DOCTOR OF PHILOSOPHY

Simulation of the micromagnetic behaviour of nanoelements by an adaptive wavelet method

Hines, Geneviève

Award date:
2003

Awarding institution:
University of Wales, Bangor

Simulation of the micromagnetic behaviour of nanoelements by an adaptive wavelet method.

Submitted to the University of Wales
In Candidature for the Degree of
Doctor of Philosophy

by

Geneviève Hines

January 2003

School of Informatics,
University of Wales, Bangor
Dean Street,
Bangor,
Gwynedd LL57 1UT
U.K.

Summary

This thesis presents an adaptive wavelet method for the simulation of the time evolution of the magnetization of nanoelements and the results obtained with the numerical simulations.

The evolution of the magnetization ($\boldsymbol{m}$) of a nanoelement situated in a non-magnetic medium is modeled mathematically with the Landau-Lifshitz equation. One term in that equation, the demagnetizing field, is derived from the solution of a Poisson equation whose right hand side depends on $\boldsymbol{m}$. Of these two equations, coupled via the magnetization, the first one is discretized by a pointwise Euler scheme while the other is solved with an adaptive wavelet method.

The multi-level features of wavelet bases are used to cope with the sharp variations in the magnetization strength at the interface between the nano-element (where $|\boldsymbol{m}| = 1$) and the surrounding non-magnetic region (where $|\boldsymbol{m}| = 0$), and in the magnetization direction within the nanoelement, which may occur under certain circumstances with the formation of narrow domain walls. The aim of the adaptive scheme is to make maximum use of an affordable number of degrees of freedom by concentrating the computational resources in the locations where the sharpest variations in $\boldsymbol{m}$ are situated. The challenge for such a method is to ensure that the size of the memory and the number of operations remains proportional to the number of degrees of freedom.

Acknowledgements

I would like to thank my supervisor, Dr Roberts, for giving me the opportunity to learn something about wavelets and for giving me the chance, through visits and conferences to meet other research students in the same field. I also would like to thank him for his friendly guidance and his support.

All my thanks to Phil Ridley and Adam Spargo for their helpful and useful comments on the physical aspects of the model and on the numerical results.

I would like to thank the authors of the Multilevel Library for making available such a powerful piece of software. It has been a real pleasure to work with it. I am also very grateful to Mario Mommer for letting me use his fast code for the calculation of inner products of wavelets. Thank you!

And finally my heart-felt thanks go to Peter, my husband, for his great way of making sense and rearranging the most back to front sentences.

Introduction

The work presented in this thesis arises from the micromagnetics modeling group at the University of Wales Bangor, where several researchers have been working on the magnetic behaviour of nano-structured permalloy. The motivation for this study lies in this type of material's potential future use for high density magnetic storage media. The material is considered discretized into a nanoelement structure at the sub-micron level and we use a mathematical model to predict behaviour such as magnetization reversal and domain formation. The simulation is done by a numerical method. The mathematical model used in this thesis is based on the Landau-Lifshitz equation.

In 2000, Philip Ridley presented in his doctoral thesis [25] results obtained for this equation by a finite element method. By contrast, the numerical method used in what follows is a combination of a finite difference method and a wavelet method. The wavelet part of the scheme is used to solve a Poisson equation with homogeneous boundary conditions on a mathematical domain representing a small magnetic region, termed nanoelement or platelet, which is embedded in a larger non-magnetic region. The wavelet solver is used repeatedly, coupled with the Landau-Lifshitz equation proper, which is an equation of motion implemented with a scheme based on finite differences, such as an Euler scheme. The difficulty arises from modelling the interface of the platelet with the outer, non-magnetic region and from the expected formation in the solution of localized less smooth features which do not remain stationary, such as the formation over time of domain walls. Different scales of resolution are necessary on the overall domain: a coarser resolution away from the nanoelement, where the magnetization is zero, and a finer resolution near and inside the nanoelement. The finite element method implemented by Ridley addresses this problem

by the use of a fixed non-uniform unstructured mesh of triangular elements, which is set at the start of the simulation. The approach presented here makes use of a wavelet adaptive strategy in order to increase the resolution scale dynamically as and where the domain walls develop. However, it is expected that the flexibility added to the method in this way will also bring an overhead in data handling and in general computational complexity. The algorithms for the wavelet part of implementation are based the following papers [3, 4, 9, 8, 26], and others as specified in the text. The programming itself was done in C++ and makes extensive use of the Multilevel Library, a research library for multi-level methods and wavelets developed at the IGPM[1], at the RWTH Aachen[2], in Germany.

This thesis is organized as follows. In the first chapter the mathematical equations used used to model the nanoelement magnetization will be presented and the discretization of these equations will be described. The second chapter gives details on wavelet bases in general and more particularly on B-spline wavelet bases defined on the unit cube. Some of the properties of the bases, relevent to the application at hand, are given. In the third chapter, an adaptive wavelet scheme for the solution of the Poisson equation is described. Chapters 4,5 and 6 are concerned with the implementation of the tools necessary for the realization of this adaptive scheme. These tools are, in order, the calculation of a sparse wavelet representation of a function, the implementation of a fast matrix-vector multiplication together with the dynamic calculation of the the entries of the Laplace operator matrix, and finally the pointwise evaluation of a sparse wavelet expansion. In chapter 7, the adaptive Poisson solver is tested numerically for an example relevent to the micromagnetics application. Finally, chapter 8 presents the results obtained for the

[1]Institut für Geometrie and Praktische Mathematik

[2]Rheinisch-Westfälische Technische Hochschule, the faculty of Mathematics, computer sciences and natural sciences of Aachen

numerical simulation of the magnetization of the nanoelement.

Chapter 1

Mathematical models of micromagnetics

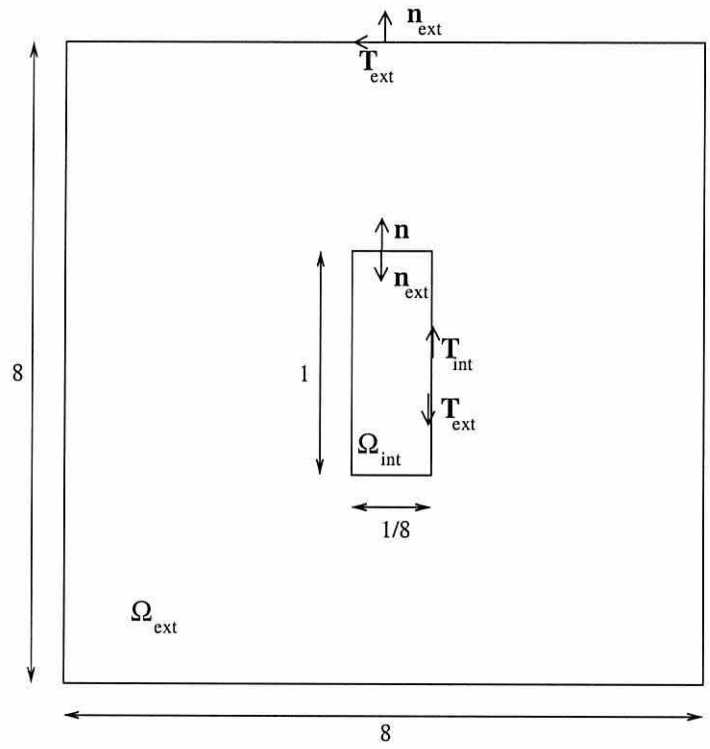
1.1 The mathematical model

The mathematical model presented here concerns the evolution of the magnetization of one nanoelement from the instant when an external magnetic field, previously applied to the nanoelement, is varied. The nanoelement considered is rectangular in shape and, as in [25], it has dimensions $0.2\mu m$ width, $1.6\mu m$ length and $0.2nm$ thickness.

The domain of solution

As a starting point, the nanoelement is represented on the two-dimensional plane as an elongated rectangle of ratio 1 to 8, embedded in an area of non-magnetic material, which is represented by a square of larger dimensions as illustrated in Figure 1.1. The 2D assumption can be justified partly by the small thickness of the nanoelement and partly by the

Figure 1.1: The domain of solution in the case of a rectangular nanoelement.



small magnitude of the out of plane component of the applied field. The magnetic region, denoted by Ω_{int}, and the non-magnetic region, denoted by Ω_{ext}, together form the domain of solution, which will be denoted by Ω.

Electromagnetics notations

The notations for the physical quantities used in the model are now given. Vectors are typeset in bold face, while normal font is used for scalar quantities, and Cartesian coordinates are assumed at all times.

For clarity, a vector field $\boldsymbol{Y}(x_1, x_2, x_3, t)$ will often be written $\boldsymbol{Y}$ when the context is clear. The following notation is assumed throughout:

- $\boldsymbol{M}$ denotes the magnetization (magnetic moment per unit volume).

- $\boldsymbol{H}_t$ denotes the total effective vector field. It is the sum of all the magnetic fields applied to the domain. The evolution of the magnetization depends solely on $\boldsymbol{H}_t$.

- $\boldsymbol{H}_{app}$ is an externally applied field. In this work as in [25] it is a uniform field in the direction of the x_1 or the x_2 axes.

- $\boldsymbol{H}_a$ denotes the anisotropy field. A uniaxial material, such as the material considered here, has exactly one preferred or easy magnetization direction. The anisotropy field is the field associated with the energy required to rotate the magnetization moment away from its preferred axis. The relation between the anisotropy energy E_a and the anisotropy field is $\boldsymbol{H}_a = -dE_a/d\boldsymbol{M}$.

- $\boldsymbol{H}_{ex}$ denotes the exchange field.

- $\boldsymbol{H}_d$ denotes the demagnetizing field.

- $\hat{\boldsymbol{e}}$ denotes a unit vector along the direction of the preferred, or easy, axis.

- M_s denotes the saturation magnetization of the material. It is a material constant which represents the strength of the sum of the $\boldsymbol{M}$ when all the magnetic moments are aligned in the external field direction.

- K denotes the anisotropy constant.

- H_k is the anisotropy field strength. It is given here by $H_k = 2K/M_s$.

- A denotes the material dependant exchange constant.

- λ denotes the dissipative constant.

- γ denotes the magneto-mechanical or gyromagnetic ratio.

The Landau-Lifshitz equation

The equation used to model the time evolution of the magnetism of a nanoelement is the Landau-Lifshitz (LL) equation of motion

$$\frac{\partial \boldsymbol{M}}{\partial t} = -|\gamma| \boldsymbol{M} \times \boldsymbol{H}_t - \frac{\lambda}{M_s^2}\big(\boldsymbol{M} \times (\boldsymbol{M} \times \boldsymbol{H}_t)\big) \tag{1.1}$$

This equation has been used successfully to model cases where the dissipative constant is small [25]. This equation in particular is suitable for the value $\lambda = 1$ which will be used in this work. The total effective field in the equation (1.1) is dependent on $\boldsymbol{M}$ and may be decomposed into the sum of the four effective fields

$$\boldsymbol{H}_t = \boldsymbol{H}_{app} + \boldsymbol{H}_{ex} + \boldsymbol{H}_a + \boldsymbol{H}_d.$$

The dependence of the fields $\boldsymbol{H}_{ex}$ and $\boldsymbol{H}_a$ on the magnetization is given by

$$\boldsymbol{H}_{ex} = \frac{A}{2M_s^2}\nabla^2 \boldsymbol{M}, \tag{1.2}$$

$$\boldsymbol{H}_a = \frac{K}{2M_s^2}(\boldsymbol{M} \cdot \hat{\boldsymbol{e}})\hat{\boldsymbol{e}}, \tag{1.3}$$

whereas $\boldsymbol{H}_{app}$ is a constant. The demagnetization field $\boldsymbol{H}_d$ is also dependent on the magnetization, and a derivation is given in the following subsection.

Derivation of the demagnetization field

This derivation of the demagnetizing field is taken from [25]. It is based on the extension of the Maxwell laws of electromagnetics to the field of micromagnetics. Let $\boldsymbol{D}$ denote the

electric flux density and $\boldsymbol{J}$ denote the electric current density. For a given magnetization moment $\boldsymbol{M}$, the demagnetizing field is given by

$$\nabla \times \boldsymbol{H}_d = \boldsymbol{J} + \frac{\partial \boldsymbol{D}}{\partial t}$$

For current free regions, $\nabla \times \boldsymbol{H}_d = 0$, and so $\boldsymbol{H}_d$ is an irrotational vector field. Now let $\boldsymbol{B}$ denote the induced field

$$\boldsymbol{B} = \boldsymbol{H}_d + 4\pi \boldsymbol{M} \qquad \text{and} \qquad \nabla \cdot \boldsymbol{B} = 0,$$

which implies that

$$\nabla \cdot (\boldsymbol{H}_d + 4\pi \boldsymbol{M}) = 0.$$

Since $\boldsymbol{H}_d$ is an irrotational field, we may introduce a scalar potential ϕ such that

$$\begin{aligned} \boldsymbol{H}_d &= -\nabla \phi, \\ \Delta \phi &= 4\pi \nabla \cdot \boldsymbol{M}, \end{aligned}$$

where ϕ can be seen to satisfy the Poisson equation. In what follows, we depart from the electromagnetics convention of denoting a potential by ϕ, in order to avoid conflict with the common notation used for scaling functions, which play an important role in wavelet methods. Instead, the potential will be denoted by U, which is conventional notation in numerical mathematics for the solution of a partial differential equation.

Using this new convention for notation, the above equations may be rewritten as

$$\boldsymbol{H}_d = -\nabla U, \tag{1.4}$$

$$\Delta U = 4\pi \nabla \cdot \boldsymbol{M}. \tag{1.5}$$

Continuing with the derivation of the demagnetizing field $\boldsymbol{H}_d$, it may be seen that outside the magnetic material, $\boldsymbol{M} = 0$, and so denoting by U_{ext} the potential in that region and by U_{int} the potential in the nanoelement, it follows that

$$\Delta U_{ext} = 0, \qquad \boldsymbol{x} \in \Omega_{ext}, \tag{1.6}$$

$$\Delta U_{int} = 4\pi \nabla \cdot \boldsymbol{M}, \qquad \boldsymbol{x} \in \Omega_{int}. \tag{1.7}$$

The interface between the magnetic and the non-magnetic regions is denoted by $\partial\Omega_{int} = \Omega_{int} \cap \Omega_{ext}$, and across this region continuity conditions are imposed between U_{ext} and U_{int}. From the continuity of the normal component of $\boldsymbol{B}$ and the tangential component of $\boldsymbol{H}_d$

$$\frac{\partial U_{int}}{\partial n} - \frac{\partial U_{ext}}{\partial n} = 4\pi \boldsymbol{M} \cdot \boldsymbol{n}, \qquad \text{on } \partial\Omega_{int}, \tag{1.8}$$

$$U_{int} = U_{ext}, \qquad \text{on } \partial\Omega_{int}, \tag{1.9}$$

where $\boldsymbol{n}$ is the unit normal away from Ω_{int}. Finally, the potential decreases to zero at infinity, which is modelled by

$$U_{ext} = 0 \quad \text{on } \partial\Omega, \tag{1.10}$$

where $\partial\Omega$ denotes the outside boundary of the finite, square, non-magnetic domain, which

is located at a distance large enough so as not to affect the solution near the nanoelement.

The Landau-Lifshitz equation in reduced units

Before doing any numerical calculation the effective field terms will be rescaled with respect to the anisotropy field strength and the Landau-Lifshitz equation will be phrased in reduced units. The anisotropy field strength is defined by

$$H_k = \frac{2K}{M_s}$$

and it follows that the reduced total effective field will be

$$\boldsymbol{h}_t = \frac{\boldsymbol{H}_t}{H_k},$$

the reduced time

$$\tau = t|\gamma|H_k,$$

the reduced dissipative constant

$$\alpha = \frac{\lambda}{|\gamma|M_s},$$

and the reduced magnetization

$$\boldsymbol{m} = \frac{\boldsymbol{M}}{M_s}.$$

The reduced magnetization is in effect a normalized $\boldsymbol{M}$, so $|\boldsymbol{m}| = 1$. The components of the total effective field in reduced units are

$$\boldsymbol{h}_a = (\boldsymbol{m} \cdot \hat{\boldsymbol{e}})\, \hat{\boldsymbol{e}}, \tag{1.11}$$

$$\boldsymbol{h}_{ex} = C_{ex} \nabla^2 \boldsymbol{m}, \qquad \text{where } C_{ex} = \frac{A}{4K}\, C_u^{-2}, \tag{1.12}$$

$$\boldsymbol{h}_d = -C_d \nabla u, \qquad \text{where } C_d = \frac{M_s^2}{2K}. \tag{1.13}$$

In the equations above, u is the solution of the Poisson equation described in the previous section, with the magnetization in reduced units, and C_u denotes the coefficient for the unit change between the unit of length used in the Poisson solver and the unit of length consistent with the unit system used for the parameters A and K.

Finally, the equation (1.1) in reduced units reads

$$\frac{\partial \boldsymbol{m}}{\partial \tau} = -\boldsymbol{m} \times \boldsymbol{h}_t - \alpha\big(\boldsymbol{m} \times (\boldsymbol{m} \times \boldsymbol{h}_t)\big). \tag{1.14}$$

Further, by writing $\boldsymbol{m} = (m_1, m_2, m_3)$ and $\boldsymbol{h}_t = (h_1, h_2, h_3)$ in terms of cartesian coordinates, and expanding the vector products, the equation (1.14) has the following component by component expression

$$\begin{aligned}
\frac{\partial m_1}{\partial \tau} &= -(m_2 h_3 - m_3 h_2) - \alpha\big(m_2(m_1 h_2 - m_2 h_1) - m_3(m_3 h_1 - m_1 h_3)\big) \\
\frac{\partial m_2}{\partial \tau} &= -(m_3 h_1 - m_1 h_3) - \alpha\big(m_3(m_2 h_3 - m_3 h_2) - m_1(m_1 h_2 - m_2 h_1)\big) \\
\frac{\partial m_3}{\partial \tau} &= -(m_1 h_2 - m_2 h_1) - \alpha\big(m_1(m_3 h_1 - m_1 h_3) - m_2(m_2 h_3 - m_3 h_2)\big).
\end{aligned}$$

Having presented the mathematical equations modelling the physical problem, it remains to describe the numerical scheme implemented in order to create numerical simulations for the behaviour of $\boldsymbol{M}$ in the nanoelement. A general outline of this scheme will be given in the next section, while the details of the wavelet numerical model and its implementation will be the subject of the next chapters.

1.2 The numerical model

The numerical model is split into two parts. One part consists of a time stepping scheme to model the equation of motion (1.14) and the other is dedicated to the calculation of the effective field components, and in particular the demagnetizing field which is derived from the solution of a Poisson equation, as seen in (1.4) and (1.5). The Poisson equation will be solved at each time step by a variational method, in terms of a wavelet basis, while the time stepping scheme will act on the physical values of the magnetization and the applied field.

1.2.1 A simple time stepping scheme: the Euler scheme.

The Landau-Lifshitz equation of motion (1.14) will be solved at discrete times τ_n, where $\tau_n = \tau_0 + n\Delta\tau$ for some inital time τ_0. The time-step $\Delta\tau$ will be kept constant during the whole simulation. The evolution of the magnetization with time will be implemented using the simplest time stepping scheme, which is the explicit forward Euler scheme. The Euler scheme is also called the tangent method and it consists of approximating the function $\boldsymbol{m}(\tau)$ by a piecewise linear polynomial. At time τ_0, the approximation coincides with the value $\boldsymbol{m}(\tau_0)$. At time $\tau_n > \tau_0$, the value of $\boldsymbol{m}$ will be approximated by $\boldsymbol{m}_n$, which is

calculated from the approximation of $\boldsymbol{m}$ at time τ_{n-1}. The scheme may summarized by

$$\begin{aligned} \boldsymbol{m}_0 &= \boldsymbol{m}(\tau_0), \\ \boldsymbol{m}_n &= \boldsymbol{m}_{n-1} + \Delta\tau \frac{\partial \boldsymbol{m}}{\partial \tau}(\boldsymbol{m}_{n-1}), \qquad \tau_n > \tau_0. \end{aligned}$$

Using the right hand side of the Landau-Lifshitz equation for $\partial\boldsymbol{m}/\partial\tau$, the scheme is

$$\boldsymbol{m}_0 = \boldsymbol{m}(\tau_0), \tag{1.15}$$

$$\boldsymbol{m}_n = \boldsymbol{m}_{n-1} - \Delta\tau\Big(\boldsymbol{m}_{n-1} \times \boldsymbol{h}_t + \alpha\big(\boldsymbol{m}_{n-1} \times (\boldsymbol{m}_{n-1} \times \boldsymbol{h}_t^{n-1})\big)\Big), \tag{1.16}$$

where $\boldsymbol{h}_t^{n-1}$ denotes the value of the total effective field calculated at step $n-1$. This method is a first order method in the sense that it is exact for polynomials of degree 1 but not for higher degree polynomials and the local truncation error is $\mathcal{O}(\Delta\tau)$. The size of the time-step $\Delta\tau$ affects the stability of the scheme and it is empirically set to a small enough value to keep the size of the errors generated under control.

1.2.2 A variational form for the derivation of the scalar potential

The problem of solving a Poisson equation with homogeneous boundary condition, stated as:

Find u such that:

$$\begin{aligned} \Delta u &= f, \qquad \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) &= 0, \qquad \boldsymbol{x} \in \partial\Omega, \end{aligned}$$

is usually restated as a variational problem:

Find $u \in H_0^1(\Omega)$ such that:

$$\int_\Omega \nabla v \cdot \nabla u = \int_\Omega v f, \quad \text{for all } v \in H_0^1(\Omega),$$

which, by the Lax-Milgram theorem, is known to have a unique solution. The aim of this section is to write the equations (1.6) and (1.7), subject to the interface conditions (1.8) to (1.10) in a form equivalent to the one above. In a two dimensional setting, the two main equations are written in variational form as:

$$\forall v \in H_0^1(\Omega), \quad \iint_{\Omega_{int}} (\Delta u_{int})\, v\, dx_1\, dx_2 + \iint_{\Omega_{ext}} (\Delta u_{ext})\, v\, dx_1\, dx_2 = \iint_{\Omega_{int}} 4\pi (\text{div } \boldsymbol{m})\, v dx_1\, dx_2.$$

Using the identity for the divergence of the product of a vector function ($\boldsymbol{F}$) by a scalar function (g),

$$\text{div } (\boldsymbol{F} g) = g \text{ div } \boldsymbol{F} + \boldsymbol{F} \cdot \textbf{ grad } g,$$

and writing Δu as div **grad** u, it follows that

$$\iint_{\Omega_{int}} (\Delta u_{int}) v dx_1\, dx_2 = \iint_{\Omega_{int}} (\text{div } (v \textbf{ grad } u_{int}) - \textbf{ grad } u_{int} \cdot \textbf{ grad } v)\, dx_1\, dx_2,$$

and similarly for u_{ext}.

By Green's theorem,

$$\iint_{\Omega_{int}} \text{div } (v \textbf{ grad } u_{int})\, dx_1\, dx_2 = \oint_{\partial \Omega_{int}^+} v(\textbf{ grad } u_{int}) \cdot \boldsymbol{n}\, ds,$$

where $\oint_{\partial\Omega_{int}^+}$ denotes the integration along curve Ω_{int} in the anticlockwise direction.

For the non-magnetic domain, which has an inside boundary ($\partial\Omega_{int}$) and an outside boundary ($\partial\Omega$),

$$\iint_{\Omega_{ext}} \text{div } (v \ \mathbf{grad}\ u_{ext})\, dx_1\, dx_2 = \oint_{\partial\Omega_{int}^-} v(\ \mathbf{grad}\ u_{ext})\cdot \boldsymbol{n}_{ext}\, ds + \oint_{\partial\Omega^+} v(\ \mathbf{grad}\ u_{ext})\cdot \boldsymbol{n}_{ext}\, ds.$$

By reversing the direction of integration on $\partial\Omega_{int}$ and using the boundary condition $v = 0$ on $\partial\Omega$, one has

$$\iint_{\Omega_{ext}} \text{div } (v \ \mathbf{grad}\ u_{ext})\ dx_1\, dx_2 = -\oint_{\partial\Omega_{int}^+} v(\ \mathbf{grad}\ u_{ext})\cdot \boldsymbol{n}\, ds$$

The variational form of the equation can be rewritten as

$$\begin{aligned} \forall v \in H_0^1(\Omega), \quad & -\iint_{\Omega_{int}} \mathbf{grad}\ u_{int} \cdot \mathbf{grad}\ v\ dx_1\, dx_2 - \iint_{\Omega_{ext}} \mathbf{grad}\ u_{ext} \cdot \mathbf{grad}\ v\ dx_1\, dx_2 \\ = & \iint_{\Omega_{int}} 4\pi(\ \text{div}\ \boldsymbol{m})\, v\ dx_1\, dx_2 - \oint_{\partial\Omega_{int}^+} v((\ \mathbf{grad}\ u_{int})\cdot \boldsymbol{n} - (\ \mathbf{grad}\ u_{ext})\cdot \boldsymbol{n})\, ds \end{aligned}$$

Using the first interface condition (1.8),

$$\begin{aligned} \forall v \in H_0^1(\Omega), \quad & \iint_{\Omega_{int}} \mathbf{grad}\ u_{int} \cdot \mathbf{grad}\ v\ dx_1\, dx_2 + \iint_{\Omega_{ext}} \mathbf{grad}\ u_{ext} \cdot \mathbf{grad}\ v\ dx_1\, dx_2 \\ = & -\iint_{\Omega_{int}} 4\pi(\ \text{div}\ \boldsymbol{m})\, v\ dx_1\, dx_2 + \oint_{\partial\Omega_{int}^+} 4\pi(\boldsymbol{m}\cdot\boldsymbol{n})\, v\ ds. \end{aligned}$$

Finally, denoting by u the prolongation of u_{ext} into Ω_{int} by u_{int}, this comes to

$$\iint_{\Omega} \mathbf{grad}\ u \,.\, \mathbf{grad}\ v\ dx_1\, dx_2 = \quad -\iint_{\Omega_{int}} 4\pi(\ \mathrm{div}\ \boldsymbol{m})\, v\ dx_1\, dx_2 + \oint_{\partial\Omega_{int}^{+}} 4\pi(\boldsymbol{m} \,.\, \boldsymbol{n})\, v\ ds, \quad \forall v \in H_0^1(\Omega), \quad (1.17)$$

where the right hand side can be seen to be independent of the solution u. This formulation is equivalent to that of the variational problem stated at the beginning of this section. The next step towards resolving this problem numerically consists of discretizing it by replacing the condition $\forall v \in H_0^1(\Omega)$ by a discrete one $\forall \psi_\lambda,\ \lambda \in \nabla$, where ∇ denotes a discrete index set and where the collection of functions $\{\psi_\lambda,\ \lambda \in \nabla\}$ forms a basis for $H_0^1(\Omega)$. These functions play the role of test functions. Further, the solution u is searched for as a linear combination of a set of basis functions for $H_0^1(\Omega)$. These are termed the trial functions. We will be using the Galerkin method, according to which the set of test functions and the set of trial functions are identical. The problem of finding a potential u for which the equation (1.17) holds is then equivalent to the problem:

Find a sequence $d_\nu \in \ell_2$ such that:

$$\sum_{\nu\in\nabla} d_\nu \iint_{\Omega} \mathbf{grad}\ \psi_\nu \,.\, \mathbf{grad}\ \psi_\lambda\ dx_1\, dx_2 = \quad -\iint_{\Omega_{int}} 4\pi(\ \mathrm{div}\ \boldsymbol{m})\, \psi_\lambda\ dx_1\, dx_2 + \oint_{\partial\Omega_{int}^{+}} 4\pi(\boldsymbol{m} \,.\, \boldsymbol{n})\, \psi_\lambda\ ds, \quad \forall \lambda \in \nabla.$$

Wavelets are the functions that have been used in this thesis for test and trial functions. Wavelet functions and some of their relevant properties will be presented in Chapter 2.

Chapter 2

Wavelets

Wavelets are functions which are used in many types of applications, from image compression, signal analysis as in seismology, to integral and differential equations in numerical analysis. They are well known for their compression properties and for the existence of a fast wavelet transform which is comparable to a fast Fourier transform with local properties in both space and frequency. In the context of the numerical scheme presented in chapter 3, we will be particularly interested in the properties which lead to

- the sparse representation of certain functions,
- the good conditioning of the Laplace operator.

Many families of wavelets have been constructed in order to suit different applications. This chapter also contains a section dedicated to presenting properties which are common to most wavelets. The section following the general description of wavelet properties is more specialized and is concerned with the description of the wavelet bases that are used in the implementation. First of all, however, wavelets are presented in their historical context.

2.1 Historical background

In the early 1980's, wavelets which could present the double advantage of locality in space (or time) and in frequency, became popular with people working on signal processing as an alternative to the Fourier transform [11]. By analogy with the Fourier transform, a wavelet function w was first thought of as the analysing function in a transform of the form:

$$S(b,a) = \frac{1}{\sqrt{a}} \int_{-\infty}^{+\infty} \bar{w}(\frac{(x-b)}{a}) s(x) dx$$

where $a \in (0, \infty)$, $b \in \mathbb{R}$, s is the analysed function, $\bar{w}$ denotes the complex conjugate of w, and w satisfies the two equations:

$$\begin{aligned} \int w(x)dx &= 0 \\ \int |w(x)|^2 dx &< \infty \end{aligned}$$

In a search for a particular discretization of the variables a and b that would give an orthonormal basis for $L^2(\mathbb{R})$, the Haar system, due to Alfred Haar in 1910, appeared as an obvious solution. This system is a construction of boxes on dyadic intervals: $2^{j/2}w(2^j x - k), j, k \in \mathbb{Z}$, with $w(x) = 1$ if $x \in (0, 1/2)$, $w(x) = -1$ if $x \in (1/2, 1)$, and $w(x) = 0$ elsewhere. However, this system shows poor smoothness and cancellation properties, and in this respect is said to be of order 0 [23]. Stromberg, in the early 80's built the first wavelets of order 1 using spline functions [27].

A major step in the history of wavelets was then the elaboration of the concept of Multi-resolution Analysis of $L^2(\mathbb{R})$, or MRA($L^2(\mathbb{R})$), by Y. Meyer and S. Mallat in the late 80's. Later, using Mallat's work on MRA($L^2(\mathbb{R})$) bases, Daubechies constructed compactly

supported, orthonormal wavelets and scaling functions of predefined regularity [17], [18]. A set of compactly supported, biorthogonal wavelets based on splines was constructed by Cohen, Daubechies and Feauveau [10] (1992). Several reviews on the construction and properties of wavelets defined on $\mathbb{R}$ have been written [2, 28, 20]. As wavelet theory progressed and more tools became available, their use spread to areas other than signal processing.

The first applications of wavelets to the solution of partial differential equations seems to have consisted of Galerkin methods on problems with periodic boundary conditions. Indeed, as was noted by Y. Meyer [24], the techniques which were first developed on the real line could be easily modified by a standard procedure of periodisation to be used on $L^2([0,1])$ in the periodic case. For problems associated with finite boundary conditions however, the need arose in the mid-90's for bases of wavelets defined on the interval, and satisfying the boundary conditions. In particular, taking as a starting point the Cohen-Daubechies-Feauveau construction [10], W. Dahmen, A. Kunoth and K. Urban developed spline biorthogonal wavelet bases for $L_2([0,1])$. These bases have an arbitrarily high degree of polynomial exactness on the whole interval. The authors have also proven that discrete norms based on expansions in these bases characterize Sobolev spaces $H^s([0,1])$ for s in a certain range that depends on the regularity of the chosen biorthogonal bases [12]. A modification of these bases into multi-dimensional wavelets satisfying certain types of boundary conditions was due to W. Dahmen and R. Schneider and described in [15]. Other constructions have generalized the unit cube to domains of more complex geometries [5, 6, 16]. With the emergence of more specialized bases, a number of numerical methods have been adapted to take advantage of their good numerical properties such as the easy preconditioning of discrete operators, the multi-level organization of the bases and the

sparsity of certain operators [9, 8].

2.2 Introduction to wavelets

Wavelets were originally defined as functions of $L_2(\mathbb{R})$ and this will also be the starting point of this section. Further subsections will detail how wavelets with different constraints, such as periodic wavelets or wavelets on the interval, have had to depart from this original model. Finally, the last subsection is dedicated to multi-dimensional wavelets.

2.2.1 Multi-resolution analysis of $L_2(\mathbb{R})$ and definition of a wavelet

'A multi-resolution analysis M of $L^2(\mathbb{R})$ is defined by means of a sequence of closed subspaces V_j, with $j \in \mathbb{Z}$, that has the following properties:

1. $V_j \subset V_{j+1}$
2. $v(x) \in V_j \iff v(2x) \in V_{j+1}$
3. $v(x) \in V_0 \iff v(x+1) \in V_0$
4. $\bigcup_{j\in\mathbb{Z}} V_j$ is dense in $L^2(\mathbb{R})$ and $\bigcap_{j\in\mathbb{Z}} V_j = \{0\}$
5. A function $\phi \in V_0$ with a non vanishing integral exists such that $\{\phi(x-l) : l \in \mathbb{Z}\}$ is a Riesz basis of V_0, i.e. every element $f \in V_0$ can be written uniquely as $f = \sum_n c_n f_n$, and positive constants A and B exist such that $A\|f\|^2 \leq \sum_n |c_n|^2 \leq B\|f\|^2$.' [20]

In the above definition, the function ϕ is a scaling function, and the spaces V_j generated by its dilates and translates $\phi_{j,k} = 2^{j/2}\phi(2^j. - k)$ for $k \in \mathbb{Z}$ are called scaling function spaces.

The wavelet space W_j may now be defined as a complementing space of V_j in V_{j+1}:

$$V_{j+1} = V_j \oplus W_j$$

A function ψ is then a wavelet if $\{\psi(. - l) : l \in \mathbb{Z}\}$ is a Riesz basis of W_0. Then, $\{\psi_{j,k} = 2^{j/2}\psi(2^j. - k) : k \in \mathbb{Z}\}$ forms a basis of W_j and $\{\psi_{j,k} : j, k \in \mathbb{Z}\}$ forms a basis for $L^2(\mathbb{R})$.

It may be noted that the presence of the $2^{j/2}$ factor in the expressions of $\phi_{j,k}$ and $\psi_{j,k}$ ensures that $\|\phi_{j,k}\|_{L_2} = \|\phi\|_{L_2}$ and similarly for the norms of $\psi_{j,k}$ and ψ, where the L_2-norm is defined as $\|f\|^2_{L_2(\Omega)} = \int_\Omega |\, f(x)|^2\, dx$. The scaling function ϕ and the wavelet ψ are referred to as the mother scaling function and the mother wavelet. The mother scaling function is usually scaled as $\int_{\mathbb{R}} \phi(x)\, dx = 1$.

2.2.2 Orthogonal and biorthogonal settings

The original wavelet bases constructed were orthogonal in the sense that they would satisfy the following orthogonality conditions:

$$\begin{aligned}
\langle \phi_{j,k}, \phi_{j,k'} \rangle &= \delta_{kk'} \qquad \forall k, k' \in \mathbb{Z} \\
\langle \psi_{j,k}, \psi_{j,k'} \rangle &= \delta_{kk'} \qquad \forall k, k' \in \mathbb{Z} \\
\langle \psi_{j,k}, \phi_{j,k'} \rangle &= 0 \qquad \forall k, k' \in \mathbb{Z}
\end{aligned}$$

The notation $\langle v, w \rangle := \int_\Omega v(x)\, w(x)\, dx$ for the inner product in $L_2(\Omega)$ will be used time and again in the remainder of this thesis. In the present situation, $\Omega = \mathbb{R}$. The orthogonality

conditions impose strict restrictions on the construction of new bases. These restrictions are relaxed in the biorthogonal setting, where a dual multi-resolution analysis $\tilde{M}$ of $L^2(\mathbb{R})$, consisting of spaces $\tilde{V}_j$ with Riesz bases given by dual scaling functions $\tilde{\phi}_{j,k}$ is used. Spaces $\tilde{V}_j$ are complemented by dual wavelet spaces $\tilde{W}_j$:

$$\tilde{V}_{j+1} = \tilde{V}_j \oplus \tilde{W}_j$$

Dual scaling functions and dual wavelets are such that:

$$\begin{aligned}
\langle \phi_{j,k}, \tilde{\phi}_{j,k'} \rangle &= \delta_{kk'} \qquad \forall k, k' \in \mathbb{Z} \\
\langle \psi_{j,k}, \tilde{\psi}_{j,k'} \rangle &= \delta_{kk'} \qquad \forall k, k' \in \mathbb{Z} \\
\langle \phi_{j,k}, \tilde{\psi}_{j,k'} \rangle &= 0 \qquad \forall k, k' \in \mathbb{Z} \\
\langle \psi_{j,k}, \tilde{\phi}_{j,k'} \rangle &= 0 \qquad \forall k, k' \in \mathbb{Z}
\end{aligned}$$

Expressed in terms of spaces,

- in the orthogonal case:

$$\begin{aligned}
V_j &\perp W_j, \qquad \forall j \in \mathbb{Z} \\
W_j &\perp W_{j'}, \qquad \forall j, j' \in \mathbb{Z} \text{ such that } j \neq j'
\end{aligned}$$

- in the biorthogonal case:

$$\begin{aligned}
V_j &\perp \tilde{W}_j, \qquad \forall j \in \mathbb{Z} \\
W_j &\perp \tilde{V}_j, \qquad \forall j \in \mathbb{Z}
\end{aligned}$$

An example for the orthogonal case is the Haar wavelet:

$$\begin{aligned} \phi_{0,0}(x) &= \chi_{[-1/2,1/2]}(x), \\ \psi_{0,0}(x) &= \chi_{[-1/2,0]}(x) - \chi_{[0,1/2]}(x), \end{aligned}$$

where $\chi_{[a,b]}(x)$ is equal to 1 if $x \in [a,b]$, and 0 otherwise. In the following, everything will be denoted as in the biorthogonal case, since the orthogonal case is included as the particular case when scaling functions and wavelets are identical to their duals.

2.2.3 Refinement and wavelet equations

The projection of a function f onto a scaling function space V_J is given by the following equation, where $\langle .,. \rangle$ denotes the $L^2(\mathbb{R})$ inner product.

$$P_{V_J} f = \sum_{k \in \mathbb{Z}} \langle f, \tilde{\phi}_{J,k} \rangle \phi_{J,k}$$

Since the space V_J may be decomposed into wavelet spaces $V_0 \oplus W_0 \oplus W_1 \oplus \cdots \oplus W_{J-1}$, the same projection may be written as :

$$P_{V_J} f = \sum_{k \in \mathbb{Z}} \langle f, \tilde{\phi}_{0,k} \rangle \phi_{0,k} + \sum_{j=0}^{J-1} \sum_{k \in \mathbb{Z}} \langle f, \tilde{\psi}_{j,k} \rangle \psi_{j,k}$$

As $V_0 \subset V_1$, the mother scaling function ϕ may be expressed exactly in the basis of V_1. Hence there exists a sequence of parameters $\{h(k) : k \in \mathbb{Z}\}$ such that the following

refinement equation or *dilation equation* holds:

$$\phi(x) = \sqrt{2}\sum_{k\in\mathbb{Z}} h(k)\phi(2x-k),$$

where $h(k) = \langle \phi, \tilde{\phi}(2.-k)\rangle = \langle \phi_{0,0}, \tilde{\phi}_{1,k}\rangle$.

