FYS-3900
Master's Thesis
in Physics

# A multiwavelet approach to the direct solution of the Poisson equation: implementation and optimization

Stig Rune Jensen

November, 2009

Faculty of Science and Technology
Department of Physics and Technology
University of Tromsø

FYS-3900

Master's Thesis

in Physics

# A multiwavelet approach to the direct solution of the Poisson equation: implementation and optimization

Stig Rune Jensen

November, 2009

# Contents

# Acknowledgements

First of all I would like to thank my *real* supervisor, Ass. Prof. Luca Frediani for all help during the last two years, and for introducing me to the facinating theory of multiwavelets. I would also like to thank my *formal* supervisor Prof. Inge Røeggen, for letting me drift off to the Chemistry Department to write my thesis.

I would like to use the opportunity to thank the Department of Chemistry at the University of Tromsø for the generous financial contribution for my trip to the European Summershool of Computational Chemistry in the fall of 2009. I would also like to thank the CTCC for their financial support for various seminars and group meetings during my time as a master student, as well as for their contribution to the above mentioned summer school.

Finally, I would like to thank the people in the CTCC group, especially Dr. Jonas Jusélius, with whom I have shared office for two years. He has quite generously let me soak up some of his impressively diverse pool of knowledge, ranging from the Linux OS and computer programming, through theoretical and computational chemistry, into the fine art of bread baking, cheese-making, and cooking in general, and all the way to his bottomless well of YouTube videos and wide range of musical taste.

# Chapter 1

# Introduction

The work of this thesis has been contributing in the development of the program package MRChem, which is a code developed at the University of Tromsø [1] that is aiming at a fully numerical treatment of molecular systems, based on Density Functional Theory (DFT). There are currently a huge number of these program packages available, each with more or less distinct features, and what separates MRChem from all of these is the choice of basis functions. While traditional computational chemistry programs use Gaussian type basis sets for their efficient evaluation of two- and four-electron integrals, MRChem is based on the multiresolution wavelet basis.

Wavelet theory is a rather young field of mathematics, first appearing in the late 1980s. The initial application was in signal theory [2] but in the early 90s, wavelet-based methods started to appear for the solution of PDEs and integral equations [3][4], and in recent years for application in electronic structure calculations [5][6][7].

## The Kohn-Sham equations

In the Kohn-Sham [8] formulation of DFT the eigenvalue equations for the electronic structure can be written

$$[-\frac{1}{2}\nabla^2 + V_{eff}(\boldsymbol{r})]\psi_i(\boldsymbol{r}) = \epsilon_i\psi_i(\boldsymbol{r}) \tag{1.1}$$

where the effective potential is the collection of three terms

$$V_{eff}(\boldsymbol{r}) = V_{ext}(\boldsymbol{r}) + V_{coul}(\boldsymbol{r}) + V_{xc} \tag{1.2}$$

where the external potential $V_{ext}$ is usually just the electron-nuclear attraction, the Coulomb potential $V_{coul}$ is the electron-electron repulsion and $V_{xc}$ is the exchange-correlation potential which (in principle) includes all non-classical effects. The functional form of $V_{xc}$ is not known.

The nuclear charge distribution is a collection of point charges, and the nuclear potential has the analytical form

$$V_{nuc}(\boldsymbol{r}) = -\sum_{\alpha=1}^{N_{nuc}} \frac{Z_\alpha}{|\boldsymbol{r} - \boldsymbol{r}_\alpha|} \tag{1.3}$$

The electronic charge distribution is given by the Kohn-Sham orbitals

$$\rho(\boldsymbol{r}) = 2 \sum_{i=1}^{N_e/2} |\psi_i(\boldsymbol{r})|^2 \tag{1.4}$$

assuming a closed shell system with double occupancy. The electronic potential is now given as the solution of the Poisson equation

$$\nabla^2 V_{coul}(\boldsymbol{r}) = 4\pi\rho(\boldsymbol{r}) \tag{1.5}$$

where the orbital-dependence of the potential makes eq.(1.1) a set of non-linear equations that is usually solved self-consistently. The current work will not be concerned with the solution of the Kohn-Sham equations, but is rather a precursor to this where some building blocks required for the DFT calculations are prepared, in particular the solution of the Poisson equation.

## The Poisson equation

Solving the Poisson equation for an arbitrary charge distribution is a non-trivial task, and is of major importance in many fields of science, especially in the field of computational chemistry. A huge effort has been put into making efficient Poisson solvers, and usual real-space approaches includes finite difference (FD) and finite element (FE) methods. FD is a a grid-based method, which is solving the equations iteratively on a discrete grid of pointvalues, while FE is expanding the solution in a basis set, usually by dividing space into cubic cells and allocate a polynomial basis to each cell.

It is a well-known fact that the electronic density in molecular systems is rapidly varying in the vicinity of the atomic nuclei, and a usual problem with real-space methods is that an accurate treatment of the system requires high resolution of gridpoints (FD) or cells (FE) in the nuclear regions. Keeping this high resolution uniformly througout space would yield unnecessary high accuracy in the inter-atomic regions, and the solution of the Poisson equation for molecular systems is demanding a *multiresolution* framework in order to achieve numerical efficiency.

There are ways of resolving these issues using multigrid techniques, and a nice overview of these methods is given by Beck [9], but this thesis is concerned with a third way of doing real-space calculations, one where the multiresolution character is inherent in the theory, namely using wavelet bases.

At this point MRChem is basically a Poisson solver. It has the capabilities of representing arbitrary functions in the multiwavelet basis, and calculate the potential originating from these functions. This is the result of the work by [1]. The current work includes the implementation of some basic arithmetic operations involving these function representations, where the space adaptivity of the grid and strict error control will be important topics. We will also look at some possible optimizations of the existing code, where computational efficiency, memory requirements and linear scaling with respect to system size will be important.

The thesis is split in two parts; theory and implementation. The theory part gives a brief overview of the mathematical theory of multiwavelets, from the basic concept of multiresolution analysis, to the representation of functions and operators in the multiwavelet basis, and ultimately to the solution of the Poisson equation. The implementation part gives a short description of the data structures and algorithms used in the MRChem program, and some details of how the mathematical theory is implemented in practice. Some numerical results are given to show the capabilities and performances of the code.

# Part I

# Theory

# Chapter 2

# The multiwavelet basis

A suitable gateway to the theory of multiwavelets is through the idea of multiresolution analysis (MRA). A detailed description of MRAs can be found in Keinert [10], from which a brief summary of the key issues are given in the following. This work is concerned with orthogonal MRA only, and for a description of the general bi-orthogonal MRA the reader is referred to Keinerts book.

## 2.1 Orthogonal MRA

A multiresolution analysis is an infinite nested sequence of subspaces of $L^2(\mathbb{R})$

$$V_k^0 \subset V_k^1 \subset \cdots \subset V_k^n \subset \cdots \tag{2.1}$$

with the following properties

1. $V_k^\infty$ *is dense in* $L^2$
2. $f(x) \in V_k^n \iff f(2x) \in V_k^{n+1}$ , $0 \leq n \leq \infty$
3. $f(x) \in V_k^n \iff f(x - 2^{-n}l) \in V_k^n$ , $0 \leq l \leq (2^n - 1)$
4. There exists a function vector $\boldsymbol{\phi}$ of length $k+1$ in $L^2$ such that

   $$\{\phi_j(x) : \ 0 \leq j \leq k\}$$

   forms a basis for $V_k^0$.

This means that if we can construct a basis of $V_k^0$, which consists of only $k+1$ functions, we can construct a basis of *any* space $V_k^n$, by simple compression (by a factor of $2^n$), and translations (to all dyadic grid points at scale $n$), of the original $k+1$ functions, and by increasing the scale $n$, we are approaching a complete basis of $L^2$. Since $V_k^n \subset V_k^{n+1}$ the basis functions of $V_k^n$ can be expanded in the basis of $V_k^{n+1}$

$$\boldsymbol{\phi}_l^n(x) \stackrel{\text{def}}{=} 2^{n/2}\boldsymbol{\phi}(2^n x - l) = \sum_l H^{(l)}\boldsymbol{\phi}_l^{n+1}(x) \tag{2.2}$$

where the $H^{(l)}$s are the so-called filter matrices that describes the transformation between different spaces $V_k^n$.

The MRA is called orthogonal if

$$\langle \boldsymbol{\phi}_0^n(x), \boldsymbol{\phi}_l^n(x) \rangle = \delta_{0l} I_{k+1} \tag{2.3}$$

where $I_{k+1}$ is the $(k+1) \times (k+1)$ unit matrix, and $k+1$ is the length of the function vector. This orthogonality condition means that the functions are orthogonal both within one function vector and through all possible translations on one scale, but *not* through the different scales.

Complementary to the nested sequence of subspaces $V_k^n$, we can define another series of spaces $W_k^n$ that complements $V_k^n$ in $V_k^{n+1}$

$$V_k^{n+1} = V_k^n \oplus W_k^n \tag{2.4}$$

where there exists another function vector $\boldsymbol{\psi}$ of lenght $k+1$ that, with all its translations on scale $n$ forms a basis for $W_k^n$. Analogously to eq.(2.2) the function vector can be expanded in the basis of $V_k^{n+1}$

$$\boldsymbol{\psi}_l^n(x) \stackrel{\text{def}}{=} 2^{n/2} \boldsymbol{\psi}(2^n x - l) = \sum_l G^{(l)} \boldsymbol{\phi}_l^{n+1}(x) \tag{2.5}$$

with filter matrices $G^{(l)}$. In orthogonal MRA the functions $\boldsymbol{\psi}$ fulfill the same othogonality condition as eq.(2.3), and if we combine eq.(2.1) and eq. (2.4) we see that they must also be orthogonal with respect to different scales. Using eq.(2.4) recursively we obtain

$$V_k^n = V_k^0 \oplus W_k^0 \oplus W_k^1 \oplus \cdots \oplus W_k^{n-1} \tag{2.6}$$

which will prove to be an important relation.

## 2.2 Multiwavelets

There are many ways to choose the basis functions $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$ (which define the spanned spaces $V_k^n$ and $W_k^n$), and there have been constructed functions with a variety of properties, and we should choose the wavelet family that best suits the needs of the problem we are trying to solve. Otherwise, we could start from scratch and construct a new family, one that is custom-made for the problem at hand. Of course, this is not a trivial task, and it might prove more efficient to use an existing family, even though its properties are not right on cue.

There is a one-to-one correspondence between the basis functions $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$, and the filter matrices $H^{(l)}$ and $G^{(l)}$ used in the two-scale relation equations eq. (2.2) and eq.(2.5), and most well known wavelet families are defined only by their filter coefficients. This usually leads to non-smooth functions, like the Daubechies $D_2$ wavelet family (figure 2.1).

In the following we are taking a different, more intuitive approach, which follows the original construction of multiwavelets done by Alpert [4]. We define the *scaling space* $V_k^n$ as the space of piecewise polynomial functions

$$\begin{aligned} V_k^n \stackrel{\text{def}}{=} \; & \{f : \textit{ all polynomials of degree } \leq k \\ & \textit{on the interval } (2^{-n}l, 2^{-n}(l+1)) \\ & \textit{for } 0 \leq l < 2^n, \; f \textit{ vanishes elsewhere}\} \end{aligned} \tag{2.7}$$

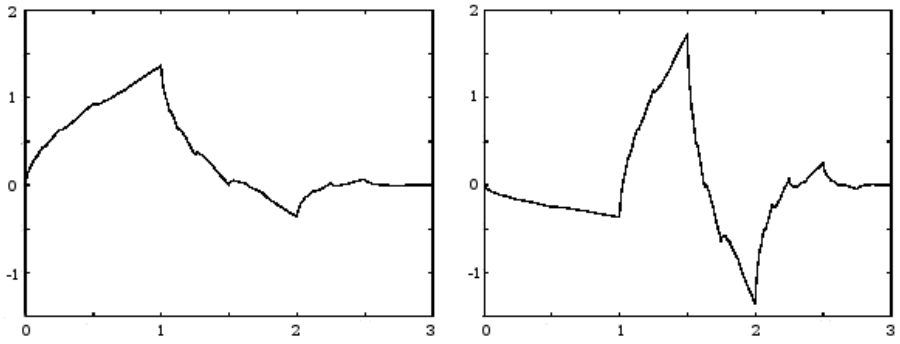


Figure 2.1: Daubechies $D_2$ scaling (left) and wavelet (right) function.

It is quite obviuos that one polynomial of degree $k$ on the interval [0,1] can be exactly reproduced by two polynomials of degree $k$, one on the interval $[0,\frac{1}{2}]$ and the other on the interval $[\frac{1}{2},1]$. The spaces $V_k^n$ hence fulfills the MRA condition eq.(2.1), and if the polynomial basis is chosen to be orthogonal, the $V_k^n$ constitutes an *orthogonal* MRA.

## 2.3 The wavelet basis

The *wavelet space* $W_k^n$ is defined, according to eq. (2.4), as the orthogonal complement of $V_k^n$ in $V_k^{n+1}$. The multiwavelet basis functions of $W_k^n$ are hence piece-wise polynomials of degree $\leq k$ on *each* of the two intervals on scale n+1 that overlaps with *one* interval on scale n. These piece-wise polynomials are then made orthogonal to a basis of $V_k^n$ and to each other. The construction of the multiwavelet basis follows exactly [4] where a simple Gram-Schmidt orthogonolization were employed to construct a basis that met the necessary orthogonality conditions. The wavelet functions for $k = 5$ are shown in figure 2.2

One important property of the wavelet basis is the number of vanishing moments. The k-th continuous moment of a function $\psi$ is defined as the integral

$$\mu_k \stackrel{\text{def}}{=} \int_0^1 x^k \psi(x) dx \tag{2.8}$$

and the function $\psi$ has $M$ vanishing moments if

$$\mu_k = 0, \qquad k = 0, \ldots, M-1$$

The vanishing moments of the *wavelet* functions gives information on the approximation order of the *scaling* functions. If the wavelet function $\psi$ has $M$ vanishing moments, any polynomial of order $\leq M-1$ can be exactly reproduced by the scaling function $\phi$, and the error in representing an arbitrary function in the scaling basis is of $M$-th order. By construction, $x^i$ is in the space $V_k^0$ for $0 \leq i \leq k$, and since $W_k^0 \perp V_k^0$, the first $k+1$ moments of $\psi_j^0$ must vanish.

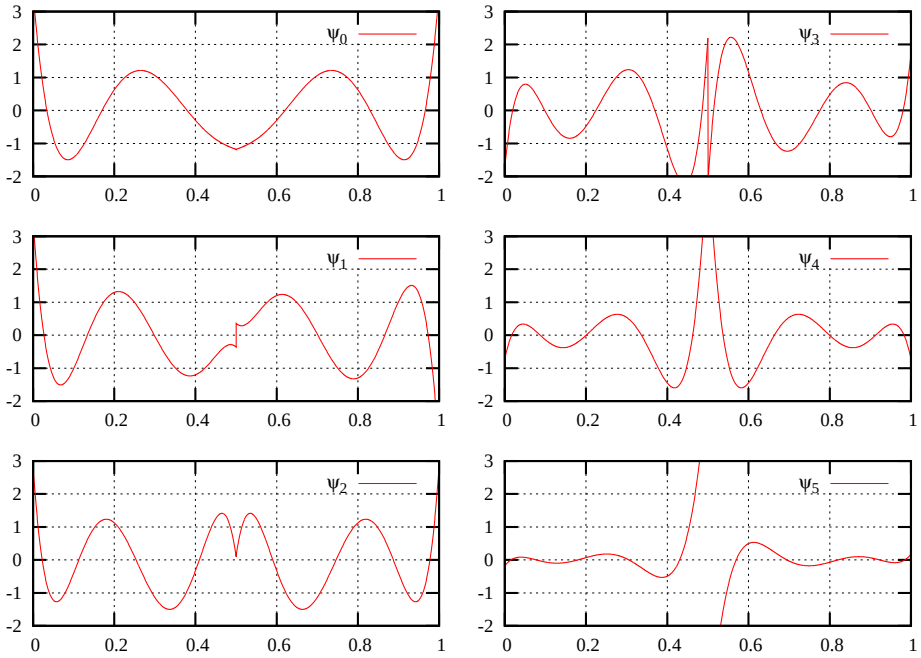


Figure 2.2: First six wavelet functions at scale zero

## 2.4 The scaling basis

The construction of the scaling functions is quite straightforward; $k+1$ suitable polynomials are chosen to span any polynomial of degree $\leq k$ on the unit interval. The total basis for $V_k^n$ is then obtained by appropriate dilation and translation of these functions. Of course, any polynomial basis can be used, the simplest of them the standard basis $\{1, x, \ldots, x^k\}$. However, this basis is not orthogonal on the unit interval and cannot be used in *orthogonal* MRA. In the following, two choices of orthogonal scaling functions will be presented, and even though they span exactly the same spaces $V_k^n$ there are some important numerical differences between the two. These differences will be considered in the implementation part of this thesis.

In order to construct a set of orthogonal polynomials we could proceed in the same manner as for the wavelet functions and do a Gram-Schmidt orthogonalization of the standard basis $\{1, x, \ldots, x^k\}$. If this is done on the interval $x \in [-1, 1]$ we end up with the Legendre polynomials $\{L_j\}_{j=0}^k$. These functions are usually normalized such that $L_j(1) = 1$ for all $j$. To make the *Legendre scaling functions* $\phi_j^L$ we transform the Legendre polynomials to the interval $x \in [0, 1]$, and $L^2$-normalize

$$\phi_j^L(x) = \sqrt{2j+1} L_j(2x-1), \ x \in [0,1] \tag{2.9}$$

The basis for the space $V_k^n$ is then made by proper dilation and translation of $\phi_j^L$. This is the original construction of scaling functions done by Alpert [4].

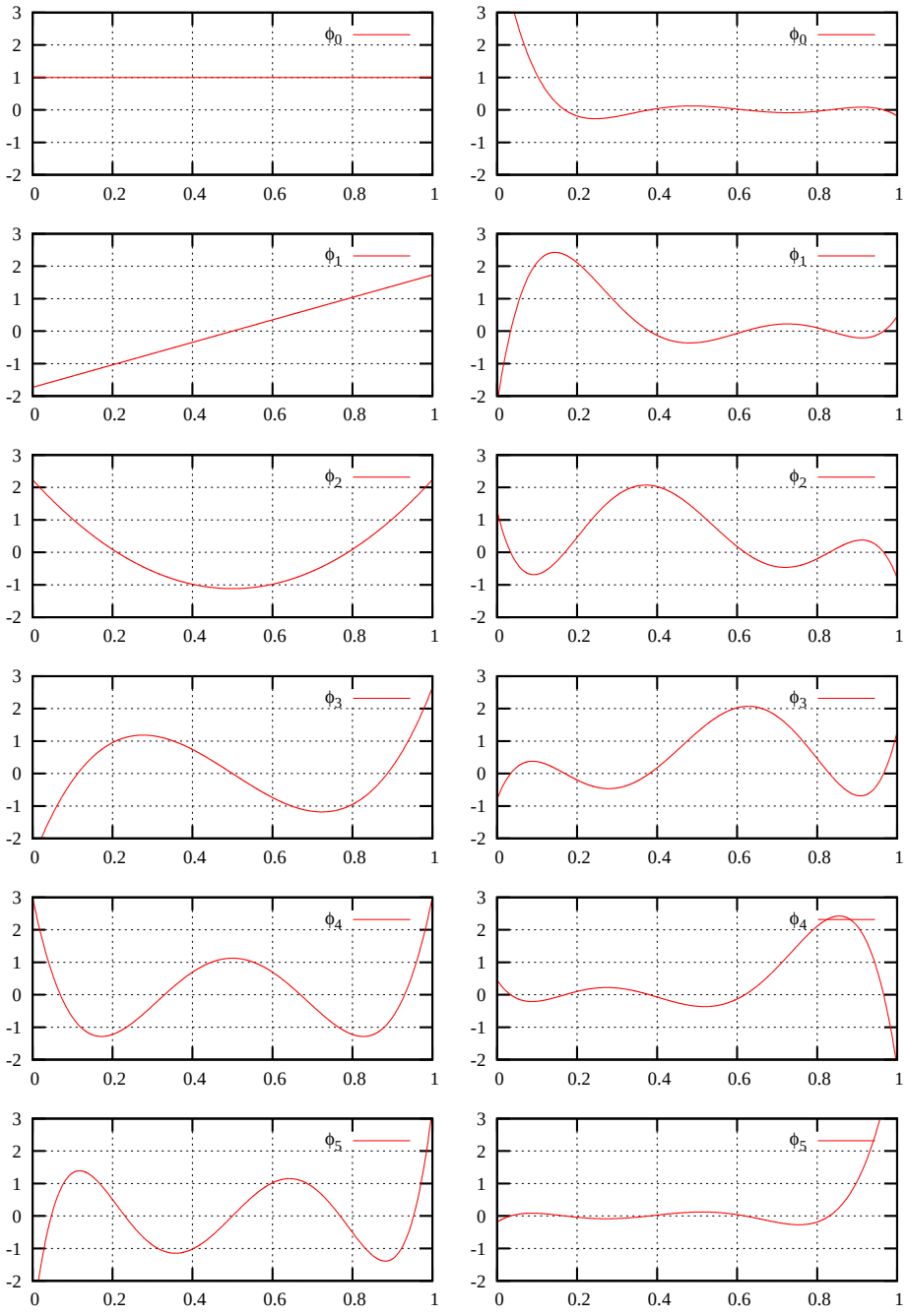


Figure 2.3: First six Legendre (left) and Interpolating (right) scaling functions at scale zero

Alpert et al. [11] presented an alternative set of scaling functions with interpolating properties. These *Interpolating scaling functions* $\phi_j^I$ are based on the Legendre scaling functions $\{\phi_j^L\}_{j=0}^k$, and the roots $\{y_j\}_{j=0}^k$ and weights $\{w_j\}_{j=0}^k$ of the Gauss-Legendre quadrature of order $k+1$, and are organized in the linear combinations

$$\phi_j^I(x) = \sqrt{w_j} \sum_{i=0}^{k_p} \phi_i^L(y_j)\phi_i^L(x), \ x \in [0,1] \tag{2.10}$$

Again the basis of $V_k^n$ is made by dilation and translation of $\phi_j^I$. The Legendre and Interpolating scaling functions of order $k = 5$ are shown in figure2.3. The construction of $\phi_j^I$ gives them the interpolating property

$$\phi_j^I(y_i) = \frac{\delta_{ji}}{\sqrt{w_i}} \tag{2.11}$$

which will prove important for numerical efficiency.

