda \lesssim \varepsilon^{-1/s}$ if $u \in \mathcal{A}^s$.

In [DSX00] and more recently also in [CDD03c], it is shown, that for a wide classes of nonlinearities, one can indeed predict $\hat{\Lambda} = \Lambda^{y(u)}$ from $\Lambda = \Lambda^u$ and ensure also that $\Lambda^{y(u)}$ is of the 'right' size, namely $\#\Lambda^{y(u)} \lesssim \varepsilon^{-1/s}$.

As we have seen earlier, these strategies are necessarily designed to cover a large class of operators charectarized only in a qualitative way, e.g. by smoothness or growth conditions. The corresponding asymptotic estimates are therefore expected to be pessimistic in qualitative terms. Thus in each concrete case a refined analysis is needed to improve the quantitative performance. The heuristics presented below aims at minimizing the overhead in the case of the prediction proposed in [DSX00].

## 7.1 Predicting Approximation Spaces for Compositions

Assume, that we are given an $\varepsilon$-accurate set $\Lambda^u$ for the function/approximant pairing $(u, u_{\Lambda^u})$. We shall briefly summerize now the method [DSX00] to predict an $\varepsilon$-accurate set $\Lambda^g$ for $(g = y(u_{\Lambda^u}), g_{\Lambda^g})$,

Suppose that the multiresolution spaces $S_j, \tilde{S}_j$ spanned by all basis functions from $\Psi$, respectively $\tilde{\Psi}$ up to level $j-1$ provide approximation orders $m$, $\tilde{m}$, respectively. Recall from Chapter 2 that this means

$$\inf_{v_j \in S_j} \|u - v_j\|_{L_2} \lesssim 2^{-mj} \|u\|_{H^m},$$

and analogously for the spaces $\tilde{S}_j$. In terms of the degrees of freedom $N$ the decay rate $2^{-mj}$ can be expressed as $N^{-m/d}$.

In fact, it has been shown in [DSX00] that when $u \in B^s_q(L_p) \cap L_\infty$, $s > 0, p, q \in (0, \infty]$ with $\|u\|_\infty \leq \sigma < \infty$ the composition $y(u)$ is still in $B^s_q(L_p)$ if $y \in C^r([-\sigma, \sigma])$, $r > s$ and $y(0) = 0$. This makes use of a result, see e.g. [RS96], stating that for $s > 0$ with

$$\frac{1}{p} \leq \frac{s}{d} + 1$$

we have that

$$\|y \circ v\|_{B^s_q(L_p)} \leq C(y, \|v\|_{L_\infty}) \, \|v\|_{B^s_q(L_p)}, \quad v \in B^s_q(L_p) \cap L_\infty. \tag{7.1.1}$$

The constant $C(y, \|v\|_{L_\infty})$ is of the form

$$C(y, \|v\|_{L_\infty}) = c(\|v\|_{L_\infty}) \varrho. \tag{7.1.2}$$

Here $c(\|v\|_{L_\infty})$ is another constant and $\varrho > 0$ satisfies

$$\|y^{(l)}\|_{L_\infty} \leq \varrho, \quad l = 1, 2, \ldots, r, \tag{7.1.3}$$

see [RS96].

If $\Lambda$ is an $M$-graded $\varepsilon$-accurate set for $(u, u_{\Lambda^u})$ it can be shown, see [DSX00], that an $M$-graded $\varepsilon$-accurate set $\Lambda^{y(u)}$ for $(y(u), \tilde{Q}_{\Lambda^{y(u)}})$ can be obtained by a fixed number of dyadic subdivisions of $\Lambda^u$.

The depth of this subdivision depends on the ratio $\|y(u)\|_{B^s_q(L_p)} / \|u\|_{B^s_q(L_p)}$. Hence $\#\Lambda^{y(u)} \lesssim \#\Lambda^u$ uniformly in $\varepsilon$. In fact, $\Lambda^{y(u)}$ is actually $\varepsilon$-accurate for $(y(u), P_{\Lambda^{y(u)}} y(u))$, where $P_{\Lambda^{y(u)}}$ is the recovery scheme from Chapter 3. Of course, to make sure that local errors of approximations to $y(u)$ can be comparable to those of $u$, the accuracy order of the dual multiresolution spaces $\tilde{S}_j$ should be at least as large as that for the primal spaces $S_j$, i.e., $\tilde{m} \geq m$ which we will always assume henceforth.

In applications, $y$ is often very regular, say $y \in C^m$. This may suggest to globally subdivide each element of $\Lambda^u$ in order to compensate the action of $y$ concerning the Besov norm, cf. [DSX00]. Note, however, that it may be

the case that this action is locally very different, hence one should consider a *local* subdivision depth

$$R_\lambda := \frac{|y(v)|_{B_q^m(L_p(\tilde{\square}_{\lambda^\circ}))}}{\|v\|_{B_q^m(L_p(\tilde{\square}_{\lambda^\circ}))}},$$

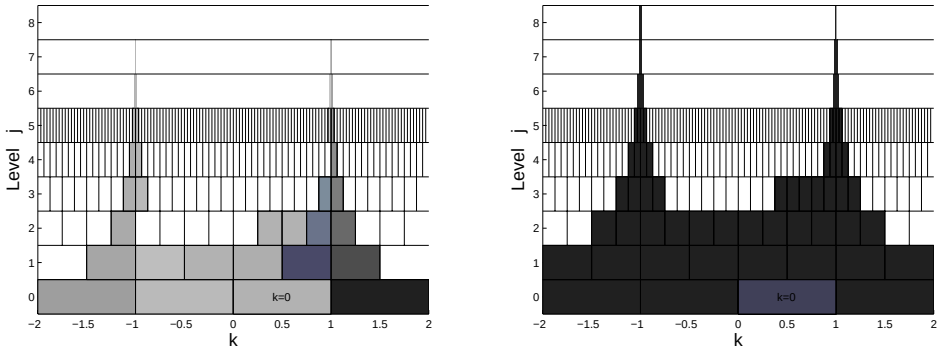
where $\tilde{\square}_{\lambda^\circ}$ is a suitable superset of the support cube $\square_{\lambda^\circ}$, e.g., the smallest cube such that

$$\tilde{\square}_{\lambda^\circ} \supseteq \bigcup_{\mu:\operatorname{supp}\tilde{\phi}_\mu \cap \square_{\lambda^\circ} \neq \emptyset} \operatorname{supp}\tilde{\phi}_\mu.$$

The feasibility of this more economic subdivision strategy depends on being provided with the knowledge of $R_\lambda$. Due to (7.1.2), one can at least derive an indicator for $R_\lambda$ and hence for a suitable local refinement depth on $\tilde{\square}_{\lambda^\circ}$, cf. (7.1.3), by computing numbers $\varrho_{\lambda^\circ}$ satisfying

$$\max_{l=1,\ldots,s} \|y^{(l)}\|_{L_\infty(\tilde{\square}_{\lambda^\circ})} \leq \varrho_{\lambda^\circ}.$$

Examples of sets $\Lambda^u, u = u_1$ and $\Lambda^g, g = u_1^2$ (represented in terms of support cubes) according to a uniform subdivision $r = 1$, cf. [DSX00] are displayed in Figure 7.1. Note that again the different gray-shades in the visualization of $\Lambda^u$ correspond to the size of the associated coefficient value, while in the prediction set $\Lambda^g$, no values are associated to the support cube.




(a) $\Lambda^\varepsilon$: Index set of approximant to $u$ (b) $\mathcal{T}^\varepsilon$: The prediction set for $g$

Figure 7.1: Index sets represented by support cubes

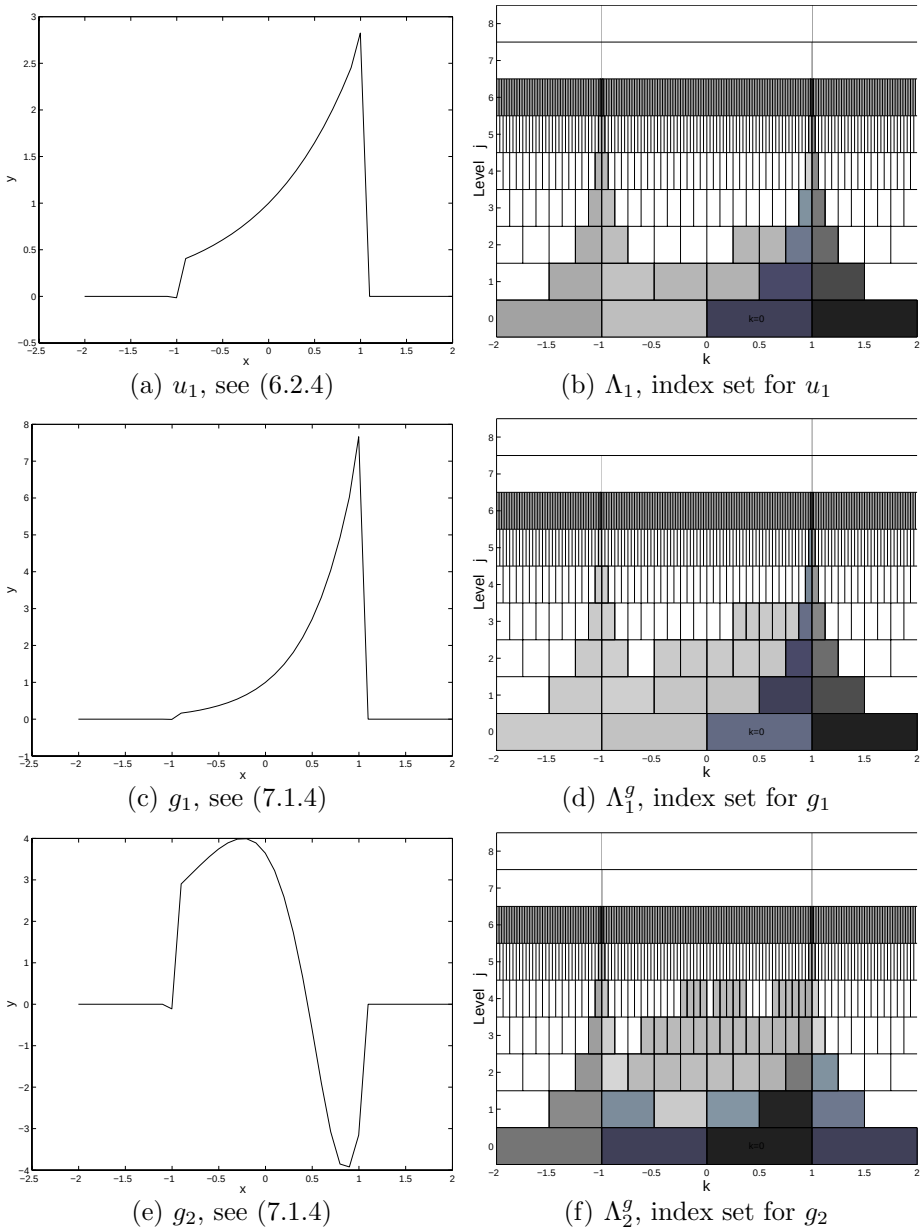
We will now illustrate the effect of the action of two different types of non-linearities on the shape of the index sets in the Figure 7.2 on the following

pages. The function under consideration is again $u_1$, see (6.2.4) In the following tests, we compare the influence of the nonlinear functions $y_1 = (\cdot)^2$ and $y_2 = 4\sin(2\cdot)$ on $u_1$. We use the abbreviations

$$g_1 := (u_1)^2, \quad \text{and} \quad g_2 := 4\sin(2u_1). \tag{7.1.4}$$

On the interval $[-2, 2]$, these functions are sampled on level 15 using the ${}_{2,2}\phi$ scaling functions. This single scale representation is then decomposed and thresholded. For illustration reasons, we chose relatively low resolution, i.e., a large thresholding parameter $\varepsilon = 0.01$ in order to end up with a small index set giving a clear picture.