As $W_0 \subset V_1$, ψ may be expressed exactly in the basis of V_1. Therefore there exists a sequence of parameters $\{g(k) : k \in \mathbb{Z}\}$ such that the *wavelet equation* holds:

$$\psi(x) = \sqrt{2}\sum_{k\in\mathbb{Z}} g(k)\phi(2x-k),$$

where $g(k) = \langle \psi, \tilde{\phi}(2.-k)\rangle = \langle \psi_{0,0}, \tilde{\phi}_{1,k}\rangle$. These relations between the scale $j = 0$ and the scale $j = 1$ are true between any scale j and $j+1$:

$$\begin{aligned}
\phi_{j,0}(x) &= \sum_{k\in\mathbb{Z}} h(k)\phi_{j+1,k}(x) \\
\phi_{j,l}(x) &= \sum_{k\in\mathbb{Z}} h(k)\phi_{j+1,2l+k}(x) \\
\psi_{j,0}(x) &= \sum_{k\in\mathbb{Z}} g(k)\phi_{j+1,k}(x) \\
\psi_{j,l}(x) &= \sum_{k\in\mathbb{Z}} g(k)\phi_{j+1,2l+k}(x)
\end{aligned}$$

Similarly, a dual refinement equation and a dual wavelet equation may be derived:

$$\begin{aligned}
\tilde{\phi}_{j,l}(x) &= \sum_{k\in\mathbb{Z}} \tilde{h}(k)\tilde{\phi}_{j+1,2l+k}(x) \\
\tilde{\psi}_{j,l}(x) &= \sum_{k\in\mathbb{Z}} \tilde{g}(k)\tilde{\phi}_{j+1,2l+k}(x)
\end{aligned}$$

Finally, by projecting $\phi_{j+1,0}$ onto V_j and W_j and $\tilde{\phi}_{j+1,0}$ onto $\tilde{V}_j$ and $\tilde{W}_j$, and using the equations above and the biorthogonality relation between $\phi_{j+1,0}$ and $\tilde{\phi}_{j+1,l}$ for $l \in \mathbb{Z}$ and

between $\tilde{\phi}_{j+1,0}$ and $\phi_{j+1,l}$ for $l \in \mathbb{Z}$, we deduce the reconstruction equations:

$$\begin{aligned} \phi_{j+1,0} &= \sum_{k\in\mathbb{Z}} \tilde{h}(-2k)\phi_{j,k} + \sum_{k\in\mathbb{Z}} \tilde{g}(-2k)\psi_{j,k} \\ \tilde{\phi}_{j+1,0} &= \sum_{k\in\mathbb{Z}} h(-2k)\tilde{\phi}_{j,k} + \sum_{k\in\mathbb{Z}} g(-2k)\tilde{\psi}_{j,k} \end{aligned}$$

The sequence $(h(n))$ is often called the scaling function filter or mask and denoted by h, while the sequence $g = (g(n))$ is the wavelet filter or mask. The dual masks are $\tilde{h} = (\tilde{h}(n))$ and $\tilde{g} = (\tilde{g}(n))$, associated with the dual basis. In the implementation, finitely supported scaling functions and wavelets are used. This implies that the corresponding masks have a finite number of non-zero entries.

The fast wavelet transform

The passage from a one-level, scaling function expression

$$\sum_{k\in\mathbb{Z}} s_{j+1,k}\, \phi_{j+1,k}$$

to an equivalent, multi-level, wavelet expression of the form

$$\sum_{k\in\mathbb{Z}} s_{j_0,k}\, \phi_{j_0,k} + \sum_{j'=j_0}^{j} \sum_{k\in\mathbb{Z}} d_{j',k}\, \psi_{j',k}$$

is called the wavelet transform of the sequence $(s_{j+1,k})$. The wavelet expression is found by the recursive application of the refinement and wavelet equations. Assuming that the scaling function expression is the scaling function expansion of a function $f \in L_2$, so that

the coefficients $s_{j,k}$ are defined by

$$s_{j,k} = \int_{-\infty}^{+\infty} f(x)\tilde{\phi}_{j,k}(x)dx,$$

applying the refinement equation to the dual scaling function $\tilde{\phi}_{j,k}$ immediately gives an expression of $s_{j,k}$ in terms of the scaling function coefficients on the higher level:

$$\begin{aligned} s_{j,k} &= 2^{(j+1)/2} \sum_n \tilde{h}(n) \int_{-\infty}^{+\infty} \tilde{\phi}_{0,0}(2^{j+1}x - 2k - n) f(x)dx \\ s_{j,k} &= \sum_n \tilde{h}(n) s_{j+1,2k+n}. \end{aligned}$$

Similarly, by the wavelet equation, we find:

$$d_{j,k} = \sum_n \tilde{g}(n) s_{j+1,2k+n}.$$

This linear dependence of the scaling function and wavelet coefficients on level j on the scaling function coefficients on level $j+1$ can be clearly expressed in matrix form as

$$s_j = \tilde{H}_j \; s_{j+1} \qquad \text{and} \qquad d_j = \tilde{G}_j \; s_{j+1},$$

where s_j, and similarly d_j and s_{j+1}, denotes the potentially infinite vector $(s_{j,k}, k \in \mathbb{Z})$ and $\tilde{H}_j$, similarly $\tilde{G}_j$, denotes a circulant matrix constructed from the entries of the mask $\tilde{h}$, repeated with a double shift to the right over consecutive rows. The composition of the $\tilde{H}_j$ and $\tilde{G}_j$ matrices is illustrated in Figure 2.1. In all practical cases, the vector s_{j+1} has a finite number of non-zero entries or it is periodic, and the masks $\tilde{h}$ and $\tilde{g}$ have compact supports which are short compared with the length of s_{j+1}. Then, the double shift in the

transformation matrices implies that the vectors s_j and d_j have shorter lengths than s_{j+1}. In fact, if s_{j+1} has length N, $\tilde{h}$ has length N_h and $\tilde{g}$ has length N_g, then s_j has length $(N + N_h)/2$ and d_j has length $(N + N_g)/2$. This means that the transformation from s_j to s_{j-1} and d_{j-1} requires roughly half as many operations as the transformation from s_{j+1} to s_j and d_j. In the periodic case, the length of s_j and d_j is exactly half that of s_{j+1}. By referring to Figure 2.1, it is easy to see that during the transformation process from level $j+1$ to level j, each coefficient $s_{j+1,}$ is multiplied by at most $\big((N_h + N_g)/2 + 2\big)$ matrix coefficients from $\tilde{H}_j$ and $\tilde{G}_j$. So $N\big((N_h + N_g)/2 + 2\big)$ multiplications take place, at most. In the periodic case, the number of entries in s_j is $N/2$ and so the transformation from level j to level $j-1$ requires $(1/2)N\big((N_h + N_g)/2 + 2\big)$. Hence the transformation from the level $j+1$ to the level j_0, requires

$$N\Big(1 + \frac{1}{2} + \cdots + \frac{1}{2^{j-j_0}}\Big)\big((N_h + N_g)/2 + 2\big)$$

multiplications. This number is linear in N. This makes the wavelet transform fast even in comparison with the fast Fourier transform which has $\mathcal{O}(N\log N)$ complexity. It is worth noting here that shorter refinement and wavelet masks make for a faster wavelet transform.

The iterative process of wavelet transformation from the level j to the level $j-2$ is represented on Figure 2.2. The matrices $\tilde{H}_{j-1}$ and $\tilde{G}_{j-1}$ have the same structure as $\tilde{H}_j$ and $\tilde{G}_j$, but they have roughly half the number of rows and columns as these matrices. The inverse transformation process is the reverse of the wavelet transform but it involves the primal masks h and g instead of the dual ones. The inverse transformation matrix from the level j to the level $j+1$ is represented in Figure 2.3.

$$\begin{bmatrix} \dots & . & . & . & . & . & \dots \\ \dots & \tilde{h}(0) & \tilde{h}(1) & \tilde{h}(2) & \tilde{h}(3) & \tilde{h}(4) & \dots \\ \dots & . & . & \tilde{h}(0) & \tilde{h}(1) & \tilde{h}(2) & \dots \\ \dots & . & . & . & . & \tilde{h}(0) & \dots \\ \dots & . & . & . & . & . & \dots \\ \dots & . & . & . & . & . & \dots \\ \dots & \tilde{g}(0) & \tilde{g}(1) & \tilde{g}(2) & \tilde{g}(3) & \tilde{g}(4) & \dots \\ \dots & . & . & \tilde{g}(0) & \tilde{g}(1) & \tilde{g}(2) & \dots \\ \dots & . & . & . & . & \tilde{g}(0) & \dots \\ \dots & . & . & . & . & . & \dots \end{bmatrix} \begin{bmatrix} \vdots \\ s_{j+1,0} \\ s_{j+1,1} \\ \vdots \\ \\ \vdots \\ s_{j+1,k} \\ s_{j+1,k+1} \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ s_{j,0} \\ s_{j,1} \\ \vdots \\ \dots \\ \vdots \\ d_{j,0} \\ d_{j,1} \\ \vdots \end{bmatrix}$$

Figure 2.1: *Transformation matrix from* V_{j+1} to $V_j + W_j$.

$$\begin{bmatrix} \\ \tilde{H}_j \\ \\ \dots\dots\dots\dots\dots\dots \\ \\ \tilde{G}_j \\ \\ \end{bmatrix} \begin{bmatrix} \\ \\ \\ s_{j+1} \\ \\ \\ \\ \end{bmatrix} = \begin{bmatrix} \\ s_j \\ \\ .. \\ \\ d_j \\ \\ \end{bmatrix} \rightarrow \begin{bmatrix} \tilde{H}_{j-1} \\ \dots\dots\dots\dots \\ \tilde{G}_{j-1} \end{bmatrix} \begin{bmatrix} \\ s_j \\ \\ \\ \\ \end{bmatrix} = \begin{bmatrix} s_{j-1} \\ \dots\dots \\ d_{j-1} \\ \dots\dots \\ \\ d_j \\ \end{bmatrix} \rightarrow \begin{bmatrix} \tilde{H}_{j-2} \\ \dots\dots \\ \tilde{G}_{j-2} \end{bmatrix} \begin{bmatrix} s_{j-1} \\ \\ \\ \\ \\ \end{bmatrix} \begin{bmatrix} \\ \\ d_{j-1} \\ \dots\dots \\ \\ d_j \\ \end{bmatrix} = \begin{bmatrix} s_{j-2} \\ \dots\dots \\ d_{j-2} \\ \dots\dots \\ d_{j-1} \\ \dots\dots \\ \\ d_j \\ \end{bmatrix}$$

$$\begin{array}{ccccccc} s_{j+1} & \longrightarrow & s_j & \longrightarrow & s_{j-1} & \longrightarrow & s_{j-2} \\ & \searrow & & \searrow & & \searrow & \\ & & d_j & & d_{j-1} & & d_{j-2} \end{array}$$

Figure 2.2: *Fast wavelet transform*

$$\left[\begin{array}{cccc|cccc} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ . & h(0) & . & . & . & g(0) & . & . \\ . & h(1) & . & . & . & g(1) & . & . \\ . & h(2) & h(0) & . & . & g(2) & g(0) & . \\ . & h(3) & h(1) & . & . & g(3) & g(1) & . \\ . & h(4) & h(2) & . & . & g(4) & g(2) & . \\ . & . & h(3) & . & . & . & g(3) & . \\ . & . & h(4) & . & . & . & g(4) & . \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{array}\right] \left[\begin{array}{c} \vdots \\ s_{j,0} \\ s_{j,1} \\ \vdots \\ \hline \vdots \\ d_{j,0} \\ d_{j,1} \\ \vdots \end{array}\right] = \left[\begin{array}{c} \vdots \\ s_{j+1,0} \\ s_{j+1,1} \\ \vdots \\ \vdots \\ s_{j+1,k-1} \\ s_{j+1,k} \\ \vdots \end{array}\right]$$

Figure 2.3: *Inverse transformation matrix from $V_j + W_j$ to V_{j+1}.*

2.2.4 Vanishing moments

An important property of a wavelet basis is the number of vanishing moments that it has. A wavelet has N vanishing moments if

$$\int_{-\infty}^{+\infty} x^i \psi(x) dx = 0 \qquad \text{for } i = 0, \ldots, N-1,$$

but not higher. Because of the biorthogonality equations, this means that the approximation order of the dual multi-resolution analysis is N, in the sense that the polynomials up to order N can be expressed exactly in terms of dual scaling functions,i.e.

$$x^i = \sum_{k \in \mathbb{Z}} \alpha_{j,k} \tilde{\phi}_k(x) \qquad i = 0, \ldots, N-1.$$

The compression properties of a wavelet also depends on its number of vanishing moments, and it is true to say that a wavelet with a higher number of vanishing moments and a shorter support has better compression properties. The compression aspect which is of interest for the present application concerns the representation of functions. Assuming that a function $f \in L_2$ is locally C^{n+1} on the support $\Omega_{j,k} = [\alpha, \beta]$ of a wavelet ψ_{jk}, the

Taylor expansion of f up to order n can be written as

$$f(x) = \sum_{k=0}^{n} \frac{1}{k!} f^{(k)}(\alpha)\,(x-\alpha)^k + \int_{\alpha}^{x} \frac{1}{n!}\,(x-t)^n\, f^{(n+1)}(t)\,dt,$$

where the remainder is an integral term (see for instance [19]). Let P_n denote the polynomial part of this expansion. If $\psi_{j,k}$ has $N > n$ vanishing moments, then the corresponding wavelet coefficient of f can be expressed as follows

$$\begin{aligned} |\,d_{j,k}| &= \Big| \int_{\Omega_{j,k}} f(x)\,\psi_{j,k}(x)\,dx \Big|, \\ &= \Big| \int_{\Omega_{j,k}} (f(x) - P_n(x))\,\psi_{j,k}(x)\,dx \Big|, \\ &= \Big| \int_{\Omega_{j,k}} \psi_{j,k}(x) \int_{\alpha}^{x} \frac{1}{n!}\,(x-t)^n\, f^{(n+1)}(t)\;dt\;dx \Big|. \end{aligned}$$

By the Cauchy-Schwartz inequality, the wavelet coefficient $d_{j,k}$ is bounded as follows

$$|\,d_{j,k}| \le \frac{1}{n!} \|\psi_{j,k}\|_{L_2(\Omega_{j,k})} \Big\| \int_{\alpha}^{x} (x-t)^n\, f^{(n+1)}(t)\;dt \Big\|_{L_2(\Omega_{j,k})}.$$

In the product above, the wavelet normalization means that the first factor is independent of j and k

$$\|\psi_{j,k}\|_{L_2(\Omega_{j,k})} = \|\psi\|_{L_2(\Omega_{0,0})}$$

and the second factor can be bounded as follows:

$$\begin{aligned} \Big| \int_{\alpha}^{x} (x-t)^n\, f^{(n+1)}(t)\;dt \Big| &\le \|(x-t)^n\|_{L^2([\alpha,x])}\, \|f^{(n+1)}\|_{L^2([\alpha,x])}, \\ &\le \|(x-t)^n\|_{L^2([\alpha,x])}\, \|f^{(n+1)}\|_{L^2(\Omega_{j,k})}, \end{aligned}$$

since $(\alpha, x) \subset \Omega_{j,k}$. Let L denote the length of the support of the mother wavelet ψ. The support of $\psi_{j,k}$ therefore has length $\beta - \alpha = 2^{-j}L$.

$$\begin{aligned} \|(x-t)^n\|^2_{L^2([\alpha,x])} &= \int_\alpha^x (x-t)^{2n}\, dt \le \int_\alpha^x (2^{-j}L)^{2n}\, dt, \qquad \alpha < x \le \beta \\ &\le \int_\alpha^\beta (2^{-j}L)^{2n}\, dt \\ &= 2^{-j}L(2^{-j}L)^{2n} = (2^{-j(n+\frac{1}{2})}L^{n+\frac{1}{2}})^2 \end{aligned}$$

Hence

$$\begin{aligned} \left\| \int_\alpha^x (x-t)^n\, f^{(n+1)}(t)\, dt \right\|_{L_2(\Omega_{j,k})} &\le \left\| 2^{-j(n+\frac{1}{2})}L^{n+\frac{1}{2}}\, \|f^{(n+1)}\|_{L_2(\Omega_{j,k})} \right\|_{L^2(\Omega_{j,k})} \\ &= (\beta-\alpha)^{\frac{1}{2}} 2^{-j(n+\frac{1}{2})} L^{n+\frac{1}{2}} \|f^{(n+1)}\|_{L_2(\Omega_{j,k})}, \\ &= 2^{-j(n+1)} L^{n+1}\, \|f^{(n+1)}\|_{L_2(\Omega_{j,k})}, \end{aligned}$$

and so

$$|\, d_{j,k}| \le \frac{1}{n!}\, L^{n+1}\, 2^{-j(n+1)}\, \|\psi\|_{L_2(\Omega_{0,0})}\, \|f^{(n+1)}\|_{L_2(\Omega_{j,k})}.$$

This shows how the size of the wavelet coefficients of a function is related to its smoothness on the support of the wavelet. The rate of decay of $|\, d_{j,k}|$, as the level j increases is $\mathcal{O}(2^{-jn}\|f^{(n)}\|_{L_2(\Omega_{j,k})})$ for $n \le N$. The wavelet expansion of f can be compressed by discarding the smaller coefficients $d_{j,k}$ and so the compression rate is better for larger values of N.

2.2.5 Construction of multivariate wavelets by tensor product

The simplest construction for multivariate wavelets consists of taking the tensor products of univariate scaling functions and wavelets. In order to use the notation of ϕ and ψ for the scaling functions and wavelets in $\mathbb{R}^n$, $n > 1$, the notation for the univariate scaling functions and wavelets, which are now considered as simple building blocks, is changed to $\xi_{j,k}$ for a scaling function and $\eta_{j,k}$ for a wavelet.

The mother scaling function in $\mathbb{R}^n$ is defined as

$$\phi(x_1, \ldots, x_n) = \prod_{i=1}^{n} \xi(x_i).$$

The translates of ϕ are denoted with a position vector $\boldsymbol{k} = (k_1, \ldots, k_n)$ as

$$\phi_{\boldsymbol{k}}(x_1, \ldots, x_n) = \prod_{i=1}^{n} \xi(x_i - k_i).$$

The dilates of ϕ are defined with the tensor product of univariate scaling functions on a single level j as

$$\phi_{j,\boldsymbol{k}} = \prod_{i=1}^{n} \xi_{j,k_i}.$$

The wavelets in $\mathbb{R}^n$ are defined by the tensor product of univariate wavelets and scaling functions. There are $2^n - 1$ different types of wavelets. The type of a multivariate wavelet is determined by a vector $\boldsymbol{e} = (e_1, \ldots, e_n)$ that records as $e_i = 0$ the use of a scaling function for the i^{th} component of the tensor product and as $e_i = 1$ the use of a wavelet.

For instance, we have

$$\psi_{\boldsymbol{e}=(1,0,0)}(x_1, x_2, x_3) = \eta(x_1)\,\xi(x_2)\,\xi(x_3).$$

The type $\boldsymbol{e} = (0, \ldots, 0)$ corresponds to a scaling function rather than a wavelet. The dilates and translates of these $2^n - 1$ mother wavelets are defined exactly as would be expected. For instance,

$$\psi_{(j,\boldsymbol{e}=(1,1,0),\boldsymbol{k})}(x_1, x_2, x_3) = \eta_{j,k_1}(x_1)\,\eta_{j,k_2}(x_2)\,\xi_{j,k_3}(x_3).$$

It is convenient to condense the notation for a wavelet index as $\lambda = (j, \boldsymbol{e}, \boldsymbol{k})$. The notation $|\lambda| = j$ is also usual. It is also sometimes useful to be able to denote in a general way the univariate components of a wavelet without having to specify whether they are scaling functions or wavelets. In this thesis, a univariate basis function is sometimes denoted by $\zeta_{j,e,k}$, where $\xi_{j,k}$ is meant if $e = 0$ and $\eta_{j,k}$ is meant instead if $e = 1$. A general notation for a wavelet is therefore

$$\psi_\lambda(x) = \prod_{i=1}^{n} \zeta_{j,e_i,k_i}(x_i).$$

The multi-variate refinement and wavelet equations are naturally deduced from their univariate counterparts.

2.3 Wavelets used in the implementation

The present application requires wavelet bases defined on a bounded domain rather than on $\mathbb{R}^n$. The bases also have to satisfy homogeneous boundary conditions. The bases satisfying these constraints cannot retain all the properties that the wavelet bases defined on the real line have. For instance, the self-similarity of all wavelets with the mother wavelet may not

hold near the boundary. However, the passage from univariate to multi-variate wavelets by tensor product does not change and so this section will only present the univariate bases used as the building blocks for the multi-variate wavelets implemented. The motivation for using these particular wavelet bases preferentially to other bases comes first from the fact that these bases have been developed for applications of the same type as the present one and as a consequence they are proven to possess many relevant properties, as will be seen in the next section. The second great motivation is the existence of software tools for the handling of these bases in applications such as elliptic partial differential equations and integral equations. These tools are gathered under the name of *Multilevel Library* and have been developed at the IGPM[1] at the RWTH Aachen[2]. The Multilevel Library has been written in the C++ programming language and although it is under continual development, it has kindly been made available on request [22].

2.3.1 Wavelet bases defined on the unit interval

W. Dahmen, A. Kunoth and K. Urban describe in [12] the construction of a family of compactly supported, biorthogonal wavelet bases for $L_2([0,1])$. Each biorthogonal pair in the family is characterized by the approximation orders of the associated primal and dual multi-resolution analyses. The order of the primal MRA is denoted by d and the order of the dual MRA is denoted by $\tilde{d}$. The family contains primal bases with MRA of any order $d \geq 1$ and the construction imposes the condition that that the sum $d + \tilde{d}$ is even.

Their construction is based on the Cohen-Daubechies-Feauveau biorthogonal spline wavelet bases for $L(\mathbb{R})$ [10].

[1] Institut für Geometrie and Praktische Mathematik
[2] The University of Aachen, Germany

The scaling function bases on $[0,1]$

First of all, a biorthogonal pair of scaling function bases from [10] is restricted to the unit interval. A minimum level j_0 is fixed for which the length of the supports of ϕ_{j_0} and $\tilde{\phi}_{j_0}$ is less than 1 and a sufficient number of functions $\phi_{j_0,k}$ and $\tilde{\phi}_{j_0,k}$ do not intersect the boundary. The scaling functions whose support intersects the boundary are carefully re-defined as a fixed linear combination of scaling functions on the same level and truncated at the boundary. This is done for the primal and the dual scaling functions independently, but the cardinalities of the new collections of primal scaling functions and dual scaling functions must be equal. The aim of the linear combination is to ensure that the approximation order of the new collection of scaling functions remains on the whole interval the same as that of the original basis from [10] and it is done in such a way that, after biorthogonalization, the two modified collections of scaling functions form a multi-resolution analysis of $L_2([0,1])$.

The new bases consist of a finite number of scaling functions. The set of indices k such that $\phi_{j,k}$ is in the basis on level j is denoted by Δ_j and similarly $\tilde{\Delta}_j$ denotes the set of indices k associated with a dual basis function on level j. This is only a convention since $\Delta_j = \tilde{\Delta}_j$.

Because the scaling functions near the boundary are no longer obtained by a simple dilation and translation from the mother scaling function, the refinement equation is replaced

by a relation of the following form :

$$\phi_{j,k} = \sum_l h_{j,k,l}\phi_{j+1,l}.$$

The coefficients $h_{j,k,l}$ still retain a certain structure, which is easier to picture in matrix form. In [12], the transpose of the refinement matrix H_j involved in the wavelet transform is denoted by $M_{j,0}$, and this convention is followed here in order to make references to [12] more consistent. The matrix $M_{j,0}$ can be decomposed into three parts. One left upper block, denoted by M_L, corresponds to the refinement equations of the left boundary adapted scaling functions. The block M_L does not depend in any way on the level j. One right bottom block, denoted by M_R, is the counterpart of M^L for right boundary adapted functions. It is also a fixed block, independent of j. Finally, the remaining part of $M_{j,0}$ is denoted by A_j and it corresponds to the refinement relations for the scaling functions which have not been boundary adapted. Accordingly, this part of $M_{j,0}$ is very similar to the transpose of the refinement matrix H_j for scaling functions defined on the real line: it is composed of one single mask, $a/\sqrt{2} = (a(l)/\sqrt{2}, l = l_1, \ldots, l_2)$, which is repeated column after column with a double shift downwards. The number of rows and columns of A_j increases with j, but the mask that composes A_j is always the same. Figure 2.4 represents the block decomposition of the refinement matrix $M_{j,0}$. The structure of $M_{j,0}$ means that it need not be stored in matrix form, and that storing M_L, M_R and the mask a is sufficient. The refinement matrix for the dual basis has the same structure, and the notations for it are $\tilde{M}_{j,0}$, $\tilde{M}_L$, $\tilde{M}_R$ and $\tilde{a} = (\tilde{a}(l), l = \tilde{l}_1, \ldots, \tilde{l}_2)$.

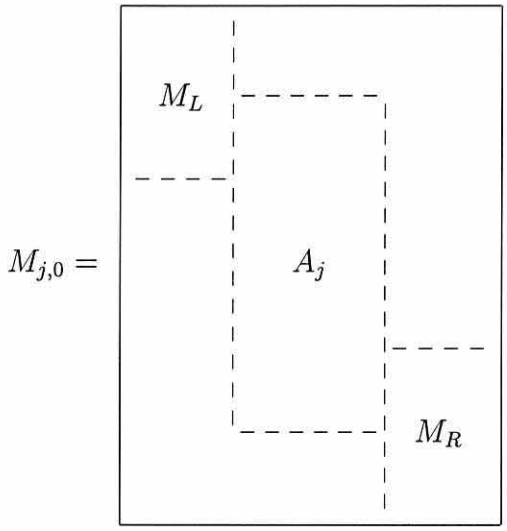


Figure 2.4: *Block structure of the refinement matrix* $M_{j,0}$.

The wavelet bases on $[0, 1]$

After a pair of MRA's with biorthogonal bases has been built for $L_2([0,1])$, wavelet biorthogonal bases are built for the wavelet spaces W_j complementing the scaling function spaces V_j into the higher level spaces V_{j+1}, and similarly for the dual spaces.

$$V_j \oplus W_j = V_{j+1} \qquad \text{and} \qquad \tilde{V}_j \oplus \tilde{W}_j = \tilde{V}_{j+1}$$

The details of the construction are given in [12]. The wavelet bases, like the scaling function bases, comprise of course of a finite number of functions. The notation ∇_j is used to represent the set of indices k such that $\psi_{j,k}$ is in the primal basis, and $\tilde{\nabla}_j$ represents the set of indices for the dual basis. The primal wavelet matrix involved in the wavelet transform is denoted by $M_{j,1}$ and it corresponds to the transpose of the matrix G_j described in section 2.2.3. The matrix $M_{j,1}$ has the same structure as $M_{j,0}$. The dual wavelet matrix is denoted by $\tilde{M}_{j,1}$ and it is similar to $M_{j,1}$ in all aspects. The structure of these matrices and the decomposition of the bases into boundary-adapted and non-boundary-adapted functions plays a very important role in the implementation of numerical methods.

Norm equivalences

First of all, let us introduce the following notations in order to express the norm equivalences below in the clear and concise form used in [12].

When j_0 denotes the coarsest level of a wavelet basis, ∇_{j_0-1} is used to denote Δ_{j_0}, the index set of the scaling functions on the coarsest level, and these scaling functions are denoted by $\psi_{j_0-1,k}$. This change of notation simply makes for shorter written expressions by including the scaling functions in the wavelet notation. Finally, the equivalence notation $a \sim b$ means that there exist two positive constants c_1 and c_2, independent of a and b, and such that $c_1 a \leq b \leq c_2 a$.

In addition to the norm equivalence between wavelet sequences in ℓ_2 and functions in L_2,

$$\|v\|_{L_2([0,1])} \sim \Big(\sum_{j=j_0-1}^{\infty} \sum_{k \in \nabla_j} |\langle v, \tilde{\psi}_{j,k} \rangle|^2 \Big)^{1/2},$$

which is equivalent to the fact that wavelet bases are Riesz bases, the authors of [26][12] have proven for any of the constructed pair of bases that there exists an equivalence between the weighted wavelet sequences in ℓ_2 and functions in $H^s([0,1])$,

$$\|v\|_{H^s([0,1])} \sim \Big(\sum_{j=j_0-1}^{\infty} \sum_{k \in \nabla_j} 2^{2sj} |\langle v, \tilde{\psi}_{j,k} \rangle|^2 \Big)^{1/2}, \qquad s \in (-\tilde{\gamma}, \gamma),$$

where γ and $\tilde{\gamma}$ are defined by

$$\gamma = \sup\{s \in \mathbb{R} : \phi \in H^s([0,1])\} \text{ and } \tilde{\gamma} = \sup\{s \in \mathbb{R} : \tilde{\phi} \in H^s([0,1])\}.$$

This property, when $s = 1$, will lead to a good, diagonal preconditioning of the Laplace operator matrix in the Poisson equation and will in this way play a role in the acceleration of the convergence of the iterative scheme that will be used to solve the equation. In the present context, the primal wavelets have to satisfy homogeneous boundary conditions and in the construction described in [15] this norm equivalence holds in $H_0^s([0,1])$ for the primal wavelets.

Chapter 3

An adaptive Poisson solver

It has already been seen how the simulation of the evolution of the magnetization $\boldsymbol{m}$ inside the nano-element has been split into two separate tasks. The first task was the calculation of the magnetization $\boldsymbol{m}_n$ at an instant τ_n, given the value of $\boldsymbol{m}$ and of its derivative with respect to τ at a set of discrete times $\tau_{n'}, n' < n$. This task is done in the physical space by a time-stepping scheme such as the Euler scheme and has been described in Chapter 1. The second task consists of the calculation at each time-step of the components of the derivative $d\boldsymbol{m}/d\tau$ according to the Landau-Lifshitz equation. The equations (1.14) and (1.11)-(1.13) show that all the components of $d\boldsymbol{m}/d\tau$ can be obtained directly from $\boldsymbol{m}$, apart from the demagnetizing field $\boldsymbol{h}_d$, which depends on the solution u of the Poisson equation (1.17). Clearly, solving this equation at every time-step is the most effort-consuming part of the whole problem.

This chapter presents a scheme for the calculation of a numerical solution to the Poisson equation. The variational formulation of the equation given in Section 1.2.2 is discretized with wavelets and expressed as a matrix equation of the type $Au = b$, where A is a sparse,

symmetric, positive definite matrix. For this reason, the matrix equation is solved by an iterative method such as the method of steepest descent or the conjugate gradient method. The properties of wavelets will have two main roles in this scheme. First, the condition number of the A matrix will be improved by a diagonal wavelet preconditioner, with the effect of accelerating the convergence of the iterative scheme. Secondly, the scheme will take advantage of the sparse wavelet representation of functions in order to express the right hand side of the equation and the solution as accurately as possible, given the number of degrees of freedom that can be afforded in practice.

3.1 Discretization of the problem

In Section 1.2.2, the Poisson equation for u was transformed into a variational problem of the form

Find $u \in H_0^1(\Omega)$ such that:

$$\int_\Omega \nabla v \cdot \nabla u = \int_\Omega v f, \quad \text{for all } v \in H_0^1(\Omega). \tag{3.1}$$

Let $\Psi = \{\psi_\lambda, \lambda \in \nabla\}$ be a wavelet basis for $H_0^1(\Omega)$, as described in Chapter 2, such that the following norm equivalence holds,

$$\|v\|_{H^1(\Omega)} \sim \Big(\sum_{\lambda \in \nabla} 2^{2|\lambda|} |\langle v, \tilde{\psi}_\lambda \rangle|^2 \Big)^{1/2}, \tag{3.2}$$

and let u be expressed as

$$u = \sum_{\lambda \in \nabla} \langle u, \tilde{\psi}_\lambda \rangle \psi_\lambda = \sum_{\lambda \in \nabla} d_\lambda \psi_\lambda.$$

The variational problem above may be discretized in terms of wavelets as

Find $(d_\lambda)_{\lambda\in\nabla} \in \ell_2$ such that:

$$\sum_{\lambda\in\nabla} d_\lambda \int_\Omega \nabla\psi_\lambda \cdot \nabla\psi_\nu = \int_\Omega f\psi_\nu, \quad \text{for all } \nu \in \nabla \tag{3.3}$$

and this is recast as a matrix equation $\boldsymbol{Cd} = \boldsymbol{b}$, where $\boldsymbol{C}$ denotes the infinite matrix with entries $c_{\lambda,\nu} = \int_\Omega \nabla\psi_\lambda \cdot \nabla\psi_\nu$, $\boldsymbol{d}$ denotes the vector $(d_\lambda)_{\lambda\in\nabla}$ and $\boldsymbol{b}$ denotes the vector with entries $b_\lambda = \int_\Omega f\psi_\lambda$. Taking advantage of the norm equivalence (3.2), the matrix $\boldsymbol{C}$ may be preconditioned by a diagonal matrix $\boldsymbol{D}$ with entries $d_{\lambda,\lambda} = 2^{-|\lambda|}$ as follows

$$\boldsymbol{DCD}\,\boldsymbol{D}^{-1}\boldsymbol{d} = \boldsymbol{Db} \qquad \text{or} \qquad \boldsymbol{Au} = \boldsymbol{f},$$

with $\boldsymbol{A} = \boldsymbol{DCD}$, $\boldsymbol{u} = \boldsymbol{D}^{-1}\boldsymbol{d}$ and $\boldsymbol{f} = \boldsymbol{Db}$.

Let $a(.,.)$ denote the continuous, bilinear form defined as

$$a(u,v) = \int_\Omega \nabla u \,.\, \nabla v, \quad \forall\, u, v \in H_0^1(\Omega).$$

Because of the norm equivalence (3.2), there exists $c_2 > 0$ such that $\|u\|_{H^1} \leq \sqrt{c_2}\|\boldsymbol{u}\|_{\ell_2}$, for all $u \in H_0^1(\Omega)$. Continuity of $a(.,.)$ implies that $|a(u,v)| \leq c_2\|\boldsymbol{u}\|_{\ell_2}\|\boldsymbol{v}\|_{\ell_2}$, for all $u, v \in H_0^1(\Omega)$. Since $a(u,v) = \boldsymbol{u}^T\boldsymbol{Av}$, the ℓ_2-norm of $\boldsymbol{A}$ is bounded from above

$$\|\boldsymbol{Au}\|_{\ell_2} = \sup_{\boldsymbol{v}\in\ell_2} \frac{\boldsymbol{v}^T\boldsymbol{Au}}{\|\boldsymbol{v}\|_{\ell_2}} \leq c_2\|\boldsymbol{u}\|_{\ell_2}.$$

Also, because $a(u,v)$ is coercive for $u \in H_0^1(\Omega)$ and the norm equivalence (3.2) holds, it follows that there exists $c_1 > 0$ such that $c_1\|\boldsymbol{u}\|_{\ell_2}^2 \leq \boldsymbol{u}^T\boldsymbol{Au}$, for all $u \in H_0^1(\Omega)$. Since

$\boldsymbol{u}^T\boldsymbol{A}\boldsymbol{u} \leq \|\boldsymbol{A}\boldsymbol{u}\|_{\ell_2}\|\boldsymbol{u}\|_{\ell_2}$, the ℓ_2-norm of $\boldsymbol{A}$ is bounded from below

$$c_1\|\boldsymbol{u}\|_{\ell_2} \leq \|\boldsymbol{A}\boldsymbol{u}\|_{\ell_2}.$$

The inequality

$$c_1\|\boldsymbol{u}\|_{\ell_2} \;\leq\; \|\boldsymbol{A}\boldsymbol{u}\|_{\ell_2} \;\leq\; c_2\|\boldsymbol{u}\|_{\ell_2}$$

has two interesting consequences for the numerical scheme. The first one is that the problem (3.3) is well-posed, in the sense that the equation $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}$ has a unique solution in ℓ_2 and that this solution depends continuously on the right hand side $\boldsymbol{f}$. The second consequence is that the condition number of any sub-section $\boldsymbol{A}_\Lambda$ of $\boldsymbol{A}$, defined as those entries $a_{\lambda,\nu}$ of $\boldsymbol{A}$ such that $\nu, \lambda \in \Lambda$ and $\Lambda \subset \nabla$, is bounded by $c_2c_1^{-1}$

$$\mathrm{cond}_2(\boldsymbol{A}_\Lambda) = \|\boldsymbol{A}_\Lambda\|_{\ell_2}\|\boldsymbol{A}_\Lambda^{-1}\|_{\ell_2} \leq \kappa = c_2c_1^{-1}.$$

This discretization of the problem (3.1) can be found in [9],[3],[8]. It has been the starting point of most recent wavelet numerical methods for elliptic problems such as the Poisson problem.

3.2 Objectives of the scheme

The object of the numerical scheme will be to find the best approximate solution to the infinite matrix equation $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}$, given that all computations are necessarily done on finite sequences of numbers. This means in particular that finite subsets $\Lambda_f, \Lambda_u \subset \nabla$ will have to be found such that f is well approximated by a finite sequence of wavelet coefficients

$(f_\lambda)_{\lambda\in\Lambda_f}$ and u by $(u_\lambda)_{\lambda\in\Lambda_u}$.

Sparse representation of functions

As seen in Chapter 2, the cancellation property of wavelets — that is, their having vanishing moments — leads to the sparse representation of certain functions, in particular those functions which have localized irregularities and are otherwise smooth. For these functions, the size of their wavelet coefficients $|d_\lambda|$ on level $|\lambda|$ decreases rapidly as the center of the support of the associated wavelet ψ_λ is situated further away from the zone of irregularity, and also as the level $|\lambda|$ increases. Consequently, a finite representation of such a function, comprising the largest N coefficients d_λ will be sparse in that it will not contain all the coefficients associated with the basis functions ψ_λ on a given level, but instead the coefficients that are present will be 'located' in a neighbourhood of the zone of irregularity that will be all the more narrow as the level of the coefficients is higher. This may be seen as the wavelet equivalent to the irregular meshes used by finite element methods. The rate of decay, as N increases, of the error made by retaining only the N largest coefficients of a sequence in ℓ_2 has been described formally, for instance in [8].

Known behaviour of the magnetization suggests an adaptive scheme

In the present situation, the numerical scheme must also fit well within the framework of the repetition of a time iteration followed by a new Poisson problem. It may be better to think of the scheme as solving $\boldsymbol{A}\boldsymbol{u}^n = \boldsymbol{f}^n$, where $\boldsymbol{f}^n$ is a function of the magnetization at time τ_n. At each step τ_n, the new Poisson problem is entirely determined by a N_n-term approximation $\boldsymbol{f}^n_{\Lambda_n}$ to the right hand side $\boldsymbol{f}^n$. Physical experiments and previous mathematical simulations such as in [25], give a good indication of two reasons why the

wavelet representation of the right hand side $\boldsymbol{f}^n$ should be sparse rather than uniform. Both reasons are traced back to sharp local variations in the magnetization $\boldsymbol{m}_n$.

Firstly, for all τ_n, the magnetization $\boldsymbol{m}_n$ presents a sharp variation at the interface $\partial\Omega_{int} = \Omega_{int} \cap \Omega_{ext}$ since $\boldsymbol{m}_n$ is zero on Ω_{ext}, the non-magnetic region. Secondly, inside the nano-element Ω_{int}, under certain conditions, the magnetization evolves with time into a domain formation. Within each domain, the magnetization is aligned parallel to one single direction. The interface between two such domains is termed the domain wall and is the location of sharp direction variations in $\boldsymbol{m}_n$. Hence, when n increases, due to the formation of domain walls, the sequence $\boldsymbol{f}^n$ should progressively contain groups of larger coefficients, which should migrate towards the final position of the domain walls, when $\boldsymbol{m}$ has stabilised.

The first reason would tend to suggest a numerical scheme based on a sparse and fixed representation of the $\boldsymbol{f}^n$, whereas the second reason suggests a sparse and adaptive representation, as n increases. As for the solution $\boldsymbol{u}^n$, it is also expected to present a fairly sharp variation on the edge of the nano-element, and since ∇u^n is one of the driving causes for the evolution of $\boldsymbol{m}_n$ with time, it is expected that the irregularities in $\boldsymbol{m}_n$ should be somewhat anticipated in u^n. Therefore it seems reasonable to expect the numerical scheme to produce a solution $\boldsymbol{u}_{\Lambda_{u_n}}$, where the set Λ_{u_n} is not necessarily the same as Λ_{f_n}, but where it could be the same as $\Lambda_{f_{n+1}}$.

