THE UNIVERSITY
of EDINBURGH

This thesis has been submitted in fulfilment of the requirements for a postgraduate degree (e.g. PhD, MPhil, DClinPsychol) at the University of Edinburgh. Please note the following terms and conditions of use:

This work is protected by copyright and other intellectual property rights, which are retained by the thesis author, unless otherwise stated.
A copy can be downloaded for personal non-commercial research or study, without prior permission or charge.
This thesis cannot be reproduced or quoted extensively from without first obtaining permission in writing from the author.
The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author.
When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given.

Parallelisation of Micromagnetic Simulations

Lesleis Nagy

THE UNIVERSITY
of EDINBURGH

Thesis submitted in fulfilment of
the requirements for the degree of
Doctor of Philosophy
to the
University of Edinburgh 2016

Declaration

I declare that this thesis has been composed solely by myself and that it has not been submitted, either in whole or in part, in any previous application for a degree. Except where otherwise acknowledged, the work presented is entirely my own.

Lesleis Nagy
March 2016

Abstract

The field of paleomagnetism attempts to understand in detail the the processes of the Earth by studying naturally occurring magnetic samples. These samples are quite unlike those fabricated in the laboratory. They have irregular shapes; they have been squeezed and stretched, heated and cooled and subjected to oxidation. However micromagnetic modelling allows us to simulate such samples and gain some understanding of how a paleomagnetic signal is acquired and how it is retained.

Micromagnetics provides a theory for understanding how the domain structure of a magnetic sample alters subject to what it is made from and the environment that it is in. It furnishes the mathematics that describe the energy of a given domain structure and how that domain structure evolves in time. Combining micromagnetics and ever increasing computer power, it has been possible to produce simulations of small to medium size grains within the so-called single to pseudo single domain state range. However processors are no longer built with increasing speed but with increasing parallelism and it is this that must be exploited to model larger and larger paleomagnetic samples.

The purpose of the work presented here is twofold. Firstly a micromagnetics code that is parallel and scalable is presented. This code is based on FEniCS, an existing finite element framework, and is shown to run on ARCHER the UK's national supercomputing service. The strategy of using existing libraries and frameworks allow future extension and inclusion of new science in the code base. In order to achieve scalability, a spatial mapping technique is used to calculate the demagnetising field - the most computationally intensive part of micromagnetic calculations. This allows grain geometries to be partitioned in such a way that no global communication is required between parallel processes - the source of

favourable scaling behaviour.

The second part of the theses presents an exploration of domain state evolution in increasing sizes of magnetite grains. This simulation, whilst a first approximation that excludes magneto-elastic effects, is the first attempt to map out the transition from pseudo-single domain states to multi domain states using a full micromagnetic simulation.

Acknowledgements

A PhD is never a solo effort and without the people listed here I don't think that I would have reached the end of my studies and produced this thesis. Everyone mentioned has played some part in me finishing: some great, some small, but all significant.

First and foremost I want to thank my supervisor Prof. Wyn Williams whose guidance, knowledge and expertise have been invaluable throughout this project - I simply wouldn't have been able to complete this work without his help and encouragement. I also want to thank Dr. Christopher Maynard for putting me on the path to this project; and Drs. Lawrence Mitchell and Christopher Johnson for acting as second supervisors - they have always been generous with their time, helpful in their advice and patient with my shortcomings!

Throughout this project I have had the privilege of working with some exceptional people. Among them are Dr. Adrian Muxworthy and Dr. Trevor Almeida and I want to thank them for the opportunity of collaborating with them and contributing to their excellent work. I also want to thank Dr. Class Abert for very useful discussions at the start of this project in regards to FEniCS and finite element modelling. In addition I'm very grateful to Prof. Karl Fabian for his help and advice, in particular with the Hubert minimiser. I also want to thank Pádraig Ó Conbhuí and Miguel Valdez for their ideas, constructive criticism and general discussions.

The viva and defence of a thesis is not easy. However I want to thank Prof. Simon Tett and Prof. Karl Fabian for agreeing to be my examiners. Their critique and suggestions were constructive and useful. I strongly believe that the final thesis is better because of their input and advice.

Finally I want to thank my friends for all their support throughout this PhD.

In particular, Paul Ross and Darren Slevin for keeping me sane over the last few years and for their regular offers of beer and coffee. I also want to thank the 'Leeds Uni Crew' consisting of: Peter Coleman, Robert Dinsdale, James Kirkbright, Joe Rodgers, Rouzbeh Safaie and Leon Savidis. Without Leon in particular I don't think I could have dragged myself across the finish line in those final weeks. Finally want to thank Gianpietro Previtali for encouraging me to go back to my studies.

I want to dedicate this work to my mother Elizabeth, my grandmother Anna and my grandfather John (who died 9th January 2003). They always believed in me, even when I didn't.

Contents

Abstract **iii**

List of Figures **x**

List of Tables **xii**

1 Introduction **1**

1.1 Aims of the Thesis 1

1.2 Magnetism in Materials 2

1.2.1 Diamagnetic Materials 4

1.2.2 Paramagnetic Materials 5

1.2.3 Ferromagnetic Materials 6

1.2.4 Diamagnetic, Paramagnetic and Ferromagnetic Hysteresis 7

1.3 Domain Structures 8

1.3.1 Demagnetising Energy 10

1.3.2 Exchange Energy 11

1.3.3 Anisotropy 16

1.4 Micromagnetic Simulations 17

1.5 Summary 19

Bibliography 20

2 The Finite Element Method **21**

2.1 Introduction 21

2.2 Derivation of the Weak Form 22

2.3 Mesh Discretisation, Test and Trial Functions 24

2.4 Constructing a Linear System 28
2.4.1 Stiffness Matrix Sparsity 29
2.4.2 Element-wise Construction of A 30
2.4.3 Boundary Conditions 31
2.4.4 Integration Over Triangles 34
2.5 Solving The System 35
2.5.1 Iterative Solvers 37
2.5.2 Sparse Matrix Formats 39
2.5.3 Bandwidth Minimisation 41
2.6 Summary 43
Bibliography 44