The pictures on the following pages show index sets $\Lambda_1 := \Lambda(u_1, \varepsilon)$ and $\Lambda_i^g, i = 1, 2$, as well as the corresponding approximations to $u_1, g_i, i = 1, 2$. The different gray values in the index plots refer to different sizes of coefficients, the darker, the larger the value. The graphs on the left-hand side are those of the wavelet expansions corresponding to the index sets displayed on the right.




(a) $u_1$, see (6.2.4) (b) $\Lambda_1$, index set for $u_1$

(c) $g_1$, see (7.1.4) (d) $\Lambda_1^g$, index set for $g_1$

(e) $g_2$, see (7.1.4) (f) $\Lambda_2^g$, index set for $g_2$

Figure 7.2: Qualitative change in the shape of the index sets for different $y$

Note that $u_1$ has jumps at $\pm 1$ and that $\Lambda_1$ has high level (level 8) coefficients at these locations. In $\Lambda_1^g$, we observe in principle the same structure. In the last example $g_2$ however, we observe a steep gradient in the center of the interval not occurring in $u_1$. This is reflected in the index set $\Lambda_2^g$ by the fact, that coefficients start to pile up around $x = 0$. Qualitatively speaking the structure of $g_2$ is comparable to $g_1$, i.e., $(\cdot)^2$ is structure preserving, while

$4sin(2\cdot)$ introduces significant structural changes, namely a 'new' cone of coefficients growing with more levels involved.

The following heuristic approach now follows the intuition of 'linking' the refinement depth to the local size of the gradient. Motivated by the chain rule, $[y(g)]' = y'(g)g'$, and (7.1.1) the idea is to refine 'proportionally' to the absolute size of $y'(g)$. One possible heuristic prediction rule derived from that ansatz is the following.

H-SUBDIV — $(\mathbf{u}(\mathcal{T}), y, M_{\max}) \to [\partial\mathcal{T}^y]$

| |
|---|
| Let $\mathcal{T}^y := \emptyset$ |
| For each $\lambda^\circ \in \partial\mathcal{T}$ do |
| Determine $r_\lambda := \dfrac{\|y'(g)\|_{L_\infty(\square_\lambda)}}{\|y'(g)\|_{L_\infty(\Omega)}}$ |
| $M_{\lambda^\circ} := \lceil r_\lambda M_{\max} \rceil$ |
| $\mathcal{T}^y \to \mathcal{T}^y \cup \{(\lvert\lambda^\circ\rvert + M_{\lambda^\circ}, \mathbf{k}) : \square_{(\lvert\lambda^\circ\rvert + M_{\lambda^\circ}, \mathbf{k})} \subset \square_{\lambda^\circ}\}$ |

Algorithm 7.1: Procedure H-SUBDIV: Heuristic prediction strategy

Note that H-SUBDIV only constructs the leaves of $\mathcal{T}^y$, which however determine the entire tree.

Before we turn to numerical examples illustrating this strategy, the following remark is in order: In this thesis, we clearly separate the two actions *prediction* and *recovery* of an adaptive evaluation scheme as two conceptually different steps. This leads to the algorithmic paradigm proposed so far, namely that we consider the prediction step as a black-box tool to be applied as a *pre-processing step* to RECOVER.

However, if the prediction technique is known and of suitable type like the techniques described above, it can be advantageous to merge the prediction step with the recovery RECOVER: If we construct $\mathcal{T}(\Lambda^\varepsilon)$ according to the above technique, we have to sweep (at least) the leaves of $\Lambda^\varepsilon$ and to store the descendants corresponding to the refinement strategy as a whole. Yet one can combine the index manipulations and the prediction techniques to perform both steps in a single sweep and during the level-iteration of RECOVER. The above refinement strategy, e.g., can be easily merged with

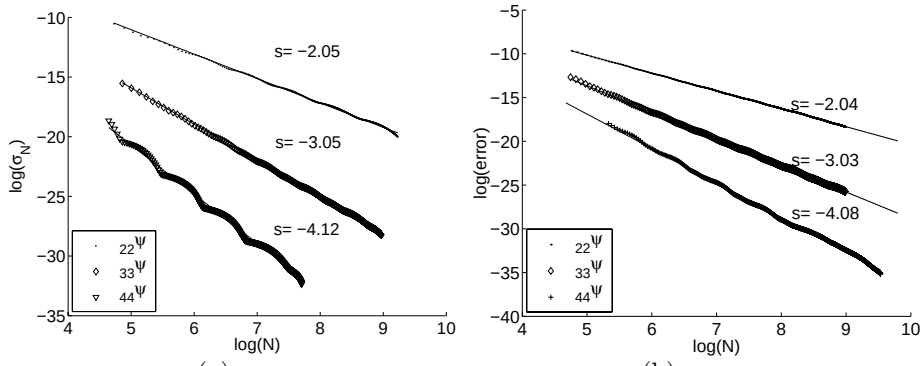
the COMPLETION routine, cf. (4.2.2), and hence - by a corresponding modification of LEAFONE cf. (4.2.2) - with RECOVER*. The benefit of this is again that the assembling and storage of the entire, possibly pessimistic prediction set is avoided.

## 7.2 Numerical Results

The principal way of carrying out the following test is the same as in Chapter 5 and we will again use the functions $u_1, u_2$:

$$u_1 = \begin{cases} exp(x) & x \in [-1,1] \\ 0 & (-2,2)\backslash(-1,1) \end{cases} \quad \text{and} \quad u_2 = exp(-100 * x^2). \tag{7.2.1}$$

As nonlinearity, we will first consider $y(\cdot) = \cdot^2$. To numerically determine the approximation rates for $y \circ u_1, y \circ u_2$, we used the same methods as before, cf. Chapter 6. The log-log plots below compare the (numerically determined) near best N-term approximation for $u_2$ with $y \circ u_2$.




(a) $u_2$ (b) $y \circ u_2$

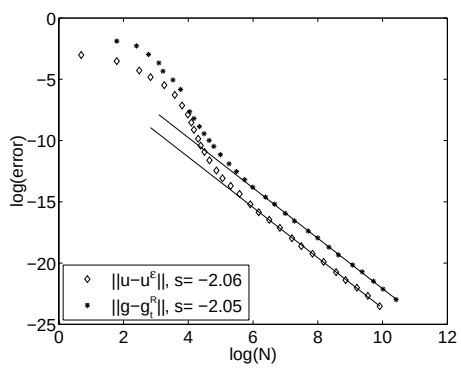
Figure 7.3: Error rates of best $N$-term approximation for $u_2$ and compositions

As expected, we observe the same error rates, i.e., the same slopes of the best fit straight line, for the original function as well as for the composition. Note also, that given a specific number of coefficients, the corresponding error for the composition is somewhat larger. Both observations are in accordance with (7.1.1). The computations of the (approximated) $\mathcal{A}^s$-norms confirm these results analogously to Table 6.2.

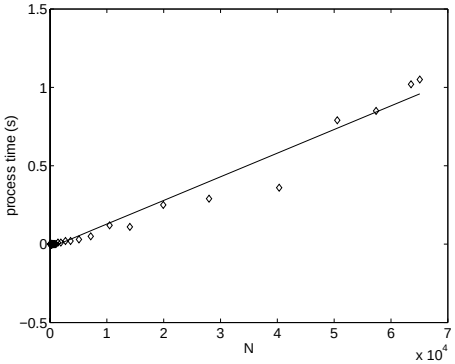
### Classical Prediction

Using a uniform scaling function representation on level 13 (32769 coefficients) we compute an approximation to $u_i, i = 1, 2$ on the interval $[-2, 2]$ serving as reference. In the same fashion, we determine a reference for $g = y \circ u$ on level 15 (131073 coefficients). After having determined $\Lambda^\varepsilon(u)$,the prediction set $\mathcal{T}(g)$ for $g$ is obtained by an $r$-fold subdivision of the support cubes in $(\Lambda^\varepsilon(u))^\circ$.

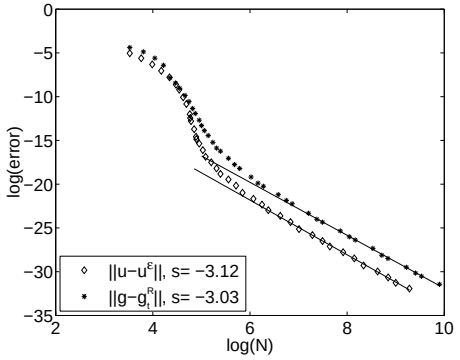
The following Figure 7.4 shows the results of the tests with $u = u_1$. Again, we displayed the log-log plots of the original approximation $\mathbf{u}^\varepsilon$ to $\mathbf{u}$ in comparison with the result of the recovery $\mathbf{g}_t^R$ of $g = y \circ u$ along with the time for the computation.



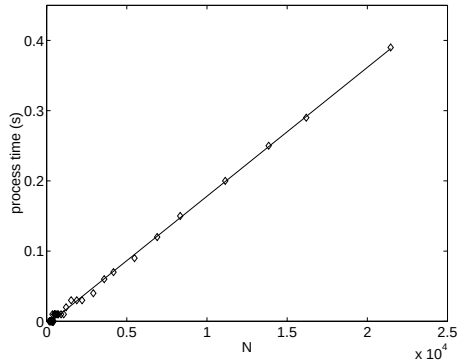

(a) Error rates using $N(2, 2)$




(b) CPU time for recovery with $N(2, 2)$




(c) Error rates using $N(3, 3)$




(d) CPU time for recovery with $N(3, 3)$

Figure 7.4: Test of the recovery scheme with $u = u_1$ and $g = u_1^2$

We observe, that the error rates match the expectations. Moreover, also here, the computation time grows linearly with the number of coefficients.

The following table lists the most interesting numbers for different sizes of $\Lambda^\varepsilon(u)$. The numbers in the first table correspond to a computation using $N(2,2)$ and $y \circ u = u_1^2$. Recall that $\Lambda^g$ is the prediction set for $g = y \circ u$ obtained by the strategy in [DSX00]. It already includes all structural requirements.

| $\#\Lambda^\varepsilon$ | 23 | 119 | 1783 | 7667 | 11570 | 13469 |
|---|---|---|---|---|---|---|
| $\|\mathbf{u} - \mathbf{u}_{\Lambda^\varepsilon}\|_{\ell_2}$ | 2.9e-3 | 7.4e-5 | 2.8e-7 | 4.2e-9 | 1.7e-11 | 2.6e-13 |
| $\#\Lambda^g$ | 104 | 332 | 3664 | 16108 | 23256 | 27040 |
| $\#\Lambda_t^R$ | 25 | 112 | 1585 | 8771 | 15711 | 18527 |
| $\|\mathbf{g} - \mathbf{g}_t^R\|$ | 2.9e-3 | 7.9e-5 | 3.2e-7 | 4.5e-9 | 1.6e-11 | 2.6e-13 |
| $\frac{\|\mathbf{g}-\mathbf{g}_t^R\|}{\|\mathbf{u}-\mathbf{u}_{\Lambda^\varepsilon}\|}$ | 0.96 | 1.06 | 1.11 | 1.07 | 0.92 | 1.0 |

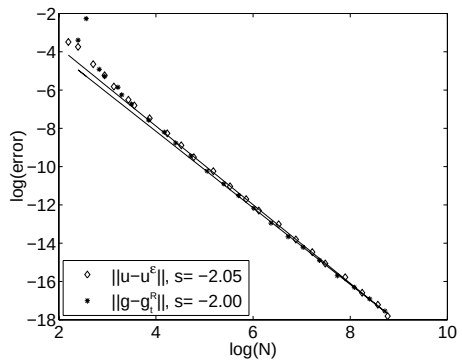
Table 7.1: Recovery for $g = u_2^2$. Parameter studies, $N(2,2)$

The following table corresponds to computations for $N(4,4)$ and $g = u_2^2$ allowing similar observations as above. All other test computations show a similar behavior.