Comparison with the aims of other adaptive wavelet schemes

Several adaptive wavelet methods for elliptic equations have been published; they are all iterative methods. Iterative methods are based on repeated matrix vector multiplications, whereas direct methods are based on the computation of a decomposition for the $\boldsymbol{A}$ matrix,

followed by forward and backward substitution. Because the wavelets used have compact support, many entries $\langle \nabla\psi_\lambda, \nabla\psi_\nu \rangle$ are zero, hence the $\boldsymbol{A}$ matrix is sparse. It is not however, a tridiagonal but a banded matrix, and it is known that the matrix decomposition leads to substantial 'fill-in', which is the reason why direct methods are avoided. When $\boldsymbol{A}$ is a finite dimensional, symmetric, positive definite matrix, several classical iterative methods are used, including the gradient and conjugate gradient methods. In the present case, $\boldsymbol{A}$ is indeed symmetric and positive definite, but is not finite-dimensional and part of the strategy of the adaptive methods consists of modifying the matrix multiplication of the classical iterative methods in order to allow dynamically chosen sub-sections of the infinite matrix to be used.

In [9] and in [8], two adaptive schemes are detailed. The overall principle of both is similar to that of multi-grid techniques, used in particular with finite element methods, in that the first iterations are made with a representation of the solution involving few unknowns, making the work involved at each iteration fairly light, and progressively, as the scheme reaches closer to the true solution, the number of degrees of freedom is increased. In this way, only few iterations involve the final, large number of degrees of freedom. The aims of the algorithm presented in [9] are the following: generate an ascending sequence of nested sets Λ_j such that $\#(\Lambda_j \setminus \Lambda_{j-1})$ stays as small as possible, and such that the error for the solution calculated with each Λ_j is reduced by some fixed factor, so $\|\boldsymbol{u} - \boldsymbol{u}_{\Lambda_{j+1}}\| \leq \theta \|\boldsymbol{u} - \boldsymbol{u}_{\Lambda_j}\|$, where $0 < \theta < 1$ does not depend on j and where $\boldsymbol{u}$ denotes the true solution. The algorithm presented in [8] has the same aims though it follows a different strategy for achieving them.

In view of the conclusions of the previous section, it may be that the algorithms from

[9] and [8] are not well suited for the problem at hand. The first reason for this is that at any time increment $n \geq 1$, the solution $\boldsymbol{u}^{n-1}_{\Lambda_{u_{n-1}}}$ to the Poisson equation at the previous time step is already a good approximation to the solution $\boldsymbol{u}^n$. Hence the 'multi-grid' strategy may not necessarily be very relevant here. The other aim of the two algorithms, which consists of the optimal choice of a super-set Λ_j, given the set Λ_{j-1}, is also a little different from what might be needed in the present situation, where the solution at time $n-1$ is probably represented in terms of the total number of unknowns that can be afforded, say N, and therefore the problem would seem perhaps more that of a redistribution of the resources from $\Lambda_{u_{n-1}}$ to Λ_{u_n}, $\#(\Lambda_{u_{n-1}}) = \#(\Lambda_{u_n}) = N$, than that of further refinement. The numerical scheme implemented for this thesis is described further in this chapter. A proof of its convergence is also given. This scheme makes use of the matrix-vector multiplication subroutine defined in [9] and and [8] and the implementation of the scheme follows closely certain suggestions made in [3].

3.3 Scheme and convergence of the scheme

Matrix multiplication

The scheme used in this thesis is an iterative scheme and as such it makes repeated use of an approximation to the matrix vector multiplication $\boldsymbol{Ax}$, where $\boldsymbol{A}$ is an infinite matrix and $\boldsymbol{x}$ is an finite vector. The algorithm used to calculate this approximation is based on a fast matrix vector multiplication subroutine called $\boldsymbol{MULT}$ and described in [9]. This subroutine calculates a finitely supported vector $\boldsymbol{w}$ such that the error $\|\boldsymbol{Ax} - \boldsymbol{w}\|_{\ell_2}$ is bounded by a known arbitrary quantity η. The algorithm for $\boldsymbol{MULT}$ works by taking advantage of the sparsity of the $\boldsymbol{A}$ matrix and also of the decrease of the modulus of

its entries $\langle \nabla\psi_\lambda, \nabla\psi_\nu \rangle$ as the difference of levels $\big||\lambda| - |\nu|\big|$ increases further and further away from the diagonal. The ***MULT*** algorithm and its implementation for this thesis are presented further in Chapter 5. Here, it is enough to say that the implementation involves a modification of ***MULT*** which imposes an upper bound on the distance between the entries of $\boldsymbol{A}$ that will be used and the diagonal. Hence, if this upper bound is denoted by $r > 0$, any entry $a_{\lambda,\nu}$ such that $\big||\lambda| - |\nu|\big| > r$ will not be used. Denoting by $\boldsymbol{A}_r$, the matrix of the entries of $\boldsymbol{A}$ that can be used, it is clear that $\boldsymbol{A}_r$ has a finite number of entries per row and column, and that it is a symmetric positive definite matrix, whose euclidean norm is bounded by the same bounds as the norm of $\boldsymbol{A}$. The scheme given below is intended to calculate an approximate solution to the equation

$$\boldsymbol{A}_r \boldsymbol{u} = \boldsymbol{f}_{\Lambda_f}, \qquad \#(\Lambda_f) = N,$$

for a given *maximum radius* r and a maximum vector length N. A bound for the norm $\|\boldsymbol{A} - \boldsymbol{A}_r\|$ is given in Chapter 5, as it is a core component of the ***MULT*** algorithm. Also, in Chapter 7, the accuracy of the solutions found for increasing r is investigated numerically on an example which is relevant to the whole model and for which a simple analytical expression for the true solution of the infinite Poisson problem is available. This example will be used to choose values for r and N, which will not fatally compromise the accuracy of the overall simulation.

Adaptive steepest gradient scheme

Let $P_N(\boldsymbol{x})$ denote the vector comprising of the N largest (in modulus) entries of $\boldsymbol{x}$. Given a maximum vector length N, a symmetric positive definite matrix $\boldsymbol{A}$ with finitely many

entries per row and column, such that $a \leq \|A\|_{\ell_2} \leq b, \quad a, b > 0$, together with a vector $\boldsymbol{f}$ with N non-zero entries, and a termination parameter ϵ, the following algorithm finds $\boldsymbol{u}_f$, an approximate vector solution with length N to the equation $\boldsymbol{Au} = \boldsymbol{f}$. This algorithm is a variant on the method of steepest descent.

Let $\boldsymbol{u}_0$ be an initial guess at the solution. If no initial guess is available, the null vector may be used instead.

$j := -1$

Initial Step

$j := j + 1$

$\bar{\boldsymbol{w}}_j := \boldsymbol{Au}_j - \boldsymbol{f}$

$\boldsymbol{w}_j := P_N(\bar{\boldsymbol{w}}_j)$

$d = \|\bar{\boldsymbol{w}}_j - \boldsymbol{w}_j\|_{\ell_2}$

$\rho_j := \boldsymbol{w}_j^T \boldsymbol{w}_j$

Loop

While ($\rho_j > \epsilon$)

$\boldsymbol{e}_j := \boldsymbol{Aw}_j$

$\alpha_j := \frac{\rho_j}{\boldsymbol{w}_j^T \boldsymbol{e}_j}$

$\boldsymbol{u}_{j+1} := \boldsymbol{u}_j - \alpha_j \boldsymbol{w}_j$

$\bar{\boldsymbol{w}}_{j+1} := \boldsymbol{w}_j - \alpha_j \boldsymbol{e}_j$

$\boldsymbol{w}_{j+1} := P_N(\bar{\boldsymbol{w}}_{j+1})$

$d := d + \|\bar{\boldsymbol{w}}_j - \boldsymbol{w}_j\|_{\ell_2}$

If $d > \|\boldsymbol{w}_j\|_{\ell_2}/2$, go to the Initial Step.

$\rho_{j+1} := \boldsymbol{w}_{j+1}^T \boldsymbol{w}_{j+1}$

$j := j + 1$

Final Step

$\boldsymbol{u}_f = P_N(\boldsymbol{u}_j)$

Convergence of the scheme

It is proved below that the sequence of the $\boldsymbol{u}_j$ converges towards the correct solution $\boldsymbol{u}$ of the equation $\boldsymbol{Au} = \boldsymbol{f}$. Such a solution exists, is unique and has a finite number of entries. The final step of the scheme defines $\boldsymbol{u}_f$ as the best $N-$term approximation of the last computed term in the sequence. The convergence of the adaptive scheme is studied with respect to the error function usually used to prove the convergence of the classical steepest descent algorithm. This error function is denoted by E and it is the A^{-1}-norm of the residual

$$E(\boldsymbol{u}_j) = \|\boldsymbol{Au}_j - \boldsymbol{f}\|_{\boldsymbol{A}^{-1}},$$

where $\boldsymbol{u}_j$ denotes the solution calculated by the adaptive scheme after the j^{th} iteration. The following shows without making any assumptions on the initial guess $\boldsymbol{u}_0$ that $E(\boldsymbol{u}_{j+1}) < E(\boldsymbol{u}_j)$.

$$\begin{aligned}
E(\boldsymbol{u}_{j+1}) &= (\boldsymbol{Au}_{j+1} - \boldsymbol{f})^T \boldsymbol{A}^{-1}(\boldsymbol{Au}_{j+1} - \boldsymbol{f}) \\
&= \Big(\boldsymbol{A}(\boldsymbol{u}_j - \alpha_j \boldsymbol{w}_j) - \boldsymbol{f}\Big)^T \boldsymbol{A}^{-1}\Big(\boldsymbol{A}(\boldsymbol{u}_j - \alpha_j \boldsymbol{w}_j) - \boldsymbol{f}\Big) \\
&= (\boldsymbol{Au}_j - \boldsymbol{f})^T \boldsymbol{A}^{-1}(\boldsymbol{Au}_j - \boldsymbol{f}) \; - \; (\boldsymbol{Au}_j - \boldsymbol{f})^T \alpha_j \boldsymbol{w}_j \\
&\quad - \alpha_j \boldsymbol{w}_j^T (\boldsymbol{Au}_j - \boldsymbol{f} \; - \alpha_j \boldsymbol{Aw}_j) \\
&= E(\boldsymbol{u}_j) \; - \; 2\alpha_j (\boldsymbol{Au}_j - \boldsymbol{f})^T \boldsymbol{w}_j \; + \; \alpha_j^2 \boldsymbol{w}_j^T \boldsymbol{Aw}_j \\
&= E(\boldsymbol{u}_j) \; - \; \frac{2\boldsymbol{w}_j^T \boldsymbol{w}_j \, (\boldsymbol{Au}_j - \boldsymbol{f})^T \boldsymbol{w}_j \; - \; (\boldsymbol{w}_j^T \boldsymbol{w}_j)^2}{\boldsymbol{w}_j^T \boldsymbol{Aw}_j}
\end{aligned}$$

And so $E(\boldsymbol{u}_{j+1})/E(\boldsymbol{u}_j) = 1 - 1/\mu_j$, where

$$\mu_j = \frac{(\boldsymbol{A}\boldsymbol{u}_j - \boldsymbol{f})^T \boldsymbol{A}^{-1}(\boldsymbol{A}\boldsymbol{u}_j - \boldsymbol{f})\, \boldsymbol{w}_j^T \boldsymbol{A}\boldsymbol{w}_j}{2\boldsymbol{w}_j^T \boldsymbol{w}_j\, (\boldsymbol{A}\boldsymbol{u}_j - \boldsymbol{f})^T \boldsymbol{w}_j \;-\; (\boldsymbol{w}_j^T \boldsymbol{w}_j)^2}$$

Let us express the true residual $\boldsymbol{A}\boldsymbol{u}_j - \boldsymbol{f}$ as the sum $\boldsymbol{w}_j + \boldsymbol{d}_j$. It follows that

$$\mu_j = \frac{(\boldsymbol{w}_j + \boldsymbol{d}_j)^T \boldsymbol{A}^{-1}(\boldsymbol{w}_j + \boldsymbol{d}_j)\, \boldsymbol{w}_j^T \boldsymbol{A}\boldsymbol{w}_j}{(\boldsymbol{w}_j^T \boldsymbol{w}_j)^2 \;+\; 2\boldsymbol{w}_j^T \boldsymbol{w}_j\, \boldsymbol{d}_j^T \boldsymbol{w}_j}$$

Let $\boldsymbol{A}_1$ denote a finite symmetric sub-matrix of $\boldsymbol{A}$ such that $A_1 \boldsymbol{w}_j = \boldsymbol{A}\boldsymbol{w}_j$ and let $w_{j,i}, 1 \leq i \leq n_1$, be the coordinates of $\boldsymbol{w}_j$ in terms of an orthogonal collection of eigenvectors for $\boldsymbol{A}_1$ with associated eigenvalues λ_i. Then

$$\begin{aligned} \boldsymbol{w}_j^T \boldsymbol{A}\boldsymbol{w}_j &= \sum_{i=1}^{n_1} (w_{j,i})^2 \lambda_i \\ &\leq \min_i \lambda_i \sum_{i=1}^{n_1} (w_{j,i})^2 \\ &\leq b\boldsymbol{w}_j^T \boldsymbol{w}_j \end{aligned}$$

Similarly, let $\boldsymbol{A}_2$ denote a finite symmetric sub-matrix of $\boldsymbol{A}^{-1}$ such that $(\boldsymbol{w}_j + \boldsymbol{d}_j)^T \boldsymbol{A}_2 (\boldsymbol{w}_j + \boldsymbol{d}_j) = (\boldsymbol{w}_j + \boldsymbol{d}_j)^T \boldsymbol{A}^{-1}(\boldsymbol{w}_j + \boldsymbol{d}_j)$, and let $w'_{j,i}$ and $d_{j,i}, 1 \leq i \leq n_2$, be the coordinates of $\boldsymbol{w}_j$ and $\boldsymbol{d}_j$ in terms of an orthogonal collection of eigenvectors for $\boldsymbol{A}_2$ with associated eigenvalues λ'_i. Then

$$\begin{aligned} (\boldsymbol{w}_j + \boldsymbol{d}_j)^T \boldsymbol{A}^{-1}(\boldsymbol{w}_j + \boldsymbol{d}_j) &= \sum_{i=1}^{n_2} (w'_{j,i} + d_{j,i})^2 \lambda'_i \\ &\leq \min_i \sum_{i=1}^{n_2} (w'_{j,i} + d_{j,i})^2 \\ &\leq \frac{1}{a} (\boldsymbol{w}_j + \boldsymbol{d}_j)^T (\boldsymbol{w}_j + \boldsymbol{d}_j). \end{aligned}$$

This gives for μ_j

$$\begin{aligned}
\mu_j &\leq \frac{b}{a} \frac{\boldsymbol{w}_j^T \boldsymbol{w}_j \, (\boldsymbol{w}_j + \boldsymbol{d}_j)^T (\boldsymbol{w}_j + \boldsymbol{d}_j)}{(\boldsymbol{w}_j^T \boldsymbol{w}_j)^2 \; + \; 2 \boldsymbol{w}_j^T \boldsymbol{w}_j \, \boldsymbol{d}_j^T \boldsymbol{w}_j} \\
\mu_j &\leq \frac{b}{a} \frac{\boldsymbol{w}_j^T \boldsymbol{w}_j \, (\boldsymbol{w}_j + \boldsymbol{d}_j)^T (\boldsymbol{w}_j + \boldsymbol{d}_j)}{(\boldsymbol{w}_j^T \boldsymbol{w}_j) \left((\boldsymbol{w}_j + \boldsymbol{d}_j)^T (\boldsymbol{w}_j + \boldsymbol{d}_j) \; - \; \boldsymbol{w}_j^T \boldsymbol{w}_j \, \boldsymbol{d}_j^T \boldsymbol{d}_j \right)} \\
\frac{1}{\mu_j} &\geq \frac{a}{b} \left(1 - \frac{\boldsymbol{w}_j^T \boldsymbol{w}_j \, \boldsymbol{d}_j^T \boldsymbol{d}_j}{\boldsymbol{w}_j^T \boldsymbol{w}_j \, (\boldsymbol{w}_j + \boldsymbol{d}_j)^T (\boldsymbol{w}_j + \boldsymbol{d}_j)} \right) \\
\frac{1}{\mu_j} &\geq \frac{a}{b} \left(1 - \frac{\boldsymbol{d}_j^T \boldsymbol{d}_j}{(\boldsymbol{w}_j + \boldsymbol{d}_j)^T (\boldsymbol{w}_j + \boldsymbol{d}_j)} \right)
\end{aligned}$$

For $E(\boldsymbol{u}_{j+1}) < E(\boldsymbol{u}_j)$ to hold, the condition $1/\mu_j > 0$ must be satisfied. It is shown below that this is guaranteed because

$$0 \leq \frac{\boldsymbol{d}_j^T \boldsymbol{d}_j}{(\boldsymbol{w}_j + \boldsymbol{d}_j)^T (\boldsymbol{w}_j + \boldsymbol{d}_j)} < 1,$$

for all steps j.

Firstly, when $\boldsymbol{w}_j = \boldsymbol{w}_i$ is calculated during an *Initial Step*, $\boldsymbol{d}_i = \boldsymbol{A}\boldsymbol{u}_i - f - \boldsymbol{w}_i$ is also equal to the difference $\bar{\boldsymbol{w}}_i - \boldsymbol{w}_i$, hence it is orthogonal to $\boldsymbol{w}_i$. It follows that $(\boldsymbol{w}_i + \boldsymbol{d}_i)^T (\boldsymbol{w}_i + \boldsymbol{d}_i) = \boldsymbol{d}_i^T \boldsymbol{d}_i + \boldsymbol{w}_i^T \boldsymbol{w}_i$ and consequently the inequality $0 \leq \frac{\boldsymbol{d}_i^T \boldsymbol{d}_i}{(\boldsymbol{w}_i + \boldsymbol{d}_i)^T (\boldsymbol{w}_i + \boldsymbol{d}_i)} < 1$ holds.

Secondly, when $\boldsymbol{w}_j$ is calculated inside the *Loop*, the norm of $\boldsymbol{d}_j$ may be bounded as follows

$$\begin{aligned}
\boldsymbol{d}_j &= \boldsymbol{A}\boldsymbol{u}_j - f - \boldsymbol{w}_j \\
&= \boldsymbol{A}(\boldsymbol{u}_{j-1} - \alpha_{j-1} \boldsymbol{w}_{j-1}) - f - \boldsymbol{w}_j \quad \text{and} \quad \alpha_{j-1} \boldsymbol{A} \boldsymbol{w}_{j-1} = \boldsymbol{w}_{j-1} - \bar{\boldsymbol{w}}_j
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{d}_j &= \boldsymbol{A}\boldsymbol{u}_{j-1} - f - \boldsymbol{w}_{j-1} + (\bar{\boldsymbol{w}}_j - \boldsymbol{w}_j) \\
&= \boldsymbol{d}_{j-1} + (\bar{\boldsymbol{w}}_j - \boldsymbol{w}_j) \\
\|\boldsymbol{d}_j\|_{\ell_2} &\leq \sum_{k=i}^{j} \|\bar{\boldsymbol{w}}_j - \boldsymbol{w}_j\|_{\ell_2} = d \\
\|\boldsymbol{d}_j\|_{\ell_2} &\leq \|\boldsymbol{w}_j\|_{\ell_2}/2.
\end{aligned}$$

This bound is sufficient to ensure that $\|\boldsymbol{d}_j + \boldsymbol{w}_j\|_{\ell_2} > \|d_j\|_{\ell_2}$ by the following sequence of implied steps.

$$\begin{aligned}
&\|\boldsymbol{w}_j\|_{\ell_2} > 2\|\boldsymbol{d}_j\|_{\ell_2} \\
&\boldsymbol{w}_j^T \boldsymbol{w}_j > 2(\boldsymbol{d}_j^T \boldsymbol{d}_j)^{1/2} (\boldsymbol{w}_j^T \boldsymbol{w}_j)^{1/2} > 2|\boldsymbol{d}_j^T \boldsymbol{w}_j| \\
&\boldsymbol{w}_j^T \boldsymbol{w}_j + 2\boldsymbol{d}_j^T \boldsymbol{w}_j > 0 \\
&\boldsymbol{w}_j^T \boldsymbol{w}_j + 2\boldsymbol{d}_j^T \boldsymbol{w}_j + \boldsymbol{d}_j^T \boldsymbol{d}_j > \boldsymbol{d}_j^T \boldsymbol{d}_j \\
&\|\boldsymbol{d}_j + \boldsymbol{w}_j\|_{\ell_2} > \|d_j\|_{\ell_2}
\end{aligned}$$

This concludes the proof that the sequence $(\boldsymbol{u}_j)_j$ converges towards the solution $\boldsymbol{u}$ of $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}$. The rate of convergence is improved when the ratio $a/b = \kappa^{-1}$ is large and when the directions of descent $\boldsymbol{w}_j$ are kept close to the directions of steepest descent $\boldsymbol{A}\boldsymbol{u}_j - f$.

Remark: it has not been shown that this scheme converges in a finite number of steps. Therefore, in practice, there should be some reasonable balance between N and ϵ. In particular, if the scheme does not terminate, the value of these parameters should be revised.

If the ***MULT*** algorithm is used to accelerate the computation of the matrix-vector products $\boldsymbol{A}\boldsymbol{u}_j$ and $\boldsymbol{A}\boldsymbol{w}_j$, the computation of d must be modified to take into account this

additional source of error and still guarantee the convergence of the scheme. Let η denote the accuracy parameter in the ***MULT*** algorithm, then the ℓ_2-norm of the error introduced at each matrix-vector multiplication is bounded by η. When $\boldsymbol{w}_i$ has been calculated in an *Initial Step*, the choice $\eta < \sqrt{\epsilon}/4$ and the modification $d := \|\bar{\boldsymbol{w}}_i - \boldsymbol{w}_i\|_{\ell_2} + \eta$ ensure that $\|\boldsymbol{d}_i\|/\|\boldsymbol{d}_i + \boldsymbol{w}_i\| < 1$. Also, the modification $d := d + \|\bar{\boldsymbol{w}}_j - \boldsymbol{w}_j\|_{\ell_2} + \alpha_j\eta$ inside the *Loop* ensures that $\|\boldsymbol{d}_j\|/\|\boldsymbol{d}_j + \boldsymbol{w}_j\| < 1$ for any step j. These modifications can be explained by writing the vector $\boldsymbol{d}_i$ as $\bar{\boldsymbol{w}}_i - \boldsymbol{w}_i + \boldsymbol{\eta}_i$, where $\|\boldsymbol{\eta}_i\|_{\ell_2} \leq \eta$ is the norm of the difference between the exact and the approximate calculation of $\boldsymbol{A}\boldsymbol{u}_i$, and by recursively writing the vectors $\boldsymbol{d}_j$ as $\boldsymbol{d}_{j-1} + (\bar{\boldsymbol{w}}_j - \boldsymbol{w}_j) - \alpha_j\boldsymbol{\eta}_j$.

3.4 Comments on the scheme

This scheme is straightforward and it is designed to be applicable to the problem at hand. Its performance will be evaluated numerically in Chapter 7 from the points of view of:

1. Whether the size of the memory storage used remains within a constant factor of N. The following terms are required to be stored in memory:

   - **vector $\boldsymbol{w}_i$ or $\boldsymbol{w}_j$:** Length N, by definition.
   - **vector $\boldsymbol{e}_j$:** Length bounded by CN, where C is a constant. This is ensured by the imposition of a *maximum radius* on the $\boldsymbol{A}$ matrix. The length will vary depending on this radius and on whether the ***MULT*** algorithm is used. The size of the coefficient C is very important in practice.
   - **vector $\boldsymbol{u}_j$:** At the first Initial Step, it has length N or less. As j increases, its length is not bounded, but the rate of increase is less than jN. Here, the

total number of iterations will play an important role, as well as the quality of the overlap between the index set of the final solution Λ_{u_f} and that of the right hand side Λ_f.

- **vector $\bar{w}_j$:** Length bounded by $C\#(\boldsymbol{u}_i) + N$ in the *Initial Step* and it is the same as the length of $\hat{\boldsymbol{e}}_j$ in the *Loop*. Here, the effect of restarting the scheme with an Initial Step when $j > 0$ is apparent.

- **matrix $\boldsymbol{A}$:** It is only necessary to keep in memory the entries which are needed for a particular multiplication, however, storing the entries of $\boldsymbol{A}$ from one iteration to the next saves much computing effort. Overall, the number of entries that are used is bounded by a constant factor times the length of the last computed $\boldsymbol{u}_j$. Once again, the size of this value for the example test will be of interest.

2. Whether the number of computing operations per iteration remains proportional to N. The computational steps may be quantified as follows:

   - The number of operations needed to perform a matrix-vector multiplication is proportional to length of the vector. When an *Initial Step* is performed in an iteration, two matrix-vector multiplications are used instead of one and for one of these multiplications, the length of the vector is that of $\boldsymbol{u}_j$.

   - The calculation of the $N-$best approximation of a vector requires the sorting of this vector's entries by decreasing modulus and the copying of the N largest entries into the new vector. The sorting operation has a computational cost of $n\log(n)$, where n is the length of the vector.

   - The computational cost of d, ρ_i and α_j is always proportional to N.

3. The speed of convergence, the accuracy of $\boldsymbol{u}_f$ as a solution of the exact Poisson problem (no *maximum radius* imposed), and the distance between the last computed $\boldsymbol{u}_j$ and $\boldsymbol{u}_f$.

The next three chapters will give some details on the computation of the main tools necessary for the implementation of this scheme and of the Landau-Lifshitz simulations in general. These concern, in order, the computation of the wavelet right hand side $\boldsymbol{f}$ of the Poisson equation, the matrix vector multiplication algorithm and the dynamic calculation of the entries for the $\boldsymbol{A}$ matrix, and the pointwise evaluation of sparse wavelet expressions. The subsequent chapters present the numerical tests performed on the adaptive Poisson solver, as well as the results obtained with Landau-Lifshitz simulations.

Chapter 4

Sparse wavelet expansion of a function f - or - Calculation of the R.H.S. of the Poisson equation

In the two-dimensional case, the system of equations which we are solving is:

$$\begin{aligned} \forall \lambda \in \Lambda, \qquad & \sum_{\mu \in \nabla} d_\mu \iint_\Omega \textbf{grad}\ \psi_\mu \textbf{ . } \textbf{grad}\ \psi_\lambda \, dx\, dy \\ = \ & -\iint_{\Omega_{int}} 4\pi (\ \mathrm{div}\ \boldsymbol{M})\, \psi_\lambda dx\, dy + \oint_{\partial \Omega_{int}^+} 4\pi (\boldsymbol{M} \textbf{ . } \boldsymbol{n})\, \psi_\lambda \, ds. \end{aligned}$$

In particular, for the simple example where $\boldsymbol{M} = (0, 1)$ inside the platelet (the region Ω_{int})

and is null outside it (the region Ω_{ext}), this is:

$$\begin{aligned} \forall \lambda \in \Lambda, \qquad & \sum_{\mu \in \nabla} d_\mu \iint_\Omega \mathbf{grad}\ \psi_\mu \cdot \mathbf{grad}\ \psi_\lambda \, dx\, dy \\ = \ & 4\pi \int_{x_a}^{x_b} \psi_\lambda(x, y_b) - \psi_\lambda(x, y_a)\, dx. \\ & \text{where } \Omega_{int} = [x_a, x_b] \times [y_a, y_b]. \end{aligned}$$

Hence the right hand side of the equation involves inner products of the form $\int f\psi_\lambda$, where ψ_λ denotes a wavelet, and the set of such wavelet coefficients forms the dual wavelet expansion of f. Because the region of interest (the platelet) occupies only a very small part of the total region of solution, a *sparse representation* of f is sought, where more wavelets are used in and near the platelet.

4.1 From full wavelet expansion to sparse

The usual projection of a function on a wavelet space, which is part of the Multilevel Library in the 1D case, consists first of all in calculating (on a regular grid) a *full scaling function representation* of f on a chosen refinement level. In a second step, the wavelet transform is applied, resulting in a *full wavelet representation* of f. Finally, the sparse wavelet representation of f is obtained by discarding unwanted wavelet coefficients of smaller magnitude.

Unlike the second and third of these steps, the first one can be implemented in a variety of ways, one of which is sketched below because it is very close to the method actually used in this project.

Projection onto the (dual) scaling function space

Let j be the level of the dual scaling function space considered. The projection onto $\tilde{V}_j$ of a function f is the expansion

$$P_{\tilde{V}_j}(f) = \sum_{\lambda \in \Delta_j} \langle f, \phi_\lambda \rangle \tilde{\phi}_\lambda.$$

The inner products $\langle f, \phi_\lambda \rangle$ are often approximated in a convenient way by replacing f by its expansion in terms of interpolating piecewise linear scaling functions, denoted ϕ^I_λ, which are the hat functions in 1D.

$$P_{\tilde{V}_j}(f) \approx \sum_{\lambda \in \tilde{\Delta}_j} \sum_{\mu \in \tilde{\Delta}_j} f(x_\mu) \langle \phi^I_\mu, \phi_\lambda \rangle \tilde{\phi}_\lambda.$$

The inner products $\langle \phi^I_\mu, \phi_\lambda \rangle$ for the one-dimensional case are calculated in the Multilevel Library by functions *integrals*, *InnerProducts* (on the real line) and *I-Integral* (on the interval) as *refinable integrals* by a spectral method based on the fact that both functions involved in the inner product are refinable functions. The library's documentation points to [29] for a reference on this subject. For the multi-dimensional case, as long as tensor-product wavelets are used, the matter boils down to multiplying together 1D-inner products.

This method presents several advantages: it is based on the point values of f, the non-zero inner products $\langle \phi^I_\mu, \phi_\lambda \rangle$ have relatively few distinct values which can be calculated accurately once and then used from memory in each sum, and finally it is a direct method in the sense that there is no need to solve a system of equations. All this combines to make

it a fast method.

The Multilevel Library also provides a method (*PolynomApprox*) which increases the order of the approximation by replacing f by a polynomial of order higher than linear. It is often advocated to use polynomials of the same order as that of the dual multi-resolution analysis, to keep and not decrease the order of the Galerkin method. For this reason, in this thesis, f was not replaced by an expansion in terms of ϕ_μ^I functions, but by a Lagrange interpolation of higher order, which will be presented further.

4.2 Method implemented: directly to the sparse representation

The method implemented for this project is based on [26] and most of the notations used and ideas presented come from that article.

The need for a full representation of a function at some point in the process of calculating its sparse wavelet representation is the drawback associated with the usual methods of projecting a function onto a wavelet space. For reasons of memory space, the number of levels used in the wavelet representation becomes limited by the number of scaling functions on the finest level despite the fact that this number has no connection with the actual size of the sparse representation of the function. Therefore in situations where the wavelet representation of a function is known beforehand to be sparse, the usual method incurs a waste: a waste of storage space and also a waste in the computing effort dedicated to the calculation of all the scaling function and wavelet coefficients that will never be used.

By contrast, an efficient method would require storage and computing resources proportional to the number of coefficients used in the final, sparse, representation of the function. Calculating directly only those wavelet coefficients that will remain in the sparse representation seems the obvious answer. It is apparent that two predictions would then be necessary: one to relate a priori the size and the quality of the sparse representation of a function, and one to predict exactly which wavelet coefficients will be involved.

4.2.1 Approximation by canonical projection

Let Λ denote a finite subset of the wavelet index set ∇, and let $\tilde{S}_\Lambda$ denote the span of the $\tilde{\psi}_\lambda$, $\lambda \in \Lambda$. The canonical projection of a function f onto $\tilde{S}_\Lambda$ is denoted $\tilde{Q}_\Lambda f$ and is the expansion

$$\tilde{Q}_\Lambda f = \sum_{\lambda \in \Lambda} \langle f, \psi_\lambda \rangle \tilde{\psi}_\lambda.$$

Global estimate

Due to the Riesz basis property of the wavelets, this expansion realizes a near-best approximation of $f \in L^2$ by a function in $\tilde{S}_\Lambda$, in the sense that

$$\|\tilde{Q}_\Lambda f - f\|_{L_2} \leq \inf_{v_\lambda \in \tilde{S}_\Lambda} \|v_\lambda - f\|_{L_2}.$$

Ideally, the wavelet expansion of the equation's RHS would be no other than its canonical projection onto a well chosen subspace $\tilde{S}_\Lambda$.

Tree structure of an index set

The choice of a subset $\Lambda \subset \nabla$ is restricted to subsets endowed with a tree structure. This choice is made for practical reasons, indeed it makes the process of handling sparse index sets much simpler and computationally cheaper. Moreover, because of the wavelet compression property, it corresponds to the natural structure of index sets defined as

$$\Lambda = \{\lambda : |\langle f, \psi_\lambda \rangle| < \epsilon\}.$$

The tree structure is very similar to that used in [7] under the name of pyramid structure. The precise definition of the tree structure is given below together with certain notations and definitions, which are used repeatedly in the same context.

- Let $\square$ denote the unit cube $[0,1]^n$ and let $\square_\lambda$ denote the dyadic cube $2^{-|\lambda|}(\boldsymbol{k}(\lambda) + [0,1]^n)$. The cube $\square_\lambda$ is associated with all the $\psi_{\lambda'}$ such that $|\lambda'| = |\lambda|$ and $\boldsymbol{k}(\lambda') = \boldsymbol{k}(\lambda)$. In dimension n, $2^n - 1$ wavelets and one scaling function are associated with it. The role of $\square_\lambda$ is to represent the relative positions and the relative sizes of the supports of different wavelets.

- The *ancestors* of λ are the indices λ' such that $\square_\lambda \subset \square_{\lambda'}$.

- The *children* of λ are the indices λ' such that $\square_{\lambda'}$ results from a single dyadic subdivision of $\square_\lambda$. The index λ is then logically called the *parent* of the λ'.

- An index set $\Lambda \subset \nabla$ has a *tree structure* if for $\lambda \in \Lambda$, all the ancestors of λ are also

in Λ.

- The *leaves* of such a set are then defined as the λ whose children are not all contained in Λ. The leaves are the indices of the wavelets which are on the local finest refinement level of a wavelet expansion. The set of leaves of Λ is denoted $\partial\Lambda$.

- A tree Λ is *M-graded* ($M \in \mathbb{N}$) if for all $\lambda \in \Lambda$, and for all λ' in the set of parents of λ,

$$\boldsymbol{k}(\lambda') + 2^{-|\lambda|+1}[-M, M+1]^n \subseteq \bigcup_{|\lambda''|=|\lambda|-1} \square_{\lambda''}, \text{ where } \lambda'' \in \Lambda.$$

Local estimate

An estimate of the local error of the approximation of $f \in L_p$ by $\tilde{Q}_\Lambda f$ is given in [26], p.12, Lemma 3.3, under the assumption that Λ has a tree structure.

Lemma: *'Assume that all ψ_λ associated with the support cube $\square_\lambda = 2^{-|\lambda|}(\boldsymbol{k} + \square)$ satisfy $\Omega_\lambda \subseteq 2^{-|\lambda|}(\boldsymbol{k} + [-M, M]^n)$, where*

$$\Omega_\lambda := supp\ \psi_\lambda \quad \text{and} \quad \tilde{\Omega}_\lambda := supp\ \tilde{\psi}_\lambda,$$

and that $\Lambda \in \nabla$ is M-graded. Then for any leaf $\lambda \in \partial\Lambda$ and any $1 \leq p \leq \infty$ one has

$$\|g - \tilde{Q}_\Lambda g\|_{L_p(\square_\lambda)} \lesssim \inf_{P \in \Pi_{\tilde{d}}} \|g - P\|_{L_p(\square^*_\lambda)},$$

where $\square_\lambda^*$ *is a cube satisfying*

$$\square_\lambda \subseteq \square_\lambda^*, \quad diam\, \square_\lambda^* \lesssim 2^{-|\lambda|},$$

and $\prod_{\tilde{d}}$ *denotes the space of polynomials on* $\mathbb{R}^n$ *of order at most* $\tilde{d}$.'

This estimate shows that the error of the wavelet approximation is locally bounded by the error of the best approximation by a polynomial of order at most $\tilde{d}$, where $\tilde{d}$ is the order of the dual multi-resolution analysis and where 'locally' is characterized by the size of the cubes $\square_\lambda, \lambda \in \partial\Lambda$.

In general, the canonical projection of a function f on $\tilde{S}_\Lambda$ cannot be calculated exactly or at low cost: a substitute for $\tilde{Q}_\Lambda$ must be used. In [26], p.25, the choice of substitute A_Λ is governed by the following requirements:

1. The number of operations needed to compute $A_\Lambda f$ remains proportional to $\#\Lambda$.

2. The local accuracy of A_Λ remains comparable to that of $\tilde{Q}_\Lambda$ as given by the estimate above.

In the problem at hand, the right hand side function f does not belong to L_2 and it is instead approximated in H^{-1}.

4.2.2 A priori choice of a sparse representation's wavelet coefficients with the BAS scheme

In the method implemented, the choice of the coefficients or, equivalently the choice of an appropriate subset Λ of ∇, is performed before any wavelet coefficients have been calculated, and therefore it must be based on a priori information on the function f. In the scheme that follows, this a priori information is contained in a positive 'error function' E, defined on the set of sub-domains of Ω. The scheme is set up to create a tree Λ for which the error E is distributed quasi-uniformly over the support-cubes $\square_\lambda$ of the tree leaves ($\lambda \in \partial\Lambda$). The scheme is called the Basic Adaptive Scheme, or BAS.

Simple requirements on function E are imposed as in [26], p.14:

- $E(\square') \leq E(\square'')$, for $\square' \subseteq \square''$, and
- $E(\square') \longrightarrow 0$, as $\text{diam}(\square') \longrightarrow 0$.

An algorithm for BAS

In [26], p.14, BAS is described by the following algorithm:
Given E, the associated BAS(E, δ) can roughly be described as follows. Set $\mathcal{B} = \square, \mathcal{G} = \emptyset$. While $\mathcal{B} \neq \emptyset$, pick $\square' \in \mathcal{B}$ and do:

1. If $E(\square') \leq \delta$, include $\square'$ in set $\mathcal{G}$,

   else include the children of $\square'$ in set $\mathcal{B}$.

2. Remove $\square'$ from set $\mathcal{B}$.

The algorithm terminates after a finite number of steps and outputs a set $\mathcal{G}_\delta = \{\square_\lambda : \lambda \in \partial\Lambda(\delta)\}$ of *good* cubes in the sense that $E(\square_\lambda) \leq \delta$ for all $\lambda \in \partial\Lambda(\delta)$. By construction, $\Lambda(\delta)$ is a tree.

4.2.3 An alternative projection operator

As was mentioned in Section 4.1, a usual substitute for the wavelet canonical projection of a function f consists of the wavelet canonical projection of a piecewise polynomial approximation of f. When a full representation of f is sought, the pieces of the polynomial are calculated on small cubes of uniform dimensions, whereas in the present situation, the size of the pieces' support must reflect the local need for accuracy. In view of the local estimate of Section 4.2.1, the correct size is given by the cube $\square_\lambda$ associated with the local leaf $\lambda \in \partial\Lambda$.

If λ is a leaf, the approximation of f on the support of ψ_λ can be done in one piece and it makes sense to replace $\langle f, \psi_\lambda \rangle$ by $\langle P_\lambda(f), \psi_\lambda \rangle$, where $P_\lambda(f)$ denotes a polynomial approximation of f on the support of ψ_λ. When λ is not a leaf, the order of the size of the support of ψ_λ is larger than that of the local leaves, indeed for large $|\lambda|$, it is close to the size of the whole domain $\Omega = \square$. In that case, the approximation of f on the support of ψ_λ must be a piecewise approximation such as $P_{\lambda'}(f)$ on each cube $\square_{\lambda'}$, where λ' is both a leaf and a child of λ. If there are m such λ', calculating one wavelet coefficient associated with λ by replacing $\langle f, \psi_\lambda \rangle$ by $\sum_{\lambda'} \langle P_{\lambda'}(f), \psi_\lambda \rangle_{L_2(\square_{\lambda'})}$ will amount to the same work as calculating m wavelet coefficients associated with leaves. This means that the total amount of work needed to approximate $\tilde{Q}_\Lambda f$ by this method cannot be proportional to the number of elements in Λ.

One way to get around this problem is to use a local scaling function expansion as an intermediate step between the function's point values and its wavelet expansion. The local scaling function expansion will consist of scaling function coefficients calculated as $\tilde{s}_\lambda = \langle P_\lambda(f), \phi_\lambda \rangle$, where $P_\lambda(f)$ is a one-piece polynomial approximation to f on the support $\sigma_{j,\boldsymbol{k}}$, $(j = |\lambda|, \boldsymbol{k} = \boldsymbol{k}(\lambda))$ of the generator function ϕ_λ. The set of λ involved, denoted $S(\partial\Lambda)$ contains all the children of the leaves of Λ and some extra indices needed for the subsequent wavelet transform. A local wavelet transform will then take the set of scaling function coefficients defined by $S(\partial\Lambda)$, to the wavelet coefficients defined by Λ.