A detailed discussion on the properties of Interpolating wavelets can be found in Donoho [12], but the case of Interpolating multiwavelets is somewhat different. An important property of Interpolating wavelets is the *smoothness* of any function represented in this basis. This property stems from general Lagrange interpolation. In the multiwavelet case the interpolating property applies *within* one scaling function vector only, which means that functions represented in this basis can be discontinous in any merging point between the different translations on any scale. This is also the case for the Legendre scaling functions, and it makes differentiation awkward in these bases.

With the basis functions in place we can now use these to construct the filter matrices that fulfill the two-scale conditions eq.(2.2) and eq.(2.5). The details of this construction are given in Alpert et al. [11], and will not be presented here, but we specifically end up with four matrices $H^{(0)}, H^{(1)}, G^{(0)}$ and $G^{(1)}$, which size and contents are dependent on the order and type of scaling functions chosen. Eq.(2.2) and eq.(2.5) thus reduces to

$$\begin{aligned} \boldsymbol{\phi}_l^n &= H^{(0)}\boldsymbol{\phi}_{2l}^{n+1} + H^{(1)}\boldsymbol{\phi}_{2l+1}^{n+1} \\ \boldsymbol{\psi}_l^n &= G^{(0)}\boldsymbol{\phi}_{2l}^{n+1} + G^{(1)}\boldsymbol{\phi}_{2l+1}^{n+1} \end{aligned} \tag{2.12}$$

## 2.5 Multiwavelets in $d$ dimensions

When dealing with multidimensional multiwavelets we open a notational can of worms that easily gets confusing. The following notation is aiming to be as intuitive as possible, and is similar to the one presented in [1].

Multidimensional wavelets are usually constructed by tensor products, where the scaling space is defined as

$$V_k^{n,d} \stackrel{\text{def}}{=} \bigotimes^d V_k^n \tag{2.13}$$

The basis for this $d$-dimensional space is given as tensor products of the one-dimensional bases.

$$\Phi^n_{\boldsymbol{j},\boldsymbol{l}}(\boldsymbol{x}) = \Phi^n_{j_1 j_2 \ldots j_d, l_1 l_2 \ldots l_d}(x_1, x_2, \ldots, x_d) \stackrel{\text{def}}{=} \prod_{i=1}^{d} \phi^n_{j_i,l_i}(x_i) \tag{2.14}$$

The number of basis functions on each hypercube $\boldsymbol{l} = (l_1, l_2, \ldots, l_d)$ becomes $(k+1)^d$, while the number of such hypercubes on scale $n$ becomes $2^{dn}$, which again means that the total number of basis functions is growing exponentially with the number of dimensions.

The wavelet space can be defined using eq.(2.4)

$$V_k^{n+1,d} = \bigotimes^d V_k^{n+1} = \bigotimes^d (V_k^n \oplus W_k^n) \tag{2.15}$$

where the pure scaling term obtained when expanding the product on the right hand side of eq.(2.15) is recognized as $V_k^{n,d}$, making the wavelet space $W_k^{n,d}$ consist of all the remaining terms of the product, which are terms that contain at least one wavelet space.

To achieve a uniform notation, we can introduce a "generalized" one-dimensional wavelet function $\{\varphi^{\alpha,n}_{j,l}\}$ that, depending on the index $\alpha$ can be either the scaling or the wavelet function

$$\varphi^{\alpha_i,n}_{j_i,l_i} \stackrel{\text{def}}{=} \begin{cases} \phi^n_{j_i,l_i} & \text{if } \alpha_i = 0 \\ \psi^n_{j_i,l_i} & \text{if } \alpha_i = 1 \end{cases} \tag{2.16}$$

The wavelet functions for the $d$-dimensional space can thus be expressed as

$$\Psi^{\alpha,n}_{\boldsymbol{j},\boldsymbol{l}}(\boldsymbol{x}) = \prod_{i=1}^{d} \varphi^{\alpha_i,n}_{j_i,l_i}(x_i) \tag{2.17}$$

Where the total $\alpha$ index on $\Psi$ separates the $2^d$ different possibilities of combining scaling/wavelet functions with the same index combination $\boldsymbol{j} = (j_0, j_1, \ldots, j_k)$. $\alpha$ is given by the binary expansion

$$\alpha = \sum_{i=1}^{d} 2^{i-1}\alpha_i \tag{2.18}$$

and thus runs from 0 to $2^d - 1$. By closer inspection we see that $\alpha = 0$ recovers the pure scaling function

$$\Psi^{0,n}_{\boldsymbol{j},\boldsymbol{l}}(\boldsymbol{x}) \equiv \Phi^n_{\boldsymbol{j},\boldsymbol{l}}(\boldsymbol{x}) \tag{2.19}$$

and we will keep the notation $\Phi^n_{\boldsymbol{j},\boldsymbol{l}}$ for the scaling function, and exclude the $\alpha = 0$ term in the wavelet notation when treating multidimensional functions.

We can immediately see that the dimensionality of the wavelet space is higher than the scaling space on the same scale $n$, specifically $2^d - 1$ times higher. This must be the case in order to conserve the dimensionality through the equation

$$V_k^{n+1,d} = V_k^{n,d} \oplus W_k^{n,d} \tag{2.20}$$

since $dim(V_k^{n+1,d}) = 2^d dim(V_k^{n,d})$.

As for the monodimensional case we can define filter matrices that transform the scaling functions at scale $n+1$, $\{\Phi_{\boldsymbol{j},\boldsymbol{l}}^{n+1}\}$, into scaling and wavelet functions at scale $n$, $\{\Psi_{\boldsymbol{j},\boldsymbol{l}}^{\alpha,n}\}_{\alpha=0}^{2^d-1}$. Details of this construction can be found in [1], where the corresponding matrices are shown to be tensor products of the monodimensional matrices.

# Chapter 3

# Function representation

With the multiwavelet basis introduced, we have a hierarchy of basis sets with increasing flexibility, and we can start making approximations of functions by expanding them in these bases.

## 3.1 Function projection

We introduce the projection operator $P^n$ that projects an arbitrary function $f(x)$ onto the basis $\{\phi^n_{j,l}\}$ of the scaling space $V^n$ (in the remaining of this text the subscript $k$ of the scaling and wavelet spaces will be omitted, and it will always be assumed that we are dealing with a $k$-order polynomial basis).

$$f(x) \approx P^n f(x) \stackrel{\text{def}}{=} f^n(x) = \sum_{l=0}^{2^n-1} \sum_{j=0}^{k} s^{n,f}_{j,l} \phi^n_{j,l}(x) \tag{3.1}$$

where the expansion coefficients $s^{n,f}_{j,l}$, the so-called *scaling* coefficients, are obtained by the usual integral

$$s^{n,f}_{j,l} \stackrel{\text{def}}{=} \langle f, \phi^n_{j,l} \rangle = \int_0^1 f(x) \phi^n_{j,l}(x) dx \tag{3.2}$$

If this approximation turns out to be too crude, we double our basis set by increasing the scale and perform the projection $P^{n+1}$. This can be continued until we reach a scale $N$ where we are satisfied with the overall accuracy of $f^N$ relative to the true function $f$.

## 3.2 Multiresolution functions

We can also introduce the projection operator $Q^n$ that projects $f(x)$ onto the wavelet basis of the space $W^n$

$$Q^n f(x) \stackrel{\text{def}}{=} df^n(x) = \sum_{l=0}^{2^n-1} \sum_{j=0}^{k} d^{n,f}_{j,l} \psi^n_{j,l}(x) \tag{3.3}$$

where the *wavelet* coefficients are given as

$$d_{j,l}^{n,f} \stackrel{\text{def}}{=} \langle f, \psi_{j,l}^n \rangle = \int_0^1 f(x)\psi_{j,l}^n(x)dx \tag{3.4}$$

According to eq.(2.4) we have the following relationship between the projection operators

$$P^{n+1} = P^n + Q^n \tag{3.5}$$

and it should be noted that $df^n$ is *not* an approximation of $f$, but rather the *difference* between two approximations. We know that the basis of $V^\infty$ forms a complete set in $L^2$, which implies that $P^\infty$ must be the identity operator. Combining this with eq.(3.5) we can decompose the function $f$ into multiresolution contributions

$$\begin{aligned} f(x) &= P^\infty f(x) \\ &= P^0 f(x) + \sum_{n=0}^{\infty} Q^n f(x) \\ &= \sum_{j=0}^{k} s_{j,0}^{0,f}\phi_{j,0}^0(x) + \sum_{n=0}^{\infty}\sum_{l=0}^{2^n-1}\sum_{j=0}^{k} d_{j,l}^{n,f}\psi_{j,l}^n(x) \end{aligned} \tag{3.6}$$

This expansion is exact, but contains infinitely many coefficients. If we want to make approximations of the function $f$ we must truncate the infinite sum in the wavelet expansion at some finest scale $N$

$$f(x) \approx f^N(x) = \sum_{j=0}^{k} s_{j,0}^{0,f}\phi_{j,0}^0(x) + \sum_{n=0}^{N-1}\sum_{l=0}^{2^n-1}\sum_{j=0}^{k} d_{j,l}^{n,f}\psi_{j,l}^n(x) \tag{3.7}$$

This expansion is completely equivalent to eq.(3.1) (with $n = N$) both in terms of accuracy and in number of expansion coefficients. However, as we have seen, the wavelet projections $df^n$ are defined as the difference between two consecutive scaling projections, and since we know, for $L^2$ functions, that the scaling projections is approaching the exact function $f$, we also know that the *wavelet* projections must approach zero. This means that as we increase the accuracy by increasing $N$ in eq.(3.7) we know that the wavelet terms we are introducing will become smaller and smaller, and we can choose to keep only the terms that are above some threshold. This makes the multiresolution representation preferred since it allows for strict error control with a minimum of expansion coefficients. This is the heart of wavelet theory.

## Wavelet transforms

The filter matrices $H^{(0)}, H^{(1)}, G^{(0)}$ and $G^{(1)}$ allow us to change between the representations eq.(3.1) and eq.(3.7). The two-scale relations of the scaling and wavelet functions eq.(2.12) apply directly to the scaling coefficient vectors $\boldsymbol{s}_l^n$, and wavelet coefficient vectors $\boldsymbol{d}_l^n$, and the coefficients on scale $n$ are obtained by the coefficients on scale $n+1$ through

$$\begin{aligned} \boldsymbol{s}_l^n &= H^{(0)}\boldsymbol{s}_{2l}^{n+1} + H^{(1)}\boldsymbol{s}_{2l+1}^{n+1} \\ \boldsymbol{d}_l^n &= G^{(0)}\boldsymbol{s}_{2l}^{n+1} + G^{(1)}\boldsymbol{s}_{2l+1}^{n+1} \end{aligned} \tag{3.8}$$

This transformation is called forward wavelet transform or wavelet decomposition of the scaling coefficients on scale $n+1$. By doing this decomposition recursively we can get from eq.(3.1) to eq.(3.7). Rearranging eq.(3.8) we arrive at the backward wavelet transform or wavelet reconstruction

$$\begin{aligned} \boldsymbol{s}_{2l}^{n+1} &= H^{(0)T}\boldsymbol{s}_l^n + G^{(0)T}\boldsymbol{d}_l^n \\ \boldsymbol{s}_{2l+1}^{n+1} &= H^{(1)T}\boldsymbol{s}_l^n + G^{(1)T}\boldsymbol{d}_l^n \end{aligned} \tag{3.9}$$

where the transposed filter matrices are used.

It should be emphasized that these wavelet transforms do not change the *function* that is represented by these coefficients, they just change the *basis set* used to represent the exact same function. This means that the accuracy of the representation is determined only by the finest scale of which the coefficients were obtained by *projection*, and a backward wavelet transform beyond this scale will not improve our approximation (but it will increase the number of expansion coefficients).

The true power of multiwavelets is that, by truncating eq.(3.7) *locally* whenever the wavelet coefficients are sufficiently small, we end up with a space adaptive basis expansion, in that we are focusing the basis functions in the regions of space where they are most needed.

## 3.3 Multiresolution functions in $d$ dimensions

The multidimensional function representation is obtained similarly to eq.(3.1) by projection onto the multidimensional basis eq.(2.14)

$$f(\boldsymbol{x}) \approx f^n(\boldsymbol{x}) = \sum_{\boldsymbol{l}} \sum_{\boldsymbol{j}} s_{\boldsymbol{j},\boldsymbol{l}}^{n,f} \Phi_{\boldsymbol{j},\boldsymbol{l}}^n(\boldsymbol{x}) \tag{3.10}$$

where the sums are over all possible translation vectors $\boldsymbol{l} = (l_1, \ldots, l_d)$ for $0 \le l_i \le 2^n - 1$, and all possible scaling function combinations $\boldsymbol{j} = (j_1, \ldots, j_d)$ for $0 \le j_i \le k$. The scaling coefficients are obtained by the multidimensional integral

$$s_{\boldsymbol{j},\boldsymbol{l}}^{n,f} \stackrel{\text{def}}{=} \langle f, \Phi_{\boldsymbol{j},\boldsymbol{l}}^n \rangle = \int_{[0,1]^d} f(\boldsymbol{x}) \Phi_{\boldsymbol{j},\boldsymbol{l}}^n(\boldsymbol{x}) d\boldsymbol{x} \tag{3.11}$$

The wavelet components are given as

$$df^n(\boldsymbol{x}) = \sum_{\boldsymbol{l}} \sum_{\boldsymbol{j}} \sum_{\alpha=1}^{2^d-1} d_{\boldsymbol{j},\boldsymbol{l}}^{\alpha,n,f} \Psi_{\boldsymbol{j},\boldsymbol{l}}^{\alpha,n}(\boldsymbol{x}) \tag{3.12}$$

where the $\boldsymbol{l}$ and $\boldsymbol{j}$ summations are the same as in eq.(3.10), and the $\alpha$ sum is over all combinations of scaling/wavelet functions (excluding the pure scaling $\alpha = 0$). The expansion coefficients are obtained by the multidimensional projection

$$d_{\boldsymbol{j},\boldsymbol{l}}^{\alpha,n,f} \stackrel{\text{def}}{=} \langle f, \Psi_{\boldsymbol{j},\boldsymbol{l}}^{\alpha,n} \rangle = \int_{[0,1]^d} f(\boldsymbol{x}) \Psi_{\boldsymbol{j},\boldsymbol{l}}^{\alpha,n}(\boldsymbol{x}) d\boldsymbol{x} \tag{3.13}$$

We can express a multidimensional function $f(\boldsymbol{x})$ by its multiresolution contributions as for the monodimensional case

$$f^N(\boldsymbol{x}) = \sum_{\boldsymbol{j}} s^{0,f}_{\boldsymbol{j},\mathbf{0}} \Phi^0_{\boldsymbol{j},\mathbf{0}}(\boldsymbol{x}) + \sum_{n=0}^{N-1} \sum_{\boldsymbol{l}} \sum_{\boldsymbol{j}} \sum_{\alpha=1}^{2^d-1} d^{\alpha,n,f}_{\boldsymbol{j},\boldsymbol{l}} \Psi^{\alpha,n}_{\boldsymbol{j},\boldsymbol{l}}(\boldsymbol{x}) \tag{3.14}$$

### Wavelet transforms in $d$ dimensions

The $d$-dimensional filter matrices were obtained by tensor products of the monodimensional filters. This means that by the tensor structure of the multidimensional basis, we can perform the wavelet transform one dimension at the time. This allows for the situation where the basis is represented at different scales in different directions. Specifically, in two dimensions, the way to go from the scaling plus wavelet representation on the square $\boldsymbol{l}$ at scale $n$ to the pure scaling representation in the four subsquares of $\boldsymbol{l}$ at scale $n+1$, we perform the transform first in one direction by dividing the square into two rectangular boxes, and then the other direction, dividing the two rectangels into four squares.

One important implication of this tensor structure is that the work done in the $d$-dimensional transform scales linearly in the number of dimensions. If the full $d$-dimensional filter matrix had been applied, the work would have scaled as the power of the dimension, hence limiting the practical use in higher dimensions. A more rigorous treatment of the multidimensional wavelet transforms can be found in [13].

## 3.4 Addition of functions

The addition of functions in the multiwavelet basis is quite straightforward, since it is represented by the mappings

$$\begin{aligned} V^n + V^n &\to V^n \\ W^n + W^n &\to W^n \end{aligned} \tag{3.15}$$

This basically means that the projection of the sum equals the sum of the projections. In the polynomial basis this is simply the fact that the sum of two $k$-order polynomials is still a $k$-order polynomial.

Consider the equation $h(x) = f(x) + g(x)$. Projecting $h$ onto the scaling space yields

$$\begin{aligned} h^n(x) &= P^n h(x) \\ &= P^n \left(f(x) + g(x)\right) \\ &= P^n f(x) + P^n g(x) \\ &= f^n(x) + g^n(x) \end{aligned} \tag{3.16}$$

and similarly

$$dh^n(x) = df^n(x) + dg^n(x) \tag{3.17}$$

The functions $f(x)$ and $g(x)$ are expanded in the same basis set and the sum simplifies to an addition of coefficients belonging to the same basis function and can be done one scale at the time.

$$\begin{aligned} h^n(x) &= f^n(x) + g^n(x) \\ &= \sum_{l=0}^{2^n-1} \sum_{j=0}^{k} s_{j,l}^{n,f} \phi_{j,l}^n(x) + \sum_{l=0}^{2^n-1} \sum_{j=0}^{k} s_{j,l}^{n,g} \phi_{j,l}^n(x) \\ &= \sum_{l=0}^{2^n-1} \sum_{j=0}^{k} \left( s_{j,l}^{n,f} + s_{j,l}^{n,g} \right) \phi_{j,l}^n(x) \end{aligned} \tag{3.18}$$

and similarly

$$dh^n(x) = \sum_{l=0}^{2^n-1} \sum_{j=0}^{k} \left( d_{j,l}^{n,f} + d_{j,l}^{n,g} \right) \psi_{j,l}^n(x) \tag{3.19}$$

The generalization to multiple dimensions is trivial, and will not be discussed at this point.