3 A Micromagnetics Code Using FEniCS 46
3.1 A Brief Overview of FEniCS 46
3.1.1 Degrees of Freedom 51
3.2 Effective Field Components - the Big Picture 51
3.3 Box Volume 54
3.4 Energy, Energy Gradient and Effective Field 57
3.4.1 Cubic Anisotropy 57
3.4.2 Exchange 62
3.4.3 Demagnetising Field 63
3.5 Energy Minimisation 69
3.5.1 The Callback Mechanism 70
3.5.2 TAO Minimiser 73
3.5.3 Hubert Minimiser 74
3.6 Time Steppers 77
3.7 MicroMag Design - The Big Picture 77
3.8 Summary 81
Bibliography 82

4 Testing and Performance Scaling 85
4.1 Testing 85
4.1.1 Demagnetising Field 86

4.1.2 Exchange and Anisotropy Field 92
4.1.3 Additional Tests 94
4.2 Performance 95
4.3 Summary 102
Bibliography 103

5 Domains, From Nano to Micro 104
5.1 Introduction 104
5.1.1 Background 105
5.2 Materials and Methods 109
5.2.1 Mesh Generation 110
5.2.2 Running Models 114
5.3 Results 116
5.3.1 Size Hysteresis, Ascending 121
5.3.2 Size Hysteresis, Descending 123
5.3.3 The 1350nm Grain 128
5.4 Conclusions 133
Bibliography 135

6 Thesis Conclusions and Future Work 139
6.1 Other Micromagnetics Codes 139
6.2 Future Work 140
Bibliography 143

Appendices 144

A Glossary of Symbols 144

B Theorems and Results 146
B.1 Vector Identities 146
B.2 Divergence Theorem 146
B.3 Calculus of Variations 147

C Mathematica Code for Stiffness Matrix Assembly 149

List of Figures

1.1 Magnetic moments in a paramagnetic material. 6
1.2 Hysteresis curves for diamagnetic, paramagnetic and ferromagnetic materials 9
1.3 Stray field domains 11
1.4 Energy vs. domain wall width 14
1.5 Domain walls 15
1.6 Anisotropy energy surfaces 17

2.1 Facet function 26
2.2 Discretisation of a planar mesh 27
2.3 Non interacting nodes 30
2.4 Local stiffness matrix 32
2.5 Sparsity pattern 33
2.6 Dirichlet matrix vector system 34
2.7 The reference element 36
2.8 CSR storage scheme for sparse matrices 40
2.9 Cuthill-Mckee bandwidth minimisation 42
2.10 Bandwidth minimised stiffness matrix. 43

3.1 A Poisson problem solution. 49
3.2 Schematic of energy, energy gradient and effective fields. 52
3.3 Box volume 55
3.4 Material, mapped and unmapped regions 65
3.5 Parallel Decomposition 69
3.6 Hubert states 77

3.7 Overall design 80

4.1 Testing demagnetisation 89
4.2 Sphere and slab demagnetising potential 90
4.2 Sphere and slab demagnetising potential continued 91
4.3 Demagnetising finite difference test 92
4.4 Exchange and anisotropy finite difference tests 93
4.4 Exchange and anisotropy finite difference tests continued 94
4.5 Exchange, Anisotropy and Demag Scaling 97
4.5 Exchange, Anisotropy and Demag Scaling continued 98
4.6 A model of scaling behaviour. 100
4.7 A model of scaling behaviour. 101

5.1 Simulated and observed holography. 107
5.2 Meshes for simulated holography 108
5.3 Standard cuboctahedra 109
5.4 Vortex core in a 100nm cuboctahedron. 117
5.5 Anisotropy and helicity information for the 100nm cuboctahedron. 119
5.6 Change in magnetisation along helicity contour. 120
5.7 Results for size hysteresis of cuboctahedra and spheres. 128
5.8 Block wall in the $\{1, 1, 1\}$ plane. 129
5.9 Estimating the wall width. 130
5.10 Anisotropy energy values for the the 1350nm cuboctahedra. . . . 131

List of Tables

4.1 Summary of geometries for the demagnetising potential. 88
4.2 Additional MicroMag tests. 95
4.3 Scaling models 96

5.1 Summary of spherical geometries 113
5.2 Summary of cuboctahedral geometries 114
5.3 Hubert minimiser parameters 115
5.4 Magnetite material parameters 116
5.5 Domain angles. 132
5.6 Domain widths. 133

Chapter 1

Introduction

1.1 Aims of the Thesis

Micromagnetism allows us to understand the process by which materials acquire and retain a signal. It is a mathematical model which allows us to study the stability of recordings, in particular naturally occurring materials such as magnetite, maghemite, hemiatite, etc. These materials are extremely important in diverse fields such as paleomagnetism, the study of the Earth's ancient field; environmental magnetism, which allows us to answer questions about the Earth's ancient climate in order to understand the modern climate in its proper context; and biomagnetism, which aims to understand the evolution of magnetically sensitive microorganisms.

Overall there are two main aims to the work presented in this thesis. Firstly a there is a development of a micromagneic code that is suitable for execution in parallel on large high performance computing (HPC) infrastructures allowing unconstrained modelling of large grains. Secondly the resulting code is used to explore the magnetic stability of recordings in grains from the nano scale up to the micron scale.

1.2 Magnetism in Materials

The mechanism by which magnetic materials acquire and retain their magnetisation is important to understanding the paleomagnetic recording process and gives rise to the domain structures found in magnetic materials. Such domain structures in turn result in Natural Remanent Magnetisation (NRM), which in the case of paleomagnetism is the recording of the Earth's ancient magnetic field. The magnetisation itself is a vector field that is comprised of individual magnetic dipole moments - these moments are macroscale approximations of the quantum-mechanical interactions between the electrons of atoms that from crystalline (*i.e.* regular) solids.