4.2.4 Choice of the set of generator functions' indices $S(\partial\Lambda)$

The set of scaling functions $\phi_\lambda, \lambda \in S(\partial\Lambda)$ must contain all the scaling functions needed to produce the wavelets $\psi_{\lambda'}, \lambda' \in \partial\Lambda$ by wavelet transform. The multi-dimensional wavelet transform involves the one-dimensional refinement and wavelet equations. Therefore, a function $\phi_{j,\boldsymbol{k}}$ is 'needed' for producing the wavelets $\psi_{\lambda'}$ associated with $\square_{\lambda'}$ if for $1 \leq i \leq n$, ξ_{j,k_i} is needed for producing $\eta_{|\lambda'|,k_i(\lambda')}$ (by the wavelet equation) or $\xi_{|\lambda'|,k_i(\lambda')}$ (by the refinement equation). The set $S(\partial\Lambda)$ is built by the algorithm outlined below.

Simple algorithm for $S(\partial\Lambda)$

For each cube $\square_{\lambda'}, \lambda' \in \partial\Lambda$,

- Let $j = |\lambda'] + 1$.
- For each spatial direction i $(1 \leq i \leq n)$:
  - Find l_i, the smallest k such that $\xi_{j,k}$ is needed for producing $\eta_{|\lambda'|,k_i(\lambda')}$ or $\xi_{|\lambda'|,k_i(\lambda')}$.

- Find m_i, the largest k such that $\xi_{j,k}$ is needed for producing $\eta_{|\lambda'|,k_i(\lambda')}$ or $\xi_{|\lambda'|,k_i(\lambda')}$.

- Insert in $S(\partial\Lambda)$ all indices $\lambda = (j, \mathbf{0}, \boldsymbol{k})$ such that $l_i \leq k_i \leq m_i$ for all i, $1 \leq i \leq n$.

Let N_g denote the maximum number of scaling functions $\xi_{j,k}$ needed to produce a scaling function $\xi_{j-1,k'}$, which may be adapted to the boundary or not. N_g does not depend on the level j. Similarly N_w, the maximum number of scaling functions $\xi_{j,k}$ needed to produce a wavelet $\eta_{j-1,k'}$, does not depend on j. It is then clear that the number of entries in $S(\partial\Lambda)$ is strictly bounded by $(\max(N_g, N_w))^n$ times the number of cubes associated with the wavelets in $\partial\Lambda$, which is $(\#\partial\Lambda)/(2^n - 1)$ if the BAS algorithm was used. So for a given spatial dimension n, the cardinality of $S(\partial\Lambda)$ is proportional to the number of leaves.

4.2.5 Generator function local expansion: calculation of the coefficients

First of all a remark. The generator function local expansion of a function f consists of a vector of coefficients $\tilde{s}_\lambda$ associated with a set of indices $S(\partial\Lambda)$. On each level j such that $\exists\lambda \in S(\partial\Lambda) : |\lambda| = j$, the coefficients in the local expansion are a subset of the coefficients in the full scaling function expansion of f on level j. This means that the sum $\sum_{\lambda\in\tilde{s}_\lambda} \tilde{s}_\lambda\tilde{\phi}_\lambda$ is *not* a representation of f because several levels are present in the expansion.

As suggested in [26], the local polynomial approximation of f used in the calculation of the $\tilde{s}_\lambda$ is a Lagrange interpolation with polynomials of order $\tilde{d}$ on $\sigma_{j,\boldsymbol{k}}$, the support of ϕ_λ.

Simple case: in one dimension and disregarding boundary issues

In one dimension, f is approximated on $\sigma_{j,k}$, the support of $\phi_{j,k}$ by

$$P_{\tilde{d}}^{\sigma_{j,k}} f = \sum_{i=1}^{\tilde{d}} f(x^i) L_i,$$

where the x^i denote $\tilde{d}$ points equally spaced on $\sigma_{j,k}^*$ and the L_i are the interpolating Lagrange polynomials

$$L_i(x) = \prod_{l=1}^{\tilde{d}} \frac{x - x^i}{x^l - x^i}.$$

The notation $\sigma_{j,k}^*$ stands for an interval which is possibly a little larger than $\sigma_{j,k}$ and whose usefulness is explained further below. Ignoring for now the problems associated with the boundaries of the domain Ω, the nodes x^i on interval $\sigma_{j,k}$ are obtained from the nodes y^i on interval $\sigma_{0,0}$ by the simple transform $x^i = 2^{-j}(y^i + k)$. As a consequence, the Lagrange polynomials $L_i^{\sigma_{j,k}}$ defined on $\sigma_{j,k}$ can be expressed as

$$\begin{aligned} L_i^{\sigma_{j,k}}(x) &= \prod_{l=1}^{\tilde{d}} \frac{x - x^i}{x^l - x^i} = \prod_{l=1}^{\tilde{d}} \frac{x - 2^{-j}(y^i + k)}{2^{-j}(y^l + k - y^i - k)} \\ &= \prod_{l=1}^{\tilde{d}} \frac{2^j x - k - y^i}{y^l - y^i} = L_i^{\sigma_{0,0}}(2^j x - k) \end{aligned}$$

The generator function coefficients $\tilde{s}_{j,k}$ are calculated as

$$\tilde{s}_{j,k} = \langle P_{\tilde{d}}^{\sigma_{j,k}} f, \phi_{j,k} \rangle = \sum_{i=1}^{\tilde{d}} f(x^i) \langle L_i^{\sigma_{j,k}}, \phi_{j,k} \rangle = \sum_{i=1}^{\tilde{d}} f(x^i) \eta_{j,k}(i)$$

where $\eta_{j,k}(i)$ denotes the inner product $\langle L_i^{\sigma_{j,k}}, \phi_{j,k}\rangle$. Denoting $\langle L_i^{\sigma_{0,0}}, \phi\rangle$ by $\eta(i)$, it is apparent that

$$\begin{aligned}
\eta_{j,k}(i) &= \int_{\mathbb{R}} L_i^{\sigma_{j,k}}(x)\phi_{j,k}(x)\, dx \\
&= \int_{\mathbb{R}} L_i^{\sigma_{0,0}}(2^j x - k)2^{j/2}\phi_(2^j x - k)\, dx \\
&= 2^{-j/2}\int_{\mathbb{R}} L_i^{\sigma_{0,0}}(y)\phi_(y)\, dy, \quad y = 2^j x - k, \quad dx = 2^{-j}dy \\
&= 2^{-j/2}\eta(i)
\end{aligned}$$

This shows that the $\tilde{d}$ coefficients $\eta(i)$ can be calculated once and used again from memory to calculate every $\tilde{s}_{j,k}$. Assuming that the necessary values of function f are available at no cost, the calculation of any coefficient $\tilde{s}_{j,k}$ requires $\tilde{d}$ *flops* (floating point operations).

The multivariate case

In $(R)^n$, we have generator functions $\phi_\lambda = \phi_{j,\boldsymbol{k}} = \prod_{i=1}^n \xi_{j,k_i}$, with supports $\sigma_{j,\boldsymbol{k}} = \sigma_{j,k_1} \times \cdots \times \sigma_{j,k_n}$. The ideal coefficient $\tilde{s}_{j,k}$ would be calculated as

$$\begin{aligned}
\tilde{s}_{j,k} &\simeq \int_{\boldsymbol{R}^n} \xi_{j,k_1}\ldots\xi_{j,k_n} f(x_1,\ldots,x_n)dx_1\ldots dx_n \\
&\simeq \int_{\boldsymbol{R}} \xi_{j,k_1}\int_{\boldsymbol{R}}\ldots\int_{\boldsymbol{R}} \xi_{j,k_n} f(x_1,\ldots,x_n)dx_n\ldots dx_1
\end{aligned}$$

In the inner-most integral function f can be considered a function of variable x_n alone. This type of integrals has been replaced in the 1D case by the sum $\sum_{i_n=1}^{\tilde{d}} 2^{-j/2} f(x_1,\ldots,x_n^{i_n})\eta(i_n)$.

$$\tilde{s}_{j,k} \simeq \sum_{i_n=1}^{\tilde{d}} 2^{-j/2}\int_{\boldsymbol{R}} \xi_{j,k_1}\int_{\boldsymbol{R}}\ldots\int_{\boldsymbol{R}} \xi_{j,k_{n-1}} f(x_1,\ldots,x_{n-1},x_n^{i_n})dx_{n-1}\ldots dx_1$$

Repeating this process n times yields the coefficient $\tilde{s}_{j,k}$ corresponding to the 1D case as

$$\tilde{s}_{j,k} = 2^{-jn/2} \sum_{i_1=1}^{\tilde{d}} \cdots \sum_{i_n=1}^{\tilde{d}} f(x_1^{i_1}, \ldots, x_n^{i_n}) \eta(i_1) \ldots \eta(i_n)$$

This shows that the coefficients $\tilde{s}_{j,k}$ can be simply calculated from the point values of f and the $\tilde{d}$ 1D-integrals $\eta(i)$, whatever the dimension of the domain. Once the $\eta(i)$ have been stored in memory, the cost of calculating any $\tilde{s}_{j,k}$ is $\tilde{d}^n$ *flops* .

Boundary adapted scaling functions

Near the boundary of the domain, the primal scaling functions have been adapted to satisfy an homogeneous boundary condition while retaining the basis' approximation order d. These modifications mean that the boundary adapted functions are no longer the straight forward translates and dilates of the other scaling functions in the basis. They are however a linear combination of the 'normal' scaling functions, truncated at the boundary.

In the one dimensional case, let $\tilde{\Delta}_j^L$ denote the set of indices k such that $\phi_{j,k}$ is a scaling function adapted to the boundary on the left of the interval. It also denotes the set of indices k such that $\tilde{\phi}_{j,k}$ is a dual scaling function adapted to the boundary on the left of the interval. Similarly, let us denote by $\tilde{\Delta}_j^R$ the corresponding set for primal and dual scaling functions adapted to the right boundary of the interval. A coarsest level j_0 is chosen such that for $j \geq j_0$, no $\phi_{j,k}$ or $\tilde{\phi}_{j,k}$ is adapted both on the right and on the left ends of the interval. For $j \geq j_0$, the sets $\tilde{\Delta}_j^L$ and $\tilde{\Delta}_j^R$ are disjoint, and all sets $\tilde{\Delta}_j^L$ have the same cardinality. Similarly, the $\tilde{\Delta}_j^R$ have all the same cardinality for $j \geq j_0$. The boundary adapted scaling functions are denoted $\phi_{j,k}^L$ and $\phi_{j,k}^R$ and they are expressed in terms of

'normal' scaling functions as:

$$\begin{aligned}\phi^L_{j,k} &= \sum_{r=l^L_{k_l}}^{l^L_{k_u}} \alpha^L_{r,k}\phi_{j,r}|_{[0,\infty)} \\ \phi^R_{j,k} &= \sum_{r=l^R_{k_l}(j)}^{l^R_{k_u}(j)} \alpha^R_{r,k}\phi_{j,r}|_{(-\infty,1]}\end{aligned}$$

These expressions are made more precise in the chapter 2, and also in the section dedicated to the implementation of the generator function local expansion, Section 4.3.3. What really matters here is that the $\alpha^L_{r,k}$ and $\alpha^R_{r,k}$ are not dependent on the level j.

$$\begin{aligned}\forall k \in \tilde{\Delta}^L_j, \quad \tilde{s}_{j,k} &= \langle P^{\sigma_{j,k}}_{\tilde{d}} f, \phi^L_{j,k}\rangle \\ &= \sum_r \alpha^L_{r,k}\langle P^{\sigma_{j,k}}_{\tilde{d}} f, \phi_{j,r}\chi_{[0,\infty)}\rangle \\ &= \sum_r \alpha^L_{r,k}\sum_{i=1}^{\tilde{d}} f(x^i)\langle L_i^{\sigma_{j,k}}, \phi_{j,r}\chi_{[0,\infty)}\rangle\end{aligned}$$

where the x^i are once again regularly spaced nodes and where

$$\chi_X(x) = \begin{cases} 1 & \text{if } x \in X \\ 0 & \text{otherwise.}\end{cases}$$

Given a certain index k, the nodes x^i associated with support $\sigma_{j,k}$ are related to the nodes y^i associated with support $\tilde{\sigma}_{j_0,k}$ as $x^i = 2^{j_0-j}y^i$. Just like in the previous section, it is immediate that

$$\forall k \in \tilde{\Delta}^L_j, \quad L_i^{\sigma_{j,k}}(x) = L_i^{\sigma_{j_0,k}}(2^{j-j_0}x).$$

As a consequence, the inner product $\langle P_{\tilde{d}}^{\sigma_{j,k}} f, \phi_{j,k}^L \rangle$ can be made independent of the level j, except for a multiplicative coefficient:

$$\begin{aligned}
&\forall k \in \tilde{\Delta}_j^L, \\
&\langle L_i^{\sigma_{j,k}}, \phi_{j,r}\chi_{[0,\infty)} \rangle &= \int_{\mathbb{R}} L_i^{\sigma_{j,k}}(x)\phi_{j,r}(x)\chi_{[0,\infty)}(x)dx \\
&&= 2^{(j-j_0)/2} \int_{\mathbb{R}} L_i^{\sigma_{j_0,k}}(2^{j-j_0}x)\phi_{j_0,r}(2^{j-j_0}x)\chi_{[0,\infty)}(x)dx \\
&&= 2^{(j_0-j)/2} \int_{\mathbb{R}} L_i^{\sigma_{j_0,k}}(y)\phi_{j_0,r}(y)\chi_{[0,\infty)}(y)dy, \quad y = 2^{j-j_0}x \\
&&= 2^{(j_0-j)/2}\eta_{j_0,r}^L(i),
\end{aligned}$$

where $\eta_{j_0,r}^L(i)$ is defined as $\int_{\mathbb{R}} L_i^{\sigma_{j_0,k}}\phi_{j_0,r}\chi_{[0,\infty)}$. Moreover, denoting by $E_{j_0}^L(k,i)$ the sums $\sum_r \alpha_{r,k}^L \eta_{j_0,r}^L(i)$, the calculation of the coefficients $\tilde{s}_{j,k}$ may be simplified to the sum

$$\tilde{s}_{j,k} = 2^{(j_0-j)/2} \sum_{i=1}^{\tilde{d}} f(x^i) E_{j_0}^L(k,i)$$

which can be done in $\tilde{d}$ *flops* once the $E_{j_0}^L(k,i)$, $k \in \tilde{\Delta}_{j,k}^L$, have been stored in memory.

The calculation of the coefficients corresponding to the right boundary adapted scaling functions can be done in exactly the same way as the calculation of the coefficients near the left boundary.

To summarize, whatever the dimension n of the domain, the calculation of any coefficient $\tilde{s}_{j,k}$ can be done in $\tilde{d}^n$ *flops*. The pre-processing needed involves the calculation of a vector of length $\tilde{d}$ containing the values of the $\eta(i)$, a matrix of dimensions $\tilde{d} \times \#\tilde{\Delta}_{j_0}^L$ containing the values of the $E_{j_0}^L(k,i)$ and a matrix of dimensions $\tilde{d} \times \#\tilde{\Delta}_{j_0}^R$ containing

the values of the $E^R_{j_0}(k,i)$. Details on these calculations are given in Section 4.3. The amount of work needed to calculate all the coefficients $\tilde{s}_{j,k}$, $(j,\boldsymbol{k}) \in S(\partial\Lambda)$ is bounded by $max(N_g, N_w)(\#\partial\Lambda/(2^n-1))\tilde{d}^n$ *flops* and so it is proportional to $\#\partial\Lambda$.

Chebyshev or regularly spaced nodes

Regularly spaced nodes are preferred to irregularly spaced nodes such as Chebyshev, because they facilitate the re-usability of the inner products $\eta(i)$.

The quality of the approximation of a function f by a Lagrange interpolation is estimated in [19], p.29, as

$$\|f - p_n\| \leq \frac{1}{(n+1)!}\|\pi_{n+1}\|\|f^{(n+1)}\|$$

where p_n denotes a Lagrange interpolation of f of degree n associated with arbitrary nodes, $\|.\|$ denotes the L^∞ norm and $\pi_{n+1}(x) = \prod_{j=0}^{n}(x - x_j)$. Further, [19] gives the following estimates:

- For evenly spaced points, $\|\pi_{n+1}\| = O((b-a)/e)^{n+1}$ when $n \longrightarrow \infty$, where $[a,b]$ is the interval of interpolation and $e = exp(1)$.
- For Chebyshev nodes, $\|\pi_{n+1}\| = 2((b-a)/4)^{n+1}$.

For the range of values of n that are likely to be used in this application ($3 \leq n \leq 6$), the quotient $(e/4)^{n+1}$ decreases from about 0.22 down to about 0.067. This does not seem enough to justify the amount of supplementary work associated with choosing Chebyshev nodes over regular ones.

4.2.6 The local wavelet transform

The local wavelet transform is the indispensable step that takes the redundant, multi-level, scaling function expansion $(\tilde{\sigma}_\lambda, \lambda \in S(\partial\Lambda))$ of a function f to its sparse wavelet expansion $(\tilde{d}_\lambda, \lambda \in \Lambda)$, where $\tilde{d}_\lambda = \int f \psi_\lambda$. The local wavelet transform makes the most of the tree structure of the set of indices Λ. Let $\mathcal{G}$ denote the set of cubes $\square_\lambda$ such that $\lambda \in \partial\Lambda$. The indices in $\mathcal{G}$ are assumed to be sorted by increasing level $|\lambda|$. Let J denote the highest level present in Λ and let j_0 denote the coarsest possible level. A simple algorithm for the local wavelet transform is the following.

For all $\square_\lambda \in \mathcal{G}$, starting with the last cube and proceeding towards the first,

- If $|\lambda| < J$, remove from $S(\partial\Lambda)$ all the indices of level $J+1$ and remove from $\mathcal{G}$ the cubes of level J. Reset J to $|\lambda|$.
- Make S_λ, the subset of $S(\partial\Lambda)$ which contains the indices of the scaling function coefficients $\tilde{s}_{J+1,k}$ which are needed for the calculation of the 2^n-1 wavelet coefficients and one scaling function coefficient associated with cube $\square_\lambda$.
- Do a one-level transform on the coefficients $\tilde{\sigma}_{\lambda'}$, $\lambda' \in S_\lambda$. The new $2^n - 1$ wavelet coefficients are stored as part of the wavelet expansion of f. The new scaling function coefficient is also stored and its index λ is inserted in the set $S(\partial\Lambda)$.
- If $|\lambda| \neq j_0$, the parent cube of $\square_\lambda$ is inserted in $\mathcal{G}$.

In the one dimensional case, the one-level transform consists of applying once the refinement and the wavelet equations to a set of coefficients. Because the scaling function and wavelet bases are defined on an interval, these equations depend on the level j involved and they

are best written in a matrix form. Let $M_{j,0}$ denote the refinement equation matrix and let $M_{j,1}$ denote the wavelet equation matrix on level j. Let also Φ_j and Φ_{j+1} denote the vectors of scaling functions on levels j and $j+1$. Similarly, let Ψ_j denote the vector of wavelets on level j. The refinement and wavelet equations in matrix form are:

$$\Phi_j = M_{j,0}^T \Phi_{j+1} \qquad \text{and} \qquad \Psi_j = M_{j,1}^T \Phi_{j+1}$$

A dual scaling function coefficient $\tilde{s}_{j,k} = \int f \, \phi_{j,k}$ is obtained from a linear combination of dual scaling function coefficients on level $j+1$ by:

$$\tilde{s}_{j,k} = \sum_{l=l_{l,0}(k)}^{l_{u,0}(k)} [M_{j,0}]_{l,k} \, \tilde{s}_{j+1,k},$$

where $l_{l,0}(k)$ denotes the lowest row index of a non-zero entry in column k of matrix $M_{j,0}$ and $l_{u,0}(k)$ denotes the highest (following the Multilevel Library's notations, the subscript l stands for 'low' and u stands for 'up'). When $\phi_{j,k}$ is not an adapted boundary condition, $l_{l,0}(k) = k+l_1$ and $l_{u,0}(k) = k+l_2$, which means that exactly $l_2 - l_1 + 1 = d+1$ scaling functions on level $j+1$ are needed. When $\phi_{j,k}$ is a left or a right boundary adapted function, the number of scaling function coefficients that are needed on level $j+1$ is equal to the length of the top left (or bottom right) block of matrix $M_{j,0}$. This length is independent of the level j.

Similarly, a wavelet coefficient $\tilde{d}_{j,k}$ is obtained as:

$$\tilde{d}_{j,k} = \sum_{l=l_{l,1}(k)}^{l_{u,1}(k)} [M_{j,1}]_{l,k} \, \tilde{s}_{j+1,k},$$

where $l_{l,1}(k)$ and $l_{u,1}(k)$ correspond to the column k of $M_{j,1}$ and the difference $l_{u,1}(k)-l_{l,1}(k)$ is also independent of the level j.

In n dimensions, the dual scaling function and wavelet coefficients are defined as

$$\tilde{s}_\lambda = \int_\Omega f \prod_{i=1}^{n} \xi_{j,k_i} \qquad \text{and} \qquad \tilde{d}_\lambda = \int_\Omega f \prod_{i=1}^{n} \zeta_{j,k_i},$$

where $\zeta_{j,k_i} = \xi_{j,k_i}$ if $e_i = 0$ and $\zeta_{j,k_i} = \eta_{j,k_i}$ if $e_i = 1$. The scaling function coefficients on level $j+1$ necessary for the calculation of coefficient $\tilde{s}_\lambda$ form the set:

$$\big\{\tilde{s}_{\lambda'} : |\lambda'| = j+1 \text{ and } k'_i \in \{l_{l,0}(k_i), \ldots, l_{u,0}(k_i)\}, \text{ for } i = 1, n\big\}.$$

Correspondingly, the set S_λ of the indices of coefficients $\tilde{s}_{\lambda'}$ on level $j+1$ necessary for the calculation of all the dual coefficients associated with the cube $\square_\lambda$ is the set:

$$S_\lambda = \Big\{(j+1, \boldsymbol{k'}) : k'_i \in \big\{\min\big(l_{l,0}(k_i), l_{l,1}(k_i)\big), \ldots, \max\big(l_{u,0}(k_i), l_{u,1}(k_i)\big)\big\}, \text{ for } i = 1, n\Big\}$$

and the one-level decomposition in n dimensions consists of the linear combination

$$\tilde{d}_\lambda = \sum_{l_1=l_{l,e_1}(k_1)}^{l_{u,e_1}(k_1)} [M_{j,e_1}]_{l_1,k_1} \cdots \sum_{l_n=l_{l,e_n}(k_n)}^{l_{u,e_n}(k_n)} [M_{j,e_n}]_{l_n,k_n}\ \tilde{s}_{j+1,\boldsymbol{l}}.$$

Assuming that the set $S(\lambda)$ contains a maximum of N^n indices and that there are a maximum of N non-zero entries in the columns of the $M_{j,0}$ and $M_{j,1}$ matrices, the sum above can be calculated in $(N^n + \cdots + N) = (N^{n+1} - N)/(N-1)$ *flops*. Since 2^n coefficients must be calculated for each cube $\square_\lambda$, the number of floating point operations needed per cube is

$2^n(N^n + \cdots + N)$ maximum. This number depends on the length of the primal filters and on the space dimension only. So the cost of the one-level decompositions performed during a local transform is proportional to number of cubes in Λ and therefore it is proportional to $\#\Lambda$. This cost can be reduced to $2N^n + \cdots + 2^n N$ per cube by calculating simultaneously the 2^n coefficients associated with one cube. The one-level decomposition is the main step in the algorithm above, but the operations of locating, inserting and deleting coefficients in the memory also incur a cost, which is the penalty for not using a matrix-type method of storage. This trade-off between the size of memory space occupied versus the effort required to handle the object stored is one of the main drawbacks of the sparse algorithms.

4.3 Implementation

The implementation is very close to the method described in the previous section. The calculation of the sparse dual wavelet expansion of a function f follows the following steps.

- Determine the set of wavelet indices that will be present in the expansion (Λ) according to a precision criterion δ (or an ideal number of wavelet coefficients N) and an error function E. This operation is encoded in a function called *FindGoodCubes*, the output of which is the set of cubes $\square_\lambda$ associated with the leaves $\lambda \in \partial\Lambda$, which uniquely determines a tree of wavelet indices Λ.

- Grade $\partial\Lambda$ so that the tree Λ is 2-graded. This is encoded in function called *GradeTree*.

- Determine the set of scaling function indices ($S(\partial\Lambda)$) necessary to obtain the tree Λ by local (dual) wavelet transform. This set is called the generator function local expansion set and the function that calculates it called *GenLocSet*.

- Calculate the local (dual) generator function (or scaling function) local expansion of f by calculating the inner products $\langle f, \phi_\lambda \rangle, \lambda \in S(\partial\Lambda)$. This task is performed by function *GenLocRHS*.

- Finally, the scaling function coefficients are transformed into wavelet coefficients by a local dual wavelet transform implemented as function *LocTransformT*.

This algorithm can be summarized as

1. $G \longleftarrow$ *FindGoodCubes* $--$ δ, N, E
2. $S \longleftarrow$ *GenLocSet* $--$ G
3. $S \longleftarrow$ *GradeTree* $--$ $S, 2$
4. $S \longleftarrow$ *GenLocRHS* $--$ S, f
5. $D \longleftarrow$ *LocTransformT* $--$ G, S

where G contains the set of *good* cubes, or set of indices $\partial\Lambda$, S encodes the set of generator function indices and coefficients and D encodes the set of wavelet indices and coefficients.

4.3.1 Implementation of the BAS algorithm in the *FindGoodCubes* function

The *FindGoodCubes* function implements the BAS algorithm (section 4.2.2) except for a few practical modifications.

The first modification concerns the initialization of the algorithm. In BAS, the starting point of the algorithm is a set $\mathcal{B}$ which contains one single element: the unit cube. In

FindGoodCubes, the set $\mathcal{B}$ can consist of any uniform dyadic refinement of the unit cube; the level of that refinement must be set in input. The intention behind this change was to prevent the set of leaves $\partial\Lambda$ from containing indices λ such that $|\lambda| < j_0 - 1$, where j_0 denotes the coarsest level of the wavelet basis used. Any possible leaf of level $j_0 - 1$ will correspond to a scaling function on level j_0.

The second modification concerns the criterion for stopping the scheme. BAS stops only when $E(\square_\lambda) \leq \delta$ for all cubes $\square_\lambda$ in the output set of good cubes $\mathcal{G}$, or $\partial\Lambda$. In *FindGoodCubes*, the scheme stops when one of three conditions is fulfilled: $E(\square_\lambda) \leq \delta$ for all $\square_\lambda \in \mathcal{G}$, or the number of elements in the union of $\mathcal{G}$ and $\mathcal{B}$ exceeds a predefined integer N, or the level j of the indices λ has reached 29, the maximum level for which the Multilevel Library works being 30. If after checking a cube $\square_\lambda \in \mathcal{B}$, the second or third condition is fulfilled, the scheme does not stop straight away, but carries on until all cubes $\square_{\lambda'} \in \mathcal{B}$ satisfying $|\lambda| = |\lambda'|$ have also been checked. Then, the scheme is stopped by the operation

$$\mathcal{B} \cup \mathcal{G} \longrightarrow \mathcal{G}$$

This is to avoid creating artifacts in the distribution of the wavelet coefficients, for instance a lack of symmetry caused by the sudden discontinuation of the scheme.

Finally, the checks $E(\square_\lambda) \leq \delta$ in BAS are replaced by $E(\square_\lambda^*) \leq \delta$ in *FindGoodCubes*. Here, $\square_\lambda^*$ denotes the union of the supports of the wavelets on level $|\lambda|$ which intersect the cube $\square_\lambda$. The cubes $\square_\lambda$ represent the size and location of the supports of different wavelets but the objects of interest are really the supports themselves.

The function header is:

```
int FindGoodCubes( set<MDIndex,less<MDIndex> > &tree,
                   set<MDIndex,less<MDIndex> > &tree-comp,
                   int level, int dim, double eps, int N,
                   const I_Basis_Bspline &basis,
                   double (*E)(RectBorder ) );
```

where 'tree' is the set of good cubes $\square_\lambda$, $\lambda \in \partial\Lambda$, and 'tree-comp' is the complementary set of cubes $\square_\lambda$, $\lambda \in \Lambda \setminus \partial\Lambda$.

Choice of an error function E

The choice of the error function E is of course problem dependent. In the problem at hand, the right hand side coefficients are the sum of inner products calculated over the inside of the magnetic domain Ω_{int} and of inner products calculated along the boundary $\partial\Omega_{int}$. In both types of inner products, the borders of the domain of integration introduce a discontinuity in f, which is expected to be smooth elsewhere. For the right hand side used in this thesis, the chosen error function E is

$$E(\square_\lambda) = \begin{cases} \operatorname{diam}(\tilde{\sigma}_{|\lambda|,\boldsymbol{k}(\lambda)}) & \text{if } \tilde{\sigma}_{|\lambda|,\boldsymbol{k}(\lambda)} \cap \partial\Omega_{int} \neq \emptyset, \\ 0 & \text{otherwise,} \end{cases}$$

where $\tilde{\sigma}_{|\lambda|,\boldsymbol{k}}$ denotes the support of $\tilde{\phi}_\lambda$ and λ is a leaf of Λ. The set Λ is reset at the start of each time-step to the set of wavelet indices used to represent the solution of the Poisson problem at the previous time step.

4.3.2 Calculation of the generator function indices set with the *GenLocSet* function

The *GenLocSet* function improves a little on the algorithm given in Section 4.2.4. The starting point is a partitioning of the unit cube into a disjoint set of *good* cubes $\mathcal{G}$, which is the output of the BAS algorithm. For each cube $\square_{\lambda'} \in \mathcal{G}$, and for each spatial dimension i, the lowest index l_i needed for a wavelet transform is only looked for if the cube's side $x_i = 2^{-|\lambda'|}\boldsymbol{k}_i(\lambda')$ borders a coarser cube $\square_{\lambda''} \in \mathcal{G}$, $|\lambda''| > |\lambda'|$. Otherwise, l_i is simply set to $2\boldsymbol{k}_i(\lambda')$. Similarly, the highest index m_i needed for a wavelet transform is only looked for if the cube's side $x_i = 2^{-|\lambda'|}(\boldsymbol{k}_i(\lambda') + 1)$ borders a coarser cube. Otherwise, l_i is simply set to $2\boldsymbol{k}_i(\lambda') + 1$.

Additional flexibility is given to the *GenLocSet* function by the introduction of three optional input parameters. Two of these parameters fix the values of the differences $L = 2k_i - l_i$ and $M = m_i - (2k_i + 1)$ arbitrarily and the third parameter causes *GenLocSet* to work with the dual basis instead of the primal basis. This flexibility is used in the process of evaluating a sparse wavelet expansion pointwise.

For each cube in the input set of good cubes, the work done by this function consists of

- n searches for a neighbour coarser cube,

- a maximum of $(\max(L, M) + 2)^n$ insertions in $S(\partial\Lambda)$.

Using the STL containers 'set' for the set of good cubes and 'map' for the set of scaling function coefficients, the complexity of a search operation and of an insertion is logarithmic in terms of the number of entries in the set or map. So the work done by the *GenLocSet* function is of the order of $\#\partial\Lambda \log(\#\partial\Lambda)$ operations. This number is not directly proportional to number of leaves in Λ but it is still independent of the number of levels involved in Λ.

The function header is:

```
int GenLocSet( map<MDIndex,double,less<MDIndex> > &S,
               const set<MDIndex,less<MDIndex> > &G,
               const I_Basis_Bspline &basis,
               int minl1 = -1, int maxl2 = -1, int pd = 0 );
```

4.3.3 Calculation of the generator function coefficients with the *GenLocRHS* function

The implementation of the calculation of the generator function coefficients of a function f differs from the ideas presented in Section 4.2.5 on one major point. When $\phi_{j,k}$ is a boundary adapted function, its support is longer than otherwise. The maximum support for a 1D-function $\phi^L_{j,k}$, $k \in \tilde{\Delta}^L_j$, is $2^{-j}[0, l_2 + l^L_{k_u}]$ and the maximum support for a 1D-function $\phi^R_{j,k}$, $k \in \tilde{\Delta}^R_j$, is $2^{-j}[l_1 + l^R_{k_l}, 1]$. When using exactly $\tilde{d}$ interpolation nodes on the support of $\phi_{j,k}$, whatever its length, a loss of accuracy is introduced near the boundary by comparison to the inside of the domain. In order to conserve the same accuracy everywhere, at the one-dimensional stage, instead of calculating a Lagrange interpolation of f on each scaling

function's support $\sigma_{j,k}$, the method implemented calculates several Lagrange interpolations of f on intervals of length 2^{-j}. Those intervals are of the form $2^{-j}[l, l+1]$, where l varies between l_1 and $l_2 - 1$ when $\phi_{j,k}$ is an inner scaling function ($\sigma_{j,k} = 2^{-j}[k + l_1, k + l_2]$), between 0 and $l_2 + l_{k_u}^L - 1$ when $k \in \tilde{\Delta}_j^L$ and between $l_1 + l_{k_l}^R(j)$ and $2^j - 1$ when $k \in \tilde{\Delta}_j^R$. The scaling function coefficients are therefore calculated as follows.

- Away from the boundary,

$$\begin{aligned}
\tilde{s}_{j,k} &= 2^{-j/2} \sum_{l=l_1}^{l_2-1} \sum_{i=0}^{\tilde{d}-1} f(x_{l,i}) \langle L_i^{[l,l+1]},\ \phi\ \chi_{[l,l+1]} \rangle, \\
&\qquad\qquad x_{l,i} = 2^{-j}\big(k + l + i/(\tilde{d}-1)\big) \\
\tilde{s}_{j,k} &= 2^{-j/2} \sum_{\nu=0}^{(\tilde{d}-1)(l_2-l_1)} f(x_\nu)\eta(\nu), \\
&\qquad\qquad x_\nu = 2^{-j}\big(k + l_1 + \nu/(\tilde{d}-1)\big)
\end{aligned}$$

- Near the left boundary ($k \in \tilde{\Delta}_j^L$),

$$\begin{aligned}
\tilde{s}_{j,k} &= 2^{(j_0-j)/2} \sum_r \alpha_{r,k}^L \sum_{l=0}^{l_2+l_{k_u}^L-1} \\
&\qquad \sum_{i=0}^{\tilde{d}-1} f(x_{l,i}) \langle L_i^{2^{-j_0}[l,l+1]},\ \phi_{j_0,r}\ \chi_{2^{-j_0}[l,l+1]} \rangle, \\
&\qquad\qquad x_{l,i} = 2^{-j}\big(l + i/(\tilde{d}-1)\big) \\
\tilde{s}_{j,k} &= 2^{(j_0-j)/2} \sum_{\nu=0}^{(\tilde{d}-1)(l_2+l_{k_u}^L)} E_{j_0}^L(k,\nu) f\big(2^{-j}\nu/(\tilde{d}-1)\big)
\end{aligned}$$

- Near the right boundary ($k \in \tilde{\Delta}_j^R$),

$$\tilde{s}_{j,k} = 2^{(j_0-j)/2} \sum_\nu E_{j_0}^R(k,\nu) f\big(2^{-j}\big(\nu/(\tilde{d}-1) + 2^j\big)\big)$$

where the sum's index ν varies from $-(\tilde{d}-1)\Big(2^j - \big(l_1 + l^R_{k_l}(j)\big)\Big)$ to 0.

This modification multiplies the amount of work for each coefficient $\tilde{s}_{j,k}$ by the number of sub-intervals used in the support $\sigma_{j,k}$ in each spatial direction.

In addition to the general *GenLocRHS* function, a more specialized function has been implemented to cater for the physical problem at hand. This function is a member of the *Interface* class, which models the platelet. The *Interface::GenLocRHS* function calculates the surface integrals, the contour integrals and the divergence of the magnetic moment $\boldsymbol{M}$, where needed.

4.3.4 Pre-processing: calculation of vector η and of matrices $E^L_{j_0}$ and $E^R_{j_0}$ in function *etakr*

The calculation of $E^L_{j_0}$ and $E^L_{j_0}$ differs from that of η essentially in the extra summation involving the coefficients $\alpha^L_{k,r}$ or $\alpha^R_{k,r}$. The values of these coefficients are given by the Multilevel Library, also this section concerns itself only with the calculation of the inner products $\langle L_i^{[l,l+1]},\ \phi\ \chi_{[l,l+1]}\rangle$, which are the main components of $E^L_{j_0}$ and $E^L_{j_0}$ as well as η.

By writing the Lagrange polynomials in terms of monomials x^s as

$$L_i(x) = \prod_{l\neq i, l=0}^{\tilde{d}-1} \frac{x - x_l}{x_i - x_l} = \sum_{r=0}^{\tilde{d}-1} L(i,r)x^r,$$

it becomes apparent that the core of the calculation consists of inner products of the form $\int_{\mathbb{R}} x^r\phi(x)\chi_{[l,l+1]}(x)dx$ or rather of a difference of inner products of the form $\int_{\mathbb{R}} x^r\phi(x)\chi_{[l,\infty)}(x)dx$,

with $0 \leq r \leq \tilde{d} - 1$ and $l_1 \leq l \leq l_2 - 1$. These inner products will be denoted μ_r^l. It is useful to note that:

- for $l \geq l_2$, $\mu_r^l = \int_n^\infty x^r \phi(x) dx = 0$, since $\text{supp}(\phi) = [l_1, l_2]$;
- for $l \leq l_1$, $\mu_r^l = \mu_r^{l_1} = \int_{-\infty}^\infty x^r \phi(x) dx$;
- for $l_1 \leq l \leq l_2$, μ_0^l can be calculated by the Multilevel Library as a sum of inner products of refinable functions (ϕ and $\chi_{[0,1]}$):

$$\mu_0^l = \sum_l^\infty \phi(x) dx = \sum_l^{l_2} \phi(x) dx = \sum_{m=l}^{l_2-1} \int_{\mathbb{R}} \phi(x)\ \chi_{[0,1]}(x - m)\ dx$$

The remaining μ_r^l will be calculated by recursion (on r) from these given values.

A recursion on the μ_r^l

Following the method used in [13], the refinement equation $\left(\phi(x) = \sum_{k=l_1}^{l_2} a_k \phi(2x - k)\right)$ is inserted in the expression for μ_r^l.