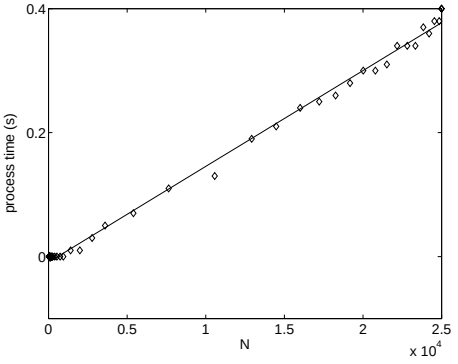
| $\#\Lambda^\varepsilon$ | 75 | 125 | 292 | 861 | 2273 |
|---|---|---|---|---|---|
| $\|\mathbf{u} - \mathbf{u}_{\Lambda^\varepsilon}\|_{\ell_2}$ | 3.6e-6 | 3.0e-7 | 4.5e-9 | 6.5e-11 | 9.9e-13 |
| $\#\Lambda^g$ | 296 | 416 | 740 | 1904 | 4723 |
| $\#\Lambda_t^R$ | 71 | 122 | 304 | 7766 | 2274 |
| $\|\mathbf{g} - \mathbf{g}_t^R\|_{\ell_2}$ | 4.2e-6 | 3.1e-7 | 4.8e-9 | 6.7e-11 | 1.09e-12 |
| $\frac{\|\mathbf{g}-\mathbf{g}_t^R\|_{\ell_2}}{\|\mathbf{u}-\mathbf{u}_{\Lambda^{\varepsilon_t}}\|_{\ell_2}}$ | 1.1 | 1.05 | 1.06 | 1.03 | 1.10 |

Table 7.2: Recovery for $g = u_2^2$: Parameter studies, $N(4,4)$

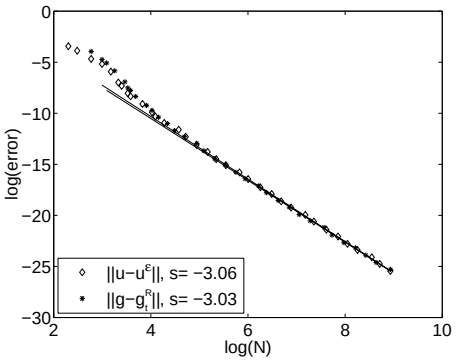
Figure 7.5 shows the corresponding results for $u = u_2, g = u_2^2$ showing the same good correspondence with the expected behavior.



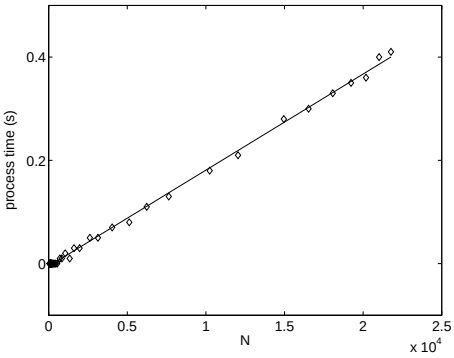

(a) Error rates using $N(2,2)$



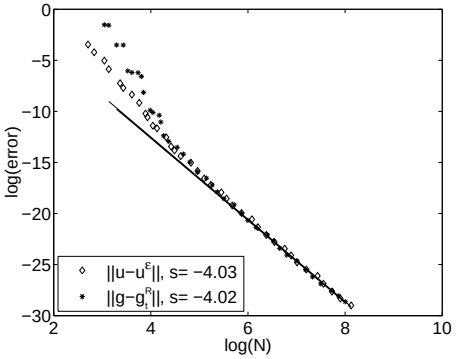

(b) CPU time for recovery with $N(2,2)$



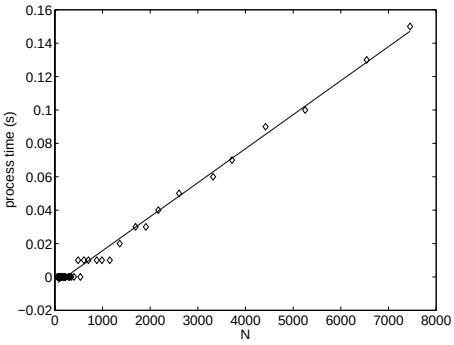

(c) Error rates using $N(3,3)$




(d) CPU time for recovery with $N(3,3)$




(e) Error rates using $N(4,4)$




(f) CPU time for recovery with $N(4,4)$

Figure 7.5: Test of the recovery scheme with $u = u_2$ and $g = u^2$

### The Heuristic Approach

We will now be concerned with $g = g_1$ and the nonlinear function $y(\cdot) = 4\sin(2\cdot)$, see Figure 7.2. For this composition, the expected error rates are determined by the same methods as before, resulting in 2 for $N(2,2)$ and 3 for $N(3,3)$.

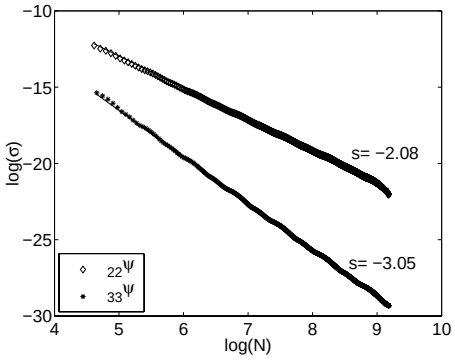



Figure 7.6: Error rates

In this test case, the shape of the index sets for $g$ and $y \circ g$ differ qualitatively, as we illustrated above, cf. Figure 7.2. Therefore, a uniform subdivision of each cube in $\Lambda^g$ might lead to a pessimistically large prediction set $\Lambda^{y\circ g}$.

We will now compute $r_\lambda$ according to Procedure H-Subdiv. Below, in order to illustrate the proceeding, we plotted the graph of $g = g_1$ as well as $y \circ g$ (left-hand side) and $|y'(g(x))|$ for $x \in [-1,1]$ on the right-hand side.

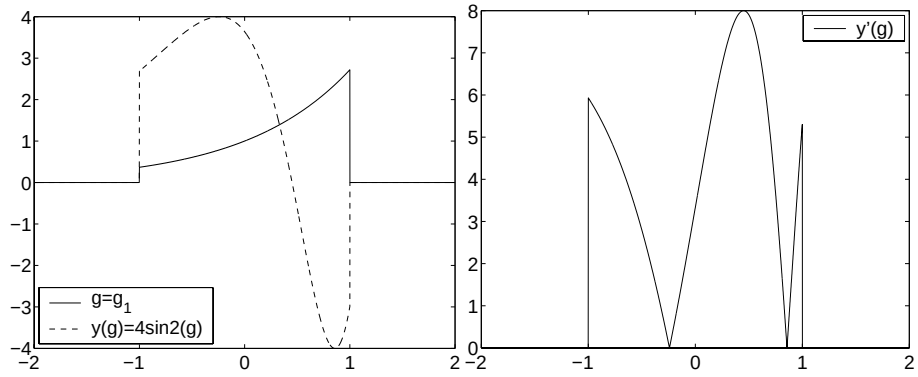



Figure 7.7: $g, y(g)$, and $y'(g)$

The following figure shows the graph of $Y(x) := |y'(g(x))|/\|y'(g(x))\|_\infty$ for $x \in [-1,1]$ (solid line) and the local subdivision indicator $r(x)$, which we derived by stating

$$r(x) = \begin{cases} 0, & Y(x) \leq 0.25, \\ 1, & Y(x) \in (0.25, 0.5], \\ 2, & Y(x) > 0.5. \end{cases} \tag{7.2.2}$$

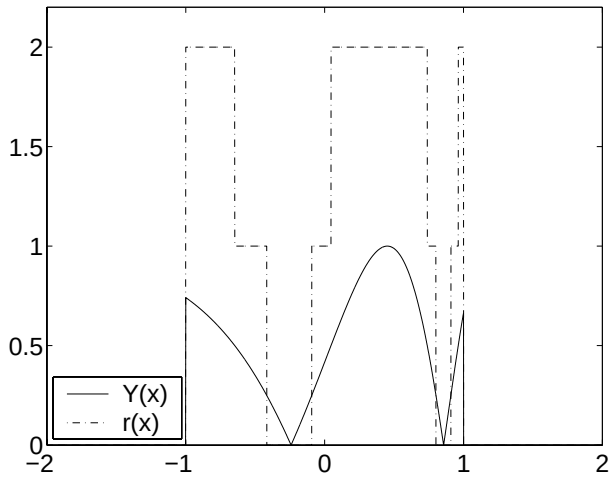
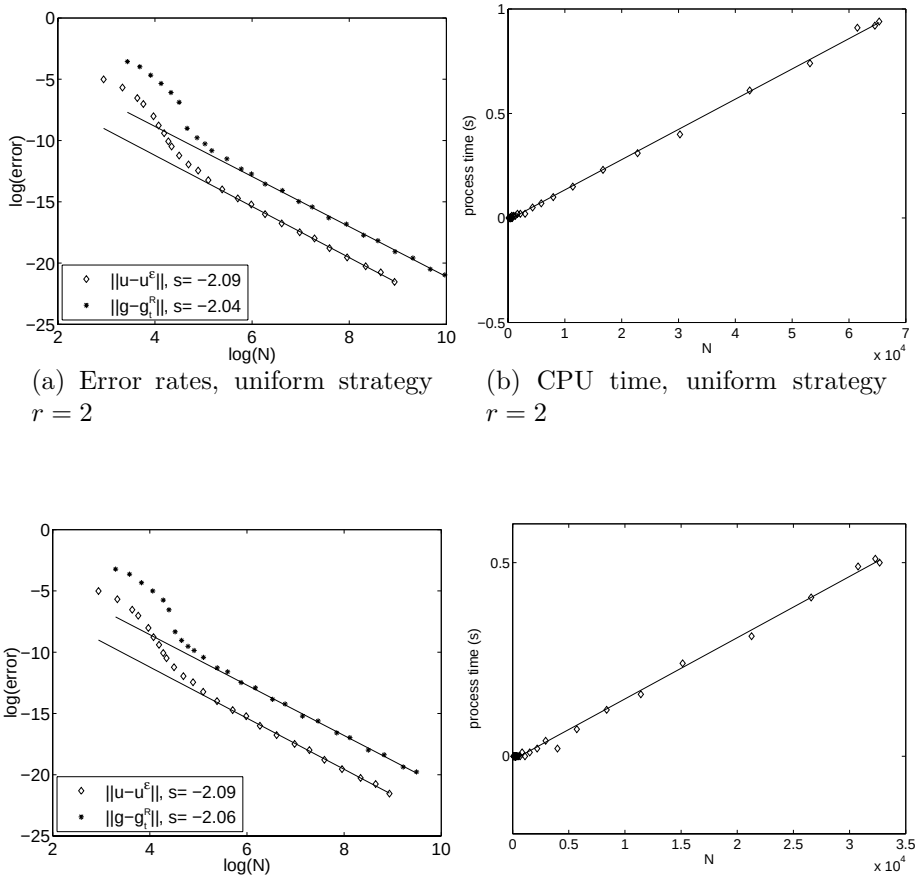



Figure 7.8: Non-uniform refinement strategy: Indicator

Of course, a rule like (7.2.2) is purely heuristic and its shape depends very much on the circumstances at hand. For any leaf $\lambda \in \partial\Lambda^g$, we will then choose $r_\lambda = \max_{x \in \square_\lambda} r(x)$ and subdivide $\square_\lambda$ in total $r_\lambda$ times. Note that this strategy leads to a uniform strategy for $y(\cdot) = \cdot^\alpha, \alpha \in \mathbb{R}$, so it is in agreement with the procedure we applied in the previous section.