All materials can be broadly classified in to separate categories of magnetism, namely diamagnetism, paramagnetism and ferromagnetism. The phenomenon of spontaneous magnetisation in ferromagnetic materials is responsible for domain structures that allow recordings to be retained for millions of years. These domain structures result primarily from the interplay between three sources of energy. As will be shown, micromagnetics is fundamentally a balancing act, attempting to find the minimum energy magnetic structure in a complex high-dimensional surface composed of three types of energy: the demagnetising energy, which arises from the static magnetic fields produced by material dipoles; the exchange energy, which arises from the preference for neighbouring dipoles to align parallel to each other and the anisotropy energy which arises from the preference of magnetic dipoles to lie in a particular direction - these directions are referred to as 'magneto-crystalline easy axes'.

In the presence of an external magnetic field, ferromagnetic materials reconfigure their domain structure in a way that responds to the applied external magnetic field. Even when the source is removed a recording of the signal remains - recorded within the domain structure. This gives rise to a property of ferromagnets whereby new recordings can 'overwrite' old information but are never completely independent of it. This hysteresis behaviour is fundamental to understanding how ferromagnetic materials record information.

The intention of this chapter is to introduce some of the basic concepts in

electromagnetism and magnetic materials. The classifications of magnetic materials are introduced, along with a discussion on how each of these effects arise. The large value for the molecular field calculated by Weiss in 1907 (Kittel, 1949) lead him to propose the presence of domains in magnetic materials - regions of uniform magnetisation. Ferromagnetic domains arise as a result of the interaction of the demagnetising, exchange and anisotropy energies and these are then discussed. Since neighbouring domains by definition consist of uniform magnetisations there must be transition regions between domains. These domain walls are then discussed with reference to how each of the three effective field energies is affected by domain wall configurations. Finally there is a a brief introduction to micromagnetics, in particular the code presented in the remainder of this thesis - its applications and why it is a useful contribution.

All materials show some form of magnetic behaviour, be it diamagnetic, paramagnetic or ferromagnetic. Three fields are responsible for the magnetic behaviour found in materials: the induction field $\boldsymbol{B}$ (with units of Tesla), the magnetic field $\boldsymbol{H}$ (with units of Am^{-1} and the magnetisation $\boldsymbol{M}$ (with units of Am^{-1}). These three quantities are related according to the constitutive equation

$$\boldsymbol{B} = \mu_0 \left(\boldsymbol{H} + \boldsymbol{M} \right), \tag{1.1}$$

where $\mu_0 = 4\pi \times 10^{-7}\ \mathrm{VsA}^{-1}\ \mathrm{m}^{-1}$ is the permeability of free space.

Magnetisation is a result of quantum mechanical interactions between electrons, namely quantised orbital and spin components. The total magnetic moment for an electron is given by

$$\langle \boldsymbol{m}^z_{\mathrm{tot}} \rangle = -\frac{\mu_{\mathrm{B}}}{\hbar} \left(2 \langle \boldsymbol{s}_z \rangle + \langle \boldsymbol{l}_z \rangle \right), \tag{1.2}$$

where $\langle \boldsymbol{m}^z_{\mathrm{tot}} \rangle$ is the expectation value of the total magnetic moment, $\mu_{\mathrm{B}} = 1.17 \times 10^{-29}$ Vms, is the Bohr magneton, $\hbar = 1.0546 \times 10^{-34}\ \mathrm{m}^2\mathrm{Kgs}^{-2}$ is the reduced Plank's constant, $\langle \boldsymbol{s}_z \rangle$ is the expectation value of the quantised spin and $\langle \boldsymbol{l}_z \rangle$ is the expectation value of the quantised angular momentum with the same units as $\hbar$ (the z suffix indicates the quantisation direction). It is the orbital component $\boldsymbol{l}_z$ that is used to define the Bohr magneton μ_{B} (Stöhr and Siegmann, 2006). It

should be noted that $\boldsymbol{s}_z$ comes from the intrinsic spin of an electron and that $\boldsymbol{l}_z$ comes from the moment due to the electron 'orbiting' around the atomic nucleus. Now the expectation of the magnetic moment $\langle \boldsymbol{m}^z_{\text{tot}} \rangle$ has the same units as the Bohr magneton and so it may be expressed as the quantity of elementary magnetic moments (Bohr magnetons) required to produce a macroscopic moment of 1 V m s. Then the bulk dipole moment is

$$\boldsymbol{m} = 0.855 \times 10^{29} \mu_{\text{B}}. \tag{1.3}$$

This is then related to magnetisation in the constitutive equation (1.1) by the relation $\boldsymbol{m} = \boldsymbol{M}V$, where magnetisation is the net dipole moment per unit volume (Stöhr and Siegmann, 2006).

The final thing to note, is that experimentally the orbital component of (1.2) is zero in iron, cobalt and nickel. This is because the outer, so called d3 electrons, are responsible for magnetic behaviour in these elements. When these elements participate in chemical bonding, neighbouring atoms affect the d3 electrons and the angular component is said to be 'quenched'. Thus the magnetic moment in these geologically important elements is primarily due to electron spin (Dunlop, 2001; Stöhr and Siegmann, 2006).

1.2.1 Diamagnetic Materials

The diamagnetic effect is ubiquitous to all elements. It results from the precession of atomic moments in the presence of an externally applied field and is in opposition to that field (Dunlop, 2001). This precession arises from the orbital component of atomic electrons. In weak fields, the electron orbitals are generally randomly orientated across atoms. However strong external fields distort the electron orbitals. Classically the external field may be thought of as an additional force acting on the electron moment, causing it to precess much like the precession of a spinning top. Diamagnetism is important only in materials which do not show any of the other forms of material magnetism. In general the contribution from electrons is small due to the fact that orbits are randomly aligned and it requires a strong external field to cause the precessional effect. Diamagnetic susceptibility is typically orders of magnitude smaller than paramagnetic

and ferromagnetic susceptibility (Tabor, 1991). It should also be noted, that the diamagnetic effect is not significant in terms of understanding spontaneous organisation of dipoles in the ferromagnetic materials modelled in this thesis.