For $l_1 \le l \le l_2$,

$$\begin{aligned}
\mu_r^l &= \int_l^\infty x^r \Big(\sum_{k=l_1}^{l_2} a_k \ \phi(2x-k) \Big) dx, \\
\mu_r^l &= \frac{1}{2} \sum_{k=l_1}^{l_2} a_k \int_{2l-k}^\infty [(y+k)/2]^r \ \phi(y) \ dy, \\
& \qquad\qquad \text{where } y = 2x-k, \ dx = \frac{1}{2} dy, \\
\mu_r^l &= 2^{-(r+1)} \sum_{k=l_1}^{l_2} a_k \int_{2l-k}^\infty \Big(\sum_{s=0}^{r} \binom{r}{s} y^s k^{r-s} \Big) \ \phi(y) \ dy, \\
\mu_r^l &= 2^{-(r+1)} \sum_{k=l_1}^{l_2} a_k \sum_{s=0}^{r} \binom{r}{s} \ k^{r-s} \ \mu_s^{2l-k}, \\
2^{r+1} \mu_r^l &= \sum_{k=l_1}^{l_2} a_k \sum_{s=0}^{r-1} \binom{r}{s} \ k^{r-s} \ \mu_s^{2l-k} \ + \ \sum_{k=l_1}^{l_2} a_k \ \mu_r^{2l-k}.
\end{aligned}$$

In the last equality, the first term of the right hand side consists of data which is already known at step r, and it is denoted by b_r^l. The second term can be rearranged to avoid any coefficients μ_r^l for l outside the range $(l_1, l_2 - 1)$. This is done as follows.

$$\begin{aligned}
\sum_{k=l_1}^{l_2} a_k \ \mu_r^{2l-k} &= \sum_{p=2l-l_2}^{2l-l_1} a_{-p+2l} \ \mu_r^p, \quad \text{where } p = 2l-k, \\
\sum_{k=l_1}^{l_2} a_k \ \mu_r^{2l-k} &= \sum_{p=2l-l_2}^{l_2-1} a_{-p+2l} \ \mu_r^p, \quad \text{because } \mu_r^p = 0 \text{ for } p \ge l_2, \\
\sum_{k=l_1}^{l_2} a_k \ \mu_r^{2l-k} &= \begin{cases} \sum_{p=2l-l_2}^{l_1} a_{-p+2l} \ \mu_r^{l_1} + \sum_{p=l_1+1}^{l_2-1} a_{-p+2l} \ \mu_r^p, & \text{if } 2l - l_2 \le l_1, \\ \sum_{p=l_1}^{l_2-1} a_{-p+2l} \ \mu_r^p, & \text{otherwise,} \end{cases} \\
& \qquad \text{because } \mu_r^p = \mu_r^{l_1} \text{ for } p \le l_1, \text{ and } a_k = 0 \text{ for } k < l_1.
\end{aligned}$$

The entire equation can now be re-written for $l_1 \leq l \leq l_2 - 1$ as a matrix equation. Let us define a $(l_2 - l_1)$ by $(l_2 - l_1)$ matrix A such that

$$\begin{cases} A_{l,p} = -a_{-p+2l}, & \text{for } l_1 \leq l, p \leq l_2 - 1, \\ & \text{except if } l < \lceil (l_1 + l_2)/2 \rceil, \\ A_{l,l_1} = -\sum_{p=2l-l_2}^{l_1} a_{-p+2l}, & \text{in that case.} \end{cases}$$

Moreover, let $\boldsymbol{b}_r$ denote the vector $(b_r^{l_1}, \ldots, b_r^{l_2-1})$ and let $\boldsymbol{\mu}_r$ denote the vector $(\mu_r^{l_1}, \ldots, \mu_r^{l_2-1})$. Finally, let D_r denote the $(l_2 - l_1)$ by $(l_2 - l_1)$ diagonal matrix $2^{r+1}I$. The following matrix equation gives the coefficients μ_r^l in terms of coefficients $\mu_{r'}^{l'}$, where $r' < r$,

$$(A + D_r)\boldsymbol{\mu}_r = \boldsymbol{b}_r.$$

After the vector $\boldsymbol{\mu}_0$ has been calculated using the Multilevel Library, the remaining vectors $\boldsymbol{\mu}_r$ are recursively obtained by inverting the matrix equation above. The proof of the non-singularity of matrices $A + D_r$ is given in an appendix.

Calculation of the $L_{i,r}$

The Lagrange polynomials L_i associated with the $\tilde{d}$ nodes x_l, for $0 \leq l \leq \tilde{d} - 1$, are such that $L_i(x_l) = \delta_{i,l}$ and so

$$\sum_{r=0}^{\tilde{d}-1} L_{i,r} x_l^r = \delta_{i,l}, \quad \text{for } 0 \leq i \leq \tilde{d} - 1.$$

The $L_{i,r}$ form a matrix L which is the inverse of a Vandermonde matrix V, with entries $V_{i,r} = x_1^r$, for $0 \leq i, r \leq \tilde{d} - 1$.

Function call

The *etakr* function is called only once in the program and it has the following header:

```
void etakr( matrix &EtaL_j0, vector &Eta_j0, matrix &EtaR_j0,
            const I_Basis_Bspline &basis );
```

4.3.5 Specialization of *GenLocRHS* for the interface condition

The specialized function used to calculate the right hand side coefficients for the physical problem must calculate two types of integrals $\int_{\Omega_{int}} \text{div}\, \boldsymbol{M}\, \phi_\lambda$ and $\oint_{\partial\Omega_{int}} \boldsymbol{M} \cdot \boldsymbol{n}\, \phi_\lambda$. The issues attached to the calculation of the surface integral are:

1. how to calculate the divergence of $\boldsymbol{M}$ from a set of dyadic point values,
2. the integral is restricted to the intersection of the support of ϕ_λ with the platelet Ω_{int}.

As for the contour integral, because the multi-dimensional wavelets have a tensor product structure and because the directions of integration are parallel to the axes, they are calculated like the surface integrals, on a domain of dimension $n-1$.

Calculation of the divergence

The divergence of the magnetic moment is calculated as the divergence of a vector of local polynomial interpolations. Each component of this vector is the tensor product of n one-dimensional Lagrange interpolating polynomials.

The degree of the one-dimensional interpolating polynomials is chosen to be $\tilde{d}+1$, so that it is one more than that of the interpolation used to calculate the inner products with ϕ_λ.

The interpolating nodes are regularly spaced on a dyadic grid and they are all chosen inside the platelet. The motivation for this is that $\boldsymbol{M}$ is expected to present a sharp variation or even a discontinuity at the border between the magnetic and the non-magnetic regions. An interpolating polynomial constructed from nodes on either side of that boundary would certainly show artificial overshoots near $\partial\Omega_{int}$, unnecessarily changing the values of div $\boldsymbol{M}$ to non-representative values. The down side of this choice is that it fixes a coarsest scaling function level (j_p) around the platelet, such that in each spatial direction, the platelet is wide enough to contain $\tilde{d}+1$ dyadic nodes on level j_p. This restriction may not be unreasonable since it means that the supports of dual scaling functions $\tilde{\phi}_{j_p,\boldsymbol{k}}$ are not wider than the platelet.

Using a notation similar to that of section 4.2.5, this can be summarized by

$$\text{div } \boldsymbol{M} \approx \text{ div } \boldsymbol{P}_{\tilde{d}+1}^{\sigma_{j,\boldsymbol{k}}} \boldsymbol{M},$$

where

$$\boldsymbol{P}_{\tilde{d}+1}^{\sigma_{j,\boldsymbol{k}}} \boldsymbol{M} = (P_{\tilde{d}+1}^{\sigma_{j,\boldsymbol{k}}} M_1, \ldots, P_{\tilde{d}+1}^{\sigma_{j,\boldsymbol{k}}} M_n).$$

and

$$P_{\tilde{d}+1}^{\sigma_{j,\boldsymbol{k}}} M_i(\boldsymbol{y}) = \sum_{\boldsymbol{x}^{\boldsymbol{l}}} M_i(\boldsymbol{x}^{\boldsymbol{l}}) \prod_{m=1}^{n} L_{l_m}^{\sigma_{j,k_m}}(y_m)$$

where $\boldsymbol{x} = (x^{l_1}, \ldots, x^{l_n})$ is a point on a dyadic grid of level j and of size $(\tilde{d}+1)^n$.

By a process of re-scaling and shifting the grid, the divergence of $\boldsymbol{M}$ can be approximately calculated from the first partial derivatives of the interpolating polynomials associated with the grid $(0, ..., \tilde{d})^n$.

$$\text{div } \boldsymbol{M}(\boldsymbol{y}) \approx 2^j \sum_{i=1}^{n} \sum_{\boldsymbol{x}^l} M_i(\boldsymbol{x}^l) \frac{d}{dx_i} L_{l_i}(y_i) \prod_{m \neq i} L_{l_m}(y_m)$$

where $\boldsymbol{x}$ is now a point on a dyadic grid of level 0.

A pre-processing step evaluates the products

$$\frac{d}{dx_1} L_{l_1}(y_1) \prod_{m \neq 1} L_{l_m}(y_m)$$

at all points $\boldsymbol{y}$ on a grid $(I_0 - dt, I_0 + 2dt)^n$, where $I_0 = -\lfloor (dt+1)/2 \rfloor$ (+1 if $\tilde{d}$ is odd) and where adjacent nodes on the grid are at a distance of $1/(\tilde{d}-1)$ from each other. This grid of values has nearly the same centre node as the grid $(0, ..., \tilde{d})^n$ of interpolating nodes and it contains all the points necessary for the calculation of the inner products of div $\boldsymbol{M}$ with the scaling functions by *GenLocRHS* as previously described.

After the pre-processing step, the divergence of $\boldsymbol{M}$ is easily obtained anywhere it is needed by matrix multiplications involving the values of the M_i at node points and the pre-calculated partial derivatives of the interpolating polynomials.

Integral over Ω_{int}

The calculation of the integral of div $\boldsymbol{M}$ with a scaling function ϕ_λ on the domain Ω_{int} is done basically in the same way as the calculation of the integral over the whole of Ω in the sense that it consists of calculating the inner product of the vector η (or of a row of one of the matrices $E_{j_0}^L$ or $E_{j_0}^R$) with a vector of point values of div $\boldsymbol{M}$. However, if the intersection of the support of ϕ_λ with Ω_{int} is empty, no calculation is made as the inner product is clearly null. What is more, if the support of ϕ_λ is not completely included inside Ω_{int}, only a part of the vector η (or of a row of one of the matrices $E_{j_0}^L$ or $E_{j_0}^R$) is used. That part is what corresponds to the piecewise interpolation of div $\boldsymbol{M}$ on the dyadic cubes of level $|\lambda|$ which are strictly inside Ω_{int}.

The header for the *interface::GenLocRHS* function is as follows. The values of $\boldsymbol{M}$ can be given as a set of point values or as a function.

```
void Interface::GenLocRHS( map< MDIndex,double,less<MDIndex> > &S,
                           const I_Basis_Bspline &basis,
                           const PointValues &M );

void Interface::GenLocRHS( map< MDIndex,double,less<MDIndex> > &S,
                      const I_Basis_Bspline &basis,
                      vector (*M)(const vector &x, Interface *pt) );
```

4.3.6 Implementation of the local wavelet transform in the *LocTransformT* function

The implementation of the local wavelet transform closely follows the algorithm from section 4.2.6. The implementation of the one-level decomposition is based on two recursive functions *OneLocDecompos0T* and *OneLocDecompos1T*. In the one-level decomposition which calculates the coefficients associated with a cube $\square_\lambda$, each of of these functions is given to start with the same set of scaling function coefficients in input, for instance N^n coefficients. The *OneLocDecompos0T* function applies to these coefficient the column k_1 of matrix $M_{j,0}$

$$\sum_{l_1=l_{l,0}(k_1)}^{l_{u,0}(k_1)} [M_{j,0}]_{l_1,k_1}\ \tilde{s}_{j+1,\boldsymbol{l}}$$

with for result a set of N^{n-1} coefficients $s^{(0)}_{j+1,(l_2,\ldots,l_n)}$. Independently, the *OneLocDecompos1T* function applies to these coefficient the column k_1 of matrix $M_{j,1}$

$$\sum_{l_1=l_{l,1}(k_1)}^{l_{u,1}(k_1)} [M_{j,1}]_{l_1,k_1}\ \tilde{s}_{j+1,\boldsymbol{l}}$$

with for result a set of N^{n-1} coefficients $s^{(1)}_{j+1,(l_2,\ldots,l_n)}$. The *OneLocDecompos0T* and *OneLocDecompos1T* functions terminate by calling both functions with for input sets the set of coefficients that has just been calculated. The iterations stop when the output sets contain only one coefficient. This coefficient is one of the 2^n coefficients associated with $\square_\lambda$. Figure 4.3.6 may help visualize the process.

It was mentioned in section 4.2.6 that the variables' storage handling can be costly when they are not stored in matrix-type structures. The implementation uses the Stan-

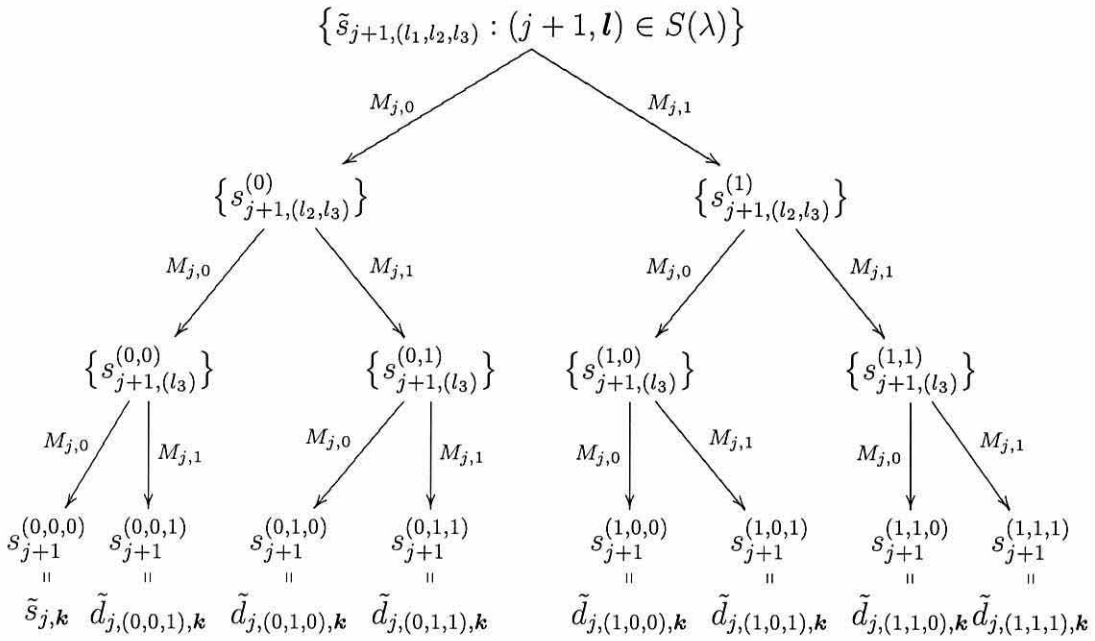


Figure 4.1: Example of a one-level decomposition in 3D for the coefficients associated with the cube $\square_\lambda$, where $\lambda = (j, \boldsymbol{e}, \boldsymbol{k})$.

dard Template Library containers 'set' and 'map' to store the set of cubes (denoted by G), the set of scaling function coefficients (denoted by S) and the set of wavelet coefficients (denoted by D). The maximum number of storage handling operations per cube $\square_\lambda, \lambda \in \Lambda$, is of N^n searches through S, the insertion of one coefficient in S and of $2^n - 1$ coefficients in D. The STL documentation guarantees that the average time needed to insert or to find an element in a set (or map) is at most proportional to the logarithm of the number of entries in the set (or map). Finally, the algorithm erases coefficients from S and G every time a cube has a lower level than the previous cube. Erasing an element takes constant time.

The function header is:

```
void LocTransformT( map< MDIndex,double,less<MDIndex> > &D,
                    set< MDIndex,less<MDIndex> >        &G,
                    map< MDIndex,double,less<MDIndex> > &S,
```

```
const I_Basis_Bspline &basis );
```

4.4 Numerical example

The example presented here is very simple. It consists of calculating a sparse dual wavelet expansion for the one-dimensional function $f(x) = \chi_{[3/8,5/8]}(x), x \in [0,1]$. The wavelet expansion is then evaluated pointwise. Let $\{x_i\}_{i=1}^N$ be a set of N points where the wavelet expansion has been evaluated and let e_i be the error made at point x_i. The wavelet expansion is compared with the original function f according to two criteria: the maximum pointwise error is $e_{max} = \max_i(|e_i|)$, and an approximation to the L_2 error is $e_{L_2} = (\sum_i e_i^2(x_i - x_{i-1}))^{1/2}$. Two different biorthogonal pairs of B-spline wavelet bases are used. In keeping with the notation of the Multilevel Library, these pairs of bases are denoted $N(d), N(d,\tilde{d})$, where d and $\tilde{d}$ are the approximation order of the the primal and dual bases. The error function E used to choose the set of wavelet coefficients kept in the sparse representation was the following

$$E(\square_\lambda) = \begin{cases} 0 & \text{if } \tilde{\sigma}_{j,k} \cap [3/8, 5/8] = \emptyset, \\ \text{diam}(\tilde{\sigma}_{j,k}) & \text{otherwise.} \end{cases}$$

The number of coefficients used in the local scaling function expansion is denoted by N_s, the number of coefficients in the wavelet expansion is denoted by N_D and the maximum wavelet level involved is denoted by J. The results are summarized in tables 4.1 to 4.2. Table 4.3 shows the results obtained when the set of wavelet coefficients retained is chosen after the full wavelet expansion has been calculated, by discarding wavelet coefficients of modulus less than 10^{-12}.

δ	0.1	0.01	0.001	10^{-4}	10^{-5}
e_{max}	0.3076	0.3076	0.3076	0.3076	0.3076
e_{L_2}	0.0397	0.0140	0.0050	0.0012	0.0004
N_S	134	224	326	462	564
N_D	51	73	95	123	147
J	7	10	13	17	20

Table 4.1: *Bases N(2),N(2,4) - direct to sparse representation.*

δ	0.1	0.01	0.001	10^{-4}	10^{-5}
e_{max}	0.3924	0.3924	0.3924	0.3924	0.3924
e_{L_2}	0.0499	0.0177	0.0044	0.0016	0.0005
N_S	78	162	274	358	442
N_D	43	64	94	116	140
J	6	9	13	16	19

Table 4.2: *Bases N(3),N(3,3) - direct to sparse representation.*

e_{max}	0.2936	0.2936	0.2936	0.2936	0.2936	0.2936	0.2936
e_{L_2}	0.0990	0.0700	0.0495	0.0350	0.0248	0.0175	0.0124
N_S	31	63	127	255	511	1023	2047
N_D	25	35	45	55	61	69	77
J	4	5	6	7	8	9	10

Table 4.3: *Bases N(2),N(2,4) - from full representation to sparse, threshold on wavelet coefficients:* 10^{-12}.

The first observation concerning the tabulated results is that the maximum error e_{max} does not vary with the parameter δ. The maximum error corresponds to the overshoot of the wavelet expansion near the discontinuities. Near the discontinuities, the local scaling function expansion of f is on the highest level and the error is evaluated at dyadic points $2^{-J_\delta}k$, corresponding to the indices (J_δ, k) of the scaling functions used. When the value of δ is decreased, the level J_δ increases, but this is only a re-scaling of the problem: when J_δ is fine enough, the scaling function coefficients involved are all calculated as

$$\tilde{s}_{J_\delta,k} = 2^{-J_\delta/2} \sum_{l=l_1}^{l_2-1} \sum_{i=0}^{\tilde{d}-1} f(x_{l,i}) \langle L_i^{[l,l+1]},\ \phi\ \chi_{[l,l+1]} \rangle,$$

$$x_{l,i} = 2^{-J_\delta}\big(k + l + i/(\tilde{d}-1)\big)$$

which depends on J_δ only by the factor $2^{-J_\delta/2}$ and by the values of f on a stencil of $(l_2-l_1)(\tilde{d}-1)+1$ points, regularly spaced at intervals of $2^{-J_\delta}/(\tilde{d}-1)$. For $f(x) = \chi_{[3/8,5/8]}(x)$, the $(l_2 - l_1)(\tilde{d}-1)+1$ values $f(x_{l,i})$ do not depend on the scale of the stencil but only on its relative position to the discontinuity. So, if the local scaling function expansion about point $x = 3/8$ contains the function $\phi_{J_\delta,k}$, there exists k' such that for parameter $\delta' < \delta$, $\phi_{J_{\delta'},k'}$ is part of the local expansion about $x = 3/8$ and $\tilde{s}_{J_\delta,k} = 2^{(J_{\delta'}-J_\delta)/2}\tilde{s}_{J_{\delta'},k'}$. Therefore $\tilde{s}_{J_\delta,k}\phi_{J_\delta,k}$ has the same values on a grid of level J_δ as $\tilde{s}_{J_{\delta'},k'}\phi_{J_{\delta'},k'}$ on a grid of level $J_{\delta'}$. The maximum error e_{max} does not depend on J_δ. However the error becomes more and more localized near the discontinuities as J_δ increases, as is shown by e_{L_2}.

The second observation is that the error function E, although very simple, is good at predicting which wavelet coefficients should be retained in the expansion, as can be seen from the comparison of tables 4.1 and 4.3. A similar error function will be used for the right hand side of the Poisson equation with interface condition.

Chapter 5

Fast matrix vector multiplication and dynamical calculation of the derivative operator matrix entries

The global wavelet scheme (presented in chapter 3) that solves iteratively the Poisson equation makes use of matrix-vector multiplications of the type $\boldsymbol{Av}$, where $\boldsymbol{A}$ is an infinite matrix and $\boldsymbol{v}$ is an infinite vector with a finite number of non-zero entries. These multiplications have to be approximated by a finite number of computing operations.

5.1 The *MULT* algorithm

5.1.1 Properties of this algorithm

In [9], the authors describe an algorithm for a routine ***MULT*** such that (Proposition 5.10, p.22 [8]) '*Given a tolerance $\eta > 0$ and a vector V with finite support, the algorithm*

MULT *produces a vector* W *which satisfies* [...]

$$\|CV - W\|_{l_2(\nabla)} \leq \eta,$$

where the matrix $\boldsymbol{A}$ is denoted by C. Moreover, for a class $\mathcal{A}_{s^*}$ of s^*-admissible matrices $\boldsymbol{A}$, the ***MULT*** algorithm has the following properties, listed in Requirements 5.3, p.18, [8]:

- '*The size of the output* Λ_η *is bounded by*

$$\#(\Lambda_\eta) \leq C\|V\|^{1/s}_{\ell^w_\tau(\nabla)}\eta^{-1/s}.$$

- *The number of* **arithmetic operations** *needed to compute* W_η *does not exceed* $C\{\eta^{-1/s}\|V\|^{1/s}_{\ell^w_\tau(\nabla)} + N\}$ *with* $N := \#suppV$.

- *The number of* **sorts** *needed to compute* W_η *does not exceed* $CN\log N$.'

Here, η denotes the tolerance, Λ_η denotes the set of wavelet indices present in the output vector W_η, s is such that $0 < s < s^*$ and $\tau = (s + 1/2)^{-1/2}$. Finally, C denotes a positive constant depending only on s when s tends to infinity.

The space $\ell^w_\tau(\nabla)$ is called 'weak ℓ_τ' and in [9] p.13 it is defined for $\tau < 2$ as:

$$\ell^w_\tau(\nabla) = \{\boldsymbol{v} \in \ell_2(\nabla) : \#\{\lambda : 2^{-j} \geq |v_\lambda| \geq 2^{-j-1}\} \leq c2^{j\tau}, j \in \mathbb{Z} \text{ for some } c < \infty\},$$

with semi-norm

$$|\boldsymbol{v}|_{\ell^w_\tau(\nabla)} = \inf\{c > 0 : v^*_n \leq cn^{-1/\tau}\}, n \geq 1,$$

where $\boldsymbol{v}^*$ denotes the sequence comprising of the modulus of the entries of $\boldsymbol{v}$ sorted by decreasing order. A norm is then defined on $\ell_\tau^w(\nabla)$ by

$$\|\boldsymbol{v}\|_{\ell_\tau^w(\nabla)} = \|\boldsymbol{v}\|_{\ell_2} + |\boldsymbol{v}|_{\ell_\tau^w(\nabla)}.$$

Assuming that $\boldsymbol{v}$ is in $\ell_\tau^w(\nabla)$ means that the size of the entries in the sequence $\boldsymbol{v}$ is expected to decrease at a certain rate, while assuming that $\boldsymbol{A}$ is in $\mathcal{A}_{s^*}$ is linked to the compressibility properties of the matrix $\boldsymbol{A}$. A subset of $\mathcal{A}_{s^*}$ is the set $\mathcal{C}_{s^*}$ of s^*-compressible matrices defined as follows.

Definition 5.8, p.21, [8] *A matrix C is called s^*-compressible if for each $0 < s < s^*$ and for some positive sequence $\{\alpha_j\}_{j\in\mathbb{N}_0}$ there exists for each $j \in \mathbb{N}_0$ a matrix C_j having at most $\alpha_j 2^j$ non-zero entries per row and columns such that $\|C - C_j\|_{\ell_2} \leq \alpha_j 2^{-sj}, j \in \mathbb{N}_0$. The class of s^*-compressible matrices is denoted by $\mathcal{C}_{s^*}$.*

When B-spline wavelet bases [12][14] are used, the preconditioned matrices $\boldsymbol{A}$ comprising of entries $a_{\lambda,\lambda'} = 2^{-(|\lambda|+|\lambda'|)} \int \nabla\psi_\lambda \cdot \nabla\psi_{\lambda'}$ are shown in [9] p.16, p.17 be s^*-compressible,

$$\|\boldsymbol{A} - \boldsymbol{A}_j\| \leq C2^{-js} \quad j \in \mathbb{N} \text{ and } 0 < s < s^*, \tag{5.1}$$

with $ns^* \leq \sigma - n/2$. The parameter σ is given in [21] p.2003, as $0 < \sigma < \rho - s$, where ρ is the Sobolev regularity of the primal wavelets and s is the coefficient used in the re-scaling matrix $\boldsymbol{D} = \text{diag}(2^{-s(|\lambda|)}, \lambda)$ ($\boldsymbol{A} = \boldsymbol{DCD}$ is the preconditioned derivative operator matrix), which is 1 in the case of a second order elliptic operator. In this case, the compressed $\boldsymbol{A}_j$

are found by a truncation process. Let $\tilde{a}_{\lambda,\lambda'}$ denote the entries of $\boldsymbol{A}_j$, then

$$\tilde{a}_{\lambda,\lambda'} = \begin{cases} a_{\lambda,\lambda'}, & \||\lambda| - |\lambda'|\| \leq j/n, \\ 0, & \text{else.} \end{cases} \tag{5.2}$$

It is noted in [9] p.26, that the bound $s^* \leq \sigma/n - 1/2$ is not best possible as the authors of [3] p.31-32 have found a bound $s^* \leq d - 3/2$, where d is the order of the primal MRA, for 1-dimensional elliptic operator matrices constructed with spline wavelets. For instance, when the wavelets used have $d = 2$, the predicted s^* would be $3/2 - 1/2 - 1 = 0$ and the value found in [3] was $1/2$.

5.1.2 Description of the algorithm

The algorithm is based on a decomposition of the vector $\boldsymbol{v}$ as the sum

$$\boldsymbol{v} = \sum_{j=0}^{J} \boldsymbol{v}_{[j]}, \qquad N = \#supp\ \boldsymbol{v} \ \text{ and } J = \lceil \log_2 N \rceil$$

where $\boldsymbol{v}_{[0]} := \boldsymbol{v}_1$ and $\boldsymbol{v}_{[j]} := \boldsymbol{v}_{2^j} - \boldsymbol{v}_{2^{j-1}}$, while $\boldsymbol{v}_{2^j}$ is the vector comprising of the 2^j largest entries of $\boldsymbol{v}$. An approximation to $\boldsymbol{Av}$ is then found as

$$\boldsymbol{w}_k := \sum_{j=0}^{k} \boldsymbol{A}_{k-j} \boldsymbol{v}[j], \tag{5.3}$$

where the blocks $\boldsymbol{v}_{[j]}$ are multiplied with columns of the $\boldsymbol{A}_{k-j}$ matrices, the more compressed as the the size of the entries in the $\boldsymbol{v}_{[j]}$ decreases. The error made by the approxi-

mation $\boldsymbol{w}_k$ is estimated by the norm

$$\|\boldsymbol{A}\boldsymbol{v} - \boldsymbol{w}_k\|_{\ell_2} = \|\boldsymbol{A}(\boldsymbol{v} - \boldsymbol{v}_{2^k}) + \sum_{j=0}^{k}(\boldsymbol{A} - \boldsymbol{A}_{k-j})\boldsymbol{v}[j]\|_{\ell_2} \tag{5.4}$$

$$\leq c_2\|\boldsymbol{v} - \boldsymbol{v}_{2^k}\| + \sum_{j=0}^{k} a_j\|\boldsymbol{v}[j]\| \tag{5.5}$$

where c_2 is the norm-2 of $\boldsymbol{A}$ and the a_j are the norm-2 of the $\boldsymbol{A} = \boldsymbol{A}_j$ matrices.

The ***MULT*** algorithm consists mainly of estimating a suitable k such that $\|\boldsymbol{A}\boldsymbol{v} - \boldsymbol{w}_k\| \leq \eta$ is guaranteed.

Below is the description of ***MULT*** which can be found in [8] p.22.

MULT$[\eta, C.V] \rightarrow (W, \Lambda)$:

1. *Sort the non-zero entries of the vector* $\boldsymbol{V}$ *and form the vectors* $\boldsymbol{V}_{[0]}, \boldsymbol{V}_{[j]}$, $j = 1, \ldots, \lfloor log\ N \rfloor$ *with* $N := \#supp\boldsymbol{V}$. *Define* $\boldsymbol{V}_{[0]} := \boldsymbol{0}$ *for* $j >$ *log* N.

2. *Compute* $\|\boldsymbol{V}_{[j]}\|^2_{\ell_2(\nabla)}, j = 0, \ldots, \lfloor log\ N \rfloor + 1$ *and* $\|V\|^2_{\ell_2(\nabla)} = \sum_{j=0}^{N} \|\boldsymbol{V}_{[j]}\|^2_{\ell_2(\nabla)}$.

3. *Set* $k = 0$.

   (a) *Compute the right hand side* R_k *of (5.5) for the given value of* k.

   (b) *If* $R_k \leq \eta$ *stop and output* k; *otherwise replace* k *by* $k + 1$ *and return to (a).*

4. *For the output* k *of (3) and for* $j = 0, 1, \ldots, k$, *compute the nonzero entries in the matrices* C_{k-j} *which have a column index in common with one of the nonzero entries*

of $V_{[j]}$.

5. *For the output* k *of (3), compute* $\boldsymbol{W}_k$ *as in (5.4.2) and take* $\boldsymbol{W} := \boldsymbol{W}_k$ *and* $\Lambda = supp\boldsymbol{W}$.

5.2 Implementation

The implementation of ***MULT*** follows closely the suggestions made in the paper [3] concerning both the type of data structures to use and the algorithm for the matrix-vector multiplication from equation 5.3.

5.2.1 Data structures

The storage of the vector $\boldsymbol{v}$ must accommodate for the fact that the index set of non-zero entries in $\boldsymbol{v}$ is lacunary and that it does not have a particular structure. Hence, it is not feasible to map the wavelet index set of non-zero entries in $\boldsymbol{v}$ to a sequence of integers other than explicitly. The storage must also be dynamic and allow access any entries in a minimum of operations. As suggested in [3], vectors of wavelets coefficients have been implemented with the container *map* from the Standard Template Library [1]. A *map* in STL is a list of pairs of objects, the first object an element of an ordered index set, the second object an element of a set of *values*. The list is sorted by ascending order on the first object of each pair, which is termed the *key*. The STL also defines a number of functions to handle *maps* efficiently. These include an *insert* function, to insert a new pair at its correct place in the *map*, a *find* function to give access to the value corresponding to a given *key*,

an *erase* function to delete part of the *map*, and so on.

The flexibility of the STL resides in its template nature which means that the set of *keys* and the set of *values* are entirely defined by the user. For a vector $\boldsymbol{v}$ of wavelet coefficients, I have defined a class of multi-dimensional wavelet indices called *MDIndex* comprising of four components:

- d the dimension of the associated wavelet,
- j the wavelet level,
- e a vector of integers corresponding to the 'type' of the wavelet,
- and k a vector of integers corresponding to its position.

The order function is defined on the subsets of indices of the same dimension and it is the following: for wavelet indices λ and λ', λ more than λ' if, in that order:

- $j(\lambda) > j(\lambda')$,
- $e_i(\lambda) > e_i(\lambda')$ and $e_m(\lambda) = e_m(\lambda')$ for $m < i$,
- $k_i(\lambda) > k_i(\lambda')$ and $k_m(\lambda) = k_m(\lambda')$ for $m < i$,

The storage of the matrix $\boldsymbol{A}$ has the similar requirements to that of the vector $\boldsymbol{v}$. The multiplication procedure described in ***MULT*** requires easy access to each column of $\boldsymbol{A}$, and within a column, sequential access to each non-zero entry is also required. Once again, as suggested in [3], the matrix $\boldsymbol{A}$ has been implemented by means of a map of maps. The first map associates a wavelet index with a column of the matrix, and each column is a map that associates a wavelet index (the row index) with a value.

Finally, another type of storage is needed to implement $\boldsymbol{v}^*$, the sequence of the entries of $\boldsymbol{v}$ sorted by decreasing magnitude. The vectors $\boldsymbol{v}_{2^j}$ and $\boldsymbol{v}_{[j]}$ are indeed blocks of $\boldsymbol{v}^*$. A *map* storage is here would be inappropriate because the set of wavelet coefficients is only a partially ordered set, and therefore it cannot be used as the set of *keys*. The STL defines a container called *multimap* to map a set of partially ordered *keys* to a set of *values*. In the case of $\boldsymbol{v}^*$, the *keys* are the magnitude of the wavelet coefficients and the *values* are the wavelet indices.

5.2.2 The *Mult* and *FastMatVecMultGH* functions

The *Mult* function implements directly the algorithm ***MULT*** and the *FastMatVecMultGH* function is basically a copy of the piece of code given in [3] p.20. The main difference with this function is due to the fact that for [3], the Laplace operator matrix was pre-calculated before starting the program, while here it is calculated dynamically column per column, as needed. Every time a column $a_{;\lambda}$ of a certain $\boldsymbol{A}_J$ is needed, the *FastMatVecMultGH* function checks whether that column exists in storage, and if so it further checks the wavelet coefficient of lowest level $a_{\lambda',\lambda}$ is such that $|j(\lambda') - j(\lambda)| = J/n$ and if it is less than J/n, the column is complemented with the non-zero coefficients $a_{\nu,\lambda}$ such that $\min(j(\lambda) - J/n, j_0) \leq \nu(j) < \lambda'(j)$. Similarly, the column is complemented with entries of higher row level, if needed. If the needed column is altogether absent from storage, it is inserted with the appropriate range of levels for the rows wavelet indices. These operations are performed by a function called *InsertColumnSubset*. The dynamic calculation of the individual entries of the Laplace operator matrix can be very time consuming but it saves memory space. One consequence of the pre-calculation of the matrix is that it imposes an

overall maximum level on the scheme. Instead, when the entries are calculated dynamically, the number of entries in storage is the restricting factor.

A difference with the algorithm ***MULT***, is the possibility to restrict the level increase between the vector $\boldsymbol{v}$ and the vector $\boldsymbol{w}_k$. This functionality can be switched on or off by changing the definition of a macro name

```
__ALI
```

(for Allowed Level Increase). The value of the maximum level allowed is defined in the macro

```
_ALI
```

. Another functionality restricts instead the size of the radius $|j - j'|$ used for each column-coefficient multiplication. In this case, the equation 5.3 is replaced by

$$\boldsymbol{w}_k := \sum_{j=0}^{k} \boldsymbol{A}_{\min(k-j,J_r)} \boldsymbol{v}[j],$$

where J_r denotes the maximum radius imposed. This functionality can be switched on or off by changing the definition of the macro name

```
__MR
```

(for Maximum Radius). The value of the maximum radius is defined in the macro

```
_MR
```

Both functionalities can also be switched on at the same time.

The function headers are:

```
void Mult( map<MDIndex,double,less<MDIndex> > &w_j,
           const multimap<double,MDIndex,less_absolute> &u_j,
   LaplaceOp &A, double Eps, int CML = 0 );

void FastMatVecMultGH( map<MDIndex,double,less<MDIndex> > &w_j,
         const multimap<double,MDIndex, less_absolute > &v_lambda,
         LaplaceOp &A, int J, int CML = 0, int ALI = 0 );
```

5.2.3 Pre-processsing step: calculation of the matrix norms c_2 and a_j

The ***MULT*** algorithm requires the evaluation of R_k, an upper bound for the error $\|\boldsymbol{A}\boldsymbol{v} - \boldsymbol{w}_k\|$ given in (5.5). It is apparent that the parameters c_2 and a_j must be approximated in a pre-processing step. By definition $c_2 = \|\boldsymbol{A}\|_2$ and $a_j = \|\boldsymbol{A} - \boldsymbol{A}_j\|_2$. It is known that $a_j \leq C_A 2^{-sj}$ for some $C_A > 0$. In [3] p.31, it has been proven for the 1D-case that $s = d - 3/2$, where d denotes the order of the primal MRA. Therefore, it remains to estimate C_A and c_2 as well as s for higher dimensions. This has been done for the spline wavelet basis with $d = 3$ and $\tilde{d} = 3$ in one and two dimensions.

These parameters are approximated numerically from finite approximations to the ma-

$$\boldsymbol{A} = \left[\begin{array}{c|c|c|c} [S_{j_0}S_{j_0}] & [S_{j_0}W_{j_0}] & [S_{j_0}W_{j_1}] & \cdots \\ \hline [W_{j_0}S_{j_0}] & [W_{j_0}W_{j_0}] & [W_{j_0}W_{j_1}] & \cdots \\ \hline [W_{j_1}S_{j_0}] & [W_{j_1}W_{j_0}] & [W_{j_1}W_{j_1}] & \cdots \\ \hline \vdots & \vdots & \ddots & \ddots \end{array}\right] \begin{array}{l} [S_{j_0}] \\ [W_{j_0}] \\ [W_{j_1}] \\ [W_j] \end{array}$$

Figure 5.1: *Block decomposition of matrix* $\boldsymbol{A}$.

trix $\boldsymbol{A}$. The 1-norm of a $n \times n$ matrix is defined as

$$\|B\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{n} |b_{i,j}|.$$

Because $\|B\|_2 \le \|B\|_1$ for any matrix B, an upper bound for the 2-norm of a finite dimensional matrix is the maximum row sum.

Denoting by $[S_{j_0}S_{j_0}]$ the block of $\boldsymbol{A}$ comprising of entries $a_{\lambda,\lambda'} = \int \nabla\phi_\lambda \cdot \nabla\phi_{\lambda'}$, $|\lambda| = |\lambda'| = j_0$, and by $[S_{j_0}W_j]$ the blocks of $\boldsymbol{A}$ comprising of entries $a_{\lambda,\lambda'} = \int \nabla\phi_\lambda \cdot \nabla\psi_{\lambda'}$, $|\lambda| = j_0$ and $|\lambda'| = j$, and denoting in the same spirit the remaining blocks of $\boldsymbol{A}$ by $[W_jS_{j_0}]$ and $[W_jW_{j'}]$, the matrix $\boldsymbol{A}$ can be split into blocks according to the level of the wavelets used to calculate its entries. This is represented on Figure 5.1.

The largest row sum in $\boldsymbol{A}-\boldsymbol{A}_J$ is searched by calculating the row sums block per block. Let $p_k([S_{j_0}W_j])$ denote the sum of the entries on the k^{th} row of block $[S_{j_0}W_j]$, let $p_k([W_jW_{j'}])$ denote the sum of the entries on the k^{th} row of block $[W_jW_{j'}]$ and so on. Then

$$\|\boldsymbol{A}-\boldsymbol{A}_J\| = \max_{j',k} \lim_{j_m\to\infty} \sum_{j=J/n}^{j_m} p_k([W_{j'}W_{j'+j}]) + \sum_{j=J/n}^{j'-j_0} p_k([W_{j'}W_{j'-j}]),$$

where the first term corresponds to a sum of entries in blocks on or above the diagonal blocks, while the second term corresponds to a sum of entries in blocks below the diagonal blocks. In the sum above, when j' or $j'-j$ is equal to j_0, the sum is made on blocks $[S_{j_0}]$ or $[\dot{S}_{j_0}]$ too.