The next diagrams compare the results of the recovery scheme for the prediction set $\Lambda^g$ determined by uniform 2-fold subdivision and by the heuristic strategy according to Figure 7.8. In both cases, the result of the recovery process shows the expected error rate and both results are very similar. Yet, comparing the CPU times, displayed on the right-hand side, the advantage of the heuristic strategy becomes clear: The size of the predicted set $\Lambda^g$, which serves as input for the recovery scheme, is about twice as large in the uniform case compared with the heuristically determined index set, which consequently results in a doubled CPU time.




(a) Error rates, uniform strategy $r = 2$

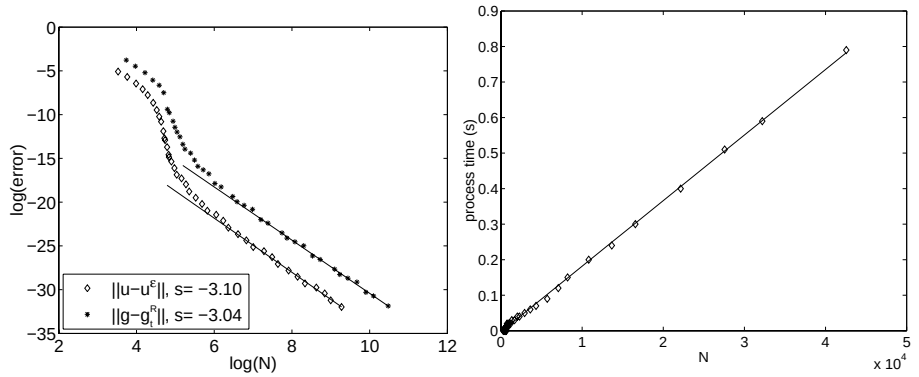
(b) CPU time, uniform strategy $r = 2$

(c) Error rates, heuristic strategy

(d) CPU time, heuristic strategy

Figure 7.9: Uniform and heuristic prediction strategy using $N(2, 2)$

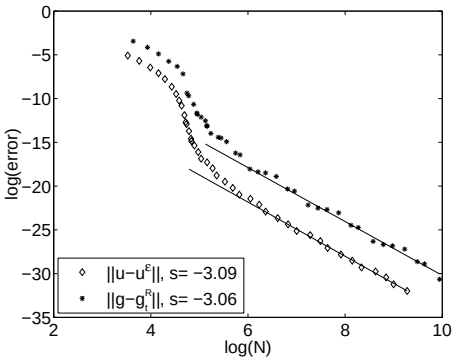
We can observe the same effects also when using different types of wavelets as the following graphs show.



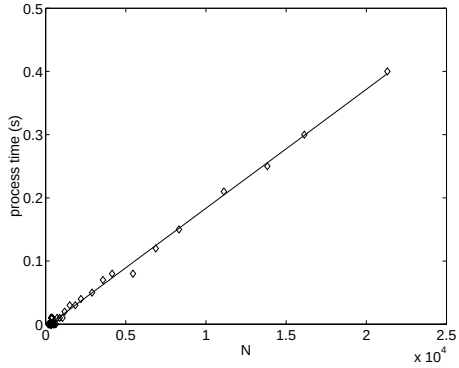

(a) Error rates, uniform strategy $r = 2$




(b) CPU time, uniform strategy $r = 2$




(c) Error rates, heuristic strategy




(d) CPU time, heuristic strategy

Figure 7.10: Uniform and heuristic prediction strategy using $N(3, 3)$

# Chapter 8

# Dual Norms

The numerical experiments of this chapter refer to the second scenario sketched in Section 3.1, namely involving topologies where $\mathcal{H}$ is compactly embedded in $L_2(\Omega)$ such as a Sobolev space $H^t$ of positive order $t$, which leads to a recovery in dual norms.

In the following tests, the original function $u$ is subject to a nonlinear mapping (of polynomial type) $y : H^t \to H^{-t}$. As before the examples are designed in such a way that $F(\psi_\lambda)$ is in $L_2$ and pointwise smooth, yet the $H^t$- and $L_2$-norms are very large compared to the (dual) $H^{-t}$-norm where the recovery is supposed to take place. The general layout of the test computations follows again Chapter 5.

## 8.1 Gauss Quadrature

First, we use the Gauss-type formulas from [BBDK01], see Section 4.1.2, for quadrature. One of our goals here is to illustrate, that this suffices in practical applications involving pointwise smooth functions.

In 1D, we consider

$$u_3(x) := 2exp(-100 * (x - 0.5)^2) \quad \text{and} \quad y(\cdot) = \cdot^5. \tag{8.1.1}$$

For the two-dimensional test we will be concerned with the same nonlinearity and

$$u_4(x_1, x_2) := u_3(x_1)u_3(x_2). \tag{8.1.2}$$

Note that according to [CDD03d], in $\mathbb{R}^d$ we have that $y : u \to u^p$ maps $H^t$ into $H^{-t}$ for $t < d/2$ if

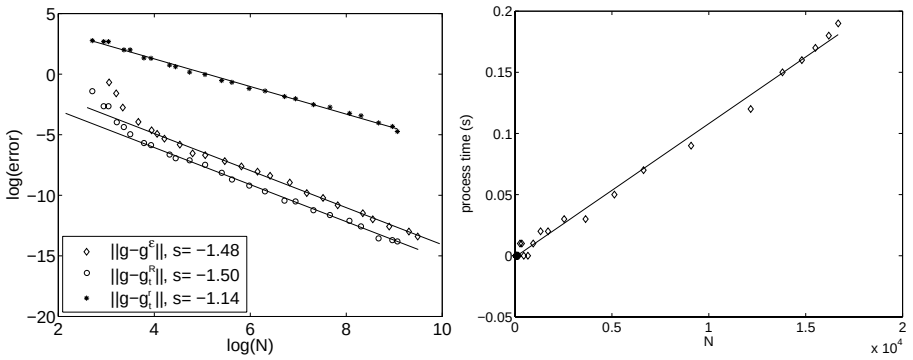
$$p < p^* = \frac{d + 2t}{d - 2t}. \tag{8.1.3}$$

For $t = 0.334$ and $d = 1$, $t = 0.667$ and $d = 2$ respectively, $y(\cdot) = \cdot^5$ is therefore close to the limit case. These choices have no practical meaning but are merely to test the dual norm case in a regime where the nonlinear mapping affects the norms in a visible way.

The size of the various norms are recorded below.

| norm | $u_3$ | $y \circ u_3$ | norm | $u_4$ | $y \circ u_4$ |
|---|---|---|---|---|---|
| $L_2$ | 0.709 | 7.576 | $L_2$ | 0.501 | 57.395 |
| $H^{0.334}$ | 7.115 | | $H^{0.667}$ | 10.548 | |
| $H^{-0.334}$ | | 0.853 | $H^{-0.667}$ | | 0.399 |

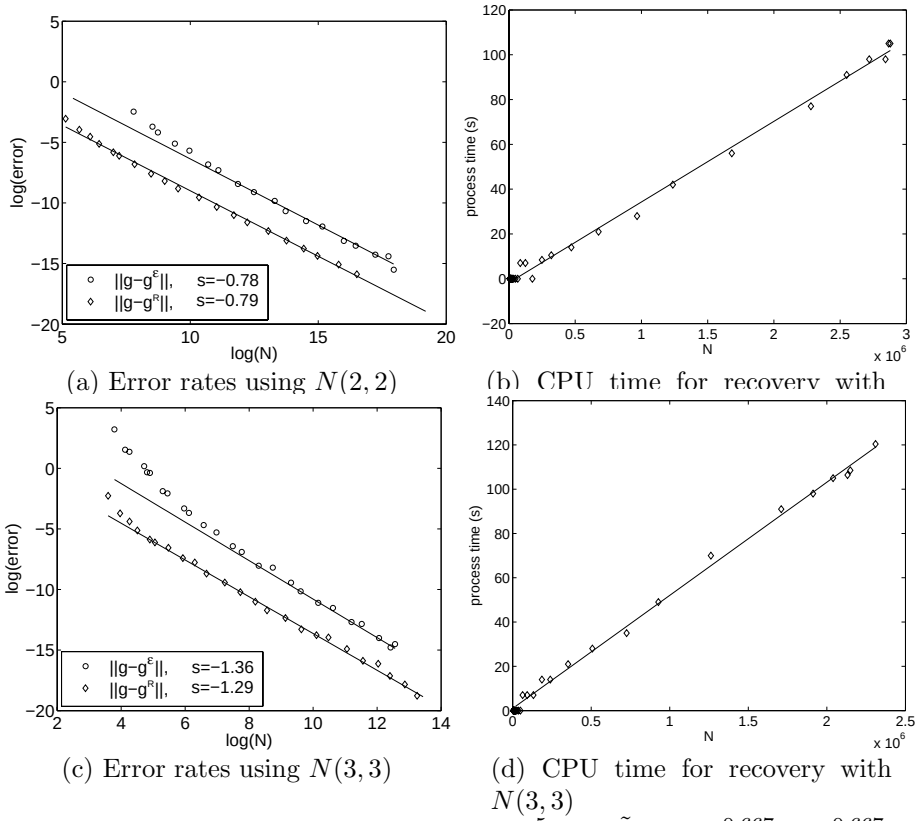
The following figures show the result of our tests in a similar fashion as before. Note however, that aside from the approximations to $u_3$ and the final result of the recovery scheme for $(y \circ u_3)^R$ we also displayed the intermediate result $(y \circ u_3)^r$, which is the output of the recovery scheme *before* scaling the coefficients by $\mathbf{D}_{\mathcal{T}}$, cf. 3.3.15. Compared to the final result, this intermediate array also has a fixed error decay rate, which is, however, by $0.35 \approx t$ lower than the one obtained for the recovery in the correct norm.




(a) Error rates (b) CPU time

Figure 8.1: Test of RECOVER: $g = (u_3)^5, (\mathcal{H}, \tilde{\mathcal{H}}) = (H^{0.334}, H^{-0.334})$, $N(2, 2)$.

The following 2D experiments, even though dealing with larger constants, also show the expected behavior.




(a) Error rates using $N(2,2)$

(b) CPU time for recovery with $N(2,2)$

(c) Error rates using $N(3,3)$

(d) CPU time for recovery with $N(3,3)$

Figure 8.2: Tests of RECOVER: $g = (u_4)^5, (\mathcal{H}, \tilde{\mathcal{H}}) = (H^{0.667}, H^{-0.667})$.

## 8.2 Least Squares Quadrature

Next we will be concerned with the pointwise smooth but oscillatory function

$$u_5(x) = (exp(-100 * (x - 0.5)) * sin(500x))^3,$$

which is to be recovered for test reasons in $H^{-1}$. Concerning the $H^{-1}$ norm we have $\|u_5\|_{H^{-1}} \approx 0.0877$, again determined by the weighted sum of a (finite) reference wavelet approximation obtained by means of a scaling function representation on a uniform grid (level 15). In order to validate the results, the inner products for this reference are computed *exactly* by means of automatic, symbolic integration using MAPLE 9.0.

For the above setting, we shall compare the recovery scheme using two different quadratures, namely the Gauss-quadrature according to [BBDK01] and the following quadrature method which is a simplification of (4.1.7).