## 3.5 Multiplication of functions

Multiplication of functions in the multiwavelet basis is somewhat more involved than addition. The reason for this is that, in contrast to eq.(3.15), the product is represented by the mapping [14]

$$V_k^n \times V_k^n \to V_{2k}^n \tag{3.20}$$

This means that the product of two functions falls outside of the MRA and needs to be projected back onto the scaling space sequence. This is easily seen in our polynomial basis; the product of two piecewise degree $\leq k$ polynomials is a piecewise polynomial of degree $\leq 2k$, which cannot be exactly reproduced by any piecewise degree $\leq k$ polynomial (other than in the limit $V^\infty$). In particular this means that the product of two functions on a given scale "spills over" into the finer scales, in the sense that

$$V^n \times V^n \to V^n \oplus \bigoplus_{n'=n}^{\infty} W^{n'} \tag{3.21}$$

Working with a finite precision it is desirable to make the product as accurate as each of the multiplicands. This is done by terminating the sum in eq.(3.21) at a sufficiently large scale $N$.

$$V^n \times V^n \to V^n \oplus \bigoplus_{n'=n}^{N-1} W^{n'} = V^N \tag{3.22}$$

Assume now that $n$ is the finest scale present in either of the multiplicands, and $N > n$ is the finest scale present in the product. An algorithm to determine the maximum scale $N$ needed in the result will be presented in the implementation

part of this thesis, and in the following it is simply assumed that $N$ is known a priori. We know that

$$V^n \subset V^{n+1} \subset \cdots \subset V^N$$

which means that the multiplication could just as well have been written

$$V^N \times V^N \to V^N$$

where the representations of the multiplicands on scale $N$ is obtained by a series of backward wavelet transforms. As pointed out before this will result in an increase in the number of coefficients without changing the *information* that we are able to extract from these functions. This *oversampling* of the multiplicands allow us to relate the scaling coefficients of the product on scale $N$ to the coefficiens of the multiplicands on the same scale.

Finally, when we have obtained the scaling coefficients of the product on scale $N$ we do a forward wavelet transform to obtain wavelet coefficients on the coarser scales. We can now throw away all wavelet terms that are sufficiently small, and we have an adaptive representation of the product.

## Scaling function multiplication

Consider the equation $h(x) = f(x) \times g(x)$. We want to represent the function $h(x)$ at some scale $N$

$$\begin{aligned} h^N(x) &= P^N h(x) \\ &= P^N \left( f(x) \times g(x) \right) \end{aligned} \tag{3.23}$$

However, as we have seen, the projection of the product eq.(3.23) does *not* equal the product of the projections, and we will actually have to perform this projection. We will of course not have available the functions $f(x)$ and $g(x)$ analytically, so the best thing we can do is

$$h^N(x) \approx P^N \left( f^N(x) \times g^N(x) \right) \stackrel{\text{def}}{=} P^N \tilde{h}(x) \tag{3.24}$$

The scaling coefficients of the product is approximated by the projection integral

$$\begin{aligned} s^{N,h}_{j^h,l} &\approx \int_0^1 \tilde{h}(x) \phi^N_{j^h,l}(x) dx \\ &= \int_0^1 f^N(x) g^N(x) \phi^N_{j^h,l}(x) dx \\ &= \int_0^1 \left( \sum_{j^f=0}^{k} s^{N,f}_{j^f,l} \phi^N_{j^f,l}(x) \right) \left( \sum_{j^g=0}^{k} s^{N,g}_{j^g,l} \phi^N_{j^g,l}(x) \right) \phi^N_{j^h,l}(x) dx \\ &= 2^N \sum_{j^f=0}^{k} \sum_{j^g=0}^{k} s^{N,f}_{j^f,l} s^{N,g}_{j^g,l} \int_0^1 \phi^0_{j^f,0}(x) \phi^0_{j^g,0}(x) \phi^0_{j^h,0}(x) dx \end{aligned} \tag{3.25}$$

and if the scale $N$ is chosen properly, the error in the coefficients can be made negligeable compared to the total error in $h^N(x)$. We see that the multiplication is related to a limited number of integrals, specifically $(k+1)^3$ different integrals involving scale zero scaling functions, regardless of how many total basis functions being used. A lot of these integrals will again be identical because of symmetry.

## Multiplication in $d$ dimensions

The generalization to multiple dimensions is quite straightforward, using the notation of eq.(2.14)

$$s^{N,h}_{\boldsymbol{j}^h,\boldsymbol{l}} = 2^N \sum_{\boldsymbol{j}^f} \sum_{\boldsymbol{j}^g} s^{N,f}_{\boldsymbol{j}^f,\boldsymbol{l}} s^{N,g}_{\boldsymbol{j}^g,\boldsymbol{l}} \int_{[0,1]^d} \Phi^0_{\boldsymbol{j}^f,\mathbf{0}}(\boldsymbol{x}) \Phi^0_{\boldsymbol{j}^g,\mathbf{0}}(\boldsymbol{x}) \Phi^0_{\boldsymbol{j}^h,\mathbf{0}}(\boldsymbol{x}) d\boldsymbol{x} \tag{3.26}$$

The only difference consists in the number of integrals, which grows exponentially in the number of dimensions. The multidimensional integral can however be decomposed into a product of monodimensional ones

$$\begin{aligned} &\int_{[0,1]^d} \Phi^0_{\boldsymbol{j}^f,\mathbf{0}}(\boldsymbol{x}) \Phi^0_{\boldsymbol{j}^g,\mathbf{0}}(\boldsymbol{x}) \Phi^0_{\boldsymbol{j}^h,\mathbf{0}}(\boldsymbol{x}) d\boldsymbol{x} \\ &\qquad = \prod_{i=1}^{d} \int_0^1 \phi^0_{j^f_i,0}(x_i) \phi^0_{j^g_i,0}(x_i) \phi^0_{j^h_i,0}(x_i) dx_i \end{aligned} \tag{3.27}$$

and we have again related all the integrals to the same small set of $(k+1)^3$ different integrals, even though the total number of basis functions quickly becomes millions and billions in several dimensions. However, the summations in eq.(3.26) runs over all $(k+1)^d$ different scaling function combinations of both $f$ and $g$, and the multiplication still seem to be a considerable task in multiple dimensions.

# Chapter 4

# Operator representation

When we now have a way of expressing an arbitrary function in terms of the multiwavelet basis, and we have the possibility of doing some basic arithmetic operations with these function representations, the next step should be to be able to apply operators to these functions. Specifically, we want to be able to compute the expansion coefficients of a function $g(x)$, given the coefficients of $f(x)$ based on the equation

$$[Tf](x) = g(x) \tag{4.1}$$

## 4.1 Operator projection

When applying the operator we will only have an approximation of the function $f(x)$ available

$$[TP^n f](x) = \tilde{g}(x) \tag{4.2}$$

and we can only obtain in the projected solution

$$[P^n T P^n f](x) = P^n \tilde{g}(x) \tag{4.3}$$

Using the fundamental property of projection operators $P^n P^n = P^n$ we get

$$[P^n T P^n P^n f](x) = P^n \tilde{g}(x) \tag{4.4}$$

We now define the projection of the operator $T$ on scale $n$ as

$$T \sim {}^nT^n \stackrel{\text{def}}{=} P^n T P^n \tag{4.5}$$

This approximation makes sense since $\lim_{n\to\infty} P^n = 1$. We can now represent the entire operation on scale $n$

$${}^nT^n f^n = \tilde{g}^n \tag{4.6}$$

Here we should note the difference between $\tilde{g}^n$ and $g^n$ in that $\tilde{g}^n$ is *not* the projection of the true function $g$, but rather the projection of the *true* $T$ operating on the *projected* $f$, and one should be concerned of whether the error $|\tilde{g}^n - g^n|$ is comparable to $|g - g^n|$, but it can be shown [1] that this will not be a problem if $f$ and $g$ have comparable norms.

## 4.2 Multiresolution operators

Making use of eq.(4.5) and eq.(3.5) we can decompose the operator into multiresolution contributions

$$\begin{aligned} T &= P^{\infty}TP^{\infty} \\ &= P^0TP^0 + \sum_{n=0}^{\infty}(P^{n+1}TP^{n+1} - P^nTP^n) \\ &= P^0TP^0 + \sum_{n=0}^{\infty}[(P^{n+1} - P^n)T(P^{n+1} - P^n) + \\ &\qquad (P^{n+1} - P^n)TP^n + P^nT(P^{n+1} - P^n)] \\ &= P^0TP^0 + \sum_{n=0}^{\infty}[Q^nTQ^n + Q^nTP^n + P^nTQ^n] \end{aligned} \tag{4.7}$$

and we simplify the notation with the following definitions, eq.(4.5) is repeated for clearity

$$\begin{aligned} {}^nA^{n'} &\stackrel{\text{def}}{=} Q^nTQ^{n'} : W^{n'} \to W^n \\ {}^nB^{n'} &\stackrel{\text{def}}{=} Q^nTP^{n'} : V^{n'} \to W^n \\ {}^nC^{n'} &\stackrel{\text{def}}{=} P^nTQ^{n'} : W^{n'} \to V^n \\ {}^nT^{n'} &\stackrel{\text{def}}{=} P^nTP^{n'} : V^{n'} \to V^n \end{aligned} \tag{4.8}$$

By truncating the sum in eq.(4.7) we get a multiresolution representation of the operator with finite precision

$$T \approx {}^NT^N = {}^0T^0 + \sum_{n=0}^{N-1}({}^nA^n + {}^nB^n + {}^nC^n) \tag{4.9}$$

### Standard representation

Suppose we have some a priori knowledge that the resulting function $g$ is required to be refined to some global finest scale $N$ in order to satisfy some accuracy condition. The matrix representation of the operation on this scale is simply

$$\begin{pmatrix} & & \\ & {}^NT^N & \\ & & \end{pmatrix} \begin{pmatrix} \\ f^N \\ \\ \end{pmatrix} = \begin{pmatrix} \\ g^N \\ \\ \end{pmatrix} \tag{4.10}$$

where the representations of $T$ and $f$ on scale $N$ are obtained by wavelet transform from their respective finest scales. This matrix equation describes the entire operation, and provided the scale $N$ has been chosen properly, the resulting function $g$ can be represented with the same accuracy as $f$. An adaptive representation of $g$ is obtained by performing a wavelet decomposition of $g^N$ into its multiresolution components, throwing away all wavelet terms that are sufficiently small.

There is (at least) one problem with this matrix representation; the matrix ${}^N T^N$ is dense, in the sense that it has generally only non-vanishing entries. This is a numerical problem more that a mathematical one, and will lead to algorithms that scale quadratically in the number of basis functions in the system, and one of the main prospects of wavelet theory is to arrive at fast (linear scaling) algorithms.

The way to approach this holy grail of numerical mathematics is to realize that the matrices $A$, $B$ and $C$ will *not* be dense (at least for the type of operators treated in this work), but rather have a band-like structure where their elements are rapidly decaying away from their diagonals. The reason for this bandedness of the matrices can be found in [3] and will not be discussed here, it suffices to say that it stems from the vanishing moments property of the wavelet functions.

The way to achieve a banded strucure of the operator is thus to decompose it according to eq.(4.9)

$$ {}^N T^N = {}^{N-1}T^{N-1} + {}^{N-1}A^{N-1} + {}^{N-1}B^{N-1} + {}^{N-1}C^{N-1} \tag{4.11} $$

The functions $f$ and $g$ can be decomposed to scale $N-1$ by simple filter operations eq.(3.8). According to eq.(4.8) ${}^n T^n$ and ${}^n C^n$ produce the scaling part of $g$, acting on the scaling and wavelet parts of $f$, respectively. Similarly, ${}^n A^n$ and ${}^n B^n$ produce the wavelet part of $g$, by acting on the wavelet and scaling parts of $f$, respectively. The matrix equation eq.(4.10) can thus be decomposed as

$$ \left(\begin{array}{c|c} {}^{N-1}T^{N-1} & {}^{N-1}C^{N-1} \\ \hline {}^{N-1}B^{N-1} & {}^{N-1}A^{N-1} \end{array}\right) \left(\begin{array}{c} f^{N-1} \\ \hline df^{N-1} \end{array}\right) = \left(\begin{array}{c} g^{N-1} \\ \hline dg^{N-1} \end{array}\right) \tag{4.12} $$

where the size of the total matrix is unchanged. What has been achieved by this decomposition is a banded structure in three of its four components, leaving only the ${}^{N-1}T^{N-1}$ part dense. We can now do the same decomposition of this ${}^{N-1}T^{N-1}$ into more banded submatrices. The function components $f^{N-1}$

and $g^{N-1}$ need to be decomposed as well. To keep everything consistent the ${}^{N-1}B^{N-1}$ and ${}^{N-1}C^{N-1}$ parts of the operator will have to be transformed accoringly. To proceed from here we need the following relations

$$\begin{aligned} {}^{n}B^{n} &= Q^{n}TP^{n} \\ &= Q^{n}T(P^{n-1}+Q^{n-1}) \\ &= Q^{n}TP^{n-1}+Q^{n}TQ^{n-1} \\ &= {}^{n}B^{n-1}+{}^{n}A^{n-1} \end{aligned} \tag{4.13}$$

and similarly

$$\begin{aligned} {}^{n}C^{n} &= P^{n}TQ^{n} \\ &= (P^{n-1}+Q^{n-1})TQ^{n} \\ &= P^{n-1}TQ^{n}+Q^{n-1}TQ^{n} \\ &= {}^{n-1}C^{n}+{}^{n-1}A^{n} \end{aligned} \tag{4.14}$$

which is the exact change in the operator that is taking place when we decompose $f^n$ into $f^{n-1}+df^{n-1}$ and $g^n$ into $g^{n-1}+dg^{n-1}$. The matrix equation will now look like

$$\left(\begin{array}{c|c|c} {}^{N-2}T^{N-2} & {}^{N-2}C^{N-2} & {}^{N-2}C^{N-1} \\ \hline {}^{N-2}B^{N-2} & {}^{N-2}A^{N-2} & {}^{N-2}A^{N-1} \\ \hline {}^{N-1}B^{N-2} & {}^{N-1}A^{N-2} & {}^{N-1}A^{N-1} \end{array}\right)\left(\begin{array}{c} f^{N-2} \\ \hline df^{N-2} \\ \hline df^{N-1} \end{array}\right)=\left(\begin{array}{c} g^{N-2} \\ \hline dg^{N-2} \\ \hline dg^{N-1} \end{array}\right) \tag{4.15}$$

We can develop this transformation recursively until we reach the coarsest scale. This multiresolution matrix representation of the operator is called the standard representation. Symbolically, we can do this decomposition of eq.(4.9) by recursive application of eq.(4.13) and eq.(4.14) where we shift more of the operator

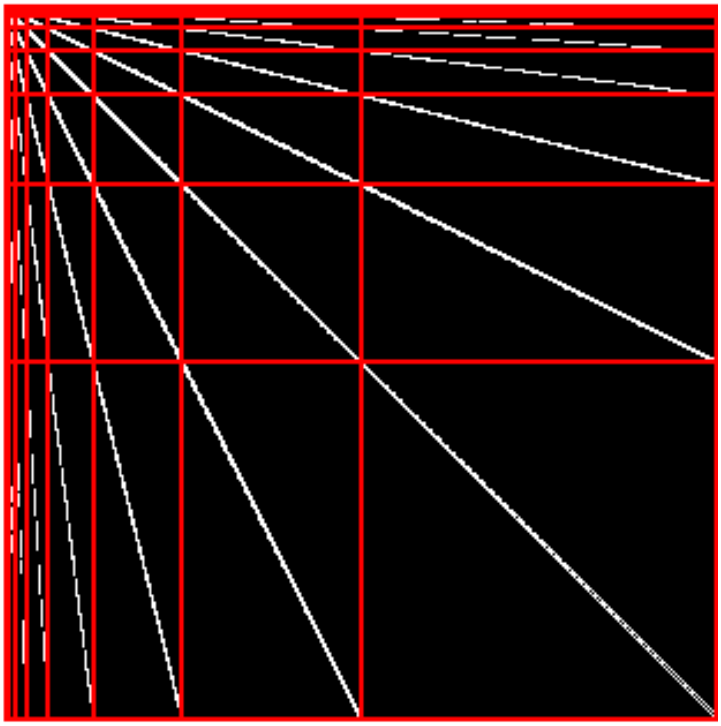

Figure 4.1: Banded structure of the standard operator matrix.

into extremely narrow banded $A$-character.

$$\begin{aligned}
{}^{N}T^{N} &= {}^{0}T^{0} + \sum_{n=0}^{N-1} {}^{n}C^{n} + \sum_{n=0}^{N-1} {}^{n}B^{n} + \sum_{n=0}^{N-1} {}^{n}A^{n} \\
&= {}^{0}T^{0} + \sum_{n=0}^{N-1} \left( {}^{0}C^{n} + \sum_{n'<n} {}^{n'}A^{n} \right) + \\
&\qquad \sum_{n=0}^{N-1} \left( {}^{n}B^{0} + \sum_{n'>n} {}^{n'}A^{n} \right) + \sum_{n=0}^{N-1} {}^{n}A^{n} \\
&= {}^{0}T^{0} + \sum_{n=0}^{N-1} {}^{0}C^{n} + \sum_{n=0}^{N-1} {}^{n}B^{0} + \sum_{n=0}^{N-1} \sum_{n'=0}^{N-1} {}^{n}A^{n'}
\end{aligned} \tag{4.16}$$

By decomposing the full operator into its multiresolution contributions we have obtained a sparse representation of the operator. The operator matrix will specifically have a "finger" structure, with a small number of bands of contributing terms, all emanating from the top-left corner, see figure 4.1.