The search for the maximum is facilitated by the fact that the entries in $\boldsymbol{A}$ depend essentially on the difference $j-j'$ of levels between the two wavelets involved in the calculation of the entry. This can be seen as follows. In dimension $n>1$, the entries of $\boldsymbol{A}$ comprise of a sum of terms of the form

$$2^{-(j+j')}\int \zeta'_{j,e_i,k_i}(x_i)\zeta'_{j',e'_i,k'_i}(x_i)\prod_{m\neq i}\int \zeta_{j,e_m,k_m}(x_m)\zeta_{j',e'_m,k'_m}(x_m).$$

This expression can be re-scaled with $\zeta'_{j,e_i,k_i}(x_i) = 2^{3j/2}\zeta'_{e_i,k_i}(2^jx_i)$ and assuming that $j'>j$, by a change of variables, the integral of the derivatives comes to

$$\begin{aligned} & 2^{3(j+j')/2}2^{-j}\int \zeta'_{j',e_i,k_i}(x_i)\zeta'_{0,e'_i,k'_i}(2^{j'-j}x_i) \\ = \; & 2^{(j+j')/2}2^{j-j'}\int \zeta'_{j',e_i,k_i}(x_i)\zeta'_{0,e'_i,k'_i}(2^{j'-j}x_i) \\ = \; & 2^{2j}\int \zeta'_{j',e_i,k_i}(x_i)\zeta'_{2^{j'-j},e'_i,k'_i}(x_i) \end{aligned}$$

Similarly, the remaining integrals can be re-scaled and they come to

$$\begin{aligned} & 2^{(j+j')/2}2^{-j}\int \zeta_{j',e_m,k_m}(x_m)\zeta_{0,e'_m,k'_m}(2^{j'-j}x_m) \\ = \; & 2^{(j'-j)/2}\int \zeta_{j',e_m,k_m}(x_m)\zeta_{0,e'_m,k'_m}(2^{j'-j}x_m) \\ = \; & \int \zeta_{j',e_m,k_m}(x_m)\zeta_{2^{j'-j},e'_m,k'_m}(x_m) \end{aligned}$$

Finally, taking into account the preconditioning factor $2^{-(j+j')}$,

$$\begin{aligned} & 2^{-(j+j')}\int \zeta'_{j,e_i,k_i}(x_i)\zeta'_{j',e'_i,k'_i}(x_i)\prod_{m\neq i}\int \zeta_{j,e_m,k_m}(x_m)\zeta_{j',e'_m,k'_m}(x_m) \\ = \; & 2^{-|j-j'|}\int \zeta'_{j',e_i,k_i}(x_i)\zeta'_{2^{j'-j},e'_i,k'_i}(x_i)\prod_{m\neq i}\zeta_{j',e_m,k_m}(x_m)\zeta_{2^{j'-j},e'_m,k'_m}(x_m), \end{aligned}$$

each term depends only on a factor $2^{(j'-j)(n+1)/2}$. What is more, because of the shift invariance of the wavelet and scaling functions which do not intersect the domain boundary, for j and j' fixed, most of the sums inside a block, for instance the $p_k([W_jW_{j'}])$, are equal. So, in order to calculate all the sums corresponding to blocks on or above the diagonal, it suffices to calculate for increasing j the $p_k([S_{j_0}W_j])$ and the $p_k([W_{j_0}W_j])$ associated with rows corresponding to functions whose support intersects the support of the boundary adapted functions, and to one scaling function and 2^{n-1} wavelets whose support does not intersect the support of boundary adapted functions.

Since the norm $\|\boldsymbol{A}-\boldsymbol{A}_J\|$ is expected to behave like $C2^{-Js}$, for fixed k a linear relationship is searched between the logarithm of the $p_k([S_{j_0}W_j])$ and the difference $j-j_0>1$. The same thing is done for the $p_k([W_{j_0}W_j])$. In the case of the spline basis with $d=3,\tilde{d}=3$ in dimension 1, these results have been calculated for $j_0\leq j'\leq j_0+11$ and are reproduced

at the end of this chapter. The regressions, calculated using *Excel 2000*, show a very good fit, with a $1 - r^2 \leq 10^{-12}$. All rows have the same slope $-s = 1.5$ but the intercept varies. So the sums $s_k([S_{j_0} W_j])$ are given by a relation $C_k 2^{-s(j-j_0)}$ and similarly the $p_{k'}([W_{j_0} W_j])$ are given by a relation $C_{k'} 2^{-s(j-j_0)}$.

In order to calculate the sums corresponding to blocks below the diagonal, it suffices to calculate for increasing j the sums $p_k([W_j S_{j_0}])$ and $p_k([W_j W_{j_0}])$ for the same choice of row k as before. In the 1D-case of the spline basis with $d = 3, \tilde{d} = 3$, these sums were found to decrease very quickly with the difference of levels $j - j_0$. This makes sense since on each row, in blocks below the diagonal, the number of non-zero entries is independent of the difference $j - j'$, while the size of the entries is multiplied by a coefficient $2^{-|j-j'|}$.

Overall, it was found in the 1D-case of the spline basis with $d = 3, \tilde{d} = 3$, that a large coefficient $C_k \approx 14.4$, corresponding to a scaling function row, was the largest of all and that its influence even outweighed the additional sum on blocks below the diagonal that would be added to a wavelet row. As a consequence, for that basis, an upper bound for the coefficients a_j has been estimated by

$$a_j \leq \frac{14.4}{1 - 2^{-1.5}} 2^{-1.5(j+1)}, \quad j >= 1$$

Also, a very large sum in the block $[S_{j_0} S_{j_0}]$ makes clear that the second scaling function row has the largest sum in $\boldsymbol{A}$. As a consequence, for that basis, the coefficient c_2 has been bounded by

$$c_2 \leq p_2([S_{j_0} S_{j_0}]) + p_2([S_{j_0} W_{j_0}]) + \frac{14.4}{1 - 2^{-1.5}} 2^{-1.5} \approx 1244.3$$

The parameter $s = 1.5 = d - 3/2$ matches the value given in [3]. For the 2D-case, the calculations have been further simplified by the assumption that the solution lies far enough from the boundary to justify the removal of all the rows in $\boldsymbol{A}$ that correspond to boundary adapted basis functions. The reason for this is that the sums in diagonal blocks are all significantly outweighed by sums corresponding to boundary adapted scaling functions. Since these functions are unlikely to be used in the problem at hand, keeping their associated rows in the calculations might mean an over-estimation of the parameter c_2. The same calculations as in the 1D-case have been repeated in the 2D-case and they have provided the following upper bounds for the a_j and c_2

$$c_2 \leq 1707.7 \qquad \text{and} \qquad a_j \leq 4.38 \frac{2^{-1.04(j+1)}}{1 - 2^{-1.04}}.$$

5.3 Dynamical calculation of the entries of the derivative operator matrix

In order to apply the ***MULT*** algorithm efficiently to sparse vectors, it will be convenient to calculate the entries of the derivative operator matrix $\boldsymbol{A}$ as and when the need for them arises. The caculated entries can then be stored in memory for future use. The dynamical calculation of these entries that has been implemented for this thesis is based on a method described in [4].

Because of the tensor product construction of the wavelets used, the entries of $\boldsymbol{A}$ may

be decomposed into a sum of products of one-dimensional integrals of the form

$$\int_0^1 \zeta_{j,e,k}^{(i)}(x)\, \zeta_{j',e',k'}^{(i)}(x)\, dx, \tag{5.6}$$

where $i = 0, 1$. The approach from [4] consists of calculating these inner products as a linear combination of integrals of scaling functions on a single level:

$$\int_0^1 \zeta_{j,e,k}^{(i)}(x)\, \zeta_{j',e',k'}^{(i)}(x)\, dx = \sum_{l \in \Lambda_J(\sigma_m)} \sum_{l' \in \Lambda'_J(\sigma_m)} m_{J,l}^{(j,e,k)} m_{J,l'}^{(j',e',k')} \int_0^1 \xi_{J,l}^{(i)}(x)\, \xi_{J,l'}^{(i)}(x)\, dx,$$

where $J = \max(j+e, j'+e')$, the $\xi_{J,l}^{(i)}$ are the i^{th} derivatives of the scaling functions and σ_m is the shortest support of the two functions $\zeta_{j,e,k}$ and $\zeta_{j',e',k'}$. The set $\Lambda_J(\sigma_m)$ is defined as the set of indices l such that the support of $\xi_{J,l}$ has a non-empty and non-trivial intersection with the set $\sigma_m \cap \mathrm{supp}(\zeta_{j,e,k})$. The set $\Lambda'_J(\sigma_m)$ is defined similarly. The coefficients $m_{J,l}^{(j,e,k)}$ are derived from the refinement and wavelet matrices.

The inner products

$$\int_0^1 \xi_{J,l}^{(i)}(x)\, \xi_{J,l'}^{(i)}(x)\, dx$$

are all equal, up to a multiplicative factor, to an inner product of scaling functions on the coarsest level j_0

$$\int_0^1 \xi_{j_0,p}^{(i)}(x)\, \xi_{j_0,p'}^{(i)}(x)\, dx.$$

In a preprocessing step, all the non-zero inner-products of the above form are calculated and stored in memory as:

$$\begin{cases} \int_{\mathbb{R}} \xi_{j_0}^{(i)} \xi_{j_0,p'-p}^{(i)}, & \text{if } \operatorname{supp}(\xi_{j_0,p}) \cap \{0,1\} = \emptyset \text{ and } \operatorname{supp}(\xi_{j_0,p'}) \cap \{0,1\} = \emptyset, \\ \int_{[0,1]} \xi_{j_0,p}^{(i)} \xi_{j_0,p'}^{(i)}, & \text{otherwise.} \end{cases}$$

The *I-Integral* function from the Multilevel Library calculates accurately the integrals of products of refinable functions and of their lower derivatives by solving an eigenvalues problem. Given the refinement matrix of two scaling function bases defined on the unit interval, this function computes simultaneously all the non-vanishing inner-products of the form above, whether the functions involved are boundary-adapted or not. The *I-Integral* is the function used in the pre-processing step.

After the pre-processing step, the more effort-consuming part of the method consists of the calculation of the coefficients $m_{J,l}^{(j,e,k)}$. It is proved in [4] that the overall amount of work needed to calculate one one-dimensional integral such as (5.6) in the worst case is proportional to $j_{diff} = \max(J-j, J-j')$. In the implementation of this method for its use by the fast matrix vector multiplication algorithms presented in the previous sections, it has been advantageous to use the fact that the entries of $\boldsymbol{A}$ need to be calculated column by column to avoid the repeated calculation of the coefficients $m_{J,l}^{(j,e,k)}$ corresponding to the column index.

During the preliminary tests on the matrix $\boldsymbol{A}$ and on the Poisson solver, the computation of the entries of $\boldsymbol{A}$ has been found to be very slow, in particular when entries $a_{\lambda,\lambda'}$ with a fairly large difference $|\lambda - \lambda'|$ were used. For this reason, the method first imple-

mented has been replaced for most calculations by a piece of software written by Mario Mommer, a research student at the University of Aachen. His software is not (yet) part of the Multilevel Library, but he has kindly given me access to it, thereby speeding up my program noticeably. His software directly calculates the one-dimensional integrals of B-spline scaling functions and wavelets by taking advantage of their polynomial expressions. In this way, the dependence on the scale difference j_{diff} is removed.

Chapter 6

Pointwise evaluation of a sparse wavelet expansion

6.1 Interface between the time-stepping scheme and the wavelet Poisson solver

The process of recovering point values from a wavelet expansion is required by numerical schemes involving wavelets. This evaluation may be needed at different stages in the scheme and for different reasons. Recall that the evolution of the magnetization $\boldsymbol{m}$ over time is governed by the Landau-Lifshitz equation, where the time derivative of $\boldsymbol{m}$ is a function of $\boldsymbol{m}$ itself and of four applied fields:

- $\boldsymbol{h}_{app}$, a constant,
- $\boldsymbol{h}_a = (\boldsymbol{m} \cdot \hat{\boldsymbol{e}})\, \hat{\boldsymbol{e}}$,
- $\boldsymbol{h}_{ex} = C_{ex}\, \nabla^2 \boldsymbol{m}$,

- and $\boldsymbol{h}_d = -C_d \nabla u$,

  where u is the solution of Poisson solver.

In the present situation, the time-stepping scheme used for the equation of motion of the magnetization is implemented in the physical space, while the Poisson solver used to calculate the demagnetization field, which is a component of the magnetization's time derivative, is implemented in the wavelet space. As a consequence, the wavelet expansion of the solution of the Poisson solver must be evaluated at every time-step. It will be seen that the gradient of the solution, h_d, can be evaluated directly from the wavelet expansion of u.

The choice of the points where the wavelet expansion is evaluated depends on the local refinement of this expansion, and since the wavelet solver is adaptive, the refinement may vary from one run of the solver to the next. Hence, the set of points where the value of the derivative $\frac{d\boldsymbol{m}}{d\tau}$ is known is dictated by the wavelet solver. However, the Euler time-stepping scheme

$$\begin{aligned} \boldsymbol{m}_0 &= \boldsymbol{m}(\tau_0), \\ \boldsymbol{m}_n &= \boldsymbol{m}_{n-1} + \Delta\tau \frac{d\boldsymbol{m}}{d\tau}(\boldsymbol{m}_{n-1}), \qquad \tau_n > \tau_0. \end{aligned}$$

clearly makes the assumption that the values of $\boldsymbol{m}_{n-1}$ and of its derivative $\frac{d\boldsymbol{m}_{n-1}}{d\tau}$ are known at the same nodes. For this reason, the point-value expression of the magnetization $\boldsymbol{m}_{n-1}$ must be updated by some form of interpolation so that it corresponds to the same nodes as the point-value expression of $\frac{d\boldsymbol{m}_{n-1}}{d\tau}$. This update will be done by projecting $\boldsymbol{m}_{n-1}$ onto the dual wavelet space, and evaluating this expansion at the correct set of nodes. The first advantage of this method is that it is systematic and that the lack of a pattern between the

set of nodes at time $n-1$ and the set of nodes at time n does not present any difficulties. The second advantage is that it makes it easy to use the same approximation order for this interpolation as the one that has been used for the calculation of the Poisson equation right hand side. Finally, it is possible to calculate accurately and for a minimum cost, the values of the Laplacian of $\boldsymbol{m}_{n-1}$ at the same time as the values of $\boldsymbol{m}_{n-1}$. This immediately gives the exchange field $\boldsymbol{h}_{ex}^{n-1}$ at the correct nodes, while the anisotropy field $\boldsymbol{h}_a$ is derived in a straight-forward manner from the values of $\boldsymbol{m}_{n-1}$. It can be noted that the evaluation of the effective fields and of the magnetization will only be performed at points situated inside the nano-element as the magnetization is always theoretically null outside it.

The role of the pointwise evaluation of a sparse wavelet expansion is therefore an important one in the scheme. The description of its implementation will be the subject of this chapter. The process of evaluation consists of two main steps. The first one is the converse of the wavelet transform described in section 4.2.6. In this step, the sparse wavelet transform is taken back to a local scaling function expansion. The second step is the pointwise evaluation of the local scaling function expansion. This evaluation takes place on grid fine enough to contain at least every node of the form $2^{-(j+1)}\boldsymbol{k}$, where $d_{(j,\boldsymbol{e},\boldsymbol{k})}$ is a coefficient in the sparse wavelet expansion. The presence of several levels of scaling functions makes this a delicate task and the choice of the scaling function coefficients present in the local scaling function expansion plays an important part in ensuring that the evaluation is done at all the desired nodes. The next section presents an algorithm for the transformation of a sparse wavelet expansion into a local scaling function expansion, given the set of indices corresponding to scaling function coefficients needed for the evaluation. Next, an algorithm for the evaluation is given, where the set of scaling function coefficients present in the local

expansion determines where the expansion is evaluated. It is also explained how the partial derivatives of the local scaling function expansion can also be evaluated, at little extra cost. Finally, the last section is concerned with the choice of the scaling functions indices. In particular, this choice must ensure that the wavelet expansion of the right hand side of the Poisson equation at time τ_n can be obtained from the pointwise magnetization $\boldsymbol{m}_n$ for the same set of wavelet indices as were present in the expansion of the potential u^{n-1} at the previous time-step.

6.2 A local inverse transform

The inverse transform implemented in this work consists of the passage from a wavelet sequence $D = (d_\lambda, \lambda \in \Lambda_D)$, which is the sparse wavelet representation of a function f,

$$f(\boldsymbol{x}) = \sum_{\lambda \in \Lambda_D} d_\lambda \, \psi_\lambda(\boldsymbol{x}), \qquad \forall \boldsymbol{x} \in [0,1]^n$$

to a sequence of scaling function coefficients $S = (s_\lambda, \lambda \in \Lambda_S)$, where the set Λ_S may include indices on several levels and where $s_\lambda = \langle f, \tilde{\phi}_\lambda \rangle$. The local scaling function expansion is therefore redundant in the sense that

$$f(\boldsymbol{x}) \neq \sum_{\lambda \in \Lambda_S} s_\lambda \, \phi_\lambda(\boldsymbol{x}), \qquad \text{for many } \boldsymbol{x} \in [0,1]^n.$$

The dual inverse transform, which takes a sequence of coefficients $(\tilde{d}_\lambda, \lambda \in \tilde{\Lambda}_D)$ to a sequence $(\tilde{s}_\lambda, \lambda \in \tilde{\Lambda}_S)$, where $\tilde{s}_\lambda = \langle f, \phi_\lambda \rangle$, has also been implemented. The two inverse transforms work in the same way, the role of the primal and dual basis functions and masks being reversed. For this reason, this section will only present the first inverse trans-

form.

The inverse transform is based on the use of an inverse transform matrix, as represented on Figure 2.3. For the wavelet bases used in the implementation, the inverse matrix between levels j and $j+1$ is the concatenation of the primal refinement and wavelet matrices $M_{j,0}$ and $M_{j,1}$, that were described in section 2.3.1, but now in the multi-dimensional case. Hence, if $\boldsymbol{d}_j$ is the vector of the full wavelet expression of the projection of a function f onto the wavelet space W_j, and $\boldsymbol{s}_j$ and $\boldsymbol{s}_{j+1}$ are the vectors of the full scaling function expressions of the projection of f onto the spaces V_j and V_{j+1}, the inverse transform between levels j and $j+1$ can be written in matrix form as

$$\left[\begin{array}{c|c} M_{j,0}, M_{j,1} & \\ & \end{array}\right] \left[\begin{array}{c} \boldsymbol{s}_j \\ \hline \boldsymbol{d}_j \end{array}\right] = \left[\begin{array}{c} \boldsymbol{s}_{j+1} \\ \\ \end{array}\right].$$

In the case of a sparse wavelet expansion, the columns of the inverse matrix that are associated with the absent entries in $\boldsymbol{d}_j$ and $\boldsymbol{s}_j$ are not used.

The principle of the algorithm is to start from the set of scaling function and wavelet coefficients in Λ_D on the coarsest level j_0, and to calculate a sparse set $\bar{S}_{j_0+1}$ of scaling function coefficients on level j_0+1 by multiplication with the correct columns of the inverse transform matrix. Within the set $\bar{S}_{j_0+1}$, it is useful to identify three disjoint subsets: the subset $S_{j_0+1} = S \cap \bar{S}_{j_0+1}$, which consists of desired coefficients, the subset S'_{j_0+1} of coefficients that are not in S but that will be needed for the calculation of coefficients s_λ, $\lambda \in \Lambda_S$, $|\lambda| > j_0+1$, on higher levels, and finally the subset of coefficients that are not needed. Clearly, unlike the first two subsets, the last one should not be stored in memory.

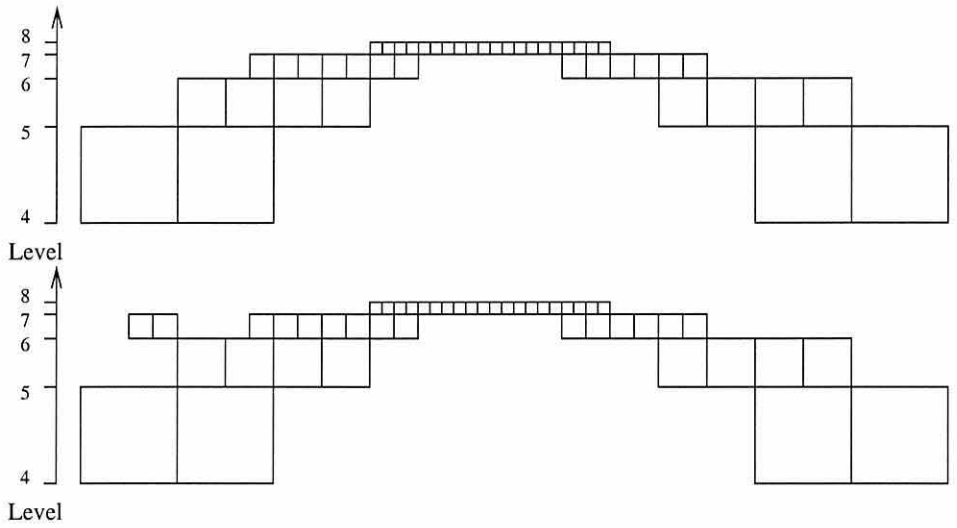


Figure 6.1: *Structure of an acceptable set Λ_S (top) and example of an unacceptable set (below). Each square, with side length 2^{-j} and lateral positioning $2^{-j}k$, represents a 1D-index $\lambda = (j,k)$.*

The whole process is iterated for increasing levels.

In order to differentiate between the set S'_{j+1} and the set of useless coefficients, we impose on the index set Λ_S a certain structure. Λ_S must have the structure of the set of leaves of a graded tree, with the relaxed condition that the parent of a leaf may also be in Λ_S provided that it is the neighbour of an index, which is already in Λ_S. In this way, some 'overlap' between consecutive levels is allowed in Λ_S. This is illustrated by a one-dimensional example on Figure 6.1. Let Λ denote the tree associated with Λ_S and let $\partial\Lambda$ denote its set of leaves. Let us define Λ_G, the index set that complements $\partial\Lambda$ into Λ. Then if Λ has a large enough grading coefficient, a coefficient s_λ, $\lambda \notin \Lambda_S$, is in S'_{j+1}, and therefore must be kept, if $\lambda \in \Lambda_G$.

An algorithm for the local inverse transform procedure

Now that all the relevant sets have been defined, an algorithm for the local inverse transform is presented.

- Start of procedure: copy the scaling function coefficients of coarsest level j_0 directly from D to S.
- Starting from the coarsest level j_0 to the finest J: for each level j:
  1. Copy the level $j+1$ of Λ_G into $\Lambda_{S'}$.
  2. Span the level $j+1$ of Λ_S and remove from $\Lambda_{S'}$ the coefficients that also belong to Λ_S and finally set to 0 the values of the coefficients on level j in S and S'.
  3. Span level j of D and, for each entry d_λ, calculate the set of scaling function coefficients of level $j+1$ which will receive some contribution from d_λ in the reconstruction process. This set is the set of row indices μ such that the entries (μ, λ) in the inverse transformation matrix are non-zero. Add the contribution from d_λ to the coefficients s_μ, if $\mu \in \Lambda_S$ or $\mu \in \Lambda_{S'}$.
  4. Span the level j of S and do the same as above.
  5. Span level j of S' and do the same as above.
  6. Delete all values in S' and all indices in $\Lambda_{S'}$.

The calculation of the contribution of a wavelet coefficient d_λ to a coefficient s_μ is simply the product $M_{j,1}(\mu, \lambda)\, d_\lambda$, while the contribution of scaling function s_λ to a coefficient s_μ is calculated by $M_{j,0}(\mu, \lambda)\, s_\lambda$. The wavelet and refinement matrices for the multi-dimensional bases are never stored in memory, as their entries are easily deduced from the univariate wavelet and refinement matrices.

6.3 Evaluation of a local scaling function expansion

The method implemented for the evaluation of the local scaling function expansion is based on the decision that the value at any one point would be calculated from scaling functions on a single level and that scaling functions on level j would be evaluated on a grid of dyadic nodes of level j.

The set of points where the evaluation takes place is determined by the set of scaling function coefficients and by the diameter of the support of the scaling functions. Let us assume that a point $\boldsymbol{x}_j$, on the dyadic grid of level j, lies inside the support of exactly N scaling functions of level j. If not all the N coefficients associated with these scaling functions are present in the local scaling function expansion, the point $\boldsymbol{x}_j$ is considered to be 'missing', and the expansion is not evaluated at that point. Because of this, the set of scaling function indices Λ_S must be chosen carefully prior to the inverse transform, so that no 'gaps' of missing values are created at the transitional area between two levels of the expansion.

In the expansion, each term $s_\lambda \phi_\lambda$ is evaluated at dyadic nodes on level $|\lambda|$ from the values of the n-dimensional mother scaling function at dyadic nodes on level 0, which are calculated and stored in memory at the start of the routine. If any derivatives of the expansion are also required, the values of the derivatives of the mother scaling function are also stored in memory and for each dyadic point in the support of ϕ_λ, an array of values is stored, with one cell for each derivative. The evaluation of boundary-adapted scaling functions necessitates a little more work than that of the other scaling functions because they are a

linear combination of the dilates and translates of the mother scaling function, truncated at the boundary. The calculation of the values of the mother scaling function and of its derivatives is performed by a routine from the Multilevel Library called *EvalValues*, which evaluates refinable functions from their refinement masks according to a method presented in [29].

The output of the local evaluation function implemented consists of the set of point values of all the derivatives required. This set is stored as an object of class *PointValues*, which is defined as an STL *map*, associated with an integer variable called *Res* that contains the level of the finest dyadic nodes present in the set of point values. Each cell in the map corresponds to a node $\boldsymbol{x}_{j,\boldsymbol{k}} = 2^{-j}\boldsymbol{k}$ and its *key* is its position vector $2^{j_{Res}-j}\boldsymbol{k}$ on the dyadic grid of resolution level *Res*. The *value* of the cell is a vector, whose components are the values of the chosen derivatives at the node $\boldsymbol{x}_{j,\boldsymbol{k}}$.

The steps of the implementation of the overall evaluation of a local scaling function expansion are summarized in the algorithm below.

Algorithm

- Make the multi-variate refinement mask by tensor products of the univariate refinement mask $\sqrt{2}\,\boldsymbol{a}$ from section 2.3.1.
- Calculate and store the values of the derivatives of the mother scaling function ϕ at the dyadic nodes inside its support. The type and the number of the derivatives needed is specified in input. The mother scaling function itself is understood as the partial derivative of order 0 in all variables.

- Span the set of scaling function coefficients S backwards. (The coefficients in S are sorted by increasing level j.) For each $s_{j,\boldsymbol{k}} \in S$,

  1. If $s_{j,\boldsymbol{k}}$ is the first coefficient encountered on level j after all the coefficients on level $j+1$ have been passed, the values calculated at nodes on level $j+1$ must first be dealt with. These values form a set V.

     (a) For each derivative i needed, a coefficient C_i is calculated. This coefficient is the multiplicative constant that takes the values of the mother scaling function's derivatives to the derivatives of a scaling function on level $j+1$. It is calculated as $C_i = 2^{(j+1)\alpha_i}\sqrt{2^{(j+1)n}}$, where α_i is the total order of the derivative and n is the dimension of the physical domain.

     (b) If $j+1 > j_0$, for each node in V, a contribution counter is checked, to see whether the number of 'contributions' at that point matches the number of scaling functions on level $j+1$ that are non-zero at that point.

     → If there is a match, the node and its associated values, multiplied by C_i, are included, if not already present, in the output set of values, which is denoted by P.

     → Otherwise, the point is discarded.

     (c) Else, for each node in V, the node and its associated values, multiplied by C_i, are included in P.

     (d) The set V is cleared of all nodes and values.

  2. Make a vector of coefficients $s'_{j,\boldsymbol{k}'}$ to replace $s_{j,\boldsymbol{k}}\phi_{j,\boldsymbol{k}}$ by an expression of the form $2^{jn/2}\sum_{\boldsymbol{k}'} s'_{j,\boldsymbol{k}'}\phi(2^j\boldsymbol{x} - \boldsymbol{k}')$. Calculate also the set of nodes N where $\phi_{j,\boldsymbol{k}}$ should be non-zero.

3. For each node $\boldsymbol{l}/2^j \in N$,

   (a) If the node is not present in V, included it, associated with an array of derivative values set to zero and a contribution counter set to zero too.

   (b) Calculate the contribution of $s_{j,\boldsymbol{k}}\phi_{j,\boldsymbol{k}}$ from the values of ϕ and of its derivatives, and add it to the array of values.

   (c) Increment the contribution counter.

The number of operations involved in this algorithm can be decomposed as follows. The number of operations done per coefficient $s_\lambda \in S$ is bounded by a constant factor of $n+n_c\, n_d$, where n_c is the maximum number of translates and dilates of ϕ used in the composition of boundary-adapted scaling functions, and n_d is the number of derivatives required. In addition to this, for each node present in the set V there is a further (constant) number of operations, the calculation of the number of scaling functions whose support overlap at the node. The total number of nodes which have been in V at some point in the algorithm can be bounded by a constant number times the cardinality of S. Finally, for each node in the output set P, the algorithm requires one *insert* in the *map* P, n_d multiplications, and one search through P for each occurence of the node in V (as a node on dyadic grids of different levels). In summary, the number of operations required by this algorithm can be said to vary linearly with the number of scaling function coefficients present in S.

Finally, a specialized version of the above algorithm has been implemented to evaluate the scaling function expansion only at points situated in the nano-element. This is done by a simple check: if the support of $\phi_{j,\boldsymbol{k}}$ has an empty or trivial intersection with the rectangle Ω_{int} representing the nano-element, the term $s_{j,\boldsymbol{k}}\phi_{j,\boldsymbol{k}}$ is not evaluated.

6.4 Choice of a set of scaling function coefficients

The choice of a set of scaling function coefficients that will be used for the evaluation of a sparse wavelet expansion must be made in two situations. The first one is when the demagnetizing field needs to be evaluated, after a run of the wavelet Poisson solver. In this case, the choice of the coefficients must make the grid of point values representative of the local refinement of the wavelet expansion. The second situation is the case of an update of the grid where the magnetization is evaluated. The point of the update is that the magnetization must be evaluated at the same nodes as the demagnetizing field. In this case, the set of scaling function indices used for the demagnetizing field is also used for the magnetization. Although this last choice may be economical because one set of indices is calculated instead of two, it also makes the choice of this set a little more difficult because the magnetization is expanded in the dual wavelet basis, whereas the demagnetizing field is obtained by the evaluation of an expansion in the primal basis. This section is concerned with the choice of a set of scaling function indices that will be suitable for both situations.

To recapitulate, the choice of a set of scaling function coefficients must take care of the following issues:

1. There must be sufficiently many coefficients, and they must be organized in such a structure as to make the primal and dual local inverse transforms described previously correct and relatively efficient.

2. There must be sufficient overlap between successive levels of refinement to avoid the creation of gaps in the grid of point values. This must be the case for the evaluation of a primal or a dual expansion. At the same time, the overlap must be as small as

possible to avoid the repeated evaluation of certain nodes. This problem has been described in the previous section.

3. In practice, point values are only needed inside the nano-element, and therefore it would be efficient to choose, as far as possible, only those coefficients that will have a contribution to make to point values in that region of the domain.

4. The choice of the coefficients must make the grid of point values representative of the local refinement of the wavelet expansion. Finally, the grid of point values must contain all the nodes necessary for the calculation of the scaling function expansion of the Poisson equation's right hand side at the next time-step.

The third point is partly answered by a function called *TrimW* , which removes from D all the coefficients d_λ such that ψ_λ has an empty or trivial intersection with Ω_{int}. This solution is not completely satisfactory and for the needs of the inverse transform, the set S will still have to contain coefficients $s_{j,\boldsymbol{k}}$ such that $\text{supp}(\phi_{j,\boldsymbol{k}}) \cap \Omega_{int} = \emptyset$. These coefficients, however, will be ignored (after identification) by the specialized version of the local evaluation function.

The management of the remaining issues has been based on a set of *cubes*, very similar to the set of *good cubes* at the core of the process of projection onto the wavelet space, presented in chapter 4. It has already been mentioned, in relation to point 1 in the list above, that the set of scaling function coefficients must be organized around a tree structure. Let Λ_D denote the set of the wavelet indices present in the sparse wavelet expansion, as output from the *TrimW* function. Assuming that Λ_D is a tree, and that Λ_G is the associated set of *good cubes* , or equivalently the set of leaves of that tree ($\Lambda_G = \partial\Lambda_D$),

the function *GenLocSet* defined in section 4.3.2 would create a set of scaling function indices with an acceptable structure. Moreover, the optional input parameters of the same *GenLocSet* can be used to specify the extent of the overlap between two different levels of refinement, thus answering the second part of point 1 and point 2.

In fact, there is no guarantee that Λ_D is a tree, so instead, Λ_G is defined as the set of leaves of the smallest tree containing Λ_D. The calculation of Λ_G is done by a function called *AssocGoodCubes*, which also calculates the complement of Λ_G in the tree. This complement is here denoted by $\Lambda_{G'}$. Next, the tree $\Lambda_G \cup \Lambda_{G'}$ is graded by a function called *GradeTree*. Then, it is possible to use the *GenLocSet* function to deduce from Λ_G the set of scaling indices Λ_S. The next two subsections will present the *AssocGoodCubes* function and the *GradeTree* function, while the last section will detail the calculation of the correct values for the *GenLocSet* input parameters that determine the extent of the overlap between two refinement levels.

An algorithm for the *AssocGoodCubes* function

The *AssocGoodCubes* has for input Λ_D and for output the sets Λ_G and $\Lambda_{G'}$, such that $\Lambda_G \cap \Lambda_{G'} = \emptyset$ and $\Lambda_G \cup \Lambda_{G'}$ is the smallest tree containing Λ_D.

1. Initialization of Λ_G with the set of dyadic nodes on level $j_0 - 1$ in the unit cube $[0,1]^n$.
2. For each wavelet index $\lambda \in \Lambda_D$, proceeding by increasing level λ,
   (a) Search Λ_G for cube $\square_\lambda$ or for an antecedent $\square_\mu \supset \square_\lambda$, $|\mu| < |\lambda|$, of that cube.
   (b) If $\square_\lambda \in \Lambda_G$, proceed to the next index in Λ_D.

(c) Else remove $\square_\mu$ from Λ_G and insert it into $\Lambda_{G'}$ instead. For increasing levels j from $|\mu|+1$ to $|\lambda|-1$,

i. Insert into $\Lambda_{G'}$ the child of $\square_\mu$, which is also the parent on level j of cube $\square_\lambda$.

ii. Insert into Λ_G the remaining of the children of $\square_\mu$.

iii. Reset $\square_\mu$ as the parent of $\square_\lambda$ on level j. Increment j.

(d) When $j = |\lambda|$, insert all the children of $\square_\mu$ into Λ_G.

The ouput sets Λ_G and $\Lambda_{G'}$ can be substantially larger than the input set Λ_D, and clearly all the more so that the structure of Λ_D is remote from that of a tree, since if Λ_D only contains a few wavelet indices on a high level, the union $\Lambda_G \cup \Lambda_{G'}$ also contains all their antecedents.

The *GradeTree* function

The *GradeTree* function has four input variables: the complementing sets Λ_G and $\Lambda_{G'}$, and two integers m_1 and m_2. In each direction i, $1 \leq i \leq n$, *GradeTree* grades $\Lambda_G \cup \Lambda_{G'}$ by m_1 leaves (on level $j-1$) before a region on level j and by m_2 leaves after, where j varies between the finest level in Λ_G and the coarsest level +2. One cube can be added in addition to the specified m_1 or m_2 in order for cubes on level j to exactly cover a certain number of cubes on level $j-1$. At the end of *GradeTree*, Λ_G is the set of leaves of the graded tree and $\Lambda_{G'}$ is its complement.

The calculation of the extent of the overlap between two levels

The *GenLocSet* function works by including in the set of scaling function indices Λ_S the children of all the cubes present in the set of *good cubes* Λ_G. Then, the 'patches' of consecutive cubes on the same level j are augmented, in all directions where the patch borders a patch on a coarser level, by L cubes before the patch and by M cubes after it. In order to calculate the values of L and M that will satisfy all the constraints imposed in the set Λ_S, it is simpler to reason in the one-dimensional setting. The values of L and M will be calculated independently for each constraint and then, out of the values calculated, the largest ones will be retained.

Suitable L and M for a correct inverse transform. Let $[l_1, l_2]$ denote the support of the primal mother scaling function and $[\tilde{l_1}, \tilde{l_2}]$ denote the support of $\tilde{\phi}$. In this paragraph, values for L and M are calculated to make the primal inverse transform correct. The calculation for the dual inverse transform follows the same lines, but $\tilde{l_1}$ and $\tilde{l_2}$ are then used instead of l_1 and l_2. The inverse transform will be correct if the set $S_j \cap S'_j$ contains all the scaling function coefficients on level j needed for the reconstruction of the scaling function coefficients in S_{j+1}. Because of the structure imposed on the sets S_j and S'_j, this requirement can be expressed as a condition on the length of the grading of the tree Λ. Before an overlap is added, Λ is already graded with a coefficient denoted by G. From the reconstruction equations, it can be observed that the calculation of the coefficient $s_{j+1,m}$ requires the knowledge of the coefficients $s_{j,k}$ for

$$\left\lfloor \frac{m - l_2 + 1}{2} \right\rfloor \leq k \leq \left\lceil \frac{m - l_1 - 1}{2} \right\rceil, \tag{6.1}$$

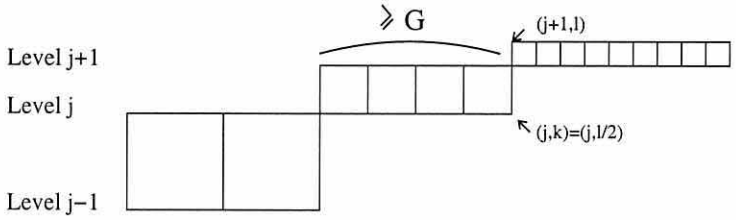


Figure 6.2: *Part of G-graded tree Λ before addition of the left overlap L*

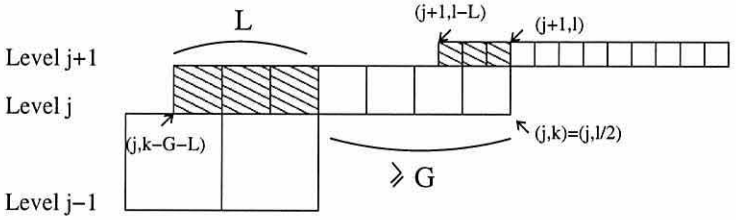


Figure 6.3: *Part of G-graded tree Λ after addition of the left overlap L*

where $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x. Let us consider a section of Λ where the indices are on level $j+1$, and let $(j+1, l)$ denote the index situated at the left end of that section. After a hypothetical overlap of length L has been added to the tree Λ, the end of the section on level $j+1$ has been moved left to the index $(j+1, l-L)$. According to (6.1), for the correct reconstruction of the coefficient $s_{j+1,l-L}$, the coefficient most to the left that S_j must contain has for index $(j, \lfloor (l-L-l_2+1)/2 \rfloor)$. On the other hand, as illustrated on Figure 6.3, the index at the left end of the section on level j, to the left of the section on level $j+1$, is $(j, k-G-L)$, where $k = l/2$. As a consequence, the following relation on L has been established

$$\frac{l}{2} - G - L \leq \left\lfloor \frac{l-L-l_2+1}{2} \right\rfloor.$$

By rearranging this inequality, it can be shown that a sufficient condition on L for the inverse transform to be correct is

$$L \geq l_2 + 1 - 2G.$$

A similar calculation for the length of the overlap added on the right hand side gives the following sufficient condition on M

$$M \geq -l_1 - 2(G-1).$$

L and M for a grid without gaps. As for the previous paragraph, the primal case is explained in details and the dual case can be deduced by writing $\tilde{l_1}$ and $\tilde{l_2}$ instead of l_1 and l_2. First, the calculation of the overlap on the left hand side. The notation of the previous paragraph is used again here. The last coefficient at the right end of the section on level j, has for index $(j, k-1)$. The scaling function associated with that coefficient has for support $2^{-j}[k-1+l_1, k-1+l_2]$ and so the last node on the right, where the local scaling function expansion on level j can be evaluated, is $x_j = 2^{-j}(k+l_1)$. In order to avoid a gap in the grid of point values, the first node where the local scaling function expansion on level $j+1$ must be evaluated is the next node along, $x_{j+1} = 2^{-(j+1)}m$, where $m = 2(k+l_1+1)$. In order to be able to calculate the value of a local expansion on level $j+1$ at point x_{j+1}, the scaling function $s_{j+1,p}$, where $m = p+l_2-1$, must be contained in S_{j+1}. Another expression for p is

$$p = m - l_2 + 1 = 2k + 2(l_1+1) - l_2 + 1.$$

The index at the left end of the section on level $j+1$, before the addition of an overlap, is $(j+1, 2k)$. The index at the end of the section, after the addition of the overlap, must be at least as far left as $(j+1, p)$. This imposes the following constraint on L:

$$L \geq 2k - p \qquad \text{and} \qquad 2k - p = -2(l_1 + 1) + l_2 - 1.$$

A similar calculation for the length of the right hand side overlap gives the following sufficient condition for M:

$$M \geq 2(l_2 - 1) - l_1 - 2.$$

A good grid of point values for the projection function *GenLocRHS* In the overall time-stepping scheme, at each iteration, the primal local evaluation function is used to calculate the values of $\boldsymbol{h}_d^n$ and the dual local evaluation function is used to calculate the values of $\boldsymbol{m}_n$ and $\boldsymbol{h}_{ex}^n$. The local scaling function expansions evaluated all have the same index set Λ_S. The set Λ_S is calculated by the *GenLocSet* function, given the set of good cubes Λ_G and two overlap parameters L and M. The values of $\boldsymbol{m}_{n+1}$ are calculated on the grid of values that is the intersection of the grid calculated by the primal and the dual evaluation functions. At the next iteration, the Poisson solver is initiated by the calculation of the local scaling function expansion of the right hand side. This is done by the *GenLocRHS* function, which requires in input the grid of point values of $\boldsymbol{m}_{n+1}$. This imposes a constraint on this grid. The scaling function coefficients calculated in *GenLocRHS* are associated with a set of local scaling function indices, denoted here by Λ_R. The set Λ_R, like Λ_S, is calculated by *GenLocSet* and from the same set of good cubes Λ_G. The overlap parameters used in that case are the default values, which are $L1 = \max(|\, l_1|, |^w l_1|)$ and

$M1 = \max(l_2, {}^w l_2)$. Here, ${}^w l_1$ and ${}^w l_2$ are the indices of the first and last non-zero entries in the wavelet mask ($\psi(x) = \sum_{l={}^w l_1}^{{}^w l_2} b_l \, \phi(2x - l)$), and of course l_1 and l_2 are the indices of the first and last non-zero entries in the refinement mask ($\phi(x) = \sum_{l=l_1}^{l_2} a_l \, \phi(2x - l)$).