Given an overall target accuracy $\varepsilon > 0$, for each level $j$ do:

1. Let r=1.

2. Determine $\mathbf{q}_j^r$ as described in (4.1.4) and compute

$$\mathbf{d}_j^r(g) := \mathbf{G}_{j,\tilde{\Theta}} \cdots \mathbf{G}_{j+r-1,\tilde{\Psi}} \mathbf{q}_{j+r}^r. \tag{8.2.1}$$

3. If

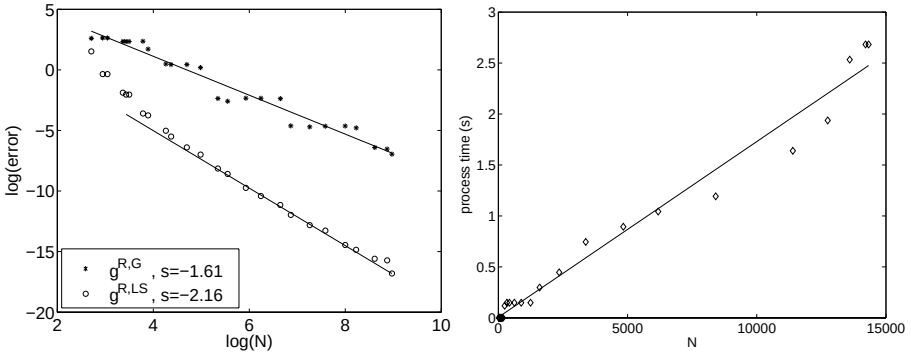
$$\left( \sum_{l=j+r}^{j+r'-1} \omega_j^2 \|\mathbf{G}_{l,\tilde{\Theta}} \mathbf{G}_{l+1,\tilde{\Phi}} \cdots \mathbf{G}_{j+r-1,\tilde{\Phi}} \mathbf{q}_{j+r'}\|_{\ell_2(\mathcal{J})}^2 \right)^{1/2} > \varepsilon(\#\Lambda_j/\#\Lambda). \tag{8.2.2}$$

set $r \to r+1$ and go to i) else accept $\mathbf{q}_j^r$, cf. (4.1.8). Here we take $r' \in \{r+1, r+2\}$.

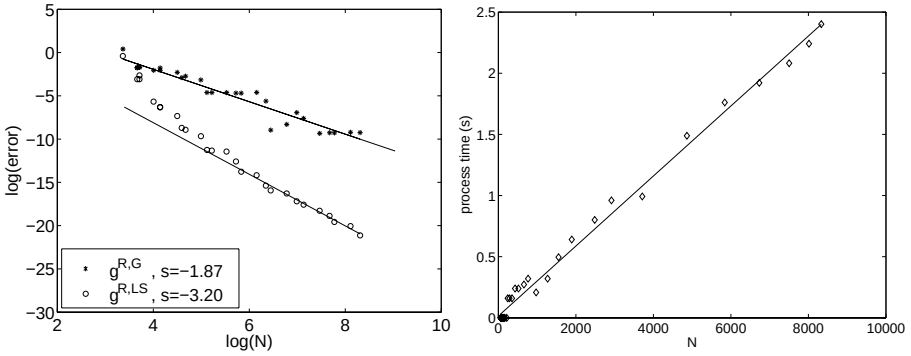
The following graphs show the results of our computations with $N(2,2)$ and $N(3,3)$. We used Gauss quadratures with two points, the number of sampling points for the least-squares formula is $|Y(N(2,2))| = |Y(N(3,3))| = 4$. The observed average refinement parameter is $r = 2$.

Note, that the error of the recovery scheme using the Gauss quadrature, labeled $(y \circ u)^{R,G}$ in the plot, does not show the desired behavior. It stagnates from time to time and even though we can observe an overall error decay, the rate is certainly insufficient. The reason is, that the error stemming from the Gaussian quadrature dominates the error of the recovery scheme. On the other hand, the performance of the recovery scheme using the least-squares method described above, labeled $(y \circ u)^{R,LS}$ is satisfactory as (8.2.2) ensures, that the quadrature error does not dominate.

On the right hand side, we display the CPU-times referring to to *least-squares* computation. More than 70% of the CPU time is consumed by checking the error criterion (8.2.2), namely to apply $\mathbf{G}_{j,\tilde{\Theta}}$ and to compute the norm in (8.2.2).




(a) Error rates using $N(2,2)$ (b) CPU time for recovery, $N(2,2)$




(c) Error rates using $N(3,3)$ (d) CPU time for recovery, $N(3,3)$

Figure 8.3: 1D-Tests of the recovery, least-squares quadrature, $g = (u_5)^3$

# Part IV

# Realization

**Overview:** Part IV gives an overview of the construction principles and the general layout of the C++-code realizing the schemes described in this thesis.
Chapter 9 collects some general ideas concerning the realization of (wavelet) adaptive schemes that guided the coding.
In Chapter 10 we briefly present the architecture and the main ingredients of our realization.

# Chapter 9

# Conceptual Remarks

The ongoing successful developments of theoretical methods to treat operator equations give rise to a growing demand for efficient computer programs based on these findings. However, the difficulties one encounters, when carrying out such realizations, should not be underestimated.

For one thing, the adaptive paradigm requires sophisticated data handling due to the fact of dynamically changing index structures. Secondly, the theory of adaptive wavelet schemes is focused on *asymptotical* behavior and therefore, constants usually do not play an important role. When transforming the results of this analysis into computer code, the main focus sometimes is quite opposite to that of the theoretical investigation. For a realization, constants have to be considered and especially the basic and therefore frequently used operations require careful treatment. Basic operations like e.g., the selecting of certain indices from a given set, are usually not explicitely analyzed in theory. Yet, from a practitioners point of view, these operations do contain the risk to spoil the optimality of the scheme by hidden sub-optimalities. Also, there is always the danger, that they *de facto* dominate the performance in practice due to huge constants.

Therefore, all ingredients of a realization have to satisfy two crucial conditions: They have to

- preserve the theoretical complexity of the algorithm,
- provide a reasonable scaling of the algorithm.

The latter is to prevent larger constants to spoil the performance of the code in the sense that the asymptotical gain of the scheme lies beyond reach of realistic numerical tests. Both conditions demand careful investigations of the theoretical schemes including *all* of the required operations concerning floating point operations, memory allocations, and data access. Moreover,

when turning to implementation, an appropriate design of the program is most important.

Another issue is, that asymptotical analysis which drives the algorithms to be realized is always (necessarily) led by the worst case scenario in a whole class of problems, hence estimates would be too pessimistic for the concrete case at hand. Nevertheless one wants the optimal result for each given problem when it comes to computations. A simple example is an error estimator stating, that we will reach a target accuracy in at most $N$ (expensive) iteration steps for each member of a whole class of problems, yet the problem at hand allows to reach it in $N'$ steps, $N' << N$. Therefore, beside preserving the asymptotical quality of the theoretical scheme, it is necessary to include mechanisms and heuristics that tune the algorithm to fit better to each current application. In fine tuned complex schemes, this often is not trivial.

Adaptive wavelet theory – although still very progressively developing – meanwhile seems to come to quite a clear view on how to treat given problems in principle. It can be observed that even the newer schemes are largely described by very few basic operations that can be formulated independently of the concrete problem at hand: The recovery scheme, various thresholding routines often using the binning routines, and a special fast matrix vector product, cf. [CDD01, BDS04], also incorporating RECOVER. At least in theory, one can combine these basic tools, to suite a great variety of situations.

To provide this flexibility also in the realization of these tools, while insisting on their optimality, is a great challenge. Attempts in this direction are documented in [Met02, BBC$^+$02, Bar01, Jür01, Kon01] and also this thesis wants to make its contribution. However, the ways of how to efficiently treat the various subproblems critical for any implementation of adaptive wavelet schemes, are just beginning to be developed and need much more investigation and experience.

In this part, we will give an overview of the implementation of the algorithms discussed so far. Our main focus is on the binning process and above all the application of the recovery scheme. Concerning the implementation of adaptive solvers, in the form as presented in [CDD01] we refer to [BBC$^+$02, Bar01, Jür01].

We present our methodology and realization in three steps. First of all, we shall review all of the above mentioned schemes and algorithms trying to extract their demands on an implementation. This will lead us to certain design principles that guided the organization and the implementation of all the material. Then, we shall give an survey on the building blocks of

our code.

One very general design principle is to separate objects and algorithms. Objects are understood to be (mathematical) elements being manipulated by algorithms. They are characterized by a set of properties remaining unchanged in the various contexts. We would like to identify those objects and implement them with their *minimal* set of properties. Any additional features should be formulated as algorithms. Such a strategy is crucial, especially when considering the constant improvement in theory resulting in continously changing details in related numerical schemes.

We drive at a realization of the binning routines and of RECOVER including its applications. We have to mention that the software presented here does not yet incorporate non-periodic wavelet bases on bounded domains. However, our design is open for a future extension in this direction.

## 9.1 Key Requirements for a Realization

Considering a realization, note that the recovery scheme as well as the adaptive algorithms, e.g. [CDD01, CDD03c], themselves are based on coefficients that encode level, location and type in a unique index $\lambda$. Moreover, an access function $V(\lambda)$ associates a single value to $\lambda$, hence the coefficient $\mathbf{d}_\lambda$ is a *pair* of the form $(\lambda, V(\lambda))$. The unique index serves as a key, or identifier to address the coefficient. Therefore, we have to provide a possibility to quickly access the value if the index is given.

Concerning sets of coefficients, recall that the binning (or sorting) routines are solely based on the values of the coefficients. They do not necessarily induce any structure based on index information. Hence we have to efficiently manage *unstructured* insertion and deletion. Also, the binning schemes require, that sets of such coefficients can be passed through fastly. Here, no ordering concerning level or location is required.

When it comes to the recovery scheme, the essential characteristic from a data-management point of view is that this algorithm is organized based on *level information*. Here it is crucial to fastly access and pass through the coefficients of a given level. An ordering according to spacial position again does not play any role.

Hence, we need an index management, that allows fast *random access*, randomly ordered passing *through the entire set* and *through a given level*.

Concerning the quadrature routines needed in the recovery scheme, we have to rely on a method that at least evaluates the primal scaling functions and their linear combinations at arbitrary points in an efficient way.

### 9.1.1 How to Treat Wavelet Expansions?

The *mathematical* object we are working with most of the time is a wavelet expansion

$$g(\cdot) = \sum_{\lambda \in \Lambda} c_\lambda \psi_\lambda = \sum_{\lambda \in \Lambda} \tilde{c}_\lambda \tilde{\psi}_\lambda.$$

Reviewing the above mentioned algorithms and their demands, we note that in most instances we concentrate on operations based only on the corresponding *coefficient set* as we do not actually work with the function, but with its representative sequence $\mathbf{c}_\Lambda$ of coefficients $c_\Lambda$. Some of the operations we want to perform depend on the basis the coefficients belong to in the sense that they e.g., require information on the mask. As one example, this is true for decomposition and reconstruction. The majority of operations on the indices, however, are completely independent of the *context* of a basis, consider e.g., insertion, deletion or sorting. Also, more complex algorithms like GRADING do not depend on the underlying basis. They are even independent of the value $V(\lambda) = c_\lambda$ associated to the index $\lambda$.

This observation motivates the principle decision to create index structures, i.e., structures representing indices, coefficients (pairings of an index and a value) and sets of these two types without incorporating any information referring to a specific basis, i.e., one can handle sets of coefficients without the context of a basis. This matches our idea to create objects with a minimal set of characteristic properties and provides additional flexibility.