## 4.3 Integral operators

We now turn our attention to a specific type of operator; the one-dimensional integral operator given in the form

$$[Tf](x) = \int K(x,y)f(y)dy \tag{4.17}$$

where $K$ is the two-dimensional operator kernel. The first step is to expand the kernel in the multiwavelet basis

$$K^N(x,y) = \sum_{l_x,l_y} \boldsymbol{\tau}^{N,N}_{l_x l_y} \boldsymbol{\phi}^N_{l_x}(x)\boldsymbol{\phi}^N_{l_y}(y) \tag{4.18}$$

where the expansion coefficients are given by the integrals

$$\boldsymbol{\tau}^{n_x,n_y}_{l_x l_y} = \int\int K(x,y)\boldsymbol{\phi}^{n_x}_{l_x}(x)\boldsymbol{\phi}^{n_y}_{l_y}(y)dxdy \tag{4.19}$$

Inserting eq.(4.18) into eq.(4.17) yields

$$\begin{aligned} {}^N T^N f^N(x) &= \int \left( \sum_{l_x,l_y} \boldsymbol{\tau}^{N,N}_{l_x l_y} \boldsymbol{\phi}^N_{l_x}(x)\boldsymbol{\phi}^N_{l_y}(y) \right) f(y)dy \\ &= \sum_{l_x,l_y} \boldsymbol{\tau}^{N,N}_{l_x l_y} \boldsymbol{\phi}^N_{l_x}(x) \int f(y)\boldsymbol{\phi}^N_{l_y}(y)dy \end{aligned} \tag{4.20}$$

where the last integral is recognized as the scaling coefficients of $f$

$${}^N T^N f^N(x) = \sum_{l_x,l_y} \boldsymbol{\tau}^{N,N}_{l_x l_y} \boldsymbol{\phi}^N_{l_x}(x)\boldsymbol{s}^{N,f}_{l_y} \tag{4.21}$$

We can now identify $\boldsymbol{\tau}^{N,N}_{l_x l_y}$ as the matrix elements of ${}^N T^N$ and eq.(4.21) is the matrix equation eq.(4.10) written explicitly. As pointed out, the matrix ${}^N T^N$ is dense and we would generally have to keep all the terms in eq.(4.21), therefore we want to decompose it to contributions on coarser scales. We introduce the following definitions, eq.(4.19) is repeated for clearity

$$\begin{aligned} \boldsymbol{\tau}^{n_x,n_y}_{l_x l_y} &= \int\int K(x,y)\boldsymbol{\phi}^{n_x}_{l_x}(x)\boldsymbol{\phi}^{n_y}_{l_y}(y)dxdy \\ \boldsymbol{\gamma}^{n_x,n_y}_{l_x l_y} &= \int\int K(x,y)\boldsymbol{\phi}^{n_x}_{l_x}(x)\boldsymbol{\psi}^{n_y}_{l_y}(y)dxdy \\ \boldsymbol{\beta}^{n_x,n_y}_{l_x l_y} &= \int\int K(x,y)\boldsymbol{\psi}^{n_x}_{l_x}(x)\boldsymbol{\phi}^{n_y}_{l_y}(y)dxdy \\ \boldsymbol{\alpha}^{n_x,n_y}_{l_x l_y} &= \int\int K(x,y)\boldsymbol{\psi}^{n_x}_{l_x}(x)\boldsymbol{\psi}^{n_y}_{l_y}(y)dxdy \end{aligned} \tag{4.22}$$

Equation eq.(4.21) can then be decomposed as

$$\begin{aligned}
[Tf]^N(x) = &\sum_{l_x,l_y} \boldsymbol{\tau}_{l_x l_y}^{N-1,N-1} \boldsymbol{\phi}_{l_x}^{N-1}(x) \boldsymbol{s}_{l_y}^{N-1} \\
&+ \sum_{l_x,l_y} \boldsymbol{\gamma}_{l_x l_y}^{N-1,N-1} \boldsymbol{\phi}_{l_x}^{N-1}(x) \boldsymbol{d}_{l_y}^{N-1} \\
&+ \sum_{l_x,l_y} \boldsymbol{\beta}_{l_x l_y}^{N-1,N-1} \boldsymbol{\psi}_{l_x}^{N-1}(x) \boldsymbol{s}_{l_y}^{N-1} \\
&+ \sum_{l_x,l_y} \boldsymbol{\alpha}_{l_x l_y}^{N-1,N-1} \boldsymbol{\psi}_{l_x}^{N-1}(x) \boldsymbol{d}_{l_y}^{N-1}
\end{aligned} \tag{4.23}$$

where we can identify $\boldsymbol{\alpha}_{l_x l_y}, \boldsymbol{\beta}_{l_x l_y}$ and $\boldsymbol{\gamma}_{l_x l_y}$ as the matrix elements of $A, B$ and $C$, respectively, and eq.(4.23) is again the matrix equation eq.(4.12) written explicitly. In this expression the last term involving the $\boldsymbol{\alpha}$ coefficients will be extremely sparse, and this sum can be limited to $l_x, l_y$ values that differ by less than some predetermined bandwidth $|l_x - l_y| < \Lambda^{N-1,N-1}$. The rest of the expression eq.(4.23) involves at least one scaling term, so we seek to decompose them further.

The first term in eq.(4.23) can be decomposed in the same manner as eq.(4.21), and the $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ terms can be partially decomposed, following the arguments of eq.(4.14) and eq.(4.13), respectively. If we do this all the way to the coarsest scale, we obtain

$$\begin{aligned}
[Tf]^N(x) = &\boldsymbol{\tau}_{00}^{0,0} \boldsymbol{\phi}_0^0(x) \boldsymbol{s}_0^0 \\
&+ \sum_{n_y=0}^{N-1} \sum_{l_y} \boldsymbol{\gamma}_{0 l_y}^{0,n_y} \boldsymbol{\phi}_0^0(x) \boldsymbol{d}_{l_y}^{n_y} \\
&+ \sum_{n_x=0}^{N-1} \sum_{l_x} \boldsymbol{\beta}_{l_x 0}^{n_x,0} \boldsymbol{\psi}_{l_x}^{n_x}(x) \boldsymbol{s}_0^0 \\
&+ \sum_{n_x=0}^{N-1} \sum_{n_y=0}^{N-1} \sum_{l_x,l_y} \boldsymbol{\alpha}_{l_x l_y}^{n_x,n_y} \boldsymbol{\psi}_{l_x}^{n_x}(x) \boldsymbol{d}_{l_y}^{n_y}
\end{aligned} \tag{4.24}$$

This is the explicit expression for the standard representation of an operator in the multiwavelet basis eq.(4.16). In eq.(4.24) the majority of terms are included in the last quadruple sum, which is limited to include only terms $|l_x - l_y| < \Lambda^{n_x,n_y}$, making the total evaluation much more efficient than eq.(4.21).

## 4.4 The Poisson operator

In order to solve the Poisson equation using the methods described above, we need to rewrite it to an integral form. The equation, in its differential form, is given as

$$\nabla^2 V(\boldsymbol{x}) = 4\pi\rho(\boldsymbol{x}) \tag{4.25}$$

where $\rho(\boldsymbol{x})$ is the known (charge) distribution, and $V(\boldsymbol{x})$ is the unknown (electrostatic) potential. It is a standard textbook procedure to show that the solu-

tion can be written as the integral

$$V(\boldsymbol{x}) = \int G(\boldsymbol{x}, \boldsymbol{y})\rho(\boldsymbol{y})d\boldsymbol{y} \tag{4.26}$$

where $G(\boldsymbol{x}, \boldsymbol{y})$ is the Green's function which is the solution to the *fundamental* equation with *homogeneous* (Dirichlet) boundary conditions

$$\begin{aligned} \nabla^2 G(\boldsymbol{x}, \boldsymbol{y}) &= \delta(\boldsymbol{x} - \boldsymbol{y}) \\ G(\boldsymbol{x}, \boldsymbol{y}) &= 0 \qquad\qquad , \boldsymbol{x} \in \textit{boundary} \end{aligned} \tag{4.27}$$

This equation can be solved analytically and the Green's function is given (in three dimensions) simply as

$$G(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{||\boldsymbol{x} - \boldsymbol{y}||} \tag{4.28}$$

This is the well known potential arising from a point charge located in the position $\boldsymbol{y}$, which is exactly what eq.(4.27) describes.

## Numerical separation of the kernel

The Green's function kernel as it is given in eq.(4.28) is not separable in the cartesian coordinates. However, since we are working with finite precision we can get by with an *approximate* kernel as long as the error introduced with this approximation is less than our overall accuracy criterion. If we are able to obtain such a *numerical* separation of the kernel, the operator can be applied in one direction at the time, allowing us to use the expressions derived above for one-dimensional integral operators to solve the three-dimensional Poisson equation. This is of great importance, not only because we do not have to derive the $d$-dimensional operator equations, which is at best notationally awkward, but also because it again reduces the scaling behavior to become linear in the dimension of the system.

The Poisson kernel can be made separable by expanding it as a sum of Gaussian functions, specifically

$$\frac{1}{||\boldsymbol{r} - \boldsymbol{s}||} \approx \sum_{\kappa=1}^{M_\epsilon} a_\kappa e^{-b_\kappa(\boldsymbol{r}-\boldsymbol{s})^2} \tag{4.29}$$

where $a_\kappa$ and $b_\kappa$ are parameters that needs to be determined, and the number of terms $M_\epsilon$, called the separation rank, depends on the accuracy requirement and on what interval this expansion needs to be valid. Details of how to obtain this expansion can be found in [1], and will not be treated here, but it should be mentioned that the separation rank is usually in the order of 100, e.g. it requires $M_\epsilon = 70$ to reproduce $1/r$ on the interval $[10^{-5}, \sqrt{3}]$ in three dimensions with error less than $\epsilon = 10^{-8}$.

# Part II

# Implementation

# Chapter 5

# The MRChem program

## 5.1 Data structures

In the following a brief introduction is given to the important data structures that are used in the MRChem program. This will introduce the nomenclature used in the rest of the thesis.

### `Node`

The `node` is the multidimensional box on which the set of scaling and wavelet functions that share the same support are defined. The `node` is specified by its scale $n$, which gives its size ($[0, 2^{-n}]^d$) and translation vector $\boldsymbol{l} = (l_1, l_2, \ldots, l_d)$, which gives its position. The `node` holds the $(k+1)^d$ scaling coefficients *and* $(2^d - 1)(k+1)^d$ wavelet coefficients that share the same scale and translation. It will also keep track of its parent and all $2^d$ children `nodes`, giving the `nodes` a tree-like structure.

### `Tree`

The `tree` data structure is basically the multiwavelet representation of a function. The name originates from the fact that a one-dimensional function is represented as a binary tree of `nodes` (octal tree in three dimesions) emanating from the single root `node` at scale zero. The `tree` keeps the entire tree of `nodes`, from root to leaf, and each `node` keeps both the scaling and wavelet coefficients. This means that there is a redundancy in the function representation (a non-redundant representation would be either to keep the scaling coefficients at the root and the wavelet coefficients of all the `nodes` from root to leaf, *or* just keep the scaling and wavelet coefficients of the leaf `nodes` only), but it proves easier just to keep all these coefficients in memory, rather than having to obtain the missing coefficients by filter operations (because they will all be needed when we start manipulating these function `trees` by addition, multiplication and operators).

## 5.2 Adaptive algorithm

The algorithm used to obtain adaptive representations of functions was presented in [1]. This is a fully on-the-fly algorithm in the sense that one sets out from the coarsest scale and refine the representation scale by scale locally only where it is needed. This in oppose to the originally proposed algorithm where one calculates the uniform representation eq.(3.1) on some finest scale and then do a forward wavelet transform and discards all wavelet terms below some threshold to obtain the adaptive representation eq.(3.7). One obvious advantage of this adaptive algorithm is that we do not need any a priori knowlegde of the global finest scale.

**Algorithm 1** Generation of adaptive Multiwavelet representation of a function

```
1:  while number of nodes on current scale N_s > 0 do
2:    for each node at current scale do
3:      compute scaling and wavelet coefficients
4:      if node needs to be refined then
5:        mark node as non-terminal
6:        allocate children nodes
7:        update list of nodes at the next scale
8:      else
9:        mark node as terminal
10:     end if
11:   end for
12:   increment scale
13: end while
```

The algorithm consists of two loops, the first runs over the ladder of scales, from coarsest to finest, while the second is over the `nodes` present at the current scale. Once the expansion coeffients of the current `node` are known, a split check is performed based on the desired precision. If the `node` does not satisfy the accuracy criterion, the `node` is marked as non-terminal and its children `nodes` are allocated and added to the list of `nodes` needed at the next scale. If the `node` does not need to be split, the `node` is marked as terminal and no children `nodes` are allocated. In this way, once the loop over `nodes` on one scale is terminated, the complete list of `nodes` needed on the *next* scale has been obtained. Now the scale counter is incremented and the loop over `nodes` on the next scale is started. The `tree` is grown until no `nodes` are needed at the next finer scale.

There are of course two points in this algorithm that need to be treated further, the first being the actual computation of the expansion coefficients (line 3). This can be done in one of four ways (projection, addition, multiplication or operator application) and will be treated in the subsequent sections. The second point is how to perform the split check (line 4).

The split check is performed to decide whether or not the function is represented accurately enough on the current `node`, based on a predefined relative precision $\epsilon$. Formally, this relative precision requires that

$$||f - f_\epsilon|| < \epsilon ||f||_2 \tag{5.1}$$

where $f_\epsilon$ is our approximation of $f$. However, this check cannot be performed since the *true* function $f$ is generally not known. The check that will be performed is based on the fact that the scaling projections will approach the exact function with increasing scale. Consequently, the wavelet projections, which is the difference between two consecutive scaling projections, must approach zero. This means that we can use the norm of the wavelet coefficients on one `node` as a measure for the accuracy of the function represented by this `node`, and we will use the following threshold for the wavelet coefficients on the terminal `nodes`

$$||\boldsymbol{d}_{\boldsymbol{l}}^n|| < 2^{-n/2}\epsilon||f_\epsilon||_2 \tag{5.2}$$

A justification for this choice of thresholding can be found in [1] and references therein. This algorithm is very general, and as pointed out it can be used to build adaptive representations of functions regardless of how the expansion coefficients are obtained, and in the following sections we will look at four different ways of doing this.

## 5.3 Function projection

The first step into the multiresolution world will always have to be taken by projection of some analytical function onto the multiwavelet basis. Only then can we start manipulating these representations by additions, multiplications and operators.

### Legendre scaling functions

In a perfect world, the projection in eq.(3.2) could be done exactly, and the accuracy of the projection would be independent of the choice of polynomial basis. In the real world the projections are done with Gauss-Legendre quadrature and the expansion coefficients $s_{j,l}^{n,f}$ of $f(x)$ are obtained as

$$\begin{aligned} s_{j,l}^{n,f} &= \int_{2^{-n}l}^{2^{-n}(l+1)} f(x)\phi_{j,l}^n(x)dx \\ &= 2^{-n/2}\int_0^1 f(2^{-n}(x+l))\phi_{j,0}^0(x)dx \\ &\approx 2^{-n/2}\sum_{q=0}^{k_q-1} w_q f(2^{-n}(y_q+l))\phi_{j,0}^0(y_q) \end{aligned} \tag{5.3}$$

where $\{w_q\}_{q=0}^{k_q-1}$ are the weights and $\{y_q\}_{q=0}^{k_q-1}$ the roots of the Legendre polynomial $L_{k_q}$ used in $k_q$-th order quadrature.

By approximating this integral by quadrature we will of course not obtain the exact expansion coefficients. However, it would be nice if we could obtain the *exact* coefficients whenever our basis is flexible enough to reproduce the function exactly, that is if $f(x)$ is a polynomial of degree $\leq k$. The Legendre quadrature holds a $(2k-1)$-rule which states that the $k$-order quadrature is exact whenever the integrand is a polynomial of order $2k-1$. By choosing $k_q = k+1$ order quadrature we will obtain the exact coefficient whenever $f(x)$ is a polynomial of

degree $\leq (k+1)$ when projecting on the basis of $k$-order Legendre polynomials, and we will use quadrature order $k+1$ througout.

In the multidimensional case the expansion coefficients are given by multidimensional quadrature

$$s_{\boldsymbol{j},\boldsymbol{l}}^{n,f} = 2^{-dn/2} \sum_{q_1=0}^{k} \sum_{q_2=0}^{k} \cdots \sum_{q_d=0}^{k} f(2^{-n}(\boldsymbol{y_q} + \boldsymbol{l})) \prod_{i=1}^{d} w_{q_i} \phi_{j_p,0}^{0}(y_{q_i}) \tag{5.4}$$

using the following notation for the vector of quadrature roots

$$\boldsymbol{y_q} \stackrel{\text{def}}{=} (y_{q_1}, y_{q_2}, \ldots, y_{q_d}) \tag{5.5}$$

This quadrature is not very efficient in multiple dimensions since the number of terms scales as $(k+1)^d$. However, if the function $f$ is separable and can be written $f(x_1, x_2, \ldots, x_d) = f_1(x_1) f_2(x_2) \cdots f_d(x_d)$, eq.(5.4) can be simplified to

$$s_{\boldsymbol{j},\boldsymbol{l}}^{n,f} = 2^{-dn/2} \prod_{i=1}^{d} \sum_{q_i=0}^{k} f_i(2^{-n}(y_{q_i} + l_i)) w_{q_i} \phi_{j_i,0}^{0}(y_{q_i}) \tag{5.6}$$

which is a product of small summations and scales only as $d(k+1)$.

The Legendre polynomials show very good convergence for *polynomial* functions $f(x)$, and are likely to give more accurate projections. However, most interesting functions $f(x)$ are *not* simple polynomials, and the accuracy of the Legendre scaling functions versus a general polynomial basis might not be very different.

## Interpolating scaling functions

By choosing the quadrature order to be $k+1$ a very important property of the Interpolating scaling functions emerges, stemming from the specific construction of these functions eq.(2.10), and the use of the $k+1$ order quadrature roots and weights. The interpolating property eq.(2.11) inserts a Kronecker delta whenever the scaling function is evaluated in a quadrature root, which is exactly the case in the quadrature sum. This reduces eq.(5.3) to

$$s_{jl}^{n,f} = \frac{2^{-n/2}}{\sqrt{w_j}} f(2^{-n}(x_j + l)) \tag{5.7}$$

which obviously makes the projection $k+1$ times more efficient.

In multiple dimensions this property becomes even more important, since it effectively removes all the nested summations in eq.(5.4) and leaves only one term in the projection

$$s_{\boldsymbol{j},\boldsymbol{l}}^{n,f} = f(2^{-n}(\boldsymbol{y_j} + \boldsymbol{l})) \prod_{i=1}^{d} \frac{2^{-n/2}}{\sqrt{w_{j_i}}} \tag{5.8}$$

This means that in the Interpolating basis the projection is equally effective regardless of the separability of the function $f$.

### Obtaining the wavelet coefficients

The wavelet coefficients are formally obtained by the projection of the function onto the wavelet basis, and we could derive expressions similar to the scaling expressions based on quadrature. There are however some accuracy issues connected to this wavelet quadrature, so we will take another approach that utilizes the wavelet transform. We know that we can obtain the scaling and wavelet coefficients on scale $n$ by doing a wavelet decomposition of the scaling coefficients on scale $n+1$ according to eq.(3.8). Line 3 of the algorithm is thus performed by computing the scaling coefficients of the $2^d$ children of the current `node` by the appropriate expression (Legendre or Interpolating), followed by a wavelet decomposition. In this way, wavelet projections are not required.

### Quadrature accuracy

We know that by using quadrature in the projection, we are only getting approximate scaling coefficients, and some remarks should be given on this matter. Consider the quadrature performed at scale zero. In $d$ dimensions this gives a total number of $(k+1)^d$ quadrature points distributed in the unit hypercube $[0,1]^d$, while the quadrature on scale one will give $(k+1)^d$ quadrature points on *each* of the $2^d$ children cubes contained in the same unit hypercube. This will obviousy increase the accuracy of the quadrature, and the scaling coefficients obtained at scale one must be considered more accurate than the ones obtained at scale zero. This means that by improving our representation of $f$ by increasing the scale, we are not only increasing the basis set, we are also increasing the accuracy of every single expansion coefficient.

If we look back to the computation of wavelet coefficients above, we see that we are gaining accuracy by doing the projection at scale $n+1$ followed by a wavelet transform, compared to a projection of the scaling and wavelet terms separately at scale $n$. This also means that once the adaptive algorithm has terminated and we have a representation of satisfactory accuracy at some local finest scale, we should perform a complete wavelet transform all the way from finest to coarsest scale, which will update the expansion coefficients on the coarser scales to be of the same *quadrature* accuracy as the coefficients at the finest scale.

## 5.4 Addition of functions

The recipe for the addition of two function `trees` is given quite intuitively by eq.(3.16) and eq.(3.17) as a simple vector addition of the scaling and wavelet coefficients on corresponding `nodes`

$$\begin{aligned} s^{n,h}_{\boldsymbol{j},\boldsymbol{l}} &= s^{n,f}_{\boldsymbol{j},\boldsymbol{l}} + s^{n,g}_{\boldsymbol{j},\boldsymbol{l}} \\ d^{n,h}_{\boldsymbol{j},\boldsymbol{l}} &= d^{n,f}_{\boldsymbol{j},\boldsymbol{l}} + d^{n,g}_{\boldsymbol{j},\boldsymbol{l}} \end{aligned} \tag{5.9}$$

These expressions are independent of the type of scaling functions used. We see that by applying the adaptive algorithm to build this addition `tree`, we automatically end up with the correct `nodal` structure (refinement), *and* the correct coefficients, and no `tree`parsing is required afterward to remove unneccesary

`nodes`, or update non-terminal coefficients as for the projection.

If the situation arise where the algorithm tells us that a `node` is required in the result that is not present in one of the `trees` representing $f$ and $g$, this `node` will have to be generated by wavelet transform before the addition can be carried out. These *generated* `nodes` will only contain scaling coefficients because the information about its wavelet coefficients is simply not available in the current `tree` representation, and further projections would have to be carried out to obtain this information. This will of course not be done, because once the projections of the functions $f$ and $g$ are done, these representatons are considered independent functions and are no longer related to any analytical functions (this relation will be lost anyway once you start manipulating the function by operators).

It can also be noted that if it occurs that *both* the $f$ and $g$ `nodes` are missing, there is no need to generate these `nodes` since no new information will be obtained in their addition that is not already available at a coarser scale in the result `tree`. The sum of these `nodes` will of course only have zero-valued wavelet coefficients, and is by definition not needed in the result.