The first observation that matters here, is that for a common index set Λ_S, the grid of point values produced by the dual evaluation routine is a subset of the grid produced by the primal evaluation routine because the dual scaling functions have a longer support ($\tilde{l_1} \leq l_1$ and $\tilde{l_2} \geq l_2$, [12]). As a consequence, it is sufficient to calculate L and M such that the grid produced by the dual evaluation routine is correct for *GenLocRHS*.

The second observation is that *GenLocRHS* needs $\tilde{d} + 1$ point values on level j to calculate a scaling function coefficient $\tilde{s}_{j,k}$. Depending on the location of (j, k) with respect to the inner domain Ω_{int}, these points must be either centered about the node (j, k), or they may all be situated to the left or to the right side of node (j, k) if node (j, k) is situated on the boundary of Ωint.

The calculation of a suitable value for L starts with considering the scaling function index situated at the end of a section on level j of the set of indices Λ_R, after the overlap L_1 has been added. This index is denoted by (j, l). Among the point values that may be needed for the calculation of $\tilde{s}_{j,l}$ by *GenLocRHS*, the one situated most to the left is associated with the point $x_j = (j, l - \tilde{d})$. Then, among the scaling function coefficients from Λ_S that will be needed to evaluate a dual scaling function expansion on level j at point x_j, the one 'situated' most to the left has for index (j, p), where $l - \tilde{d} = p + \tilde{l_2} - 1$. Another expression for p is $p = l - \tilde{d} - \tilde{l_2} + 1$. Then, writing L as $L_1 + L_2$, this imposes the following condition

on $L2$:

$$L_2 \geq l - p \qquad \text{and} \qquad l - p = \tilde{d} + \tilde{l_2} - 1.$$

As a consequence, a sufficient condition on L is

$$L \geq \max(|\, l_1|, |^w l_1|) + \tilde{d} + \tilde{l_2} - 1.$$

A similar calculation for the right hand side gives the following condition on M

$$M \geq \max(l_2, ^w l_2) + \tilde{d} - \tilde{l_1} - 1.$$

Finally, the values for L and M used in the calculation of Λ_S are the smallest values that satisfy the three sets of conditions established above. In practice, the last set of conditions dominates the other two.

Chapter 7

Numerical tests on the Poisson solver

The suitability of the adaptive scheme described in Chapter 3 will be assessed in this part. An important tool in the assessment of the accuracy of the scheme will be the knowledge of a closed form analytical expression for the solution of the Poisson equation corresponding to the first iteration of the evolutive scheme, for a rectangular nano-element and a uniform initial magnetization $\boldsymbol{m} = (0, 1, 0)$. The choice of certain parameters such as a suitable *maximum radius* for the derivative operator matrix and the number of degrees of freedom involved will also motivate these numerical tests. Finally, the number of operations and the size of the memory storage needed will be studied with respect to the number of degrees of freedom.

7.1 An analytical solution

When the magnetization is $\boldsymbol{m} = (0, 1, 0)$, the solution of equation 1.17 inside a magnetic region of rectangular shape $c \leq x_1 \leq a$, $d \leq x_2 \leq b$, is given in [25] by the formula

$$\begin{aligned} u(x_1, x_2) \;=\; & 2\left[\left(\frac{x_1 - a}{2}\right)\ln\left(\frac{(x_1 - a)^2 + (b - x_2)^2}{(x_1 - a)^2 + (x_2 - d)^2}\right) + (b - x_2)\tan^{-1}\left(\frac{x_1 - a}{b - x_2}\right)\right. \\ & \left. - (x_2 - d)\tan^{-1}\left(\frac{x_1 - a}{x_2 - d}\right)\right] \\ & -2\left[\left(\frac{x_1 - c}{2}\right)\ln\left(\frac{(x_1 - c)^2 + (b - x_2)^2}{(x_1 - c)^2 + (x_2 - d)^2}\right) + (b - x_2)\tan^{-1}\left(\frac{x_1 - c}{b - x_2}\right)\right. \\ & \left. - (x_2 - d)\tan^{-1}\left(\frac{x_1 - c}{x_2 - d}\right)\right] \end{aligned}$$

In the numerical tests presented further, $\boldsymbol{m}$ will always be equal to $(0, 1, 0)$ and the nano-element will be a rectangle $[c, a] \times [d, b]$, where $c = 32/64$, $a = 33/64$, $d = 28/64$ and $b = 36/64$ and the error made by the numerical calculation will be evaluated from the analytical expression above.

7.2 Tests with the usual matrix-vector multiplication

In a first series of tests, the adaptive Poisson algorithm is run with $N = 2^{10}$, $\epsilon = 10^{-6}$ and for increasing values of the maximum radius $r = 0, \ldots, 3$. Every matrix-vector multiplication is calculated in the usual manner.

7.2.1 The right hand side f

The right hand side of the equation is calculated with *GenLocRHS*. It has originally 1172 wavelet coefficients but it is limited to its 2^{10} terms of largest modulus before the start of

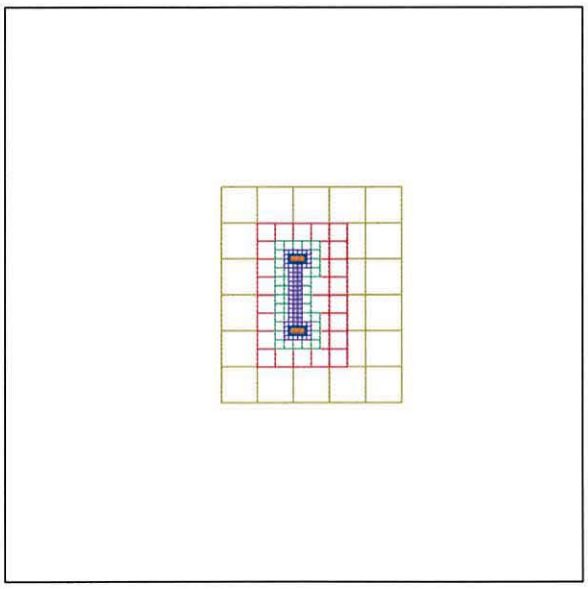


Figure 7.1: *Distribution of the coefficients of* $\boldsymbol{f}$ *over* Ω. *Colour code for the coefficient levels: khaki=4, red=5, green=6, purple=7, blue=8, brown=9, orange=10.*

the scheme. The wavelet coefficients present in $\boldsymbol{f}$ are spread between levels 4 and 10 and their distribution over the domain is represented on graphs 7.1-7.3, where every square represents a cube $\square_\lambda$, meaning that at least one of the four coefficients with level $|\lambda|$ and position vector $\boldsymbol{k}(\lambda)$ is present in $\boldsymbol{f}$. The level of the coefficients is colour coded as well as being represented by the squares sizes. On figure 7.1, the frame represents the boundary of Ω, while the nanoelement, situated in the center of the graph, is represented by a rectangle shaded in gray. It can be seen on the close-up pictures that the coefficients of highest level are situated about the top and bottom edges of the nano-element. The size of the nano-element has been chosen small enough to minimize the effect of the simulation of Ω by a bounded domain and for larger nano-elements.

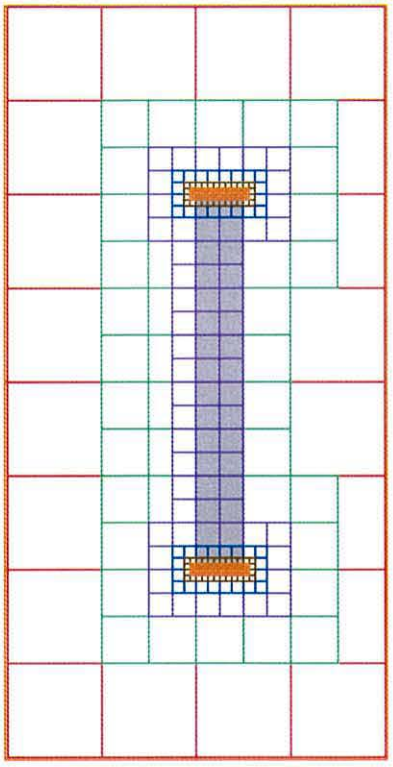


Figure 7.2: *Distribution of the coefficients of* $\boldsymbol{f}$ *over* Ω. *Close-up on the nano-element. Colour code for the coefficient levels: red=5, green=6, purple=7, blue=8, brown=9, orange=10.*

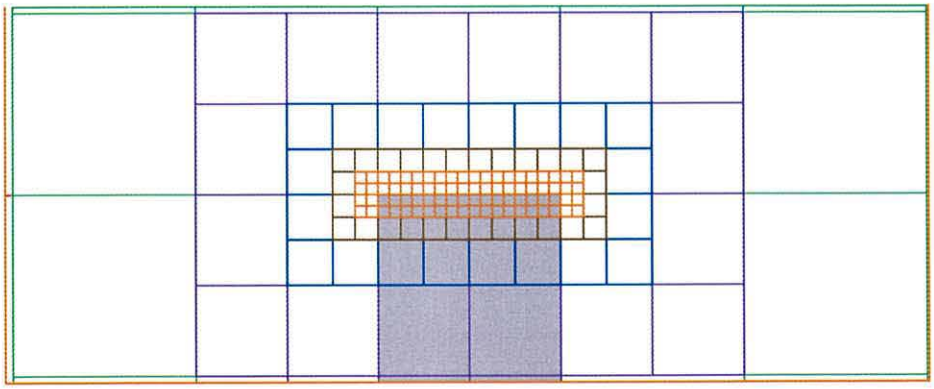


Figure 7.3: *Distribution of the coefficients of* $\boldsymbol{f}$ *over* Ω. *Close-up on the top edge of the nano-element. Colour code for the coefficient levels: green=6, purple=7, blue=8, brown=9, orange=10.*

7.2.2 Test results

For each run of the Poisson solver with a maximum radius r, the following data has been collected on table 7.4:

Iter. : the number of iterations taken.

Init. Step : the number of Initial Steps taken.

$\#\boldsymbol{u}_j$: the number of entries in the vector $\boldsymbol{u}_j$ at the last iteration and before the compression step.

$\#(\boldsymbol{A}_r\boldsymbol{u}_j)$: the number of entries in the vector $\boldsymbol{A}_r\boldsymbol{u}_j$, where $\boldsymbol{u}_j$ is the solution before the compression step. This gives an upper bound for the size of the output of the matrix-vector multiplications.

$\|\boldsymbol{A}_r\boldsymbol{u}_j - \boldsymbol{f}\|$: the residual calculated as the norm-2 of the vector of wavelet coefficients $\boldsymbol{A}_r\boldsymbol{u}_j - \boldsymbol{f}$.

$\|\boldsymbol{A}_r\boldsymbol{u}_f - \boldsymbol{f}\|$: the residual calculated after compression of the solution.

$\|E_u\|$: the relative error on the potential, calculated by the formula

$$\|E_u\| = \left(\frac{\sum_{\boldsymbol{x}\in G(\Lambda_u)} |u_f(\boldsymbol{x}) - u(\boldsymbol{x})|^2}{\sum_{\boldsymbol{x}\in G(\Lambda_u)} |u(\boldsymbol{x})|^2} \right)^{1/2},$$

where $G(\Lambda_u)$ denotes a set of points inside Ω_{int}, $u_f(\boldsymbol{x})$ is the evaluation of the wavelet expansion $\boldsymbol{u}_f$ and u is the analytical solution.

Looking first at the accuracy of the potential calculated numerically, table 7.4 shows that for all the tests, the size of the residuals for the problem $\boldsymbol{A}_r\boldsymbol{u} = \boldsymbol{f}$, calculated before or after the compression of the solution, is of the order of $\sqrt{\epsilon}$. The relative error E_u decreases mostly between the tests done with $r = 0$ and $r = 1$, and less so for higher values of r. This may be related to the inaccuracy caused by the wavelet representation of the right hand side and the limitation to N degrees of freedom of the representation of the descent direction $\boldsymbol{w}_j$. Even so, the size of E_u seems to be acceptable from an applications point of view [25]. The number of iterations needed for a stopping parameter $\epsilon = 10^{-6}$ is fairly small, considering the size of the system solved. These are positive points, which need to be balanced against the resources involved by the scheme. It seems clear from the values of $\#\boldsymbol{w}$ that the memory space needed to store the variables is too large by comparison with the target number of unknowns $N = 1024$. Also, the number of Initial Steps taken by each of the tests is very large, since such a step is taken more often than every second step. As explained in section 3.4, this has a serious adverse consequence on the number of operations performed by the solver. The next two sections show the results of tests made on modified versions of the solver in the view of reducing the resources used. In the first section, the matrix vector multiplication is replaced by a modified version of the ***MULT*** algorithm, while in the second section, the Poisson solver scheme is 'simplified' by removing the convergence check on d and thereby disabling any second calls to the Initial Step.

7.3 Tests with an approximate matrix-vector multiplication

The logical replacement for the usual matrix vector multiplication is the ***MULT*** algorithm. It has been found however that in this particular case, the parameter k calculated by my implementation of ***MULT*** was always so large that there would be in effect no difference with a normal multiplication. The modification used in this series of tests consists in fixing the parameter k to $\log_2(N)$. A maximum radius r is still imposed, and everything apart from the mutiplication routine being kept as in the previous section, the results resented on table 7.5 have been obtained.

These results show that on this example the residuals for the problem $\boldsymbol{A}_r\boldsymbol{u} - \boldsymbol{f}$ are of the same order as those calculated with a normal matrix vector multiplication, and more importantly, the size of the relative error E_u has not increased. The number of entries in the output of the matrix vector multiplication has been reduced significantly. The number of Initial Steps used is as large as previously.

In the next series of tests, a 'quick' adaptive scheme is used, where the Initial Step is used only once, disregarding the size of the parameter d. The matrix vector multiplications are preformed with the modified ***MULT*** algorithm. The results are presented on table 7.6. Because only one Initial Step is taken in this case, the length of the output of $\boldsymbol{A}_r\boldsymbol{u}_j$ is not a good indicator of the memory needed for the scheme, for this reason, table 7.6 also presents under the heading $\#(\boldsymbol{A}_r\boldsymbol{w}_j)$ the maximum length of the output of a matrix multiplication in the scheme. Clearly, the 'quick' scheme reduces the number of operations (only one matrix-vector multiplication, and the input vector has always length

N) and the size of the storage needed per iterations. Another positive point is that the difference between the residual calculated with solution before and after compression is much smaller thatn with the original scheme, suggesting that less effort is wasted by the solver. On this example, by comparison with the tests made on the original scheme (table 7.4), a deterioration of the relative error can be noted: from 2.73E-02 to 2.81E-02 when $r = 1$ and from 2.36E-02 to 2.68E-02 when $r = 2$. However, because the size of E_u is still of the same order and because the resources involved have been so much reduced, it seems reasonable to favor the 'quick' scheme to the original one, also all the tests presented from now on use the 'quick' algorithm and the modified $\boldsymbol{MULT}$ algorithm.

The next tests aim at checking whether decreasing the threshold parameter ϵ would improve the errors E_u. The results on table 7.7 show that when ϵ is decreased from 10^{-6} to 10^{-9}, the relative errors decrease slightly, at the expense of a three to four-fold increase in the number of iterations.

From the reduction in E_u on table 7.6, it appears that a maximum radius of at least 1 must be used. Also, when $N = 1024$, it seems that there is no advantage in taking r greater than 2. The same tests have been repeated with $N = 2048$ and it can be observed on table 7.8 that in this case too, the most important reduction of the relative error E_u is achieved by the increase of the maximum radius from 0 to 1. The representation of the right hand side of the equation with N wavelet coefficients is another factor of the difference between the solution calculated by the scheme and the solution to the infinite Poisson problem. The results presented on table 7.8 have been obtained with the same vector $\boldsymbol{f}$ as all the results presented so far. This vector contains only 1024 non-zero entries. When

running the same test, with $r = 1$ and a vector $\boldsymbol{f}$ of length 2048, the relative error can be seen to decrease from 2.86E-02 to 2.68E-02. The effect of the compression step at the end of the scheme on the size of the residual has been reduced in the 'quick' algorithm, however in the tests done with the 'quick' algorithm, this step still reduces the number of wavelet coefficients used to represent the solution approximately by half. The coefficients that are removed are mainly situated in the non-magetic part of the domain. The coefficients present in the solution before and after compression are represented on graphs 7.9 to 7.11.

As a summary, it appears that for this example, the 'quick' adaptive algorithm converges to an approximate solution with an acceptable accuracy. The number of iterations necessary is small and when the modified ***MULT*** algorithm is used for the matrix vector multiplications, the size of the vectors used remains proportional to the target number of degrees of freedom N, the multiplicative factor being about 10, when $r = 1$. The choice of $r = 1$ for the value of the maximum radius seems to give the best compromise between the accuracy of the solution and the cost of the scheme in terms of memory usage and operations count. In the next section, this algorithm and the chosen parameters are put to the test in the coupling of the Poisson solver with the evolutive Landau-Lifshitz equation.

r	Iter.	Init. Step	$\#\boldsymbol{u}_j$	$\#(\boldsymbol{A}_r\boldsymbol{u}_j)$	$\|\boldsymbol{A}_r\boldsymbol{u}_f - \boldsymbol{f}\|$	$\|\boldsymbol{A}_r\boldsymbol{u}_j - \boldsymbol{f}\|$	$\|E_u\|$
0	35	21	2965	11544	1.03E-03	5.30E-03	5.63E-02
1	26	20	5295	82882	1.30E-03	5.79E-03	2.73E-02
2	26	20	6267	421398	1.39E-03	4.95E-03	2.36E-02
3	26	20	6227	1404883	1.51E-03	4.42E-03	2.37E-02

Figure 7.4: *Tests with unmodified MULT multiplication and* $N = 1024$.

r	Iter.	Init. Step	$\#\boldsymbol{u}_j$	$\#(\boldsymbol{A}_r\boldsymbol{u}_j)$	$\|\boldsymbol{A}_r\boldsymbol{u}_f - \boldsymbol{f}\|$	$\|\boldsymbol{A}_r\boldsymbol{u}_j - \boldsymbol{f}\|$	$\|E_u\|$
1	26	20	5441	36320	1.21E-03	6.14E-03	2.69E-02
2	26	21	6447	51913	1.21E-03	6.25E-03	2.37E-02
3	26	21	6430	66445	1.28E-03	6.48E-03	2.38E-02
4	26	21	6291	76352	1.28E-03	6.61E-03	2.38E-02

Figure 7.5: *Tests with a modified algorithm* $\boldsymbol{MULT}$ *and* $N = 1024$.

r	Iter.	Init. Step	$\#\boldsymbol{u}_j$	$\#(\boldsymbol{A}_r\boldsymbol{u}_j)$	$\|\boldsymbol{A}_r\boldsymbol{u}_f - \boldsymbol{f}\|$	$\|\boldsymbol{A}_r\boldsymbol{u}_j - \boldsymbol{f}\|$	$\|E_u\|$	$\#(\boldsymbol{A}_r\boldsymbol{w}_j)$
0	32	1	2390	11247	1.40E-03	1.65E-03	5.43E-02	8186
1	20	1	2438	23234	1.39E-03	1.48E-03	2.81E-02	11871
2	21	1	2394	41083	1.48E-03	1.56E-03	2.68E-02	20031
3	21	1	2371	49454	1.50E-03	1.55E-03	2.65E-02	22194

Figure 7.6: *Tests with a modified algorithm* $\boldsymbol{MULT}$*, the 'quick' adaptive scheme and* $N = 1024$.

r	Iter.	$\|E_u\|$
1	67	2.68E-02
2	80	2.57E-02

Figure 7.7: *Tests with* $\epsilon = 10^{-9}$ *and* $N = 1024$.

r	Iter.	Init. Step	$\#\boldsymbol{u}_j$	$\#(\boldsymbol{A}_r\boldsymbol{u}_j)$	$\|\boldsymbol{A}_r\boldsymbol{u}_f - \boldsymbol{f}\|$	$\|\boldsymbol{A}_r\boldsymbol{u}_j - \boldsymbol{f}\|$	$\|E_u\|$	$\#(\boldsymbol{A}_r\boldsymbol{w}_j)$
0	33	1	4015	15395	5.25E-03	5.89E-03	4.98E-02	13695
1	22	1	4112	45471	6.97E-03	7.60E-03	2.86E-02	17921
2	22	1	4195	80054	8.22E-03	9.30E-03	2.58E-02	25792
3	22	1	4167	90284	8.12E-03	8.80E-03	2.58E-02	33085

Figure 7.8: *Tests with a modified algorithm* $\boldsymbol{MULT}$*, the 'quick' adaptive scheme and* $N = 2048$.

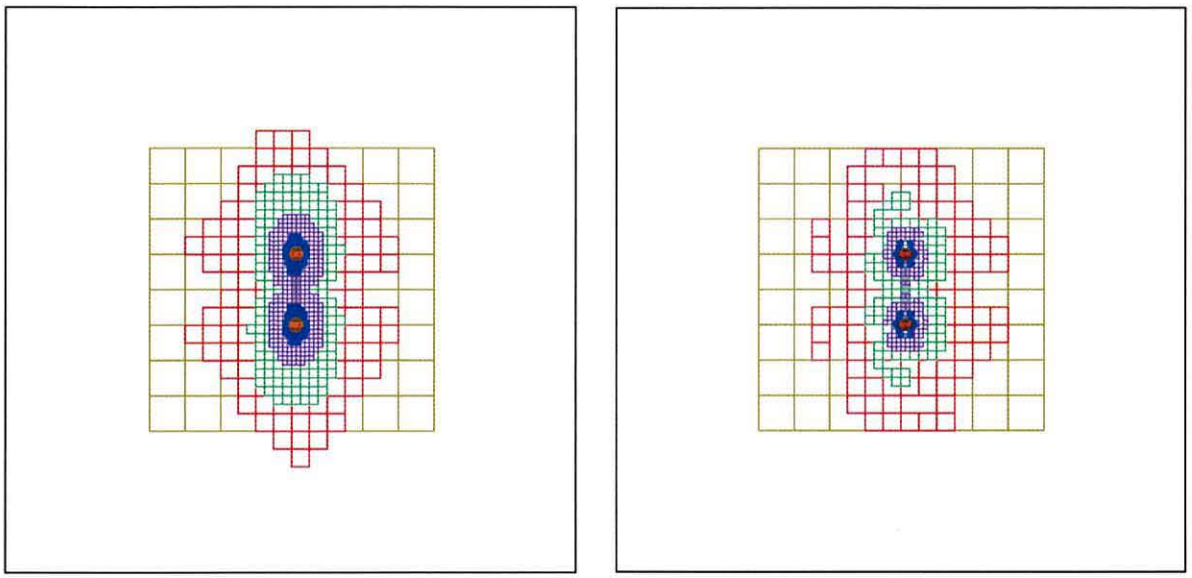


Figure 7.9: *Distribution of the coefficients of the solution before (left) and after (right) compression. Colour code for the coefficient levels: khaki=4, red=5, green=6, purple=7, blue=8, brown=9, orange=10.*

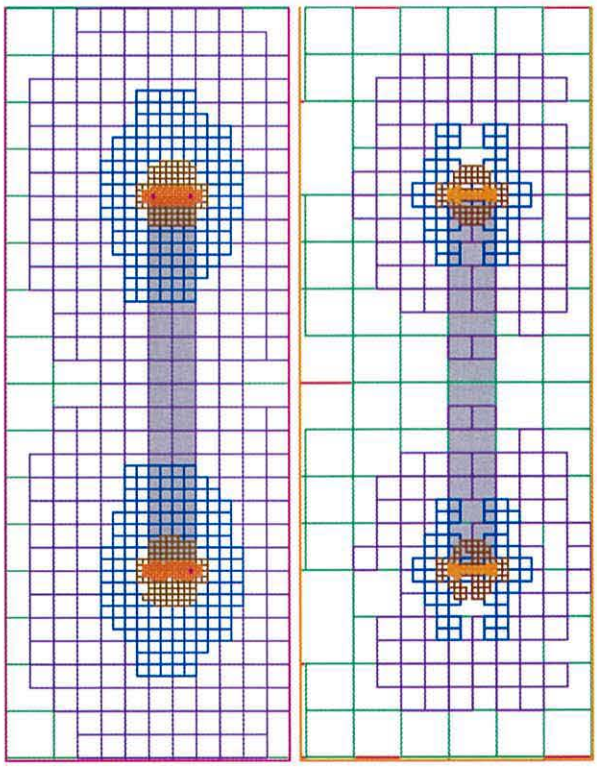


Figure 7.10: *Distribution of the coefficients of* $\boldsymbol{u}_j$ *(left) and* $\boldsymbol{u}_f$ *(right). Close-up on* Ω_{int}*. Colour code for the coefficient levels: red=5, green=6, purple=7, blue=8, brown=9, orange=10, pink =11.*

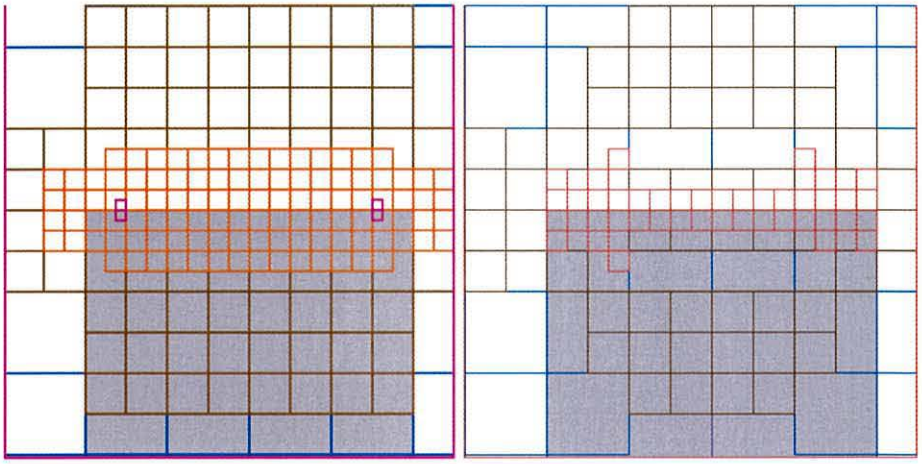


Figure 7.11: *Distribution of the coefficients of* $\boldsymbol{u}_j$ *(left) and* $\boldsymbol{u}_f$ *(right). Close-up on the top edge of* Ω_{int}*. Colour code for the coefficient levels: blue=8, brown=9, orange=10, pink =11.*

Chapter 8

Preliminary results

In this chapter, two simulations of the evolution of the magnetization of a rectangular nanoelement are presented. In both simulations the magnetization is initially uniform, parallel to the x_2-axis, which is the direction of the length of the nanoelement, $\boldsymbol{m}_0 = (0, 1, 0)$. The anisotropy vector is parallel to the x_1-axis, the direction of the width of the nanoelement, $\boldsymbol{h}_a = (1, 0, 0)$. The applied field, responsible for the initial magnetization, is assumed to be zero at all times $\tau > \tau_0$. The remaining physical parameters are given the following values: the anisotropy constant is fixed to $K = 5.10^5\, erg/cm^3$, the saturation magnetization is set to $M_S = 800\, emu/cm^3$ and the dissipative constant is unity ($\alpha = 1$). The two simulations presented differ by the value given to the exchange constant A. In the first simulation $A = 10^{-5}\, erg/cm$, and in the second simulation $A = 10^{-6}\, erg/cm$. A larger value for the exchange constant A implies that larger sections of the magnetic material behave as separate units. The numerical parameters are given the following values. The parameters needed for the Poisson solver are given the values accepted at the end of the previous chapter: $r = 1$ and $N = 2048$. The size of the time-step $\Delta\tau$ used in the Euler scheme was worked out empirically by running the programme for decreasing values of $\Delta\tau$ until

numerical stability was achieved. The time-step for the first simulation is set to $\Delta\tau = 10^{-2}$ and for the second simulation to $\Delta\tau = 5.10^{-2}$. After the choice of the time-step values, because the Euler scheme does not conserve the magnetization strength, the programme was modified to normalize the magnetization pointwise at every time-step.

Figures 8.1-8.4 and 8.6-8.8 represent the evolution of the magnetization inside the nanoelement, as calculated by the two simulations. For each simulation, the first graphs correspond to times τ fairly near to one another and close to the initial time $\tau_0 = 0$, and they show the most obvious changes in the magnetization of the nanoelement. The last graphs correspond to later times, they are more spread out in time and they show how the magnetization settles in a given configuration. The formation of vortices and later domains can be observed. The evolution of the distribution of the wavelet coefficients used during the two simulations is also graphically represented on Figures 8.5 and 8.9. The graphs focus on the area of the nanoelement. As the first simulation takes place, refinement occurs over the whole nanoelement (the gray area on the graph), and more particularly along the vertical edges. Coarsening occurs away from the nanoelement and also at its top and bottom edges. As the second simulation takes place, refinement can again be observed over the entire nanoelement and along its vertical edges, but also in a certain measure, along horizontal accumulations across the nanoelement. It may be observed also from the arrow plots 8.1-8.4 and 8.6-8.8 that the domain walls formed during the second simulation are rather more narrow than those formed during the first simulation. The horizontal accumulation of wavelet coefficients in this case may be due to the domain walls. Finally, a general observation is that the displacement of the wavelet refinement during both simulations seems fairly slow. Consequently, the efficiency of the scheme could be improved

by alternating a succession of non-adaptive time-steps with an adaptive one. During the non-adaptive time-steps, the Poisson solver would follow a Galerkin scheme on a fixed wavelet subspace of $L^2(\Omega)$. The efficiency gain would occur in the Poisson solver and at the update stage, when the magnetisation $\boldsymbol{m}_{n-1}$ is updated to be evaluated at the nodes where $\boldsymbol{m}_n$ is calculated. In particular, a scheme more accurate than the Euler scheme, such as a scheme where the calculations of $\boldsymbol{m}$ at time-steps $\tau_{n'}, n' < n-1$ are used to calculate $\boldsymbol{m}_n$, could be all the more efficiently implemented, since the update step would be less frequently required.

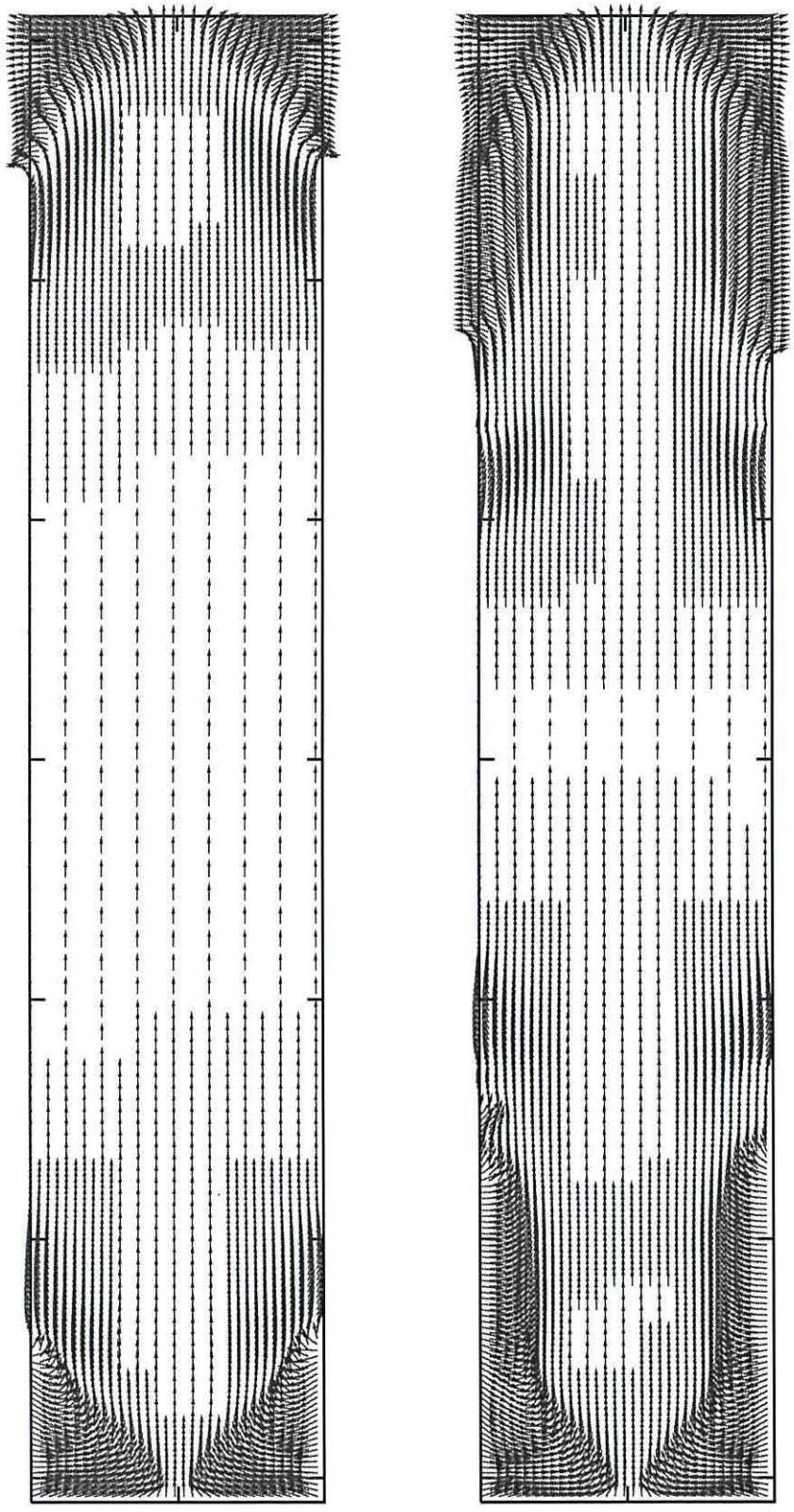


Figure 8.1: *Simulation 1.* Magnetization of the nanoelement at times $\tau = 1.18$ (left) and $\tau = 2.20$ (right)

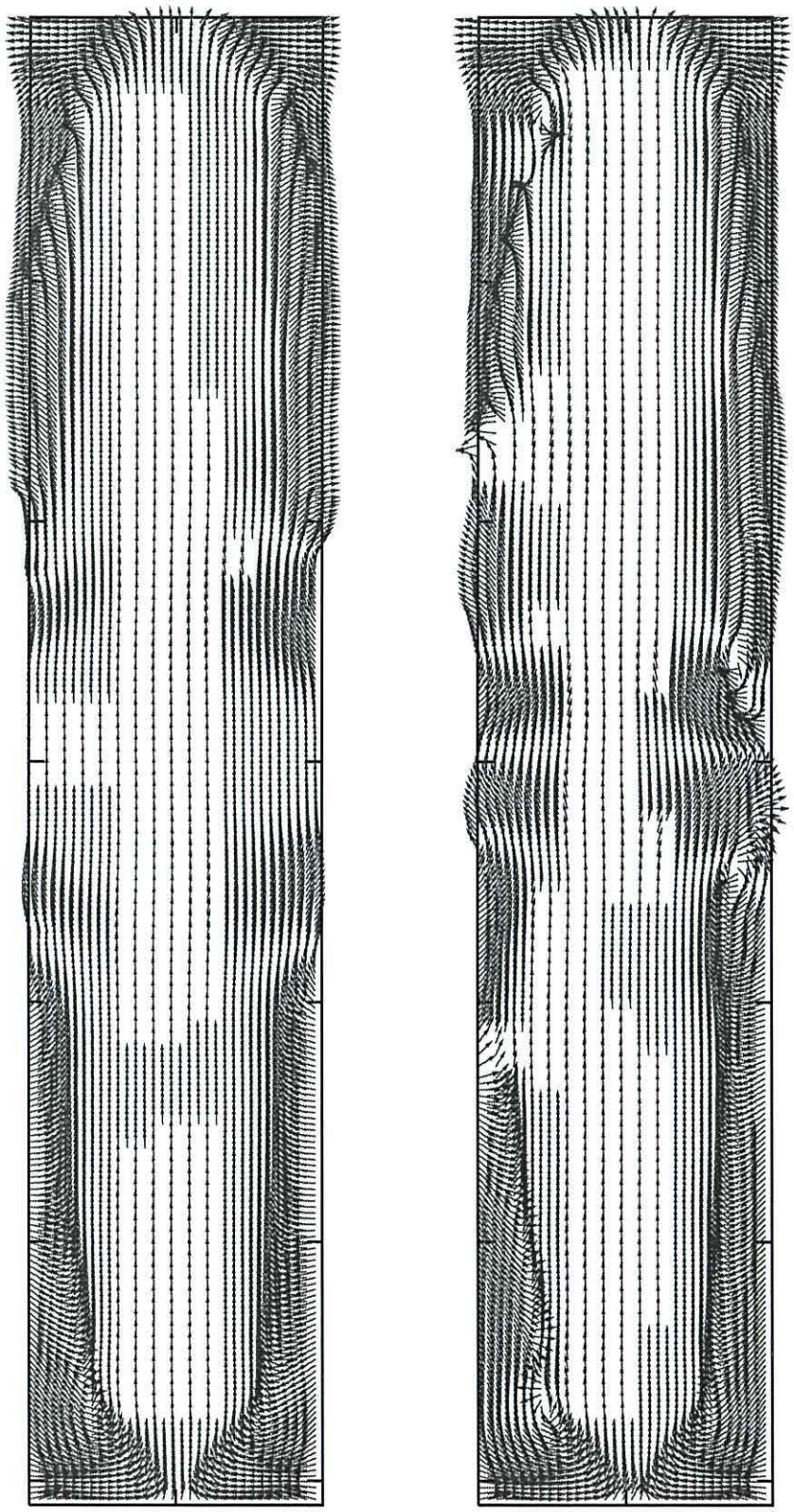


Figure 8.2: *Simulation 1.* Magnetization of the nanoelement at times $\tau = 3.0$ (left) and $\tau = 3.7$ (right)