1.2.2 Paramagnetic Materials

Whereas diamagnetism results from an opposition to an applied field, paramagnetism results from the partial alignment of magnetic moments with an external field. Thermal effects mean that moments do not align precisely as seen in figure 1.1. Curie's law (eqn. 1.4) relates paramagnetic response to temperature

$$\chi_p = \frac{N\mu_{\mathrm{B}}^2\mu_0}{kv}\left(\frac{1}{T}\right), \tag{1.4}$$

where N is the total number of moments, μ_{B} is the Bohr magneton, μ_0 is the permeability of free space, k is the Boltzmann constant, v is the volume of the grain and T is the temperature (Tauxe et al., 2009). The derivation of (1.4) follows a thermodynamical argument where $\chi_p = M/H$, *i.e.* the paramagnetic susceptibility is equal to the gradient of the magnetisation M in response to an applied field H. By considering a thermodynamical system with two micro-states of M - one which is aligned parallel with an external field and one that is aligned anti-parallel with an external field; it is possible to calculate the expectation value for the magnetisation M. This may then be used to derive (1.4), the full derivation is presented in Dunlop (2001).

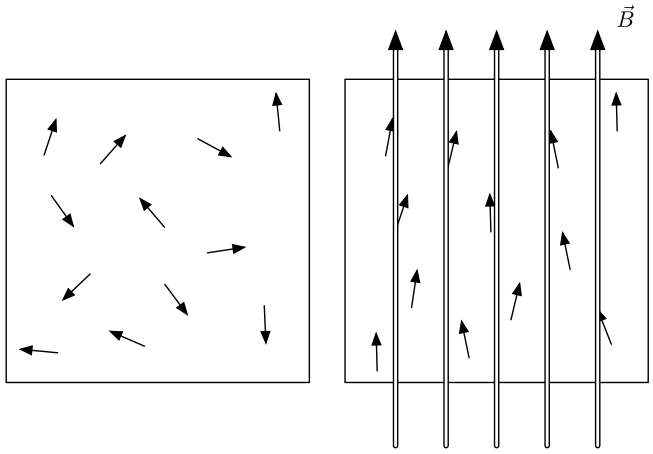


Figure 1.1: Magnetic moments in a paramagnetic material; without (a) and with (b) externally applied field [image taken from Spaldin (2010)].

1.2.3 Ferromagnetic Materials

The final class of magnetic materials exhibit a property called spontaneous magnetisation - a measurable external field even in the absence of an applied field. Such materials are called ferromagnetic.

Spontaneous magnetisation arises due to the cooperative behaviour between atoms that results in the alignment of atomic electron spins, in particular electrons in the 3d sub-shell of certain magnetic elements (e.g. iron, cobalt and nickel), resulting in a net spin (Dunlop, 2001). This seems counterintuitive, since it would be expected that opposing spin-magnetic moments of electrons near to each other would cancel out - resulting in a favourable energy configuration. However this is not the case, since it is possible to show that electrons with cooperative spins result in a lower energy configuration if the parent atoms of those spins are involved in bonding with other atoms - as is the case in iron and ferromagnetic materials. This can be demonstrated by considering the covalent bonding of a hydrogen molecule H_2, where the electrons are shared between atoms. In this scenario it is better (energetically speaking) for the two electrons comprising the molecule to adopt parallel spins. This calculation is rather involved and outlined in (Stöhr and Siegmann, 2006; Dunlop, 2001), however this effect is responsible

for the exchange coupling of neighbouring magnetisations (as discussed below).

Historically, the approach for estimating the ferromagnetic response is due to Weiss. This treatment, assumes a paramagnetic response above a given critical temperature (called the Curie temperature), where as spontaneous ordering in to domains (regions of cooperating magnetisation) occurs below this critical temperature. This spontaneous ordering was assumed to be a result of a molecular magnetic field by Weiss, in analogy with the Lorentz field responsible for the ferroelectrical response. Since the material is assumed to behave paramagnetically, the fundamental technique for deriving the temperature dependent susceptibility of a ferromagnet is similar to the way in which it is derived for paramagnets - however there is an additional term $\boldsymbol{H}_w$. This is the Weiss field that is assumed to be proportional to the average magnetisation of a domain *i.e.* $H_w = \lambda \boldsymbol{M}$.

$$\chi_f = \frac{N\mu_b^2\mu_0}{kv}\left(\frac{1}{T - T_c}\right). \tag{1.5}$$

This is the Curie-Weiss law and it illustrates how susceptibility in ferromagnetic materials tends to infinity as T approaches the Curie temperature T_c. Intuitively this makes sense, since thermal effects dominate at higher temperatures when magnetic moments become randomly aligned. However the Weiss field is extraordinarily strong. For example Kittel (1949) calculates the magnitude of the Weiss molecular field (for a material with a Curie temperature on the order of 10^3 K) to be on the order of 10^9 $^{\mathrm{A}}/_{\mathrm{m}}$.

1.2.4 Diamagnetic, Paramagnetic and Ferromagnetic Hysteresis

So far diamagnetic, paramagnetic and ferromagnetic samples have been considered individually but how do they relate to each other? Diamagnetism is easily differentiated from the paramagnetic/ferromagnetic response since it is temperature independent and the resulting magnetisation is in opposition to an applied field. Like the diamagnetic response, paramagnetism requires a sample to be in an

external field, however the response is temperature dependent. The main difference between a paramagnetic/diamagnetic sample and a ferromagnetic sample is the presence of the spontaneous magnetisation that that imparts a memory. Thus when an external field is removed, a remanence of the original field is preserved. This is readily observed in the hysteresis loops shown in figure 1.2. Diamagnetic/paramagnetic materials do not retain information of the original applied field when it is removed due to thermal effects, however ferromagnetic materials do and this is highlighted by the fact that the hysteresis curve for a ferromagnetic material is a loop.