### Addition accuracy

No absolute accuracy will be lost during an addition. The reason for this is given by the relations eq.(3.15), which simply states that the projection of the sum equals the sum of the projections. However, *relative* accuracy might be lost if the additon *reduces* the norm of the function. If this is the case, each of the functions $f$ and $g$ would have been projected with a higher wavelet norm threshold compared to a direct projection of the analytical sum (this will lead to the situation mentioned above where both the $f$ and $g$ `nodes` are missing).

## 5.5 Multiplication of functions

As it was presented in chapter three, the multiplication was a "leaf-to-root" algorithm that would start by doing the multiplication at some predetermined finest scale, followed by a wavelet transform to obtain the `nodes` at coarser scales, throwing away all `nodes` that are not needed. By doing the multiplication this way there is no accuracy problems related to the multiplication, provided that the finest scale $N$ in eq.(3.22) is chosen properly. Since we now are doing the multiplication on a finer scale than either $f$ or $g$ were originally represented, no information from these functions is lost in the multiplication.

The projection integral in eq.(3.25) is again done by Gauss-Legendre quadrature and we end up with two double summation

$$s^{N,h}_{j^h,l} \approx 2^{N/2} \sum_{q=0}^{k} w_q \left( \sum_{j^f=0}^{k} s^{N,f}_{j^f,l} \phi^0_{j^f,0}(y_q) \right) \left( \sum_{j^g=0}^{k} s^{N,g}_{j^g,l} \phi^0_{j^g,0}(y_q) \right) \phi^0_{j^h,0}(y_q) \tag{5.10}$$

But what is a proper choice of finest scale $N$ in the product? To avoid this question it is desireable to incorporate the multiplication in the previous adap-

tive algorithm. To do this we need to be able to calculate the result at an arbitrary scale $n$, that is not necessarily beyond the finest scales of $f$ and $g$. Using quadrature, all the information we need from the multiplicands is their pointvalues in the quadrature roots $\{y_q\}_{q=0}^k$ at scale $n$.

$$s_{j^h,l}^{n,h} \approx \sum_{q=0}^{k} w_q \left(f(y_q) \times g(y_q)\right) \phi_{j^h,l}^n(y_q) \tag{5.11}$$

But now the question arises of what scale the functions $f$ and $g$ shall be evaluated. The best we can do is to evaluate $f$ and $g$ at their respective finest scales. If we do this throughout the algorithm, there are still no accuracy issues, since we are using our best approximations available for the $f$ and $g$ pointvalues, which is simply the best we can achieve.

The problem with this approach is that we will now get expressions that couple different scales of $h$, $f$ and $g$. This will specifically mean that we can no longer exploit the characteristic property of Interpolating wavelets for the evaluation of the $f$ and $g$ pointvalues, since the quadrature roots on different scales do not coincide. This means that in order to get a numerically efficient multiplication, we need to relate the product on one scale to the multiplicants on the same scale.

## Legendre scaling functions

This uncoupling is easily done by using the expression in eq.(5.10) at an arbitrary scale $n$

$$s_{j^h,l}^{n,h} \approx 2^{n/2} \sum_{q=0}^{k} w_q \left( \sum_{j^f=0}^{k} s_{j^f,l}^{n,f} \phi_{j^f,0}^0(y_q) \right) \left( \sum_{j^g=0}^{k} s_{j^g,l}^{n,g} \phi_{j^g,0}^0(y_q) \right) \phi_{j^h,0}^0(y_q) \tag{5.12}$$

The generalization to multiple dimensions gives no surprises, but by expanding the vector notation in $d$ dimensions it becomes clear that multiplication will become a time consuming process in the Legendre basis.

$$\begin{aligned} s_{\boldsymbol{j}^h,\boldsymbol{l}}^{n,h} \approx 2^{dn/2} \sum_{q_1=0}^{k} \sum_{q_2=0}^{k} \cdots \sum_{q_d=0}^{k} \Bigg( & \left( \prod_{i=1}^{d} w_{q_i} \right) \\ & \times \left( \sum_{j_1^f=0}^{k} \sum_{j_2^f=0}^{k} \cdots \sum_{j_d^f=0}^{k} s_{\boldsymbol{j}^f \boldsymbol{l}}^{n,f} \left( \prod_{i=1}^{d} \phi_{j_i^f l_i}^0(y_{q_i}) \right) \right) \\ & \times \left( \sum_{j_1^g=0}^{k} \sum_{j_2^g=0}^{k} \cdots \sum_{j_d^g=0}^{k} s_{\boldsymbol{j}^g \boldsymbol{l}}^{n,g} \left( \prod_{i=1}^{d} \phi_{j_i^g l_i}^0(y_{q_i}) \right) \right) \\ & \times \prod_{i=1}^{d} \phi_{j_i^h l_i}^0(y_{q_i}) \Bigg) \end{aligned} \tag{5.13}$$

The scaling behavior of this expression is $(k+1)^{2d}$. From eq.(5.13) we can see that the only function evaluations that are actually taking place concern the

$k+1$ different scaling functions evaluated in the $k+1$ different quadrature roots. These $(k+1)^2$ function values need to be evaluated only once, and fetched from memory whenever needed in the expression eq.(5.13), which will speed up the process.

## Interpolating scaling functions

Multiplication in the Interpolating basis in $d$ dimensions follows exactly the expression for the Legendre basis eq.(5.13), and now the true power of the Interpolating scaling functions is revealed, in that it is specifically designed to return Kronecker deltas when evaluated in the quadrature roots. Inserting this property in eq.(5.13) we get

$$\begin{aligned} s^{n,h}_{\boldsymbol{j}^h\boldsymbol{l}} \approx 2^{dn/2} \sum_{q_1=0}^{k}\sum_{q_2=0}^{k}\cdots\sum_{q_d=0}^{k}\Bigg(\Bigg(\prod_{i=1}^{d} w_{q_i}\Bigg) \\ \times\Bigg(\sum_{j_1^f=0}^{k}\sum_{j_2^f=0}^{k}\cdots\sum_{j_d^f=0}^{k} s^{n,f}_{\boldsymbol{j}^f\boldsymbol{l}}\Bigg(\prod_{i=1}^{d}\frac{\delta_{j_i^f,q_i}}{\sqrt{w_{q_i}}}\Bigg)\Bigg) \\ \times\Bigg(\sum_{j_1^g=0}^{k}\sum_{j_2^g=0}^{k}\cdots\sum_{j_d^g=0}^{k} s^{n,g}_{\boldsymbol{j}^g\boldsymbol{l}}\Bigg(\prod_{i=1}^{d}\frac{\delta_{j_i^g,q_i}}{\sqrt{w_{q_i}}}\Bigg)\Bigg) \\ \times\Bigg(\prod_{i=1}^{d}\frac{\delta_{j_i^h,q_i}}{\sqrt{w_{q_i}}}\Bigg)\Bigg) \\ = 2^{dn/2} s^{n,f}_{\boldsymbol{j}^h\boldsymbol{l}} s^{n,g}_{\boldsymbol{j}^h\boldsymbol{l}}\Bigg(\prod_{i=1}^{d}\sqrt{w_{j_i^h}}\Bigg) \end{aligned} \tag{5.14}$$

which leaves only *one* term in the evaluation of each coefficient of the product, making the Interpolating basis vastly superior to the Legendre basis when it comes to multiplication efficiency.

## Obtaining wavelet coefficients

The calculation of the wavelet coefficients is done in the same way as for the projection, by wavelet transform of the scaling coefficients at scale $n+1$. Line 3 of algorithm 1 is again obtained by calculation of the scaling coefficients of the $2^d$ children of the current `node` by the appropriate expression (Legendre or Interpolating), followed by a wavelet decomposition.

## Multiplication accuracy

Now we can see that some accuracy issues will arise. As before we are making approximations of the coefficients based on the quadrature projection, but more importantly, we are making this approximation based on point values of $f$ and $g$ obtained at scale $n$, which may not resemble the true function at all. Formally, however, this is not a problem, since we know that by increasing the scale we are both improving the quadrature accuracy *and* improving the quality of the $f$ and $g$ pointvalues, and we will approach the exact product in the limit of high $n$. Obviously, by following this algorithm the accuracy of each coefficient will

improve as we ascend the ladder of scales, and just as for the projection, we will need to do a complete wavelet transform, from finest to coarsest scale in order to update all non-terminal coefficients.

As for the addition the situation may arise where some `nodes` in $f$ or $g$ needs to be generated by wavelet transform, but unlike the addition, we need to do this also when the `nodes` are missing in *both* $f$ and $g$. The reason for this is that we are only dealing with scaling coefficients, which will be non-zero also for "generated" `nodes`.

## 5.6 Operator application

Applying the operator is a two-step procedure. First we need to set up the operator by projecting it onto the multiwavelet basis. The application of the operator is then obtained by performing the matrix-vector multiplication given by eq.(4.15) to obtain the full multiresolution result.

For integral operators the projection of the operator reduces to a projection of the $2d$-dimensional kernel $K(\boldsymbol{x}, \boldsymbol{y})$ onto the multiwavelet basis. This is a function projection as good as any, and follows the projection algorithm described previously in this chapter. In the case of the Poisson operator, we make separate projections for each of the terms in the separable kernel expansion eq.(4.29), and in a sense we get $M_\epsilon$ different operators will be applied.

As was pointed out in chapter 4, by separating the Poisson kernel numerically, we will be able to apply a one-dimensional operator three times to obtain the full three-dimensional result, and in the following only a one-dimensional algorithm for the operator application is presented. Applying this one-dimensional algorithm to the correct terms in the $d$-dimensional case becomes a technical issue, and more is said on this matter in [1].

The way to build the result `tree` adaptively would be to apply the operator piece by piece, specifically one row at the time, to obtain the result scale by scale, refining only where needed. The standard (S) way of doing this is to apply the entire row, that is, applying all blocks on one row, to obtain the coefficients on one `node` of the result based on the *full* multiresolution operator

$$\begin{aligned} g^0 &= {}^0T^0 f^0 + \sum_{n'=0}^{N-1} {}^0C^{n'} df^{n'} \\ dg^n &= {}^nB^0 f^0 + \sum_{n'=0}^{N-1} {}^nA^{n'} df^{n'} \end{aligned} \tag{5.15}$$

and based on this make the decision of whether to split the `node`, before moving on to the next `node` and ultimately to the next scale. Note that if we are moving beyond the finest scale of $f$, only the ${}^nB^0$ term of the operator need to be considered, since $df^n$ is then zero by definition, and the function representation of $f$ will never have to be extended beyond its finest scale in the S operator application.

## Non-standard representation

By applying the operator the standard way, we improve our final result by adding wavelet corrections one scale at the time, not altering what was already there at the coarser scales. We will now introduce a somewhat different approach, where we do not apply the entire *multiresolution* operator on every scale, but rather the *mono*resolution operator one scale at the time.

$$\begin{aligned} g^n &= {}^nT^n f^n + {}^nC^n df^n \\ dg^n &= {}^nB^n f^n + {}^nA^n df^n \end{aligned} \tag{5.16}$$

This is the so-called non-standard (NS) way of applying operators, and the matrix equation will now look like

$$\left(\begin{array}{cc|cc} {}_{N-2}T^{N-2} & {}_{N-2}C^{N-2} & & \\ \hline {}_{N-2}B^{N-2} & {}_{N-2}A^{N-2} & & \\ \hline & & {}_{N-1}T^{N-1} & {}_{N-1}C^{N-1} \\ \hline & & {}_{N-1}B^{N-1} & {}_{N-1}A^{N-1} \end{array}\right) \left(\begin{array}{c} {}_f{}^{N-2} \\ \hline {}_{df}{}^{N-2} \\ \hline {}_f{}^{N-1} \\ \hline {}_{df}{}^{N-1} \end{array}\right) = \left(\begin{array}{c} {}_g{}^{N-2} \\ \hline {}_{dg}{}^{N-2} \\ \hline {}_g{}^{N-1} \\ \hline {}_{dg}{}^{N-1} \end{array}\right) \tag{5.17}$$

The advantage of this operator application is that the coarse scale evaluations will be more efficient, in that we are using only one of the terms in each of the sums in eq.(5.15), and only when we reach the finest scale of the operator are we applying the *full* multiresolution operator.

There are at least two disadvantages of the non-standard operator. The most obvious being that the total matrix has grown in size, since we are now explicitly calculating the scaling coefficients on every scale. In our case this is not that big an overhead, since we need these scaling coefficients anyway, and in the S representation they would have to be obtained by wavelet transform after the calculation of the wavelet coefficients. In $d$ dimensions this is even less of a problem since the scaling part is then only one of the $2^d$ scaling/wavelet contributions on each scale.

The other, more important disadvantage, is that it seems that we have to evaluate the dense ${}^nT^n$ submatrices all the way to the the finest scale, and this would

shatter any hope for a linear scaling algorithm. The way around this problem is to realize that the effect of ${}^nT^n$ on $f^n$ is exactly what was calculated at the previous scale, and instead of calculating this part again, we can do a wavelet transform of the $g^{n-1}$ and $dg^{n-1}$ coefficients to obtain the ${}^nT^n$ contribution to $g^n$. In this way only the coarsest scale $T$ matrix will actually have to be evaluated, and we re-gain our fast algorithm.

As was pointed out, we are not applying the *full* multiresolution operator until we reach the finest scale, which means that the coefficients calculated at the coarser scales are somewhat incomplete. However, the representation we have on the finest scale *will* be complete, and a wavelet decomposition all the way to the coarsest scale will update all non-terminal `nodes` to include the effect of the full operator.

Another consequence of the NS form of the operator is that we might have to extend the representation of $f$ beyond its finest scale during the operator application, since the scaling part of $f$ will be non-vanishing at any scale. However, at these nodes only the $B$ part of the operator needs to be applied because the wavelet coefficients are zero for generated `nodes`.

## Adaptive algorithm

The application of the NS operator follows the same algorithm as before, but with a few additional terms. First of all, we loop over the sum in the kernel

**Algorithm 2** Algorithm for operator application

**for** each term in the kernel expansion $K(x) = \sum_{\kappa=1}^{M} K_\kappa(x)$ **do**
  **while** number of nodes at current scale is $N_s > 0$ **do**
    **for** each node at current scale **do**
      **for** each operator component ($O = \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$ or $\boldsymbol{\tau}$) **do**
        **for** each input node within bandwidth $l_x - l_y < \Lambda^{n,n}$ **do**
          **if** $(||O^{n,n}_{l_x - l_y}|| \cdot ||w^{n,f}_{l_y}|| > \delta)$ **then**
            apply operator $w^{n,g}_{l_x} := w^{n,g}_{l_x} \; + \; O^{n,n}_{l_x - l_y} w^{n,f}_{l_y}$
          **end if**
        **end for**
      **end for**
      **if** node needs to be refined **then**
        mark node as non-terminal
        allocate children nodes
        update list of nodes at the next scale
        add current node result to children by wavelet transform
      **end if**
    **end for**
    increment scale
  **end while**
**end for**

expansion. The reason for doing this, in contrast to adding these terms to one kernel representation first, is that the different terms will have different band-

widths, and by adding them all together, we would end up with a bandwidth equal to the widest individual one.

The second additional term is the double loop in the evaluation of the expansion coefficients, and the third difference is the wavelet transform after the allocation of children `nodes` to obtain the contribution from the ${}^nT^n$ part of the operator for all but the coarsest scale. In the algorithm, $O$ stands for the entries of any of the operator components $A, B, C$ and $T$, and $w$ stands for both scaling and wavelet coefficients.

One should note that there are three factors determining whether a specific entry $O^{n_x,n_y}_{l_x,l_y}$ of the operator needs to be taken into account. Firstly, we need only the operator at the current scale. Secondly, we need only the `nodes` that are within the bandwidth, and finally, we use only the terms where the product of the norms is above a given threshold (line 6 in algorithm 2). This greatly reduces the number of terms we actually need to compute.

## Obtaining the coefficients

The actual calculation of the coefficients is performed in the following way. In the NS matrix equation, the wavelet coefficients are obtained by the $\boldsymbol{\gamma}$ and $\boldsymbol{\alpha}$ parts of the operator at any scale.

$$\begin{aligned} dg^n(x) &= {}^nB^n f^n(x) + {}^nA^n df^n(x) \\ \sum_{l_x} \boldsymbol{d}^{n,g}_{l_x} \boldsymbol{\psi}^n_{l_x}(x) &= \sum_{l_x} \left( \sum_{l_y} \left( \boldsymbol{\beta}^{n,n}_{l_x l_y} \boldsymbol{s}^{n,f}_{l_y} + \boldsymbol{\alpha}^{n,n}_{l_x l_y} \boldsymbol{d}^{n,f}_{l_y} \right) \boldsymbol{\psi}^n_{l_x}(x) \right) \\ \boldsymbol{d}^{n,g}_{l_x} &= \sum_{l_y} \left( \boldsymbol{\beta}^{n,n}_{l_x l_y} \boldsymbol{s}^{n,f}_{l_y} + \boldsymbol{\alpha}^{n,n}_{l_x l_y} \boldsymbol{d}^{n,f}_{l_y} \right) \end{aligned} \tag{5.18}$$

In the calculation of the scaling coefficients, the $\boldsymbol{\tau}$ part is included only for the coarsest scale.

$$\begin{aligned} g^0(x) &= {}^0T^0 f^0(x) + {}^0C^0 df^0(x) \\ \sum_{l_x} \boldsymbol{s}^{0,g}_{l_x} \boldsymbol{\phi}^0_{l_x}(x) &= \sum_{l_x} \left( \sum_{l_y} \left( \boldsymbol{\tau}^{0,0}_{l_x l_y} \boldsymbol{s}^{0,f}_{l_y} + \boldsymbol{\gamma}^{0,0}_{l_x l_y} \boldsymbol{d}^{0,f}_{l_y} \right) \boldsymbol{\phi}^0_{l_x}(x) \right) \\ \boldsymbol{s}^{0,g}_{l_x} &= \sum_{l_y} \left( \boldsymbol{\tau}^{0,0}_{l_x l_y} \boldsymbol{s}^{0,f}_{l_y} + \boldsymbol{\gamma}^{0,0}_{l_x l_y} \boldsymbol{d}^{0,f}_{l_y} \right) \end{aligned} \tag{5.19}$$

For the general scale $n > 0$ the $\boldsymbol{\tau}$ part is substituted with a filter operation.

$$\begin{aligned} \boldsymbol{s}^{n,g}_{l_x=even} &= \left( H^{(0)T} \boldsymbol{s}^{n-1,g}_{l_x/2} + G^{(0)T} \boldsymbol{d}^{n-1,g}_{l_x/2} \right) + \sum_{l_y} \boldsymbol{\gamma}^{n,n}_{l_x l_y} \boldsymbol{d}^{n,f}_{l_y} \\ \boldsymbol{s}^{n,g}_{l_x=odd} &= \left( H^{(1)T} \boldsymbol{s}^{n-1,g}_{(l_x-1)/2} + G^{(1)T} \boldsymbol{d}^{n-1,g}_{(l_x-1)/2} \right) + \sum_{l_y} \boldsymbol{\gamma}^{n,n}_{l_x l_y} \boldsymbol{d}^{n,f}_{l_y} \end{aligned} \tag{5.20}$$

In all of these expressions the summation over $l_y$ is limited to those that differ from $l_x$ by less than the bandwidth $|l_y - l_x| < \Lambda^{n,n}$. More on the calculation of this bandwidth if given in [1].

# Chapter 6

# Results

In the following some numerical results obtained by the MRChem program is presented. Some of what is presented are the results of previous work, but are presented here to make the exposition more complete.

## 6.1 Function projection

The function projection as a whole was implemented by [1].

### Grid adaptivity

To illustrate the power of a basis that adapts locally to the function it is trying to represent, we will project the Gaussian function

$$f(\boldsymbol{x}) = e^{-1000(\boldsymbol{x}-\boldsymbol{x}_0)^2}, \qquad \boldsymbol{x}_0 = (0.2, 0.5, 0.5) \tag{6.1}$$

in three dimensions. The Gaussian function is a multiresolution function in that is has variations at all length scales, and an adaptive basis is crucial to represent it accurately with a basis set of a manageable size.