However, to switch between the single- and the multi-scale representation of a wavelet expansion involve at least either the primal or the dual mask. Moreover, such transformations usually involve several levels and do therefore require a multi-scale index management.

The requirements on the structure that provides this management and hence represents a set of indices, however may depend on the application. Therefore our representation of a wavelet basis will only provide the masks of the basis functions (and related information). It will not not incorporate any operations to *perform* a change of basis.

A structure representing a wavelet expansion (`CBSpline`) will combine a context-free index set with a wavelet. With the corresponding masks at hand, a decomposition for example can be performed by an *algorithm* working on the index set using the mask of the wavelet as data. Note, that with this design, we can relate the index set with a different basis by linking the decomposition algorithm with a different mask.

### 9.1.2 The Importance of Point Values

Section 4.1 relies on the assumption that point values at more or less *arbitrary* knots are feasible at low cost. Of course, one could think of replacing or modifying the employed quadrature and thereby circumvent or weaken the necessity of the fast point evaluations at arbitrary points. It is known that refinable functions can be evaluated at diadic grid points by successively applying the two-scale relation.

However, in this regard it is good to note that *cardinal $B$-spline wavelets* exhibit the features of Section 2.2 and allow easy point evaluations at arbitrary knots. Using *tensor products* in higher dimensions also provides all necessary properties at and offers a simple structure. For more complicated domains there exist methods to construct suitable bases using cardinal $B$-splines and tensor products, cf. Chapter 2. Hence, we can only make use of quadrature formulas in our program, if the the corresponding point values are provided by wavelet representation we intend to use. Therfore, concerning all issues related to point values, we restricted the design of our code to biorthogonal cardinal $B$-spline wavelets according to [CDF92] and related constructions. Recall also, that at least in view of the quadrature formulas described in Section 4.1.2 a favorable by-product of cardinal $B$-spline wavelets is, that their primal scaling functions are always non negative.

## 9.2 Index Management: Why No Tree Structure?

Due to the constant dynamical changing of the index set, because of the adaption processes and also due to the intensive manipulations on the structure of the index set described in Chapter 4.2, an efficient index management is crucial in any adaptive code. An index management faces the problem that in general it can not rely on any structural information that permits to save storage or eases the handling of the data. Reviewing the key operations, our schemes require, the following are the most demanded:

- Iterations on all entries of the set,
- iterations on all entries on a given level, and
- accesses to entries corresponding to a given key (check if an entry exists, update, insertion, and deletion of entries).

For RECOVER, an efficient level-wise access is crucial, but this feature is not necessarily (explicitely) required in all adaptive schemes. On the other hand, already a level dependent weighting as e.g., needed for the preconditioning, can be executed much easier and faster, if one is given the possibility to sweep the coordinates level by level. We therefore consider level-wise sweeps as one key operation of index managements in the present adaptive context.

Yet, in view of RECOVER and especially the index manipulations, e.g. LEAF, one might wonder why we do not suggest the index container to be a *tree structure*. Tree structures seem to be natural at the first glance, considering the fact that the recovery scheme is entirely based on a tree-shaped (support-cube) model of the index set and that tree structure is only a very weak restriction on an $N$-term approximation, cf. [CDD01]. Yet we nevertheless claim, that a corresponding implementation of an index-set should *not* explicitely consist of a tree. To explain this, let us compare two alternatives.

If the implementation of a set incorporated tree structure, we would have to store for each entry the position of its parent and its children. All in all, $2^d+1$ objects if we consider only level-differences of 1. *Position* here means a pointer to the storage address. In addition to that, as fast level-wise sweeps are required, one also needs to store information concerning neighbors (not necessarily siblings) on the same level. Moreover, any insertion or deletion in such a set would require the update of all involved parents, children and neighbors. As a benefit resulting from that effort, we would be able to access e.g., the parent and the children of a given index directly, i.e. without searching.

If the data structure is no tree, i.e., if we do not list the storage-position of parents/children, given an index $\lambda = (j, \mathbf{k}, \mathbf{e})$, we would have to *search* the corresponding relative index be it the parent or a child. What is the expense of that? First of all, to compute the corresponding parent index is simple and 'cheap', in the case of wavelets on $\mathbb{R}^d$ e.g., it is given by $(j-1, \mathbf{k}/2)$, where $\mathbf{k}/2 = (\lfloor k_1/2 \rfloor, \ldots, \lfloor k_d/2 \rfloor)$ and $\lfloor k_i/2 \rfloor$ is obtained by a simple binary integer division, which is a fast operation, namely a bit shift. Likewise, we can compute the set of children, see (4.2.2). Secondly due to the manipulations based solely on the value of a coefficient (e.g. COARSE), we need unstructured access. This means that given an index $\lambda$, we have to be able to access the corresponding coefficient efficiently, i.e., we have to provide a fast access function $V(\cdot)$ whether the data-structure is a tree or not, or else it is not truly suited for adaptive strategies.

Hence, instead of using storage to track the position of the relatives (resulting also in time for updates) we can combine two fast operations to acces parents/children by algorithmic means: First compute the desired index and then use the fast random access mechanism. In summary, under the given circumstances, we consider it more favorable to model an index tree by algorithm rather then by data structure.

# Chapter 10

# Implementation

## 10.1 Requirements on the Components

Let us summarize the building blocks of our attempt to implement the above mentioned computation tools for adaptive wavelet schemes based on tensor product cardinal $B$-spline wavelets on $\mathbb{R}^d$. We made the following basic decisions.

- Separate objects and algorithms.
- Objects shall have a minimal set of characterizing properties.
- The index management system shall be independent of basis information.
- A wavelet basis provides mask information. In the case of primal basis, point evaluation routines will be added.
- A wavelet expansion will combine the index management with the basis information and provide all related routines.

In view of the above mentioned requirements, we end up with the following list of objects and their properties.

**Index:** *minimal storage*
An index $\lambda$ contains a variety of information, namely $\lambda = (j, \mathbf{k}, e)$, where the level $j$ is an unsigned integer, $\mathbf{k} = (k_1, \ldots, k_2)$ are integers and $e = 0, \ldots, 2^d - 1$ is the type information. As we expect to store and access a huge number of indices the storage amount should be kept minimal.

**Coefficient:** *Pair structure, fast access, minimal storage*
Remembering Section 1.1, it is clear that a coefficient has to be implemented as a *pair structure* $(\lambda, V(\lambda))$. These pairs consists in $\lambda = (j, \mathbf{k}, e)$,

an index playing the role of the access key and the value $V(\lambda)$, usually a double. To store, handle and access a huge number of coefficients, the storage amounts for $(\lambda, V(\lambda))$ should be kept minimal and the key-value access should be as fast as possible.

**Index/Coefficient Set:** *Memory operations, fast random access, fast sweeps*
We suggest to work with sets of indices $\lambda = (j, \mathbf{k}, e)$ and sets of coefficients $(\lambda, V(\lambda))$. The key operations, our schemes require from the index structure are discussed in Section 9.2. Additionally note, that we have to provide fast dynamical memory operations, as the memory requirements for an adapted set are not known beforehand. Note that, the fact, that all $(\lambda, V(\lambda))$ have the same size offers optimization potential with respect to the standard memory management provided by the computer architecture.

**Wavelet bases:** *Mask information, point evaluation (primal)*
We need information on the mask of the wavelets. Moreover, for the $M$-grading, support information on all incorporated functions/masks is required. In the case of the primal basis, we also need to put at disposal point evaluations of the basis functions.

**Wavelet expansion:** *Efficient coefficient management, fast point evaluation, fast transformations*
A wavelet expansion has to unite the key properties of a coefficient set and the wavelet functions, namely efficient management of the coefficients and fast point evaluation. Moreover fast decomposition and reconstruction routines are necessary.

## 10.2 A Tour Through the Wavelet Library

This chapter is intended to give an overview of the software of the wavelet library `igpm_w_lib`. About 30000 lines of code, more then 50 classes and various algorithms give rise to some 400 pages of documentation when using DOXYGEN [vH04], a standard documentation-generator. This renders a comprehensive description impossible within this thesis. Nevertheless, in the following we shall mention all important features without going into technical and language dependent details. For a more complete documentation of the software and further details concerning implementation we refer to [MV99, BV].

All code was written in `C++` and the approach is clearly object oriented.

Algorithms and classes used in the wavelet library `igpm_w_lib` intend to be quite flexible in usage. With this in mind, we used the various concepts, `C++` offers in this regard (templates, iterators and streams) intensively. Before we start to describe our software, let us briefly recall the following basic facts we can not avoid referring to:

*Objects* in `C++` are usually implemented in *classes* containing *member variables* and *methods*. Methods are functions and procedures belonging to a certain class. Classes introduce new *data types*. Many algorithms and objects can be formulated in an abstract way, i.e., they are independent of the concrete context.

An example for such an object is a container of wavelet coefficients, being in principle independent of the realization of the coefficients. The only link between index and container is an interface that provides access to the value of an index. Sorting of the coefficients in the container requires the existence of some ordering relation, e.g., the usual $\leq$ for real numbers. Knowing the interfaces, we can now formulate a sorting algorithm which is independent of the concrete realization of the coefficients and of the container. `C++` provides the powerful tool of *template arguments* to formulate such abstract objects (classes) and algorithms (functions). Beside usual *run–time* parameters, classes and functions can be given *template* arguments. These parameters cannot change while the program is running, but have to be set beforehand.

Another language structure allowing abstraction processes is *inheritance*. One can e.g. implement a class $A$ and inherit to a class $B$. An object of type $B$ then is also of type $A$, but might have additional features. One advantage is that all processes working with properties of $A$ can be applied to $B$ and $B$ (in principle) offers at least the functionality of $A$. The classically cited application model for this is a manager who is an employee with additional functionality.

Abstract containers require a mechanism to provide the user with the possibility to sweep through its entries without knowing details of the inner organization of the container. This leads to the *iterator* concept. An iterator, sometimes called *generalized pointer* enables the user to sweep through complex structures as if stepping through a sequence.

To handle data exchange in a flexible way, `C++` offers so called *streams*. A stream is simply an object containing data which can be filled, read and displayed independently where the data is stemming from. For further details on `C++`, we *of course* refer to [Str00].

The interactions and dependencies of the most important classes and algo-

rithms of the `igpm_w_lib` are sketched in the diagram below. Its elements will be explained in a moment.