1.3 Domain Structures

The hysteresis loop of figure 1.2c shows some interesting properties: a saturation magnetisation, a remanent magnetisation and a coercive field. By proposing the molecular field outlined in section 1.2.3 it is possible to explain how M_{s} and M_{rs} can arise since the molecular field is responsible for organising the magnetic dipoles within the materials. However the precise details of how the hysteresis loop in figure 1.2c comes about is complex and is a result of a delicate balance between demagnetising, exchange and anisotropy energies (described below). In summary, the three energies comprise a high dimensional energy landscape and the domain structure of a grain occupies some local energy minimum in this landscape. When an external field is applied, the landscape alters, lowering energy barriers and allowing the domain structure to reconfigure and occupy a new lower energy state. When the external field is removed the energy barriers return, however the structure has been altered and so the measured signal from the magnetisation is changed. This is the way in which magnetic materials acquire a signal from an external field. The full details of this procedure were found by Landau and Lifshitz, culminating in the work by Brown (Brown, 1963).

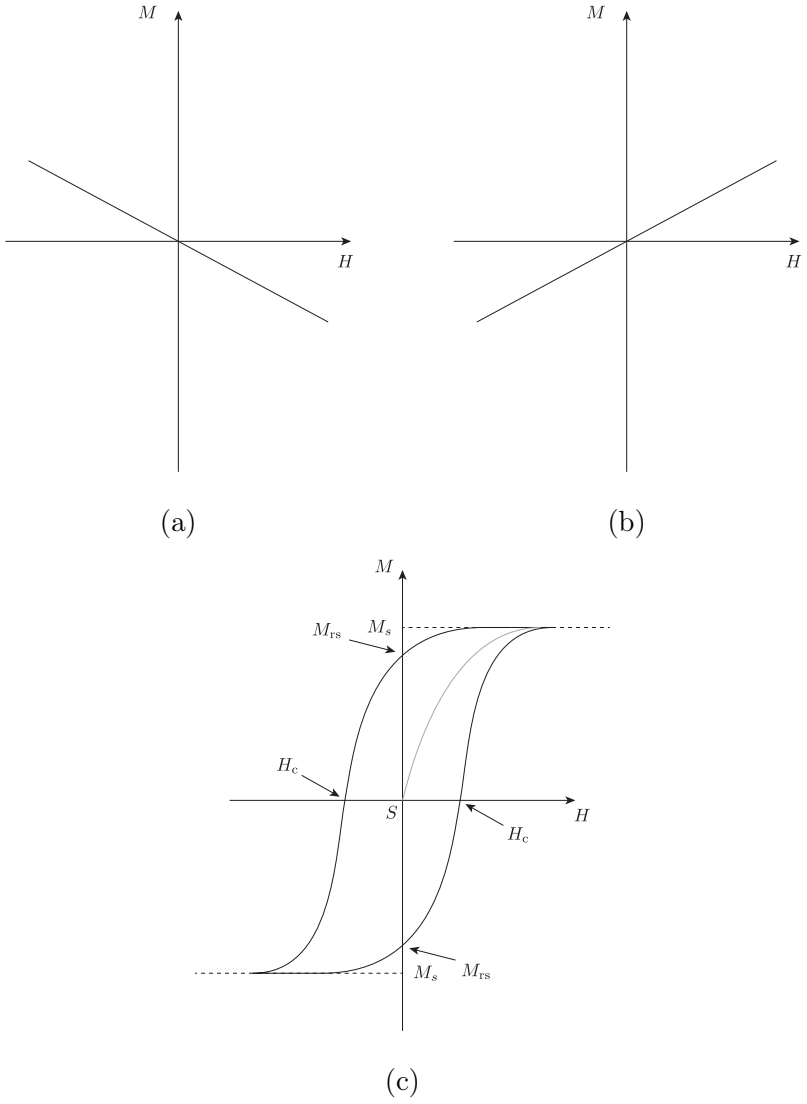


Figure 1.2: Magnetisation response curves of diamagnetic (a), paramagnetic (b) and ferromagnetic (c) material, with applied field H and magnetisation response M. For diamagnets the response is negative and small corresponding to a small negative coercivity ($\chi < 0$), for paramagnets the response is positive and small ($\chi > 0$) and for ferromagnets the response is large and positive ($\chi \gg 0$). Ferromagnets also have 'memory'. If the starting state is fully demagnetised - marked by S in subfigure (c) - and a field is applied to reach saturation magnetisation M_{s}; then reducing the field to zero results in a remanent field M_{rs}. If the applied field is continued in the negative direction then the result is that magnetisation reduces in strength until the sample becomes demagnetised once more. The field strength at which this occurs is the the coercive field H_{c}. This property means that the response curve is a loop for ferromagnets, unlike the response curves for diamagnets and paramagnets.

1.3.1 Demagnetising Energy

The demagnetising energy results from the dipole-dipole interactions between magnetic regions of a material. In this model each point of the magnetic material may be considered as a source of magnetism (having its own magnetisation). By using the superposition principle, the contribution of each point source may be summed over every other point to describe the field resulting from the demagnetising energy.

In order to lower the net demagnetising energy in a sample, the magnetisation splits in to domains. This is illustrated in figures 1.3a, 1.3b and 1.3c; as the surface 'magnetic charges' become paired, the net field becomes weaker at the surface. In figures 1.3d and 1.3e, closure domains are shown, these small surface domains are aligned orthogonally to the main magnetisation within the domains resulting in closed magnetic loops. This has the effect of eliminating the net stray field at the boundaries.