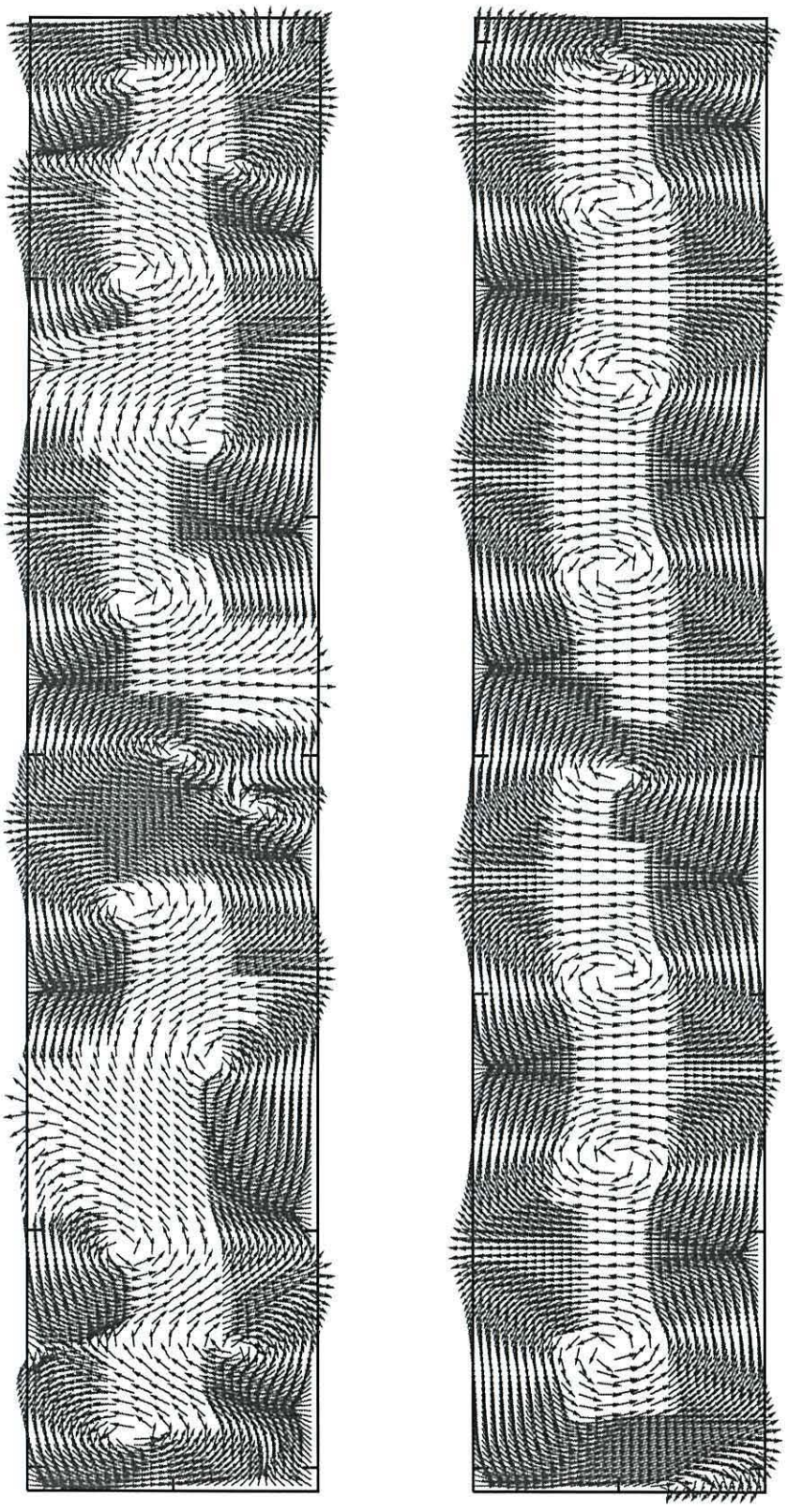


Figure 8.3: *Simulation 1.* Magnetization of the nanoelement at times $\tau = 8.6$ (left) and $\tau = 26.7$ (right)

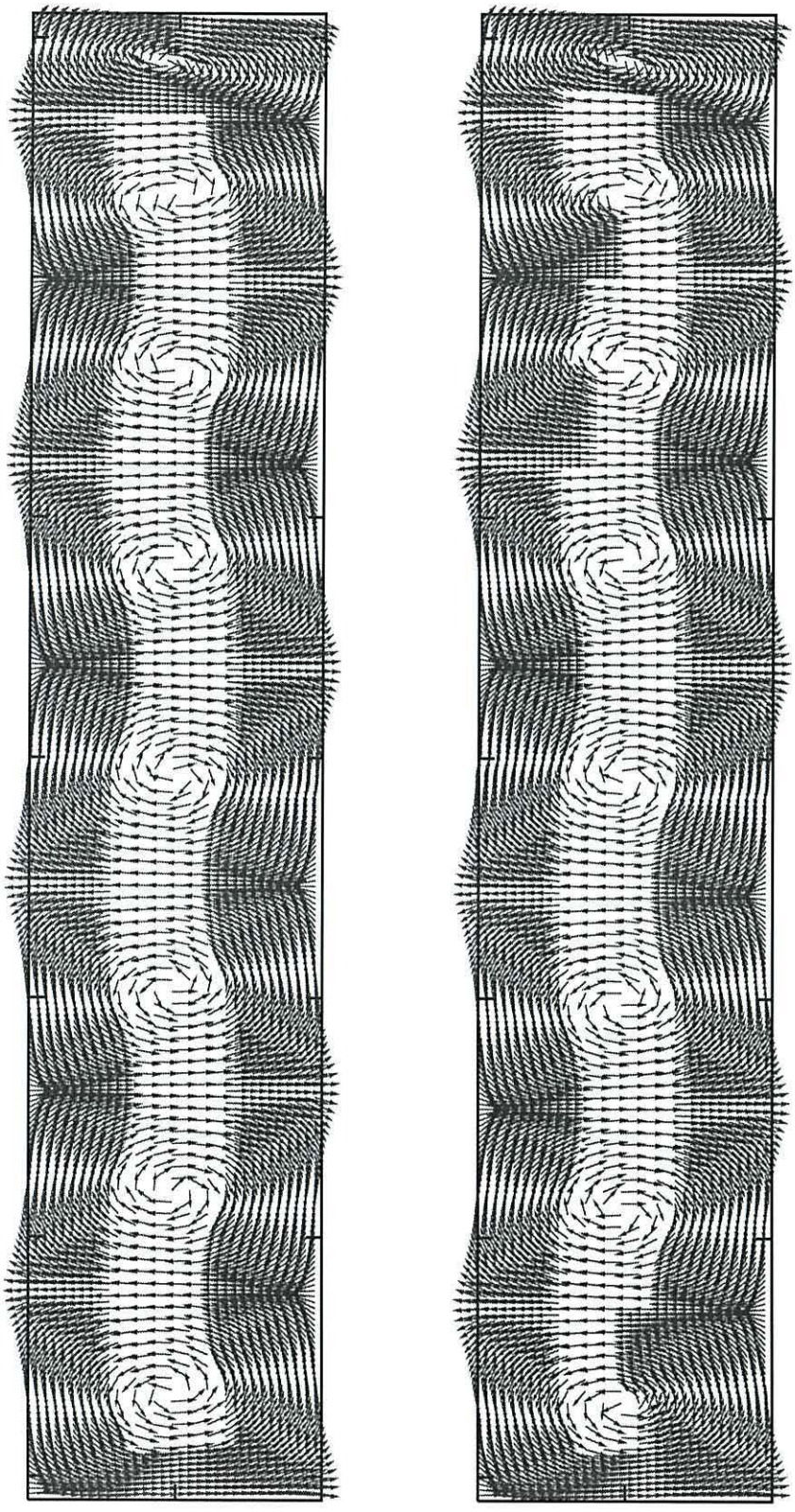


Figure 8.4: *Simulation 1.* Magnetization of the nanoelement at times $\tau = 33.1$ (left) and $\tau = 43.6$ (right)

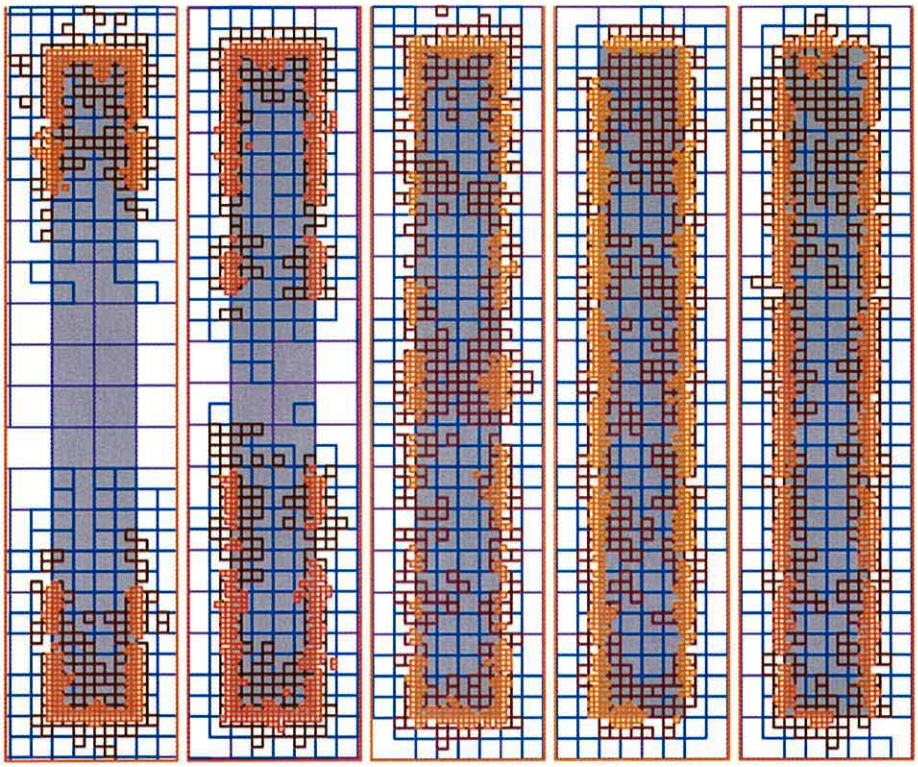


Figure 8.5: *Simulation 1.* Distribution of the wavelet coefficients at times $\tau = 1.18$, $\tau = 2.2$, $\tau = 3.7$, $\tau = 26.7$ and $\tau = 36.4$

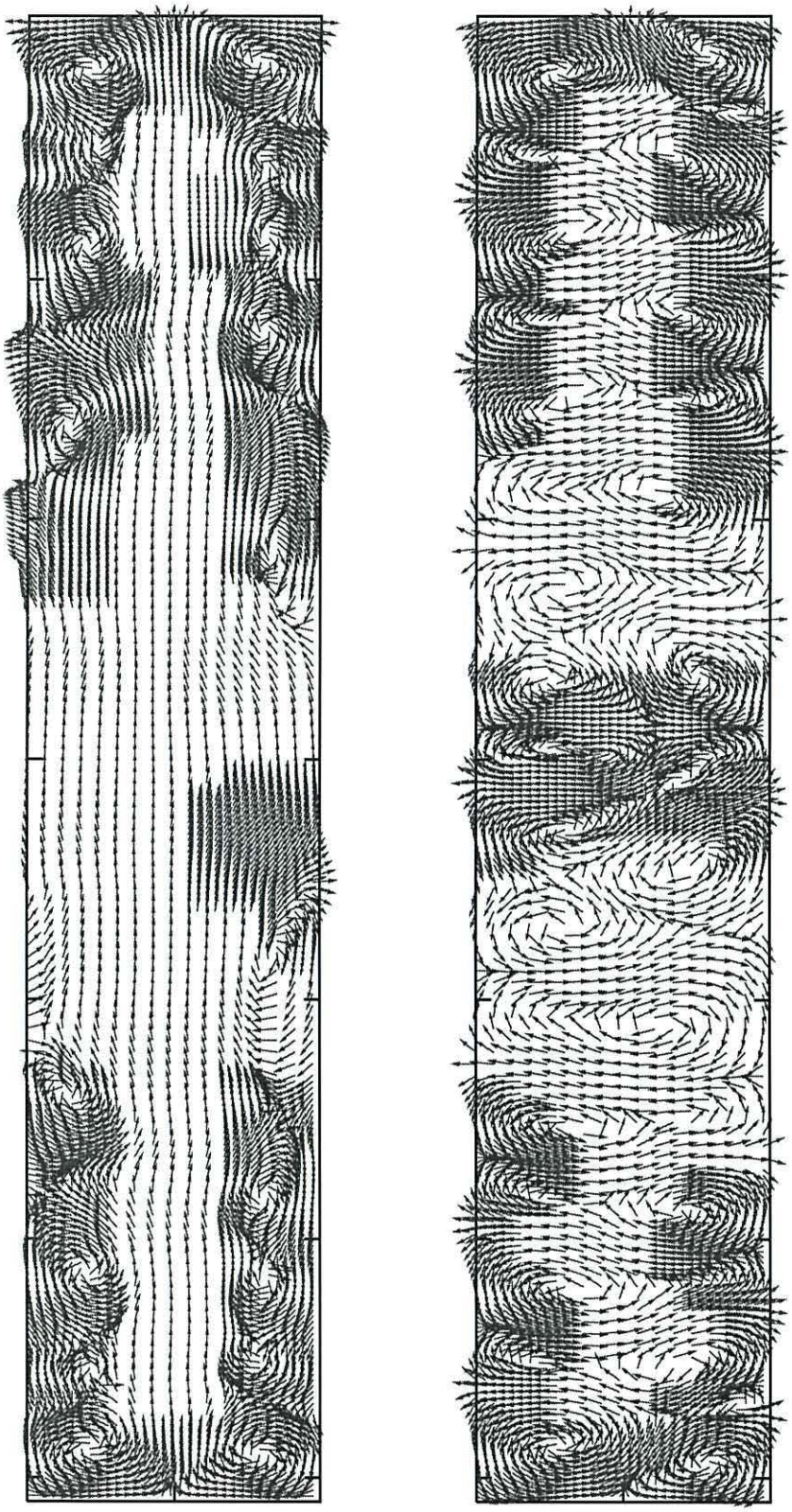


Figure 8.6: *Simulation 2.* Magnetization of the nanoelement at times $\tau = 4$ (left) and $\tau = 8$ (right)

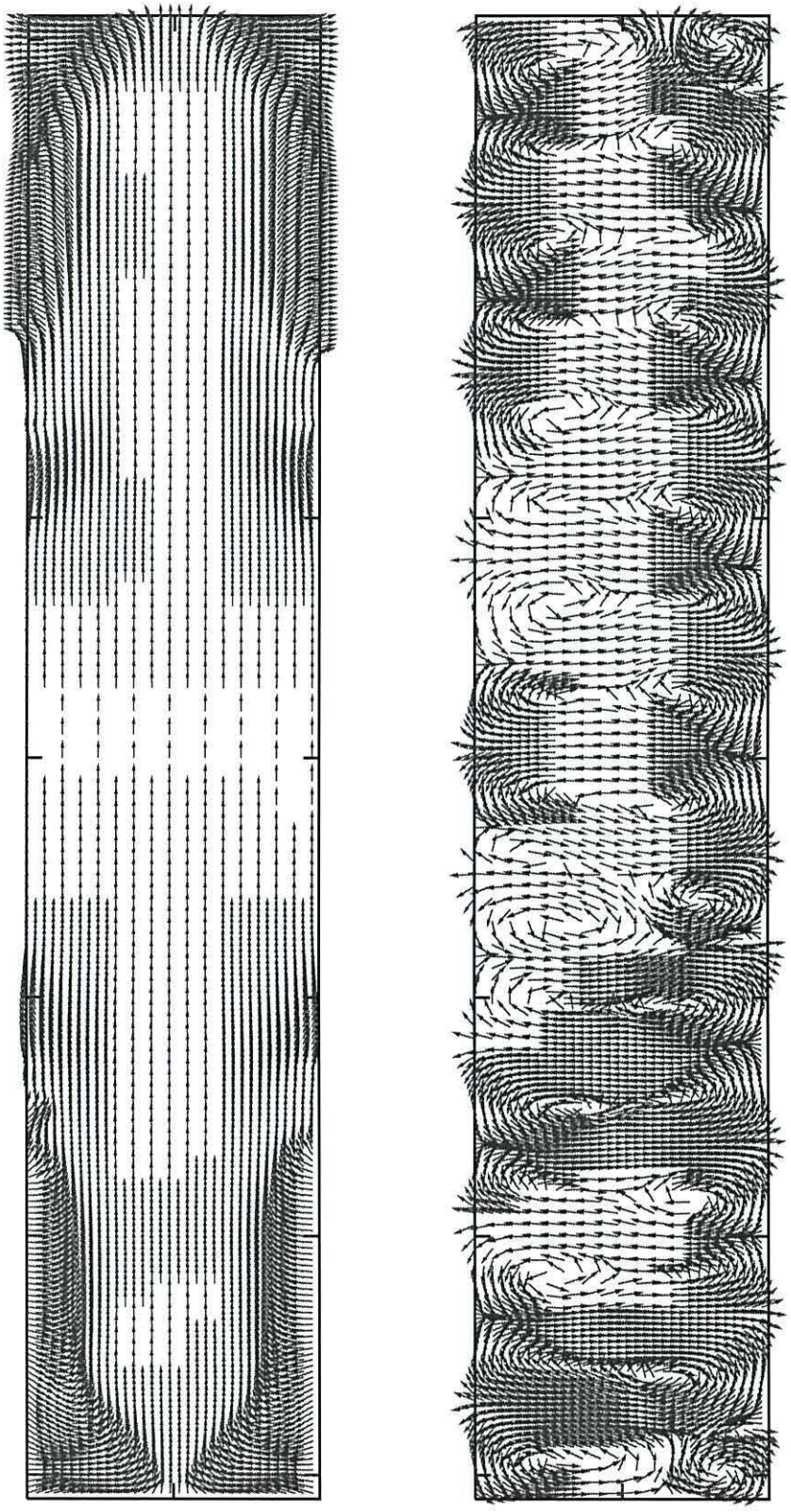


Figure 8.7: *Simulation 2.* Magnetization of the nanoelement at times $\tau = 11$ (left) and $\tau = 32$ (right)

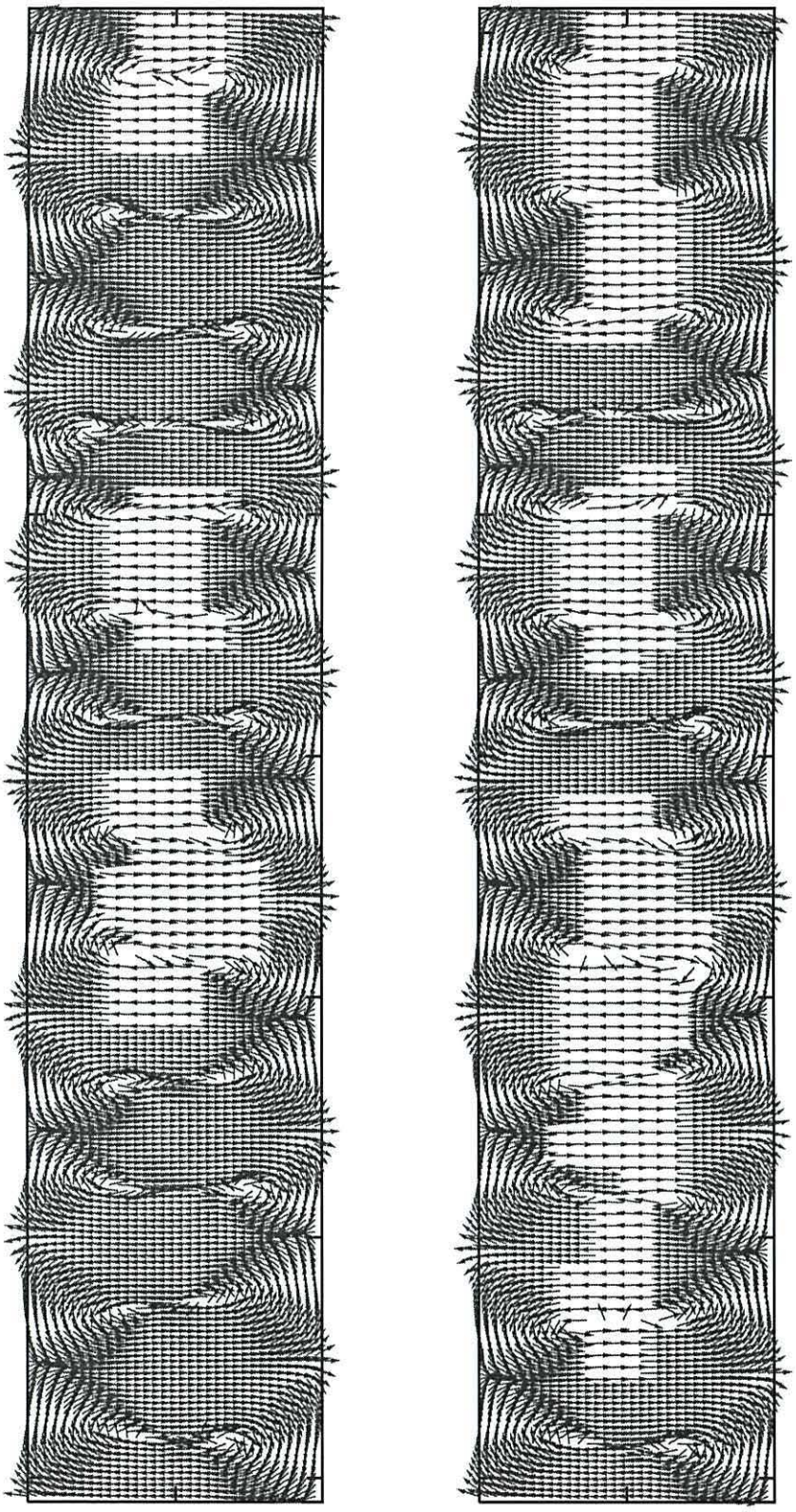


Figure 8.8: *Simulation 2.* Magnetization of the nanoelement at times $\tau = 107$ (left) and $\tau = 144$ (right)

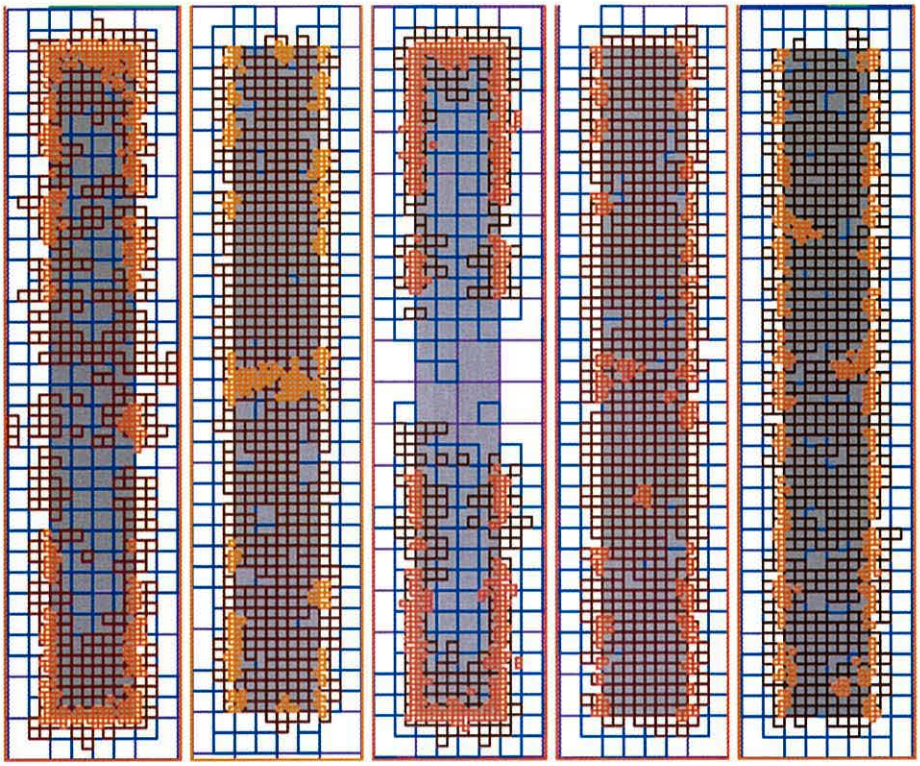


Figure 8.9: *Simulation 2.* Distribution of the wavelet coefficients at times $\tau = 4$, $\tau = 8$, $\tau = 11$, $\tau = 14$ and $\tau = 114$

Appendix A

Non-singularity of the $A + D_r$ matrix

The non-singularity of the $A + D_r$ matrix ($r \geq 1$) from Section 4.3.4 is proved here for any B-spline scaling function basis constructed in [10] and [12]. In the construction, $l_1 \leq 0$ and $l_2 > 0$ are integers such that $[l_1, l_2]$ is the support of the mother scaling function ϕ. The refinement equation is $\phi(x) = \sum_{k=l_1}^{l_2} a_k \, \phi(2x - k)$, where the a_k are defined in [10], p.540, as

$$a_k = 2^{-d} \binom{d}{k + \lfloor d/2 \rfloor}$$

and d denotes the approximation order of the scaling function basis. By convention, the refinement mask $(a_k)_k$ is considered infinite ($k \in \mathbb{Z}$) with null entries outside the range $l_1 \leq k \leq l_2$.

Matrix A is the $(l_2 - l_1)$ by $(l_2 - l_1)$ matrix defined by

$$\begin{cases} A_{l,p} = -a_{-p+2l}, & \text{for } l_1 \leq l, p \leq l_2 - 1, \\ & \text{except if } l < \lceil (l_1 + l_2)/2 \rceil, \\ A_{l,l_1} = -\sum_{p=2l-l_2}^{l_1} a_{-p+2l}, & \text{in that case.} \end{cases}$$

Matrix D_r is 2^{r+1} times the identity matrix of size $(l_2 - l_1)$ by $(l_2 - l_1)$. The $A + D_r$ matrix is singular if and only if $\det(A + D_r) \triangleq \det\big(A - (-2^{r+1})I\big) = 0$, which is the same as A having (-2^{r+1}) for eigenvalue.

The Gershgorin circle theorem states that all the eigenvalues of matrix A lie on the complex plane within the circles $\mathcal{C}_l$ centered at $z_l = A_{l,l}$ and with radii $r_l = \sum_{p \neq l} |A_{l,p}|$. First, let us determine the center and radius of the circle associated with row l_1.

$$\begin{aligned} z_{l_1} &= -\sum_{p=2l_1-l_2}^{l_1} a_{-p+2l_1} = -\sum_{p=l_1}^{l_2} a_p, \\ r_{l_1} &= \sum_{p=l_1+1}^{l_2} |a_{-p+2l_1}| = \sum_{p=2l_1-l_2}^{l_1-1} |a_p| = 0. \end{aligned}$$

By integrating both sides of the refinement equation over $\boldsymbol{R}$, it is apparent that $\sum_{k=l_1}^{l_2} a_k = 2$, so $z_{l_1} = -2$ and $\mathcal{C}_{l_1} = \{-2\}$. Let us now determine the center and radius of the remaining circles $\mathcal{C}_l$, for $\lambda_1 + 1 \leq l \leq l_2 - 1$.

$$\begin{aligned} z_l &= -a_{2l-l_2}, \\ r_l &= \sum_{p=l_1, p \neq l}^{l_2-1} |a_{2p-l_2}|. \end{aligned}$$

Because all the a_k are positive and because they add up to 2, it is clear that $-2 \leq z_l \leq 0$ and that $0 \leq r_l \leq 2$. Moreover, if $z_l = -2$, then $r_l = 0$. Therefore the intersection of the union of the circles $\mathcal{C}_l$ with the real axis is a subset of, or is equal to, the interval $]-4, 2\,]$. This proves that -2^{r+1}, $r \geq 1$, cannot be an eigenvalue for A and that $A + D_r$, $r \geq 1$, is non-singular.

Appendix B

Tables of results used in the estimation of the parameters for the *MULT* algorithm

B.1 Dimension 1

The following tables concern the Laplace operator for the one-dimensional B-spline basis on the unit interval defined in the Multilevel Library by the parameters: $d = 3$, $\tilde{d} = 3$, and biorthogonalization $method = 5$.

On the table concerning the row sums of blocks on and above the diagonal, the label S denotes that the row function is a scaling function and the label W that it is a wavelet. The k_i denote different rows. The rows present in the table are all the rows such that the support of the correponding row function has a non-trivial intersection with the support of a boundary adapted basis function. In addition to those, the table also presents the

Row level	Column level	Level diff.	Block $[W_j S_{j_0}]$	Block $[W_j W_{j_0}]$
5	4	1	1.27E+00	1.08E+00
5	4	1	1.22E+00	1.18E+00
6	4	2	3.98E-02	4.14E-02
7	4	3	2.49E-03	1.52E-03

Figure B.1: *Maximum row sums for blocks below the diagonal. Functions away from the boundary only.*

sums for one row associated with a scaling function and one row associated with a wavelet whose support only intersects the supports of inner basis functions.

$\boldsymbol{Sk_1}$			$\boldsymbol{Wk_4}$	
R Square	1.00000		R Square	1.000000
Standard Error	1.5E-06		Standard Error	3.2E-07

	Intercept	Slope		Intercept	Slope
Coefficients	0.140212	-1.4999	Coefficients	1.627615	-1.5000
Standard Error	9.7E-07	14.E-07	Standard Error	2.5E-07	3.5E-08
Lower 95%	0.140209	-1.5000	Lower 95%	1.627615	-1.5000
Upper 95%	0.140214	-1.4999	Upper 95%	1.627616	-1.5000

Observation	Residuals	Observation	Residuals
1	5.E-07	1	-2.E-07
2	3.E-07	2	-1.E07
3	2.E-07	3	-5.E-08
4	2.E-08	4	6.E-08
5	-1.E-07	5	2.E-07
6	-3.E-07	6	2.E-07
7	-5.E-07	7	4.E-07
8	-5.E-07	8	3.E-07
9	-6.E-07	9	-1.E-07
10	-2.E-06	10	-6.E-07

Figure B.2: *Typical statistics for the regressions on the log of the row sums.*

Col. level	4	4	5	6	7	8	9	10
Row level	4	4	4	4	4	4	4	4
Level diff.	0	0	1	2	3	4	5	6
S								
k_1	10.2	4.6	0.39	0.138	0.0488	0.0172	0.00608	0.00215
k_2	**1220.3**	**16.1**	5.06	1.807	0.6389	0.2259	0.07986	0.02823
k_3	30.6	1.4	0.22	0.079	0.0278	0.0098	0.00347	0.00123
W								
k_1	*	6.2	2.76	0.46	0.16	0.0580	0.0205	0.00725
k_2	*	8.2	1.29	0.47	0.17	0.0609	0.0215	0.00761
k_3	*	8.8	1.18	0.46	0.16	0.0572	0.0202	0.00715
k_4	*	8.6	0.96	0.39	0.14	0.0483	0.0171	0.00603
k_5	*	5.7	0.86	0.35	0.12	0.0435	0.0154	0.00544
k_6	*	5.1	0.88	0.35	0.12	0.0443	0.0156	0.00553
k_7	*	4.9	0.88	0.35	0.12	0.0443	0.0156	0.00553
k_8	*	4.7	0.88	0.35	0.12	0.0443	0.0156	0.00553

Col. level	11	12	13	14	15			
Row level	4	4	4	4	4			
Level diff.	7	8	9	10	11			
S						Intcpt. (I)	Slope	2^I
k_1	7.61E-04	2.69E-04	9.51E-05	3.36E-05	1.19E-05	*0.14*	*−1.5*	*1.10*
k_2	9.98E-03	3.53E-03	1.25E-03	4.41E-04	1.56E-04	*3.85*	**−1.5**	**14.4**
k_3	4.34E-04	1.53E-04	5.43E-04	1.92E-04	6.78E-04	*−0.67*	*−1.5*	*0.63*
W						Intcpt. (I)	Slope	2^I
k_1	2.56E-03	90.6E-04	3.2E-04	1.13E-04	4.0E-05	*1.89*	*−1.5*	*3.71*
k_2	2.69E-03	9.52E-04	3.37E-04	1.19E-04	4.2E-05	*1.93*	*−1.5*	*3.83*
k_3	2.53E-03	8.94E-04	3.16E-04	1.12E-04	3.9E-05	*1.87*	*−1.5*	*3.65*
k_4	1.13E-03	7.54E-04	2.67E-05	9.43E-05	3.3E-05	*1.63*	*−1.5*	*3.09*
k_5	1.92E-03	6.8E-04	2.4E-05	8.5E-05	3.0E-05	*1.48*	*−1.5*	*2.78*
k_6	1.96E-03	6.92E-04	2.45E-05	8.65E-05	3.1E-05	*1.89*	*−1.5*	*3.71*
k_7	1.96E-03	6.92E-04	2.45E-05	8.65E-05	3.1E-05	*1.50*	*−1.5*	*2.83*
k_8	1.96E-03	6.92E-04	2.45E-05	8.65E-05	3.1E-05	*1.50*	*−1.5*	*2.83*

Figure B.3: *Row sums for blocks on and above the diagonal.*

B.2 Dimension: 2

The following tables concern the Laplace operator for the two-dimensional B-spline basis on the unit interval defined in the Multilevel Library by the parameters: $d = 3$, $\tilde{d} = 3$, and biorthogonalization *method* $= 5$. Three types of rows are considered: in Sk_1, the row function $\psi_{j_0,\boldsymbol{k},e}$ is a scaling function, while in Wk_1 and in Wk_2 it is a wavelet of type $(1,0)$ and $(1,1)$, respectively.

Column level	Row level	Level diff.	Sk_1	Wk_1	Wk_2
4	4	0	**1628.302**	*	*
4	4	0	**78.010**	7.291	144.951
5	4	1	0.873	3.101	2.028
6	4	2	0.346	1.420	0.917
7	4	3	0.159	0.521	0.416
8	4	4	0.079	0.236	0.203
9	4	5	0.039	0.115	0.101
		Intercept (I)	0.521	2.131	2.052
		Slope	-1.043	**−1.042**	-1.082
		2^I	1.435	**4.379**	4.146

Figure B.4: *Row sums for blocks on and above the diagonal. Functions away from the boundary only.*

$\boldsymbol{Wk_1}$				
Row level	Column level	Level diff.	Block $[W_j S_{j_0}]$	Block $[W_j W_{j_0}]$
5	4	1	1.22E+00	1.18E+00
6	4	2	3.98E-02	4.14E-02
7	4	3	2.49E-03	1.52E-03

$\boldsymbol{Wk_2}$				
Row level	Column level	Level diff.	Block $[W_j S_{j_0}]$	Block $[W_j W_{j_0}]$
5	4	1	1.22E+00	1.18E+00
6	4	2	3.98E-02	4.14E-02
7	4	3	2.49E-03	1.52E-03

Figure B.5: *Maximum of row sums for blocks below the diagonal. Functions away from the boundary only.*

Sk_1		Wk_1	
R Square	0.9993	R Square	0.9990
Standard Error	0.0444	Standard Error	0.0607

	Intercept	Slope		Intercept	Slope
Coefficients	0.5211	-1.0432	Coefficients	2.0517	-1.0818
Standard Error	0.0730	0.0199	Standard Error	0.0637	0.0192
Lower 95%	0.2071	-1.1286	Lower 95%	1.8490	-1.1429
Upper 95%	0.8351	-0.9577	Upper 95%	2.2544	-1.0207

Observation	Residuals	Observation	Residuals
1	0.0343	1	0.0505
2	-0.0434	2	-0.0134
3	-0.0160	3	-0.0702
4	0.0251	4	-0.0212
		5	-0.0544

Figure B.6: *Typical statistics for the regressions on the log of the row sums.*

Contents

Bibliography

[1] Codeguru sites. http://www.codeguru.com/cpp/stlguide/, October 2001.

[2] B. Alpert. Wavelets and other bases for fast numerical linear algebra. In C.K. Chui, editor, *Wavelets - A tutorial in Theory and Applications*, pages 181–216. Academic Press, 1992.

[3] A. Barinka, T. Barsch, P. Charton, A. Cohen, S. Dahlke, W. Dahmen, and K. Urban. Adaptive wavelet schemes for elliptic problems - implementation and numerical experiments. IGPM report 173, RWTH Aachen, November 1999.

[4] S. Bertoluzza, C. Canuto, and K. Urban. On the adaptive computation of integrals of wavelets. *Appl. Numer. Math.*, 34:13–38, 2000.

[5] C. Canuto, A. Tabacco, and K. Urban. The wavelet element method part I: construction and analysis. Preprint 1038, Istituto di Analisa Numerica del CNR, Pavia, 1997.

[6] C. Canuto, A. Tabacco, and K. Urban. The wavelet element method part II: realization and additional features in 2D and 3D. Preprint 1052, Istituto di Analisa Numerica del CNR, Pavia, 1997.

[7] G. Chiavassa. *Algorithmes adaptatifs en ondelettes pour la résolution d'équations aux dérivées partielles.* PhD thesis, Institut de recherche sur les phénomènes hors équilibre, June 1997.

[8] A. Cohen, W. Dahmen, and R. DeVore. Adaptive wavelet methods II - beyond the elliptic case. IGPM report, RWTH Aachen, November 2000.

[9] A. Cohen, W. Dahmen, and R. DeVore. Adaptive wavelet methods for elliptic operator equations – convergence rates. *Math. Comp.*, 70:27–75, 2001.

[10] A. Cohen, I. Daubechies, and J.-C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Comm. Pure Appl. Math.*, 45:485–560, 1992.

[11] J.M. Combes, A. Grossmann, and Ph. Tchamitchian, editors. *Wavelets: Time-Frequency Methods and Phase Space*, Marseille, France, December 14-18 1987. International Conference, Springer Verlag.

[12] W. Dahmen, A. Kunoth, and K. Urban. Biorthogonal spline-wavelets on the interval — stability and moment conditions. *Appl. Comp. Harm. Anal.*, 6:132 – 196, 1999.

[13] W. Dahmen and C.A. Micchelli. Using the refinement equation for evaluating integrals of wavelets. *SIAM J. Numer. Anal.*, 30(2):507–537, April 1993.

[14] W. Dahmen and R. Schneider. Wavelets with complement boundary conditions - functions spaces on the cube. *Results in Mathematics*, 34:255–293, 1998.

[15] W. Dahmen and R. Schneider. Wavelets with complementary boundary conditions - function spaces on the cube. *Results in Mathematics*, 34:255–293, 1998.

[16] W. Dahmen and R. Schneider. Composite wavelet bases for operator equations. *Math. Comp.*, 68:1533–1567, 1999.

[17] I. Daubechies. Orthonormal bases of compactly supported wavelets. *Comm. on Pure and Applied Maths.*, XLI:909 – 996, 1988.

[18] I. Daubechies. The wavelet transform, time-frequency localization and signal analysis. *IEEE Trans.*, IT-36(5):961 – 1005, 1990.

[19] J.-P. Demailly. *Analyse numérique et équations différentielles.* Collection Grenoble Sciences. Presses universitaires de Grenoble, 1991.

[20] B. Jawerth and W. Sweldens. An overview of the theory and applications of wavelets. In *Proceedings of the Nato Advanced Research Workshop*, 7-11 September 1992.

[21] A. Kunoth, W. Dahmen, and R. Schneider. Wavelet least squares methods for boundary value problems. *SIAM J. Numer. Anal.*, 39(6):1985–2013, 2002.

[22] The Multilevel Library. http://www.igpm.rwth-aachen.de/software.html.

[23] Y. Meyer. Orthonormal wavelets. In Combes et al. [11], pages 21–37.

[24] Y. Meyer. *Ondelettes et Opérateurs.*, volume I: Ondelettes. Hermann, Paris, 1990.

[25] P.H.W. Ridley. *Finite Element Simulation of the Micromagnetic Behaviour of Nanoelements.* PhD thesis, University of Wales, Bangor, 2000.

[26] R. Schneider, W. Dahmen, and Y. Xu. Nonlinear functionals of wavelet expansions - adaptive reconstruction and fast evaluation. IGPM-Report 160, RWTH Aachen.

[27] O. Stromberg. A modified Franklin system and higher order spline systems on $\mathbb{R}^n$ as unconditionnal bases for Hardy spaces. In W Beckner and al., editors, *Conference in Harmonic Analysis in honor of Antoni Zygmund*, volume II of *Wadworth Math.*, pages 475–493.

[28] M. Unser and A. Aldroubi. Polynomial splines and wavelets - a signal processing perspective. In C.K. Chui, editor, *Wavelets-A tutorial in Theory and Applications*, pages 91 – 122. Academic Press, 1992.

[29] K. Urban. On divergence-free wavelets. *Advances in Computational Mathematics*, 4:51–82, 1995.