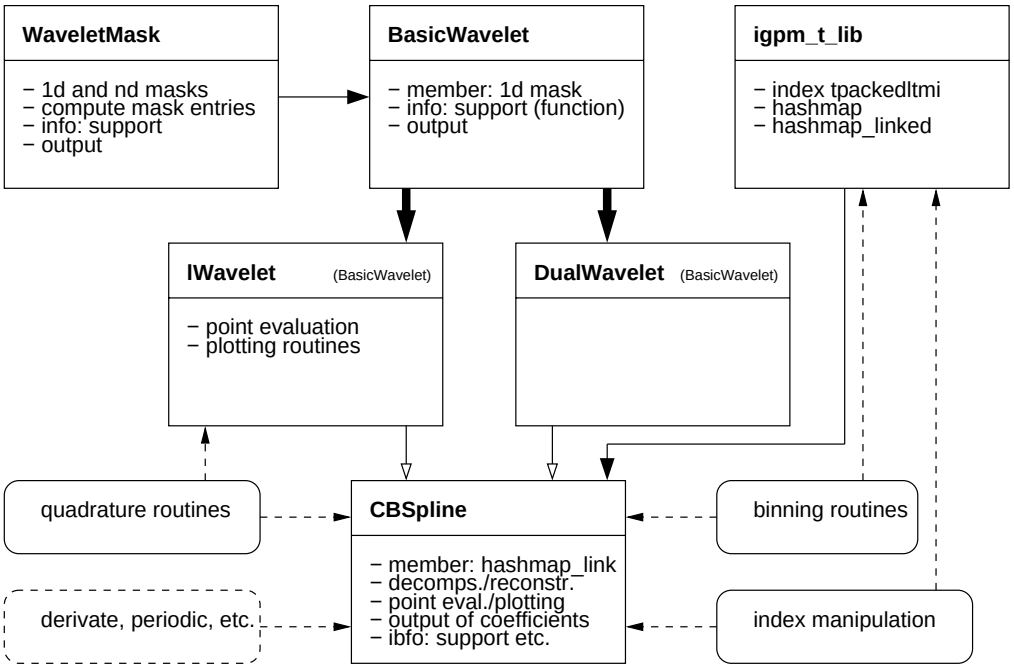



Figure 10.1: Most important classes and algorithms of the `igpm_w_lib`

In Figure 10.1 we used rectangles to visualize classes and libraries. The title contains the name and in the case if inheritance relations the name of the 'father'–class. Inheritance relations are indicated by bold full arrows. Dashed rectangles with rounded corners symbolize algorithms. Simple full arrows are used to denote that one class provides an object, i.e., a class member, for another class. Hollow arrows indicate a 'provides information'– relation. Here information is provided without creating class members. The dashed arrows show on which objects the corresponding algorithm can operate.

### 10.2.1 Index Management `igpm_t_lib`

The index management library `igpm_t_lib` we use is mainly due to Alexander Voß. To suit the demands of our schemes, some modifications were made compared to the first version of the `igpm_t_lib` described in [MV99].

### Index

An index can be stored in a compressed format implemented in the class `tpackedltmi`. Limiting the range of $j$ and $\mathbf{k}$ beforehand by certain powers of two, we can compress these information into a certain number of bytes.

Of course, fixing a maximal level contrasts the principle of adaptivity in the sense that we *do* make a priori assumption on the distribution of significant coefficients. Note however, that from a principal point of view all implementations performing on machines offering a limited range of numbers have this deficiency. In practical experiments, the user has to decide whether to choos a compressed format or to use `tmultiindex` offering the usual integer range for the application at hand.

### Hash-maps

*Hash-maps* are data containers designed to provide fast access (insertion, deletion, check of existence) to its entries. It uses a specific part of the entries, the *key*, for storage and access management. The part of the entry not being key is called *value*. Of course, it is possible that there is no value, in which case the hash-map represents a mathematical set. Otherwise, it forms an *associative container* as the hash-map performs a mapping between key and value.

The storage and access management of the hash-map is based on a so called *hash–function*. It maps the keys to integer numbers in a certain range $0, \ldots, n-1$. This integer is then used to encode the position of the entry. Obviously, the performance of the hash-map depends on the speed and the separation ability of the hash-function. It is important that possibly few keys from the range of keys are mapped onto the same integer, causing a so called *collision*. Usually the hash-function is not injective. Therefore, collisions will occur and have to be dealt with by providing an appropriate *collision strategy*. One simple, but for our purposes sufficient method is to store all colliding elements in a list. The probability of collisions obviously depends heavily on the range of integers the keys are mapped to, i.e. on $n$. That is the reason, why the performance of a hash-map depends strongly on the *fill ratio*.

The decision, whether an object is already an element of the hash-map depends only on the key, which therefore has to be stored. This is done by the so called *hash-table*. This is a (static) vector of $n$ entries containing pointers $p_i, i = 0, \ldots n-1$ to the beginning of the lists of elements with keys mapped onto $i$. Any insert routine will have to check if the key already

exists. If this is the case, the new entry has to be assigned new memory space to. Otherwise, the memory space of the existing entry is reused.

An interesting side effect of this, is that we can easily handle conditional insertion. The situation we constantly face in our schemes is, that we compute a value for $c_\lambda$ and have to either insert $(\lambda, c_\lambda)$ in the hash-map or to update an existing entry corresponding to $\lambda$. The insert routine returns a boolean value indicating whether the key exists and the (either new or already used) memory address. This enables us to update or replace the content of the memory without further operations.

While the hash-table does require only little storage, the elements of a hash map may be complex and huge data structures. However, they usually are of the same size, hence we can optimize the standard general `C++` memory management. As reallocation of memory is quite time consuming, we will allocate memory for several entries each time and enlarge this storage area only if necessary.

### Subsets of a Hash-map: Level Access

The structure described so far does not yet provide features to fastly pace all indices of a certain level as we will have to check each entry of the hash-map. A fast sweep according to a criterion requires the *grouping* of hash-map entries. This can be done by a list. Similar to the hash-table, we will have to provide a vector of pointers each pointing to the first element of a level. Furthermore, each entry points to the next entry on the same level. *First* and *next* here only refers to the order of insertion into the hash-map, i.e., the members of a list representing one level are *not* stored or sorted according to individual spatial position. Hence, we actually perform a binning with respect to the criterion 'level'.

The resulting structure `thashmap_linked` can be visualized as in Figure 10.2.

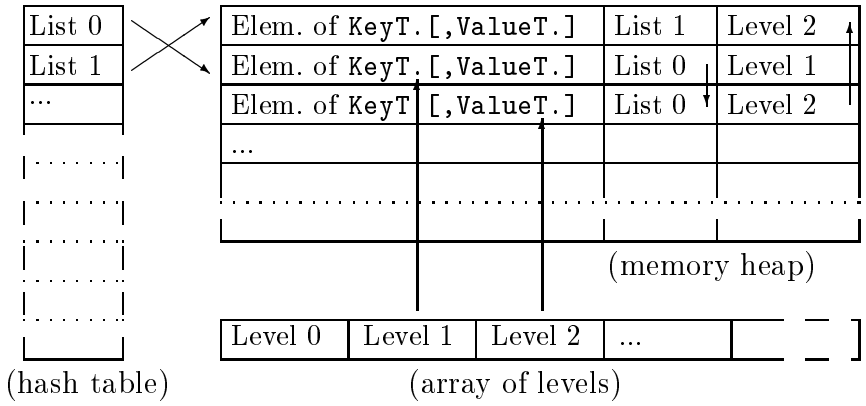



Figure 10.2: Schematic description of `hashmap_linked`

### 10.2.2 Wavelet Implementation

The key to all operations concerning the tensor product wavelets is to provide the one dimensional information and methods and combine them with efficient tools to handle the tensor product structure. All classes follow this principle.

**Masks:** `WaveletMask`

The multidimensional tensor product masks are not necessarily required for the algorithms we intend to realize. Even in higher dimensions, we therefore only compute the one dimensional mask for dual and primal functions in `WaveletMask` and provide a mechanism to provide the multidimensional array on request. An iterator enables the user to sweep the multidimensional cube of the mask and provides the information on the mask coefficients. Moreover, `WaveletMask` offers information on the support of the mask.

**Wavelet Basis Functions:** `WaveletBasic`, `Wavelet` **and** `DualWavelet`

The structure `WaveletBasic` provides all information valid for dual and primal wavelets. These are the one dimensional masks (`WaveletMask`) and support information. `Wavelet` and `DualWavelet` are inherited from that class. Concerning point evaluation for the primal scaling functions, note that it is possible to execute the evaluation without any information on the masks. This complete separation is no longer possible for the wavelet functions, as we compute their point values via the corresponding linear

combination of splines given by the scaling functions on the next higher level. This however is no problem, as we only have to sum over a fix linear combination of scaling functions. Therefore, at this point, we can avoid the necessity of more complex index structures.

Our implementation distinguishes dual and primal bases only when it comes to point evaluation. Primal bases are treated as wavelet bases (see above) with the additional possibility to be point evaluated. Hence, `Wavelet`, representing a primal function, is a `WaveletBasic`, that additionally offers methods to carry out point evaluations.

### Wavelet Expansions: `CBSpline`

The structure `CBSpline` links the data management and the wavelet basis functions providing change of basis using `WaveletMask`, support information from `WaveletBasic` and point evaluations for `Wavelet`. Of course, the multi-scale transformations could also relate to the mask information in `WaveletBasic`, however the present organization strictly separates information stemming from the mask or the corresponding function to enable independent changes in the implementation.

For convenience, `CBSpline` contains a set of coefficients (`hashmap_linked`) as member variable. All methods, however, can also be performed with any given set of coefficients. This enables the user to treat an object of type `CBSpline` either as a given fix function in form of a wavelet expansion, or as a wavelet basis context for an arbitrary set coefficients.

Note that the binning and the index-manipulative routines can operate on a container (`hashmap_link`) and on a wavelet expansion (`CBSpline`), see Figure 10.2.

### Control Elements

For all of the above classes, `igpm_w_lib` provides code to test and verify the performance. Control structures put at hand easy and standardized methods to produce and store reference data. They can be used to ensure the integrity of the schemes after modifying building blocks. This check can be also performed by the control tools. For the importance of such tools for the software development in environments with a high probability for changing requirements on the code, we refer to [Bec00].

### 10.2.3 Index-Manipulation

The routines in `WaveletSetManip.h` handle the various structural manipulations on the index set such as constructing the $M$-graded hull of a given index set or selecting the leaves. This can be done by combinations of two basic routines, namely, by deleting or inserting an $M$-square around the parent of a given key. We emphasize, that enforcing structure consumes a lot of computational effort and temporary storage. Both can be reduced by appropriately choosing the input set and by carefully combining the basic routines.

### 10.2.4 Binning

By setting up a `tool` object, all constants for the binning routine are at hand and ready to use. The corresponding routines, such as Bin-Count, Bin-Thresh, and Bin-Sort are implemented as member functions of the tool class.

### 10.2.5 Auxiliary Classes and Routines

#### Quadrature

The class `QuadFormula` provides numerical integration. The quadrature formulas used in a numerical scheme are very likely to be subject to frequent changes by the user. Therefore, to be as flexible as possible, the knots and weights will not be computed, but they have to be provided as data for an initialization. This is also due to the fact, that the corresponding data for a formula can be gained beforehand and hence is not time critical. Therefore, it does not matter where the knots and weight are stemming from. The source may be typically some data-base or an ad-hoc build program in a mathematical language. Streams offer a standardized efficient way to initialize `QuadFormula`, which then is ready for usage. We of course provide data-files for the Gaussian quadrature from Section 4.1.2 according to a variety of wavelet systems.

#### Frequently used Constants

Note, that for a given problem to compute most of these constants depend only on the level. As a consequence, during the computation there are relatively few different constants that are used and reused very often. Therefore, during compilation, i.e., before a code using the `igpm_w_lib` is

running, frequently used constants like, e.g., $2^{jd/2}$, $j$ denoting a wavelet level, are computed once and for all and stored in a rapidly accessible structure provided by `WaveletConst`.

### Plotting Routines

`tGridIterator` provides the user with a fast and easy way to pace multidimensional cubic grids. The tools in the file `igpm_matlab` arrange the output of all wavelet structures for display using MATLAB [Inc04].

### Periodization

The file `periodic` provides tools to form and handle 1-periodic wavelet bases from functions on the real line.

# List of Figures

# List of Algorithms

# Bibliography


[Ada00] R.A. Adams, *Sobolev spaces*, Academic Press, 1900.

[Bar01] T. Barsch, *Adaptive multiskalenverfahren für elliptische partielle differentialgleichungen – Realisierung, Umsetzung und numerische Ergebnisse*, Ph.D. thesis, RWTH Aachen, 2001.

[BBC+02] A. Barinka, T. Barsch, P. Charton, A. Cohen, S. Dahlke, W. Dahmen, and K. Urban, *Adaptive wavelet schemes for elliptic problems – implementation and numerical experiments*, SIAM J. Sci. Comput. **23 (3)** (2002), 910–939.