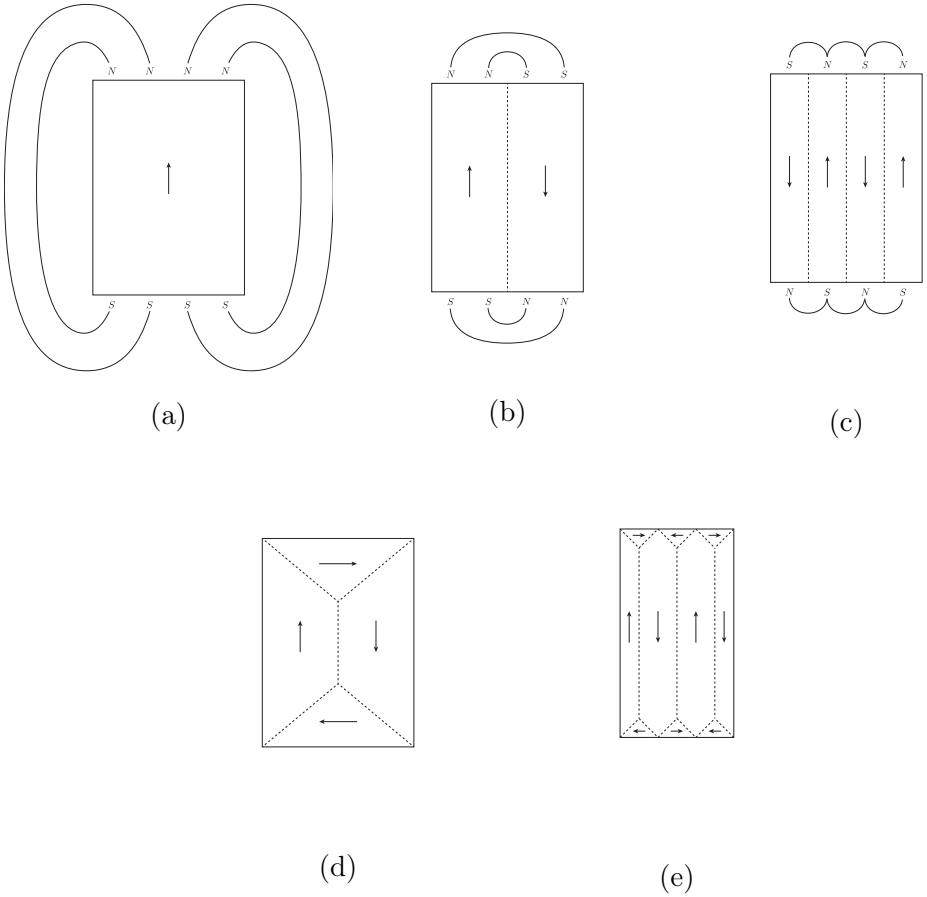


Figure 1.3: The magnetic stray field in the three samples (a), (b) and (c) is reduced in size as the sample breaks in to smaller and smaller domains. These domains are aligned anti-parallel, allowing surface poles to be paired. By introducing closure domains (d) and (e), the stray field is eliminated at the sample surface, thus the demagnetising energy is minimised (Kittel, 1949).

1.3.2 Exchange Energy

The overall energy due to magnetic configurations is reduced when a sample splits in to domains. However, anti-parallel alignment of domains comes at at cost that prevents the sample breaking up in to arbitrarily small domains. The exchange energy is a component of the effective field that prefers adjacent magnetisation vectors to be parallel aligned.

The exchange energy E_{a} is quantum-mechanical in nature and may be expressed as

$$E_{\mathrm{a}} = -A \int_{\Omega} \boldsymbol{m} \cdot \nabla^2 \boldsymbol{m} \, dV, \tag{1.6}$$

where A is a material dependent parameter called the exchange constant (which changes with regard to temperature), $\boldsymbol{m}$ is a unit vector field representing the magnetisation - *i.e* the magnetic dipoles which comprise the material and Ω is the region over which a sample is defined. Equation (eqn. 1.6) is derived by taking Taylor series expansions of expressions for exchange coupling between the electrons of neighbouring atoms in a molecule of ferromagnetic material; (Kittel, 1949) describes this procedure in detail.

In order to rearrange (1.6) to the form that will be used to calculate the exchange field (see section 3.4.2), the fact that $\boldsymbol{m}$ is a unit vector (*i.e.* $|\boldsymbol{m}| = 1$) is used

$$|\boldsymbol{m}|^2 = m_1^2 + m_2^2 + m_3^2 = 1. \tag{1.7}$$

By taking second derivates with respect to x_i, where $i = 1, 2, 3$, results in three expressions

$$\left(\frac{\partial m_1}{\partial x_i}\right)^2 + \left(\frac{\partial m_2}{\partial x_i}\right)^2 + \left(\frac{\partial m_3}{\partial x_i}\right)^2 + \frac{\partial^2 m_1}{\partial x_i^2} m_1 + \frac{\partial^2 m_2}{\partial x_i^2} m_2 + \frac{\partial^2 m_3}{\partial x_i^2} m_3 = 0. \tag{1.8}$$

Summing over i results in

$$\sum_i \left[\left(\frac{\partial m_1}{\partial x_i}\right)^2 + \left(\frac{\partial m_2}{\partial x_i}\right)^2 + \left(\frac{\partial m_3}{\partial x_i}\right)^2 \right] = -\sum_i \left[\frac{\partial^2 m_1}{\partial x_i^2} m_1 + \frac{\partial^2 m_2}{\partial x_i^2} m_2 + \frac{\partial^2 m_3}{\partial x_i^2} m_3 \right]$$

$$|\nabla \boldsymbol{m}|^2 = -\boldsymbol{m} \cdot (\nabla m_1, \nabla m_2, \nabla m_3). \tag{1.9}$$

The left hand side of (1.9) is the Frobenius norm of the gradient tensor of $\boldsymbol{m}$, where as the right hand side is seen to be equivalent to the integrand of (1.6).