Figure 6.1 show various plots of the spherical Gaussian. All plots are along the line $y = z = 0.5$, and the six plots on the left hand side shows approximations of $f(\boldsymbol{x})$ as the projection on the scaling spaces $V^0$ through $V^5$ in the basis of fifth order Legendre polynomials. The accuracy of the approximation is clearly increasing with higher resolution. From scale 3 the function starts to resemble a Gaussian, and at scale 5 the error is less than $10^{-2}$. The $f^5(x)$ projection is expanded in the full basis of $V^5$ in three dimensions, consisting of $2^{dn} = 32768$ `nodes`, each containing $(k+1)^d = 216$ basis functions. This is hardly an efficient way of representing a spherical Gaussian.

The plots on the right hand side of figure 6.1 show the wavelet projections of $f(\boldsymbol{x})$ on scales 0 through 5. The wavelet projection on scale $n$ is, by definition, the difference between the scaling projections on scale $n$ and $n + 1$, which can be seen in the plots. The wavelet plots are obtained adaptively and all terms that were sufficiently small (contributes less that $10^{-2}$ to the function) were left out.

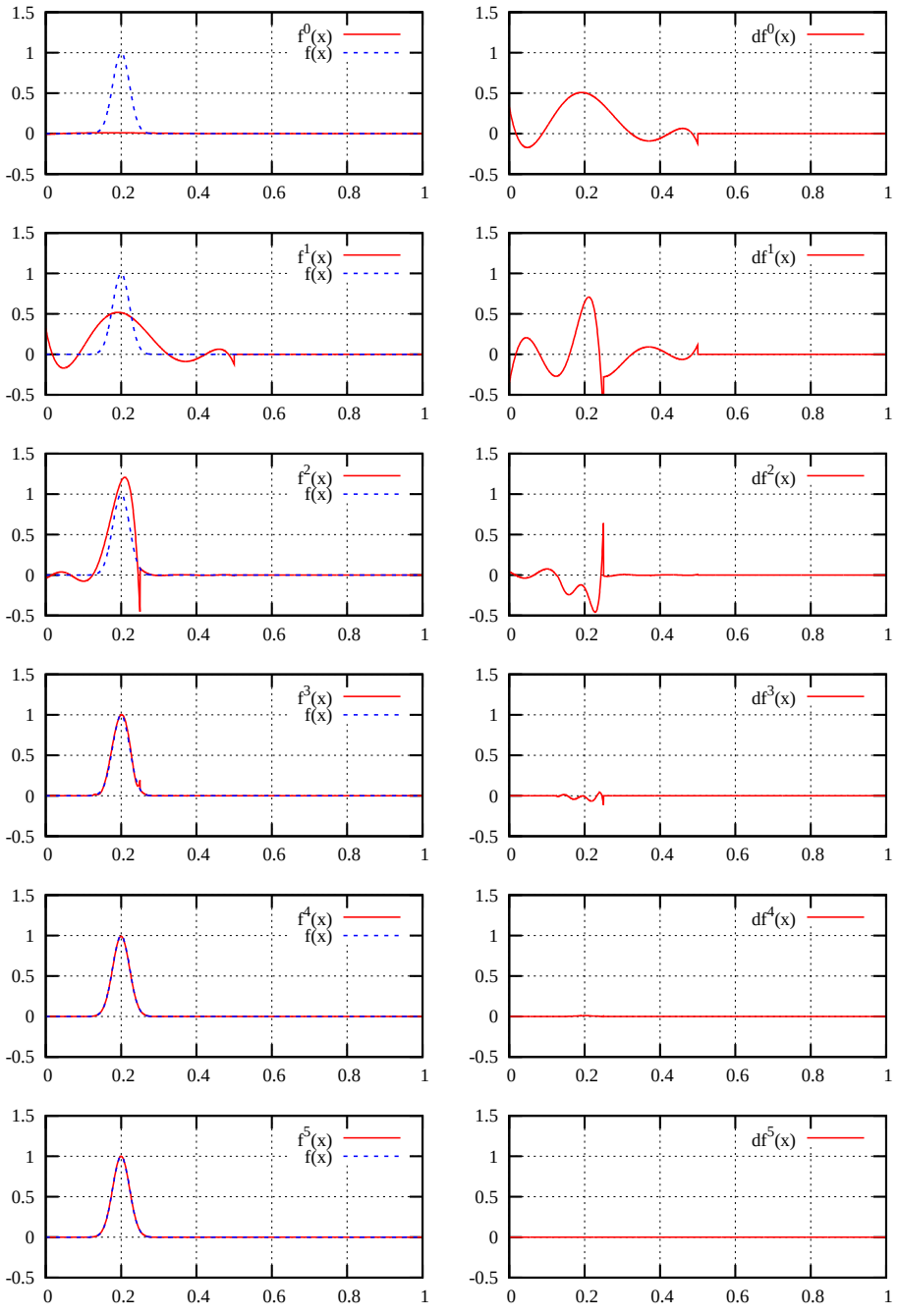


Figure 6.1: Projection of the Gaussian function eq. (6.1) on scaling spaces $V^0$ through $V^5$ (left), and wavelet spaces $W^0$ through $W^5$ (right).

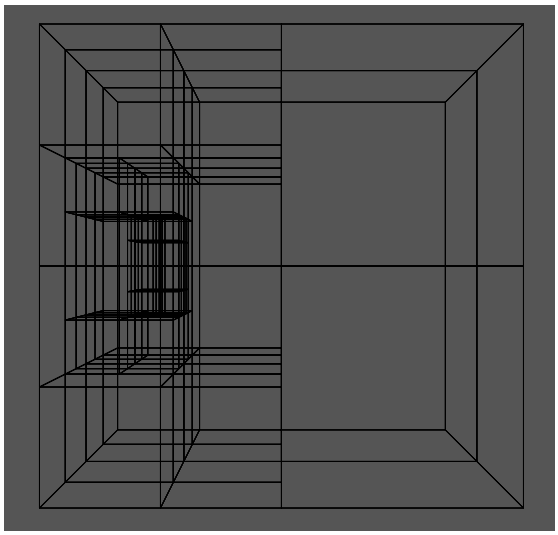

Figure 6.2: Grid refinement in the adaptive representation of the Gaussian function eq. (6.1)

The adaptive representation of $f(\boldsymbol{x})$ with the same accuracy ($10^{-2}$) corresponds to using the sum of the scaling projection on scale 0 (top left scaling plot) and the wavelet projection at scales 0 through 4 (top five wavelet plots to the right). If all the wavelet terms had been kept this would result in an equivalent expansion with respect to number of basis functions, but by throwing away small wavelet terms, we arrive at 105 contributing `nodes`. Even though the number of wavelet function on one `node` becomes $(2^d-1)(k+1)^d = 1512$, this significantly reduces the number of expansion coefficients needed to represent the functions with the *same* overall accuracy. Figure 6.2 shows the grid partitioning in the adaptive representation of $f(\boldsymbol{x})$, where we can see that the basis functions are concentrated around the center of the Gaussian.

## Accuracy

To test the accuracy of the projections we do pointwise evaluation of the error compared to the analytic function. This test is more strict than what we can really demand of the projection, since we only require the $L^2$ norm of the error to be within the accuracy threshold. Figure 6.3 shows the error, relative to the norm, of two functions $f(\boldsymbol{x})$ and $g(\boldsymbol{x})$, evaluated in 2000 points along the line given by $y = z = 0.5$. The functions are projected using 8th order Interpolating scaling functions, with various target accuracies ranging from $10^{-2}$ to $10^{-10}$.

The function $f$ is the same as above, while $g$ is a more diffuse Gaussian

$$\begin{aligned} f(\boldsymbol{x}) &= e^{-1000(\boldsymbol{x}-\boldsymbol{x}_0)^2}, \quad \boldsymbol{x}_0 = (0.200, 0.5, 0.5) \\ g(\boldsymbol{x}) &= e^{-100(\boldsymbol{x}-\boldsymbol{x}_0)^2}, \quad \boldsymbol{x}_0 = (0.485, 0.5, 0.5) \end{aligned} \tag{6.2}$$

As can be seen from the plots in figure 6.3, the pointwise errors are consistently below the requested accuracy.

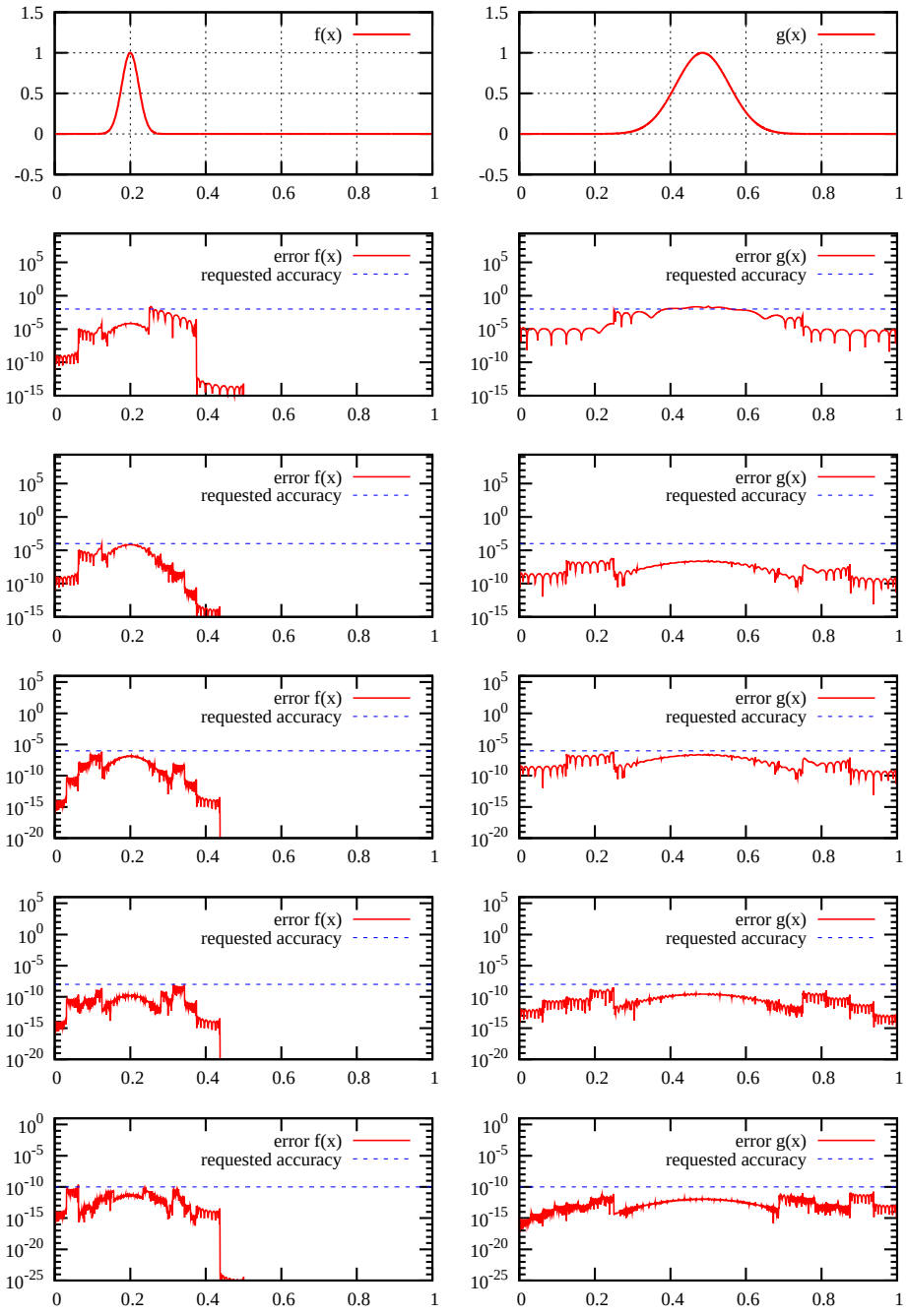


Figure 6.3: Pointwise relative error in the projection of the two Gaussians eq. (6.2) on the line $y = z = 0.5$

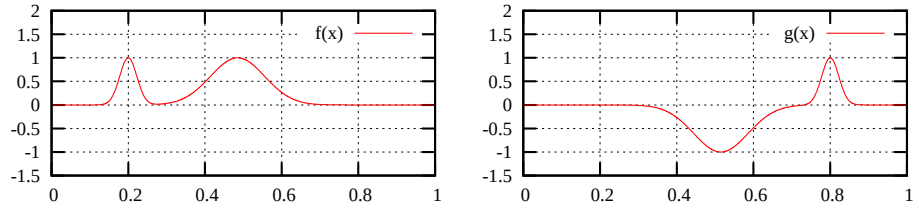


Figure 6.4: Plot of the two functions used in the test of addition and multiplication

## 6.2 Addition of functions

The addition will be carried out using the following two functions

$$f(\boldsymbol{x}) = e^{-1000(\boldsymbol{x}-\boldsymbol{x}_0)^2} + e^{-100(\boldsymbol{x}-\boldsymbol{x}_1)^2}, \qquad \begin{cases} \boldsymbol{x}_0 = (0.200, 0.5, 0.5) \\ \boldsymbol{x}_1 = (0.485, 0.5, 0.5) \end{cases} \tag{6.3}$$

$$g(\boldsymbol{x}) = e^{-1000(\boldsymbol{x}-\boldsymbol{x}_0)^2} - e^{-100(\boldsymbol{x}-\boldsymbol{x}_1)^2}, \qquad \begin{cases} \boldsymbol{x}_0 = (0.800, 0.5, 0.5) \\ \boldsymbol{x}_1 = (0.515, 0.5, 0.5) \end{cases} \tag{6.4}$$

which are shown in figure 6.4.

### Grid adaptivity

The grid of the sum will never contain more nodes than the union of the separate grids of $f$ and $g$, and no refinement will take place in the sum that is not present in either $f$ or $g$. If the norm of the sum is bigger than the norm of each individual function, the refinement of the sum might be truncated locally at a coarser scale than the separate functions. Figure 6.5 and 6.6 shows the grids of $f$ and $g$ separately, and figure 6.7 shows the grid of their sum, all obtained with accuracy $10^{-6}$ using 8th order Interpolating scaling functions.

### Accuracy

As it was pointed out, some relative accuracy might be lost during an addition if the norm of the function is *reduced*. This is actually the case in our example addition, but the square norm of the sum is only about half the norms of $f$ and $g$ separately, and the error should be in the same order of magnitude. The accuracy was tested the same way as for the projection; by pointwise calculation of the error relative to the norm along the line $y = z = 0.5$, and the results are shown in figure 6.9. In these plots we can see that the error is sometimes marginally above the threshold, but considering the strictness of this test the results are satisfactory.

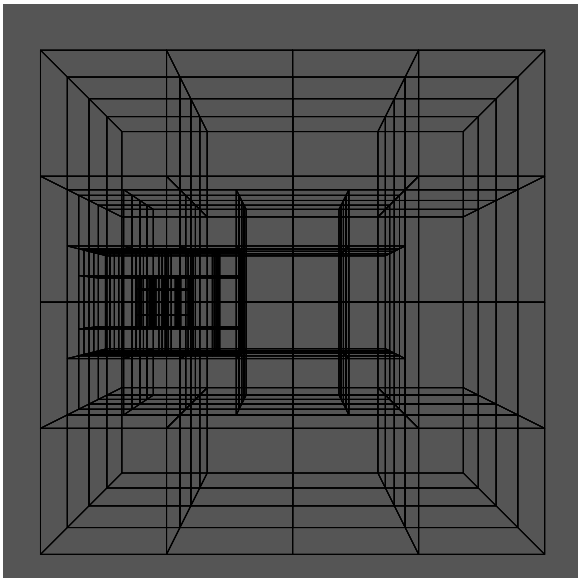

Figure 6.5: Grid refinement of the adaptive representation of the function $f(\boldsymbol{x})$ eq. (6.3) used in addition and multiplication

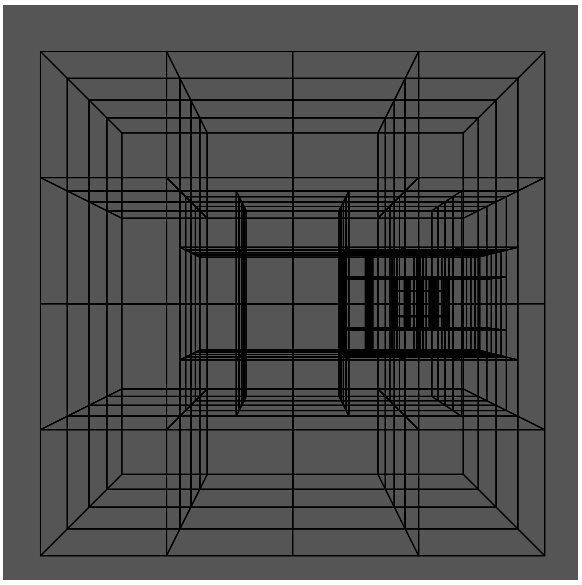

Figure 6.6: Grid refinement of the adaptive representation of the function $g(\boldsymbol{x})$ eq. (6.4) used in addition and multiplication

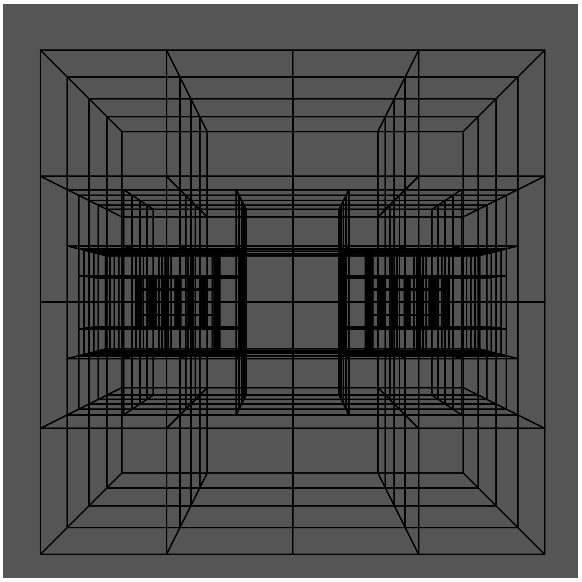

Figure 6.7: Grid refinement of the adaptive representation of the function obtained by addition of $f(\boldsymbol{x})$ eq. (6.3) and $g(\boldsymbol{x})$ eq. (6.4)

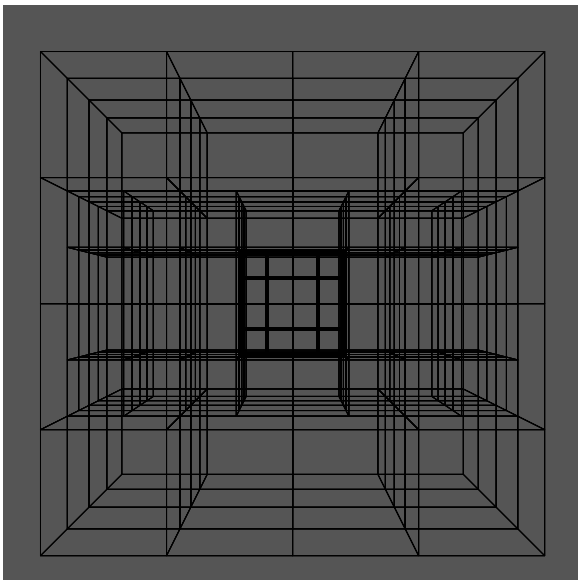

Figure 6.8: Grid refinement of the adaptive representation of the function obtained by multiplication of $f(\boldsymbol{x})$ eq. (6.3) and $g(\boldsymbol{x})$ eq. (6.4)

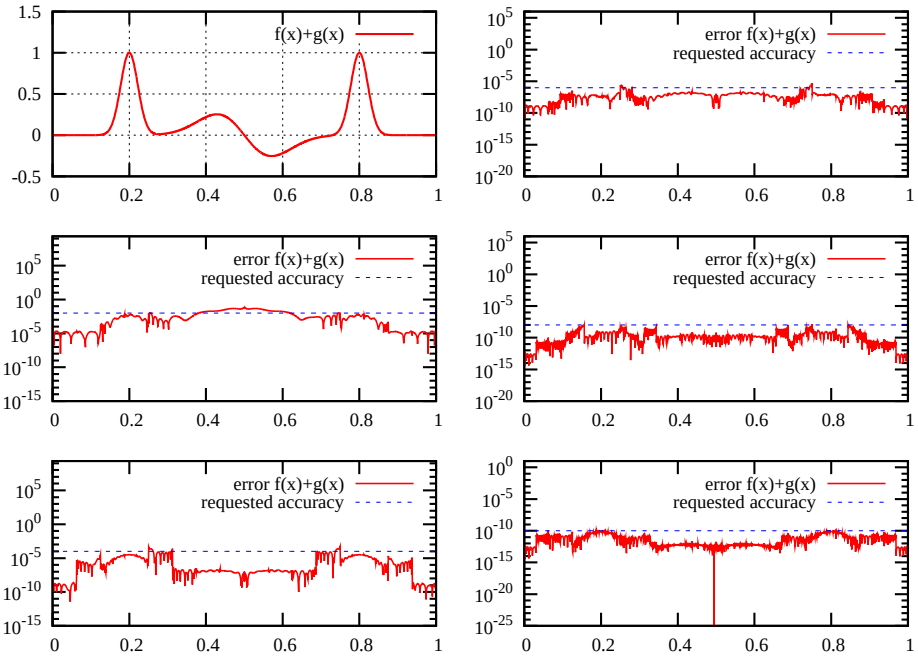


Figure 6.9: Pointwise relative error in the addition of the two functions eq. (6.3) and eq. (6.4) on the line $y = z = 0.5$

## 6.3 Multiplication of functions

The multiplication is done with the same functions as the addition, eq.(6.3) and eq.(6.4).