[BBD+02] A. Barinka, T. Barsch, S. Dahlke, M. Konik, and M. Mommer, *Some remarks on quadrature formulas for refinable functions and wavelets II: Error analysis*, Journal of Computational Analysis and Applications **4(4)** (2002), 339–362.

[BBDK01] A. Barinka, T. Barsch, S. Dahlke, and M. Konik, *Some remarks on quadrature formulas for refinable functions and wavelets*, ZAMM **81** (2001), 839–855.

[BDD03] A. Barinka, Wolfgang Dahmen, and Stephan Dahlke, *Adaptive application of operators in standard representation*, Tech. report, IGPM, RWTH Aachen, 2003, to appear in Adv. Comp. Math.

[BDS04] A. Barinka, W. Dahmen, and R. Schneider, *Fast computation of adaptive wavelet expansions*, Tech. report, IGPM, RWTH Aachen, 2004.

[Bec00] K. Beck, *Extreme programming explained. Embrace change*, Addison-Wesley USA, 2000.

[BL76] J. Berg and J. Loefstroem, *Interpolation spaces, an introd uction*, Springer, 1976.

[BV] A. Barinka and A. Voß, *The* `igpm_w_lib`, in preparation.

[CDD00] A. Cohen, W. Dahmen, and R.A. DeVore, *Adaptive wavelet methods II – beyond the elliptic case*, Tech. report, IGPM, RWTH Aachen, 2000.

[CDD01] A. Cohen, W. Dahmen, and R. DeVore, *Wavelet methods for elliptic operator equations — convergence rates*, Math. Comp. **70** (2001), 27–75.

[CDD03a] _____, *Adaptive wavelet schemes for nonlinear variational problems*, SIAM J. Numer. Anal **41(5)** (2003), 1785–1823.

[CDD03b] _____, *Adaptive wavelet techniques in numerical simulation*, Tech. report, IGPM, RWTH Aachen, 2003, to appear in *Encyclopedia of Computational Mechanics*, (R. De Borste, T. Hughes, E. Stein, eds.).

[CDD03c] A. Cohen, W. Dahmen, and R.A. DeVore, *Sparse evaluation of compositions of functions using multiscale expansions*, SIAM J. Math. Anal. **35** (2003), 279–303.

[CDD03d] _____, *Sparse evaluation of nonlinear functionals of multiscale expansions*, SIAM J. Math. Anal. **35** (2003), 279–303.

[CDDD01] A. Cohen, W. Dahmen, I. Daubechies, and R. DeVore, *Tree approximation and optimal encoding*, Applied and Computational Harmonic Analysis **11** (1001), 192–226.

[CDF92] A. Cohen, I. Daubechies, and J. Feauveau, *Bi–orthogonal bases of compactly supported wavelets*, Comm. Pure Appl. Math. **45** (1992), 485–560.

[CDP96] J.M. Carnicer, W. Dahmen, and J.M. Pena, *Local decomposition of refinable spaces and wavelets*, Appl. Comp. Harm. Anal. **3** (1996), 127–153.

[Chu92] C.K. Chui, *An introduction to wavelets*, Academic Press, Boston, 1992.

[CM00] A. Cohen and R. Masson, *Wavelet adaptive methods for second order elliptic problems, boundary conditions and domain decomposition*, Num. Math. **86** (2000), 193–238.

[CTU98] C. Canuto, A. Tabacco, and K. Urban, *Numerical solution of elliptic problems by the wavelet element method*, ENUMATH **97** (1998), 17–37.

[CTU00] ———, *The wavelet element method, part II: Realization and additional features in 2d and 3d*, Appl. Comp. Harm. Anal. **8** (2000), 123–165.

[Dah82] W. Dahmen, *Adaptive approximation by multivariate smooth splines*, J. Approx. Theory **36** (1982), 119–140.

[Dah94] ———, *Some remarks on multiscale transformations, stability and biorthogonality*, Wavelets, Images and Surface Fitting (1994), 157–188.

[Dah96] ———, *Stability of multiscale transformations*, Journal of Fourier Analysis and Applications **2** (1996), 341–361.

[Dah97] ———, *Wavelet and multiscale methods for operator equations*, Acta Mumerica **6** (1997), 55–228.

[Dah03] W. Dahmen, *Multiscale and wavelet methods for operator equations*, C.I.M.E. Lecture Notes, vol. 1825, Springer-Verald, Heidelberg, 2003.

[Dau92] I. Daubechies, *Ten lectures on wavelets*, CBMS–NSF Regional Conference Series in Applied Math, vol. 61, SIAM, Philadelphia, 1992.

[DD97] S. Dahlke and R. DeVore, *Besov regularity for elliptic boundary value problems*, Comm. Partial Differental Equations **22** (1997), 1–16.

[DeV98] R. DeVore, *Nonlinear approximation*, Acta Mumerica **7** (1998), 51–150.

[DHU00] S. Dahlke, R. Hochmuth, and K. Urban, *Adaptive wavelet methods for saddle point problems*, Mathematical Modelling and Numerical Analysis (M2AN) **34(5)** (2000), 1003–1022.

[DKU99] W. Dahmen, A. Kunoth, and K. Urban, *Biorthogonal spline–wavelets on the interval – stability and moment conditions*, Appl. Comp. Harm. Anal. **6** (1999), 132–196.

[DL93] R. DeVore and G. Lorentz, *Constructive approximation*, Grundleheren der mathematischen Wissenschaften, vol. 303, Springer Verlag, 1993.

[DM93] W. Dahmen and C.A. Micchelli, *Using the refinement equation for evaluating integrals of wavelets*, SIAM J. Numer. Anal. **30(2)** (1993), 507–537.

[DP88] R. DeVore and V. Popov, *Interpolation of besov spaces*, Trans. Amer. Math. Soc. **305** (1988), 397–414.

[DR75] P. Davis and P. Rabinowitz, *Methods of numerical integration*, Academic Presss, New York, 1975.

[DS99a] W. Dahmen and R. Schneider, *Composite wavelet bases for operator equations*, Math. Comp. **68** (1999), 1533–1567.

[DS99b] _____, *Wavelets on manifolds I: Construction and domain decomposition*, SIAM J. Math. Anal. **31** (1999), 184–230.

[DSX00] W. Dahmen, R. Schneider, and Y. Xu, *Nonlinear functions of wavelet expansions – adaptive reconstruction and fast evaluation*, Numer. Math. **86** (2000), 49–101.

[DT96] R. DeVore and V. Temlyakov, *Some remarks on greedy algorithms*, Advances in Computational Math. **5** (1996), 173–187.

[GGP00] W. Gautschi, L. Gori, and F. Pitolli, *Gauss quadrature for refinable weight functions*, Appl. Comput. Harmon. Anal. **8** (2000), 249–257.

[GW69] W. Gautschi and J.Ã. Welsch, *Calculation of gauss quadrature rules*, Math. Comp. **23** (1969), 245–260.

[Inc04] The Mathworks Inc., *Matlab, the language of technical computing*, 2004.

[Jür01] M. Jürgens, *Adaptive Multiskalenverfahren auf polygonal berandeten Gebieten*, Master's thesis, RWTH Aachen, 2001.

[KLR95] J.-P. Kahane and P.-G. Lemarié-Rieusset, *Fourier series and wavelets*, Gordon and Breach Science Publishers, Luxembourg, 1995.

[Kon01] M. Konik, *A fully discrete wavelet galerkin boundary element method in three dimensions*, Ph.D. thesis, TU Chemnitz, 2001.

[Kry62] V.I. Krylov, *Approximate calculation of integrals*, Macmillan, New York, 1962, translated by A.H. Stroud.

[LR92] P.G̃. Lemarié-Rieusset, *Analyses, multi–résolutions nonorthogonales, commutation entre projecteurs et derivation et ondelettes vecteurs à divergence nulle*, Revista Mat. Iberoamericana **8** (1992), 221–236.

[Mal89] S. Mallat, *Multiresolution approximation and wavelet orthonormal bases of $L_2(\mathbb{R})$*, Trans. Amer. Math. Soc. **315** (1989), 69–88.

[Met02] A. Metselaar, *Handling waveley expansions in numerical methods*, Ph.D. thesis, Universtiet Twente, 2002.

[Mey92] Y. Meyer, *Wavelets and operators*, Cambridge Studies in Advanced Mathematics, vol. 37, Cambridge University Press, 1992.

[Mül03] S. Müller, *Adaptive mulitscale schemes for conservation laws*, Lecture Notes in Computational Sience and Engeneering, Springer, Berlin, 2003.

[MV99] S. Müller and A. Voß, *The* `igpm_t_lib`, Tech. report, IGPM, RWTH Aachen, 1999.

[PS94a] R. Piessens and W. Sweldens, *Asymptotic error expansions of wavelet approximations of smooth functions ii*, Numer. Math. **68** (1994), 377–401.

[PS94b] ———, *Quadrature formulae and asymptotic error expansions for wavelet approximations of smooth functions*, SIAM J. Numer. Anal. **31(4)** (1994), 1240–1264.

[RS96] T. Runst and W. Sickel, *Sobolev spaces of fractional order, nemitskij operators, and nonlinear partial differential equations*, Nonlinear Analysis and Applications, De Gruyter, New York, 1996.

[Sch39] L.L. Schumaker, *Spline functions – basic theory*, Pure and applied Mathematics, Wiley-Interscience, 1939.

[SS66] A.H. Stroud and D. Secrest, *Gaussian quadrature formulas*, Prentice–Hall, New Jersey, 1966.

[Sto93] J. Stoer, *Numerische mathematik 1*, 6 ed., Springer, Berlin, 1993.

[Str71] A.H. Stroud, *Approximate calculation of multiple integrals*, Prentice–Hall, New Jersey, 1971.

[Str00] B. Stroustrup, *The C++ programming language*, Addison-Wesley, Boston, 2000.

[Sze39] G. Szegö, *Orthogonal polynomials*, American Mathematical Society Colloquium Publications, vol. 13, American Mathematical Society, Rhode Island, 1939.

[Urb95] K. Urban, *On divergence free wavelets*, Advances in Computational Mathematics **4** (1995), 51–82.

[vH04] Dimitri van Heesch, *Doxygen*, 2004, www.stack.nl/ dimitri/doxygen/.

[Woj97] P. Wojtaszczyk, *A mathematical introduction to wavelets*, Cambridge University Press, 1997.

[XZ03] Yuesheng Xu and Qingsong Zou, *Adaptive wavelet methods for elliptic operator equations with nonlinear terms*, Advances in Computational Mathematics **19(1)** (2003), 99–146.

## Lebenslauf

| | |
|---|---|
| Arne Barinka | Name |
| 27.12.1970 | Geburtsdatum |
| Trier | Geburtsort |
| | **Ausbildung** |
| 29.05.1990 | Abitur<br>Friedrich Spee-Gymnasium Trier |
| 1990-1991 | Wehrdienst |
| 1991-1993 | Grundstudium Mathematik und Physik, RWTH Aachen |
| 1993/94 | Studium Mathematik, Universita di Perugia (Italien) |
| 1994-1997 | Studium Mathematik, RWTH Aachen |
| 21.01.1997 | Diplom in Mathematik, RWTH Aachen |
| | **Beruf** |
| 1997-1999 | Sachbearbeiter für ein Projekt der Volkswagen-Stiftung<br>Institut für Geometrie und Praktische Mathematik,<br>RWTH Aachen |
| 1999-2004 | Wissenschaftlicher Angestellter<br>Institut für Geometrie und Praktische Mathematik,<br>RWTH Aachen |