Hence (1.6) may be written as

$$E_{\mathrm{a}} = A \int_{\Omega} |\nabla \boldsymbol{m}|^2 \ dV. \tag{1.10}$$

It can be easily seen that sudden transitions between parallel and antiparallel magnetisations come at a cost by considering an analytic expression for a domain wall (Keimpema et al., 2006)

$$\boldsymbol{m}\left(x, y, z\right) = M_{\mathrm{s}} \left(\cos\left(2 \arctan\left(e^{y/\lambda}\right)\right), 0, \sin\left(2 \arctan\left(e^{y/\lambda}\right)\right)\right), \tag{1.11}$$

where the parameter λ controls the width of the domain wall; with small values of λ corresponding to narrow domain walls and so rapid changes in magnetisation. If the exchange constant and saturation magnetisation are assumed to be that of magnetite ($A = 1.34 \times 10^{-11}$ J m^{-1} and $M_{\mathrm{s}} = 4.8 \times 10^{5}$ A m^{-1}) and the x, y and z lengths are 0.2μm, 5μm and 0.2μm respectively then (1.10) is seen to behave as shown in figure 1.4.

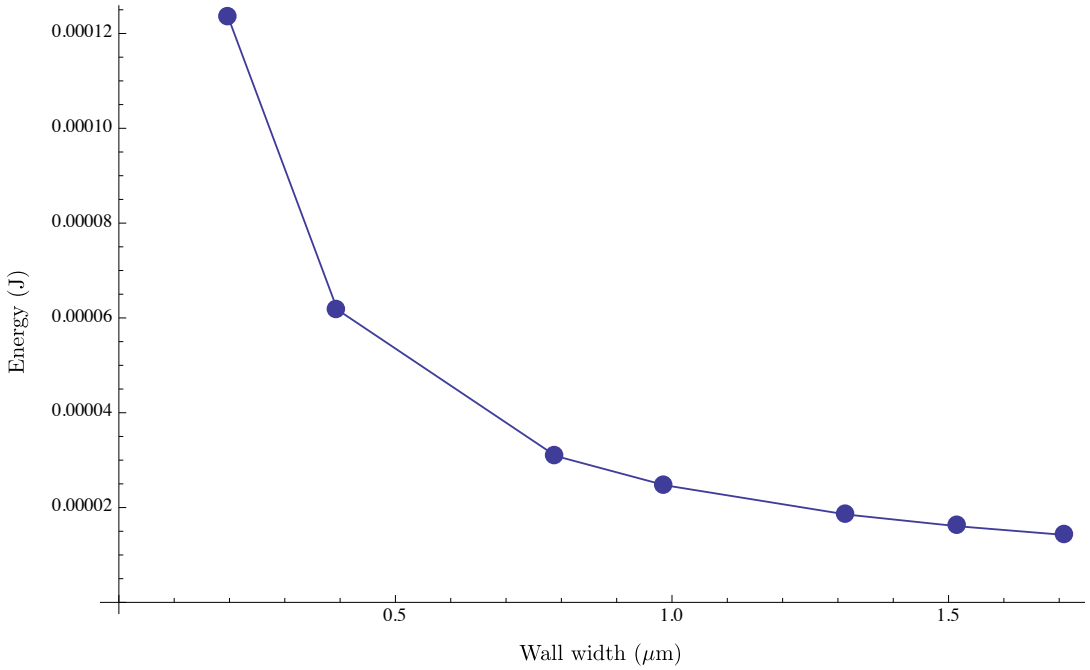
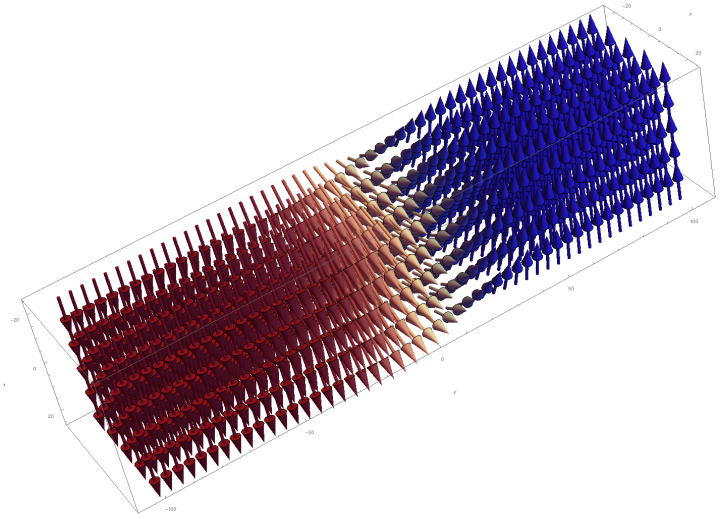
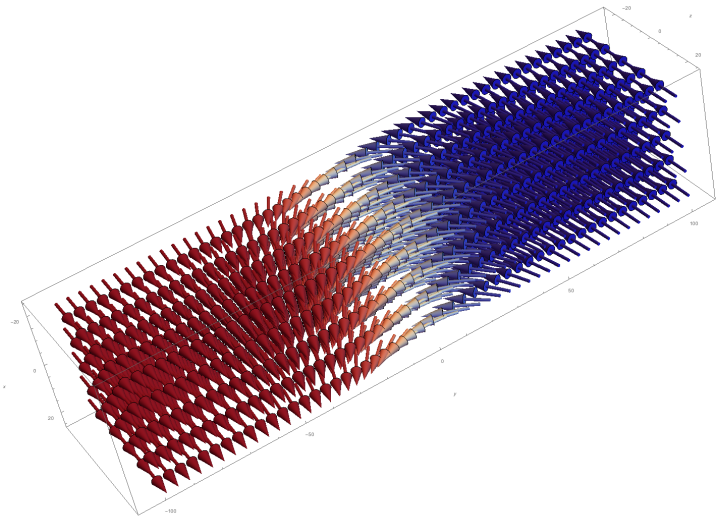


Figure 1.4: Plot of exchange energy against domain wall width using (1.11). As the domain wall becomes wider, the exchange energy likewise reduces in size.

Bloch type walls are defined as domain structures where magnetisation vectors rotate in the plane of the domain wall, as shown in figure 1.5a. Where as Néel type walls are defined as domain structures where magnetisation vectors rotate perpendicular to the domain wall as shown in figure 1.5b. Both of these types of domain strcutre may be modelled by the expression in (1.11). However in both cases narrow domain walls lead to higher energy configurations.