### Grid adaptivity

The problem of grid adaptivity the case of the product of two functions is more challenging than in the addition, since the grid of the product does not necessarily resemble the grids of the multiplicands. In our example the narrow peak in each of the functions $f$ and $g$ will vanish in the product, and the diffuse peaks will multiply to one single peak, shifted to the center of the plot. The grid of the product is shown in figure 6.8, and we see that the grid perfectly adapts to represent the new function efficiently.

### Accuracy

There were some accuracy issues related to the adaptive construction of the product representation as it was presented in the previous chapter. By performing the same accuracy test as above we can see from figure 6.10 that the relative accuracy of the product falls consistently below the accuracy threshold, except for the $\epsilon = 10^{-2}$ case, and the accuracy of the product is by all means satisfactory.

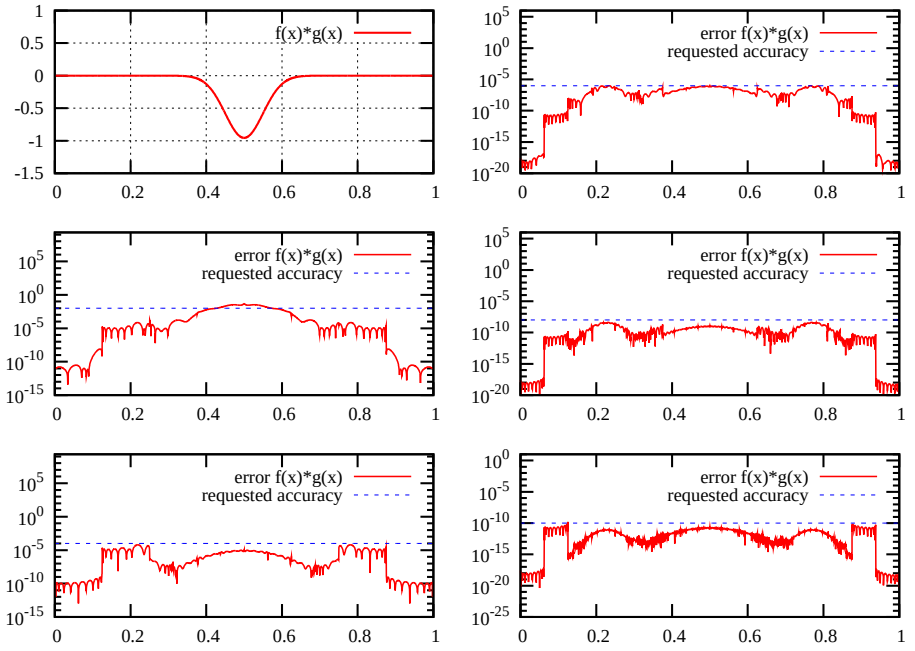


Figure 6.10: Pointwise relative error in the multiplication of the two functions eq. (6.3) and eq. (6.4) on the line $y = z = 0.5$

For the multiplication one additional accuracy test is carried out, that considers the $L^2$ norm of the error. This test is done by projecting the analytical product with accuracy two orders of magnitude higher, and looking at the norm of the difference between the analytical projection and the wavelet multiplication. The results are shown in table 6.3 and we see that the error falls way below the threshold for any of the test calculations. In this table the computation time is also included and we see that the Interpolating multiplication is very efficient.

| $k$ | $E_2$ | Time | $k$ | $E_2$ | Time | $k$ | $E_2$ | Time |
|---|---|---|---|---|---|---|---|---|
| | $\epsilon = 10^{-3}$ | | | $\epsilon = 10^{-5}$ | | | $\epsilon = 10^{-7}$ | |
| 5 | 6.1e-06 | $< 1$ | 7 | 4.4e-07 | 1 | 9 | 1.0e-09 | 1 |
| 6 | 5.1e-06 | $< 1$ | 8 | 4.7e-09 | 1 | 10 | 1.7e-10 | 2 |
| 7 | 9.5e-06 | $< 1$ | 9 | 3.0e-09 | 1 | 11 | 3.8e-11 | 2 |
| 8 | 1.2e-05 | $< 1$ | 10 | 6.4e-09 | 1 | 12 | 2.2e-11 | 2 |
| 9 | 3.1e-06 | $< 1$ | 11 | 3.9e-09 | 2 | 13 | 4.0e-11 | 2 |
| | $\epsilon = 10^{-4}$ | | | $\epsilon = 10^{-6}$ | | | $\epsilon = 10^{-8}$ | |
| 6 | 8.9e-07 | 1 | 8 | 6.0e-09 | 1 | 10 | 1.7e-10 | 3 |
| 7 | 4.4e-08 | 1 | 9 | 1.0e-09 | 1 | 11 | 2.5e-11 | 3 |
| 8 | 5.0e-08 | 1 | 10 | 3.9e-10 | 1 | 12 | 9.5e-12 | 3 |
| 9 | 2.2e-07 | 1 | 11 | 3.0e-10 | 1 | 13 | 6.0e-12 | 3 |
| 10 | 1.1e-08 | 1 | 12 | 7.1e-10 | 2 | 14 | 6.6e-12 | 3 |

Table 6.1: Precision in multiplication. $\epsilon$ is the requested accuracy, $k$ is the polynomial order of the basis, $E_2$ is the $L^2$ norm of the relative error and time is given in seconds.

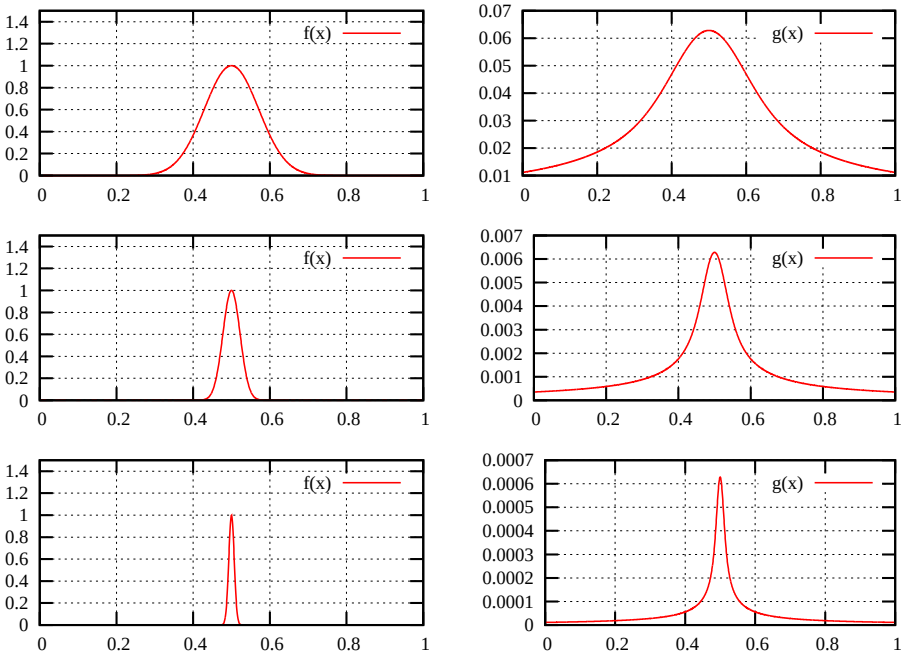


Figure 6.11: Potential $g(x)$ to the right, given by the corresponding distribution $f(x)$ to the left, all calculated as the solution of the Poisson equation.

## 6.4 Operator application

The original implementation of the operator application was done by [1], which arrived at a fully numerical solution of the Poisson equation in three dimensions

$$g(\boldsymbol{x}) = \int \frac{1}{||\boldsymbol{x} - \boldsymbol{y}||} f(\boldsymbol{y}) d\boldsymbol{y} \tag{6.5}$$

Figure 6.11 shows line plots of the calculated potential given by Gaussian distributions with different exponenents, positioned in the center of the unit cube.

### Accuracy

To test the accuracy of the operator application we apply it to a function $f$ that gives an analytic solution $g$

$$g(\boldsymbol{x}) = \sum_{i=1}^{3} e^{-\alpha(\boldsymbol{x}-\boldsymbol{x}_i)^2} \tag{6.6}$$

$$f(\boldsymbol{x}) = \nabla^2 g(\boldsymbol{x}) = \sum_{i=1}^{3} (4\alpha^2 |\boldsymbol{x} - \boldsymbol{x}_i|^2 - 6\alpha) e^{-\alpha(\boldsymbol{x}-\boldsymbol{x}_i)^2} \tag{6.7}$$

with $\alpha = 300$, $\boldsymbol{x}_1 = (0.5, 0.5, 0.5)$, $\boldsymbol{x}_2 = (0.6, 0.6, 0.5)$ and $\boldsymbol{x}_3 = (0.35, 0.6, 0.5)$. The results of this test are given in table 6.2, where we can see that the norm of the error is consistently below the threshold. These are updated results and the

| $k$ | $E_2$ | Time | $k$ | $E_2$ | Time | $k$ | $E_2$ | Time |
|---|---|---|---|---|---|---|---|---|
| | $\epsilon = 10^{-3}$ | | | $\epsilon = 10^{-5}$ | | | $\epsilon = 10^{-7}$ | |
| 5 | 4.0e-05 | 6 | 7 | 3.3e-07 | 17 | 9 | 2.7e-09 | 66 |
| 6 | 3.4e-05 | 3 | 8 | 3.4e-07 | 18 | 10 | 2.7e-09 | 49 |
| 7 | 3.1e-05 | 2 | 9 | 2.9e-07 | 10 | 11 | 2.7e-09 | 49 |
| 8 | 3.0e-05 | 3 | 10 | 2.9e-07 | 9 | 12 | 2.6e-09 | 52 |
| 9 | 2.9e-05 | 2 | 11 | 3.0e-07 | 9 | 13 | 2.6e-09 | 29 |
| | $\epsilon = 10^{-4}$ | | | $\epsilon = 10^{-6}$ | | | $\epsilon = 10^{-8}$ | |
| 6 | 5.1e-06 | 11 | 8 | 4.4e-08 | 36 | 10 | 2.5e-10 | 108 |
| 7 | 3.2e-06 | 9 | 9 | 2.8e-08 | 29 | 11 | 2.5e-10 | 77 |
| 8 | 3.0e-06 | 5 | 10 | 2.8e-08 | 31 | 12 | 2.5e-10 | 95 |
| 9 | 3.0e-06 | 5 | 11 | 2.9e-08 | 16 | 13 | 2.4e-10 | 89 |
| 10 | 3.0e-06 | 3 | 12 | 2.8e-08 | 20 | 14 | 2.4e-10 | 54 |

Table 6.2: Precision in the application of the Poisson operator where $\epsilon$ is the requested accuracy, $k$ is the polynomial order of the basis, $E_2$ is the $L^2$ norm of the error, and time is given in seconds.

improvements in computation time compared to what was presented in [1] is due to the optimization that is discussed later. These improvements are making the performance comparable to results obtained by Beylkin and coworkers [15].

## 6.5 Nuclear potential

An efficient linear scaling numerical Poisson solver with strict error control is of course interesting in its own right, but the main goal of the MRChem program is to carry out calculations on molecular systems, and a considerable amount of work has been put into extending the applicability beyond the treatment of single Gaussian distributions and into more interesting chemical systems. Accurate and efficient calculation of the nuclear potential is important in any area of computational chemistry and MRChem provides a method that is truly linear scaling with system size.

The true nuclear potential is emanating from the point charges of the nuclei in the system, which means that the charge distribution is a collection of delta functions. Delta functions cannot be represented by the multiwavelet basis, but by choosing very narrow Gaussian functions as nuclear charge distributions we are able to calculate the nuclear potential by application of the Poisson operator.

### Range of validity

By approximating the nuclear charge by a Gaussian distribution, we are making an error in the potential whenever evaluated *within* the charge distribution, that is within the radius where the charge is substantially (with respect to the requested accuracy) different from zero. This means that there is a range of validity of the potential, depending on the width of the nuclear Gaussian, and by increasing the Gaussian exponent, we can make the validity range approach the nuclear position.

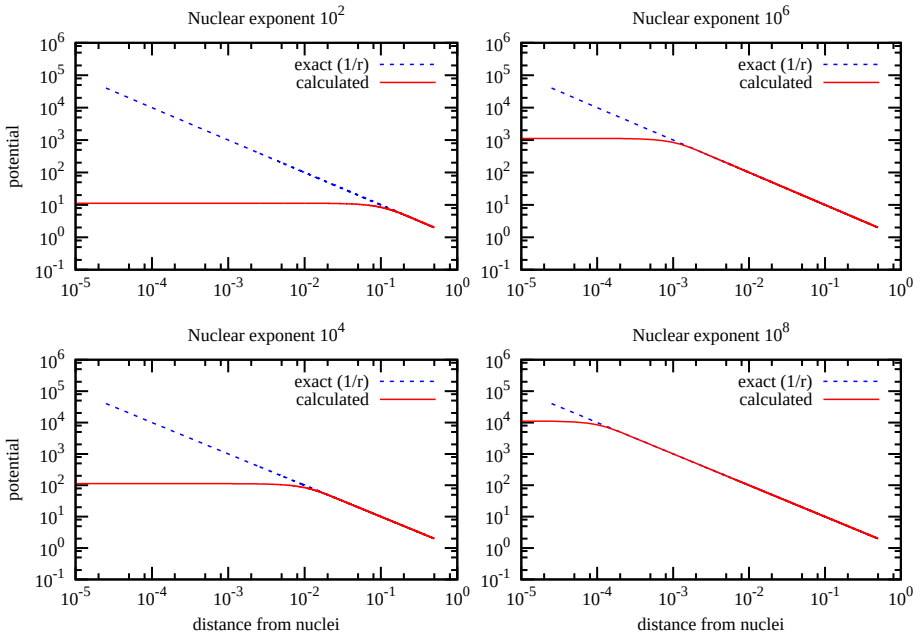


Figure 6.12: Plots of the potential from a hydrogen nucleus calculated using different Gaussian exponents for the nuclear charge.

Figure 6.12 shows the true nuclear potential along with the one calculated using different nuclear exponents, and we can see that by increasing the exponent from $10^2$ to $10^8$ we can make the range of validity of the calculated potential go from $r \sim 10^{-1}$ to $r \sim 10^{-4}$.

We know that the nuclear potential has an analytical solution, and it might seem unnecessary to obtain it by application of the Poisson operator. However, the wavelet formalism does not sit well with singular functions, and a direct projection of the $r^{-1}$ potential is not desireable. On the other hand, if we were able to obtain a *smoothed* nuclear potential with a similar validity range as above, the direct projection of this potential might be more efficient than the operator application. One such smoothed potential was obtained by the numerical separation of the Poisson integral kernel eq.(4.29). This expansion can be made valid with the same range and accuracy as the potential calculated by the operator.

## Linear scaling

One important aspect of the potential calculation is the scaling behavior with respect to system size. The banded structure of the matrix representation of the poisson operator gives prospects of linear scaling, and to test this property we calculate the nuclear potential given by an increasing number of water molecules. In this test both the number of nuclear charges *and* the volume of the system was increased. The system was expanded in one direction only, and the geometry of the $10H_2O$ case is shown in figure 6.16 together with an

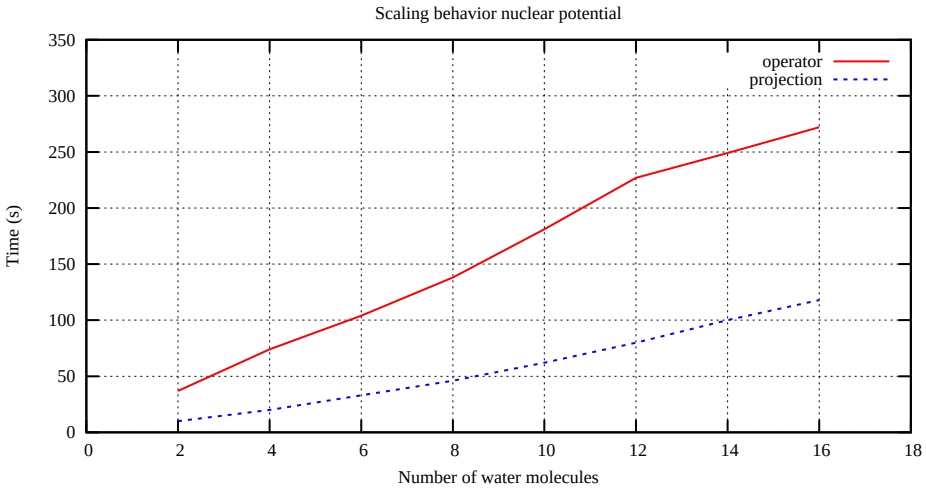


Figure 6.13: Scaling behavior of the calculation of the nuclear potential using the poisson operator and by direct projection of the kernel expansion

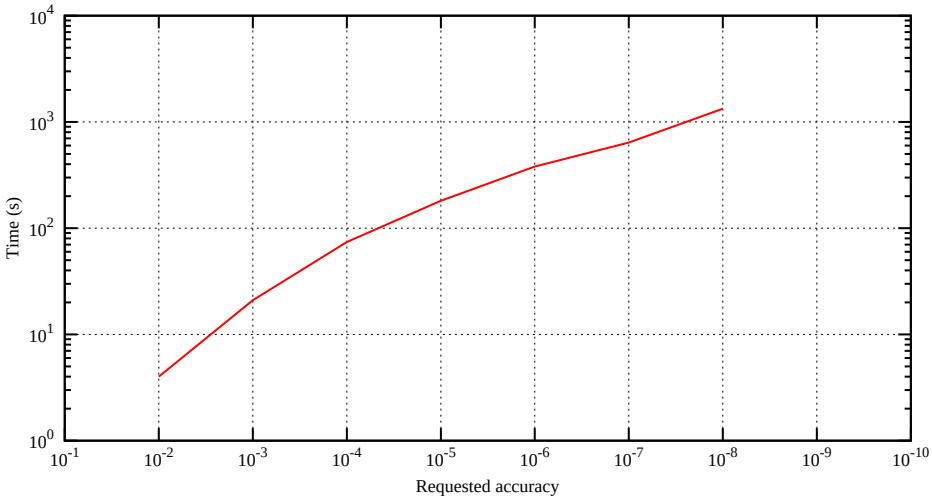


Figure 6.14: Scaling behavior with respect to requested accuracy on the calculation of nuclear potential for four water molecules by operator application.

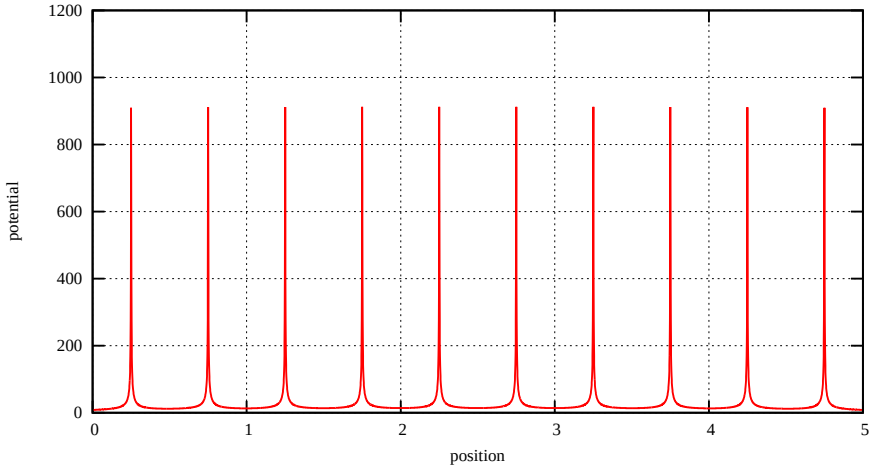


Figure 6.15: Lineplot of the nuclear potential of 10 water molecules as calculated by the Poisson operator. Line goes through the nuclear positions of the oxygen atoms.

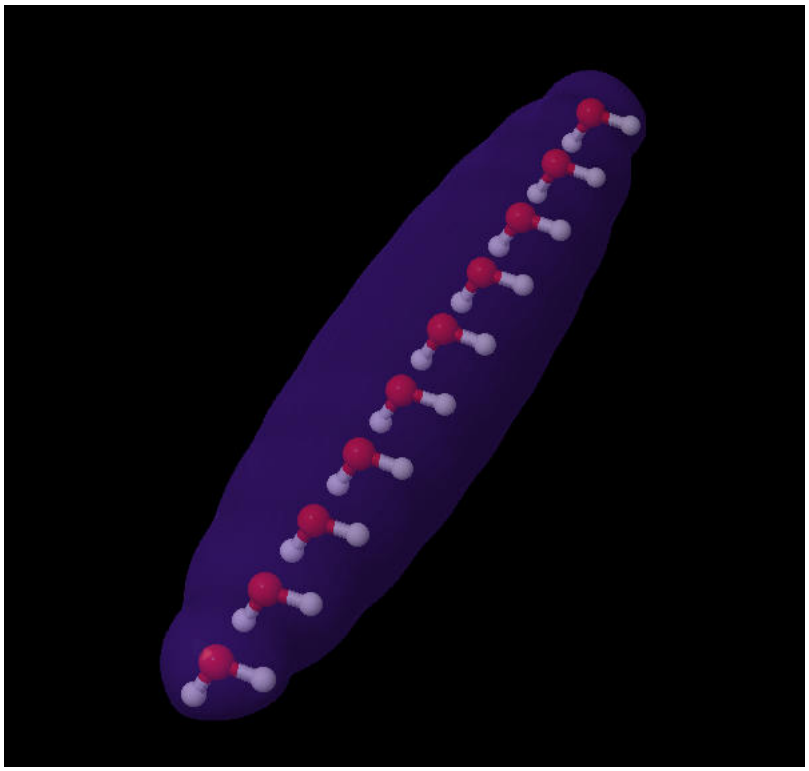

Figure 6.16: Isosurface plot of the nuclear potential of 10 water molecules as calculated by the poisson operator.

isosurface plot of the calculated potential.

Figure 6.15 shows the potential on the line that goes through the 10 oxygen atoms in figure 6.16. The calculation was done using a nuclear exponent of $10^6$ and the potential in figure 6.15 should be accurate to a distance of about $10^{-3}$ from each nuclear position. All calculations in figure 6.13 were done using a nuclear exponent of $10^6$ and accuracy $10^{-4}$, and we see a proper linear scaling of the operator application.

Figure 6.13 also shows the computation time of the direct projection of the kernel expansion to obtain the nuclear potential with the same accuracy and range of validity, and there is no question that the projection is more efficient than the operator application for nuclear potentials. Figure 6.14 shows the computation time for the nuclear potential for the $4H_2O$ system with different accuracy, calculated by application of the Poisson operator. Here we can see that the computation time more or less doubles when we increase the accuracy by an order of magnitude.

## 6.6 Electronic potential

The calculation of the electronic potential is a bit different from the nuclear potential in that there is no analytical solution and the calculation has to be done by operator application. Obviously, in order to calculate the electronic potential we need the electron density, and ultimately we would have MRChem optimizing the density based on Density Functional Theory (DFT). For the time being the electron density will be calculated using other Quantum Chemistry programs (Dalton, CFOUR) which provides the density matrix and the corresponding Gaussian basis set, which can then be projected onto the multiwavelet basis.

### Linear scaling

The scaling behavior of the calculation of the electronic potential was obtained the same way as for the nuclear potential; by calculation on an increasing number of water molecules. In these tests the electron density was projected for up to ten non-interacting water molecules and the potential were calculated on the entire system. Figure 6.18 shows the electronic potential on an isosurface of the density for the system with 10 water molecules. Figure 6.17 shows the grid partitioning used to represent the electronic potential.

Figure 6.19 shows the scaling behavior of the electronic potential calculation, which shows a proper linear scaling. One should note that the computation time of the electronic potential is almost an order of magnitude faster than the calculation of the nuclear potential on the same system with the same accuracy. The reason for this is that the electronic potential is much smoother than the nuclear potential and a deeper refinement is required in the latter.

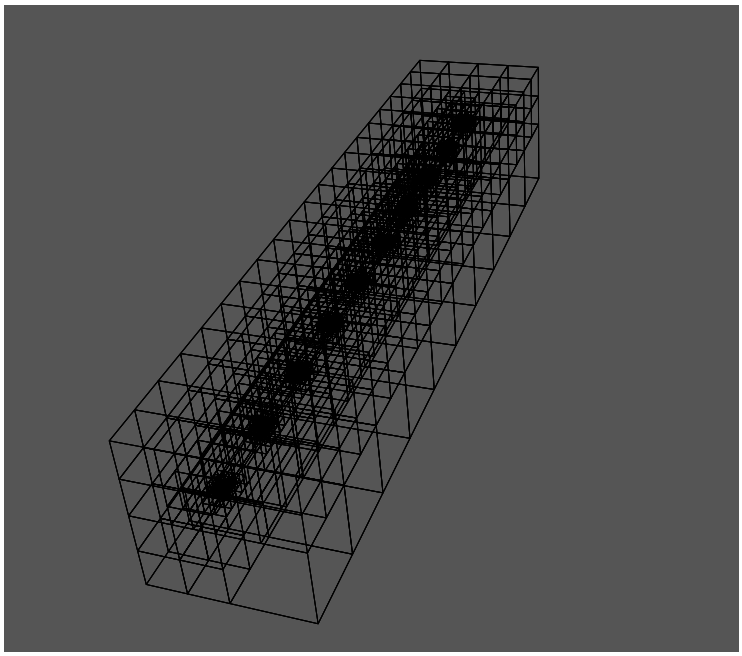

Figure 6.17: Grid refinement of the adaptive representation of the electronic potential of 10 water molecules.

## 6.7 Optimization

In the NS representation of the operator, the situation may arise where we need a `node` of the input function $f$ that is not present in the current `tree`-representation of the function, that is, the `node` must be generated by wavelet transform. This generated `node` is truly redundant in the `tree` representation, and at first we could be tempted to just calculate it using the filters, use it if it is needed in the operator application, and delete it immediately.

However, since we are dealing with a finite bandwidth of the operator, it is likely that the same generated `node` will be needed to evaluate several `nodes` of the result. In this case we would have to generate these `nodes` every time we encounter them, making the algorithm slower. To deal with this we can decide to store every new `node` as it is generated, which means that the next time the very same `node` is required, we will have it available directly, and only when the operator application has terminated these nodes are deleted.

A common limiting factor of finite element methods is the memory usage, since the number of basis functions required becomes huge, especially in multiple dimension. This is also the case for multiwavelets, and a typical function representation in three dimensions contains some thousand `nodes` and each `node` some thousand coefficients, making the total memory requirement to represent *one* function approach the GB order of magnitude. This means that we should strive to make the function representations as small as possible at any given moment.

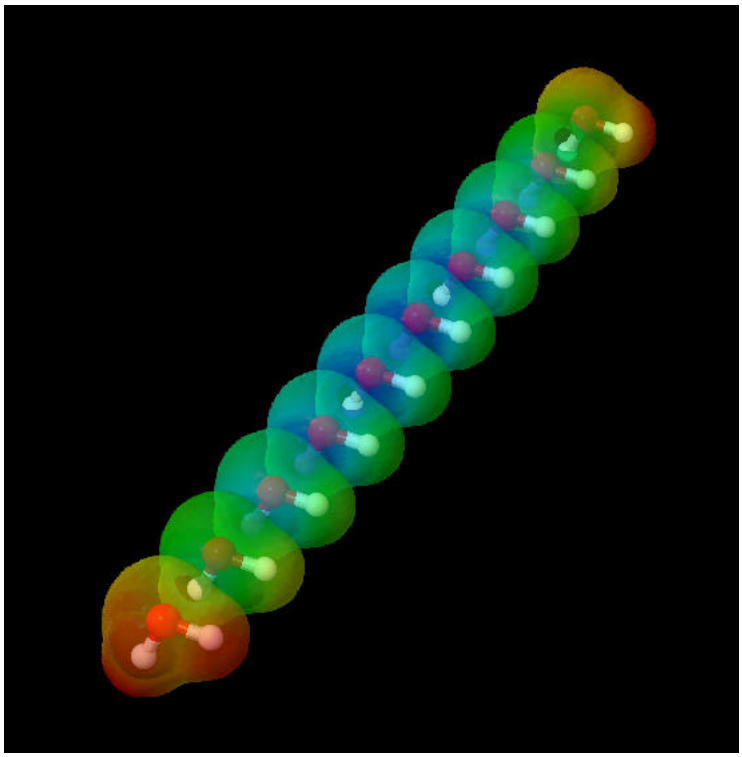

Figure 6.18: Electronic potential in color on an isosurface of the electron density for ten non-interacting water molecules.

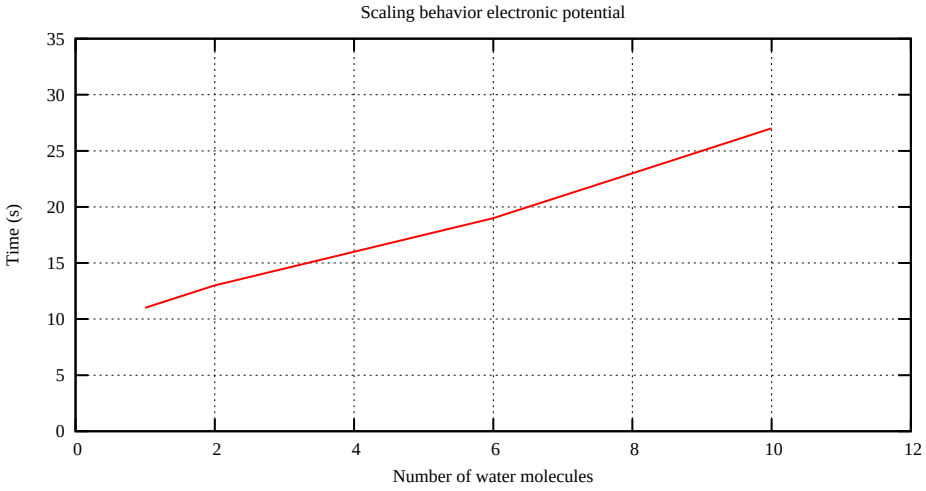


Figure 6.19: Computation time in the calculation of electronic potential for different number of water molecules.

It is therefore necessary to arrive at some compromise between the two approaches mentioned above, where we do not store all the generated `nodes` in memory, but still do not have to generate them every time we need them. In the operator algorithm there were three factors that limited the number of terms to consider in the matrix-vector multiplication; the scale, the bandwidth and the product of the norms. The latter enables one such compromise.

We can decide to keep only the norm of the generated `node` after it is obtained and used the first time, while its coefficients are deleted. The additional memory used to keep the norms is negligeable compared to the total function, and we are now in the position to perform the last thresholding test of the generated nodes without having to obtain their full set of coefficients every time, and only if the norm-test tells us that the `node` is required, the wavelet transform is carried out again.

This procedure makes sense only if *many* generated `nodes` needs to be considered in the first place, but the percentage of these that are *actually* needed in the operation is small. If only a few generated `nodes` are required, it should be possible to store them in memory altogether, and if more or less every `node` is above the norm threshold, we would be better of just calculating the full `node` *with* coefficients right away.

Table 6.3 shows some details of the electronic potential calculations performed on the water molecules in the linear scaling test in the previous section. The second column shows the number of original `nodes` in the electron density representation. The third column shows the number of *different* `nodes` that needed to be generated by wavelet transform during the application of the operator. The fourth column gives the number of times these generated `nodes` were fetched by the operator to test if the norm was sufficiently high, and the fifth column shows the number of times this norm test said that the `node` actually needed to be taken into account.

We can immediately see that since the number of times a generated `node` is fetched is in the order of millions, the proposed algorithm where the `nodes` were obtained by wavelet transform *every* time they were needed is bound to be inefficient. This is the way the original implementation was done, and this is why the computation time in the results presented in [1] were some five times slower that the results presented in table 6.2 in the previous section. The last column in table 6.3 gives the computation time using this algorithm.

Secondly, we see that the number of *different* `nodes` that are generated by the operator is 5-10 times the number of original `nodes` in the representation. In three dimensions these generated `nodes` will contain only one 8th of the coefficients of an original `node` since their wavelet coefficients will be zero. This means that keeping all the generated `nodes` for later use will approximately double the total memory requirement of representing the function. This will however dramatically improve the computation time since now only some thousand wavelet transforms needs to be carried out, in oppose to some millions in the previous algorithm. The sixth column in table 6.3 shows the computation

| $nH_2O$ | Nodes | | | | Time (sec) | | |
|---|---|---|---|---|---|---|---|
| | Orig. | Gen. | Fetched | Used | Full mem. | Opt. | Min mem. |
| 1 | 328 | 4026 | $3 \cdot 10^6$ | 17751 | 11 | 11 | 401 |
| 2 | 456 | 5378 | $4 \cdot 10^6$ | 23031 | 13 | 13 | 622 |
| 4 | 824 | 4448 | $4 \cdot 10^6$ | 22103 | 16 | 16 | 625 |
| 6 | 1128 | 5794 | $5 \cdot 10^6$ | 23784 | 19 | 19 | 849 |
| 8 | 1464 | 7200 | $7 \cdot 10^6$ | 28528 | 22 | 22 | 1008 |
| 10 | 1752 | 8834 | $1 \cdot 10^7$ | 35051 | 27 | 28 | 1484 |

Table 6.3: Number of nodes generated in the electron density during the calculation of the electronic potential for different number of water molecules.

time using the algorithm that keeps *all* the `nodes` and we can see a significant improvement compared to the first algorithm.

Now one would expect that the compromising algorithm presented above would give performances somewhere in between the two extremes, but as we can see from column seven in the table, the computation time is practically the same as for the fast "full memory" algorithm, while the maximum number of generated `nodes` *with* coefficients at any moment never exceeds 20. This means that we get the best of both worlds and we have a fast algorithm where the representation of the density function never grows noticably.

# Chapter 7

# Discussion

In the current implementation of MRChem we have achieved a fully adaptive construction of functions in any finite dimension which can be obtained by projection, addition, multiplication or application of the Poisson operator, and the representations obtained are consistently within the accuracy threshold.

With the optimization step presented in this thesis we have improved the performance of the code to give comparable results to the ones reported by Beylkin and coworkers [15] for the solution of the Poisson equation. The operator application is shown to be truly linear scaling with system size, both in the case of (practically) point charges in the calculation of the nuclear potential, and for more general charge distributions in the calculation of the electronic potential.

One aspect that is not treated in this thesis is the implementation of parallel algorithms. The wavelet formalism is well suited for parallelization where the work load is easily distributed by letting each processor calculate only parts of the `tree` representation and the communication between processors during a `tree` construction is limited to the function norm that is updated between every scale in the adaptive algorithm. This norm is required for the refinement test of each separate `node`. A lot of work has been put into the parallel implementation an the current results looks promising.

An interface to other computational chemistry programs has been implemented, where molecular geometries and a suitable starting guess for the electron density can be obtained, and we are in the position to start implementing the Kohn-Sham equations for the iterative solution of the electronic structure of molecular systems.

The Kohn-Sham equations can be rewritten in an integral form in the same manner as the Poisson equation, and the ground state can be obtained by iterative solution of the following equation

$$\psi(\boldsymbol{x}) = -2 \int H^{(\mu)}(\boldsymbol{x}, \boldsymbol{y}) V(\boldsymbol{y}) \psi(\boldsymbol{y}) d\boldsymbol{y} \tag{7.1}$$

where $H^{(\mu)}$ is the Helmholtz kernel which is given, for free boundary conditions

in three dimensions, as

$$H^{(\mu)}(\boldsymbol{x}, \boldsymbol{y}) = \frac{e^{-\mu||\boldsymbol{x}-\boldsymbol{y}||}}{||\boldsymbol{x}-\boldsymbol{y}||} \tag{7.2}$$

This kernel can be numerically separated in the same way as the Poisson kernel as an expansion of Gaussians, and can be obtained to any accuracy and range of validity [1]. In this way the solution of the Kohn-Sham equations becomes very similar to the solution of the Poisson equation, and can be done using the same machinery.

A current bottleneck in the molecular calculations is the projection of the electron density from the Gaussian basis set to the multiwavelet basis. Gaussian functions are hard to reproduce by simple polynomials and the projection of these functions is not cheap. Since the size of the density matrix grows as the square of the number basis functions, the number of Gaussian projection quickly becomes the limiting factor. In the current implementation the projection of the density easily becomes several orders of magnitude more expensive than the calculation of the potential given by this density.

However, the initial projection is meant only as a crude starting guess for the SCF iterations and it is not necessary to employ an accurate expansion with a large Gaussian basis. The initial projection can also be done with less accuracy than what is required in the result, which will speed up the process.

# List of Figures

# List of Tables

# Bibliography


[1] E. Fossgaard, *Fast numerical algorithms for solving numerically separable integral equations in multiple dimensions*, Submitted.

[2] T. N. G. Strang, *Wavelets and filter banks*, Wellesley College, 1996.

[3] G. Beylkin, *Fast wavelet transforms and numerical algorithms 1*, Comm. Pure. Applied. Math, 1991.

[4] B. K. Alpert, *A class of bases in $L^2$ for the sparse representation of integral operators*, SIAM J. Math. Anal., 1993.

[5] R. Harrison, *Multiresolution Quantum Chemistry: Basic theory and initial applications*, J. Chem. Phys., 2004.

[6] A. Niklasson, *Multiresolution density-matrix approach to electronic structure calculations*, Phys. Rev., 2002.

[7] T. A. Arias, *Multiresolution analysis of electronic structure: semicardinal and wavelet bases*, Rev. Mod. Phys., 1999.

[8] W. Kohn, *Self-consistent equations including exchange and correlation effects*, Rev. Phys., 1965.

[9] T. L. Beck, *Real-space mesh techniques in density functional theory*, Rev. Mod. Phys., 2000.

[10] F. Keinert, *Wavelets and Multiwavelets*, Chapman and Hall/CRC, 2003.

[11] B. K. Alpert, *Adaptive solution of partial differential equations in multiwavelet bases*, J. Comp. Phys., 2002.

[12] D. L. Donoho, *Interpolating wavelet transforms*, 1992.

[13] C.J.Tymczak, *Separable and nonseparable multiwavelets in multiple dimensions*, J. Comput. Phys., 2002.

[14] G. Beylkin, *On the fast algorithm for multiplication of function in the wavelet bases*, International conference wavelets and applications, Toulouse, 1992.

[15] G. Beylkin, *Fast adaptive algorithms in the non-standard form for multidimensional problems*, Appl. Comp. Harmonic Analysis, 2008.