(a)



(b)

Figure 1.5: Two different types of domain wall, the colouring is used to represent the alignment of the domains. Within the Bloch wall (a) magnetisation is coloured according to the x component of the magnetisation - rotation is from the $(1, 0, 0)$ direction in blue, to $(-1, 0, 0)$ direction in red in the plane of the wall. The Néel wall (b) is coloured according to the z component of the magnetisation - rotation is from the $(0, 0, 1)$ direction in blue to the $(0, 0, -1)$ direction in red perpendicular to the plane of the wall.

1.3.3 Anisotropy

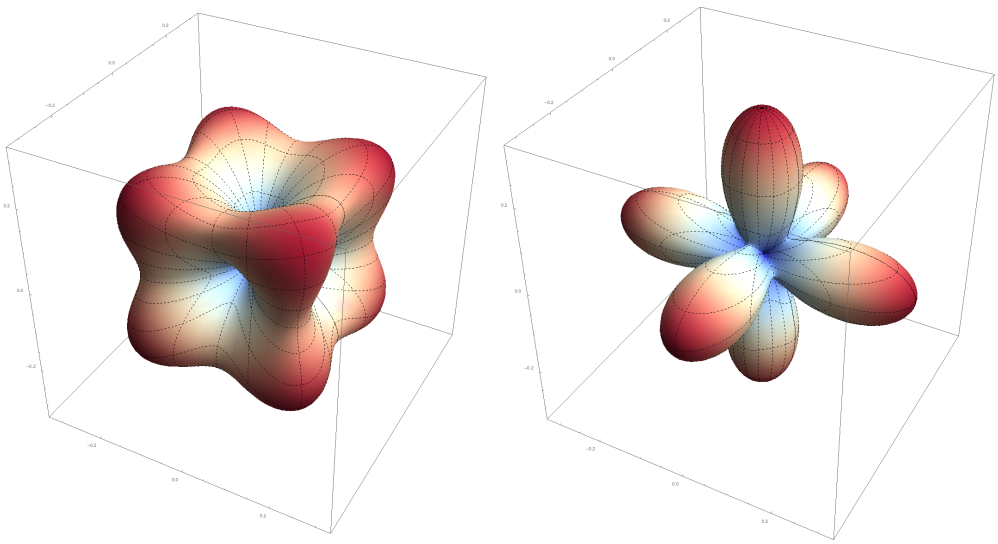
It has been shown previously that the demagnetising energy prefers small domains, where as the exchange energy prefers large gradually changing domains - in the best case one large uniform domain. This antagonism is as the heart of how domain structures form in micromagnetics. However magneto-crystalline anisotropy is an important factor final factor in determining domain structures. This component of the effective field energy is a preference for the magnetisation to lie along a particular axis. The exchange energy is usually written as

$$E_{\mathrm{a}} = K_1 V \left(\alpha_1^2\alpha_2^2 + \alpha_2^2\alpha_3^2 + \alpha_3^2\alpha_1^2\right) + K_2 V \left(\alpha_1^2\alpha_2^2\alpha_3^2\right), \tag{1.12}$$

(Dunlop, 2001) for cubic crystalline materials (which are the types of materials examined in this thesis); where α_1, α_2, α_3 represent direction cosines with respect to the $(1, 0, 0)$ axis, V is the volume of the material and K_1, K_2 are material dependent parameters. It is usual to drop the second, K_2, term for simplification since its contribution to the magneto-crystalline anisotropy is less significant (Dunlop, 2001). Furthermore, the assumption that the magnetisation is a unit vector (1.7) allows (1.12) to be written as

$$E_{\mathrm{a}} = K_1 V \left(\frac{1}{2} - \frac{1}{2}\left(\alpha_1^4 + \alpha_2^4 + \alpha_3^4\right)\right). \tag{1.13}$$

Assuming a uniformly magnetised sample of unit volume allows the energy surface resulting from (1.13) to be visualised. It can be seen that for negative anisotropies (which is the case for magnetite), the easy axes, *i.e.* axes corresponding to energy minima, correspond to the cubic diagonals.



(a) (b)

Figure 1.6: Energy surfaces for positive (a) and negative (b) magneto-crystalline anisotropies. For positive anisotropy constants K_1, the easy axes lie along $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$ directions, where as for negative anisotropy constants the easy axes lie along $(1,1,1)$, $(-1,1,1)$, $(1,-1,1)$ and $(1,1,-1)$ directions.

1.4 Micromagnetic Simulations

The theory of micromagnetics was formulated in its entirety by Brown (Brown, 1963). The total energy of a configuration of magnetisation $\boldsymbol{m}$ is given by the volume integral

$$E_{\text{tot}} = \int_{\Omega} \left(A \left|\nabla \boldsymbol{m}\right|^2 + \boldsymbol{H}_{\text{a}}(\boldsymbol{m}) - M_{\text{s}}(\boldsymbol{H}_{\text{z}} \cdot \boldsymbol{m}) - \frac{1}{2}(\boldsymbol{H}_{\text{d}} \cdot \boldsymbol{m}) \right) \, dV, \quad (1.14)$$

where the first expression $A\left|\nabla \boldsymbol{m}\right|$ corresponds to the exchange (see above), $\boldsymbol{H}_{\text{a}}$ corresponds to the anisotropy energy (see above), $\boldsymbol{H}_{\text{z}}$ corresponds to the Zeeman or externally applied field, and the final term, $-\frac{1}{2}(\boldsymbol{H}_{\text{d}} \cdot \boldsymbol{m})$ corresponds to the demagnetising field which comes from Maxwell's equations and is described in more detail in section 3.4.3.

The expression in (1.14) describes the total energy of a micromagnetic configuration. Since we are interested in minimum energy configurations, it is possible to make use of the calculus of variations and the Euler-Lagrange equations (Gelfand and Fomin, 1963) in order to minimise (1.14). By taking the functional derivative of (1.14) and then setting it to zero, Brown derived two properties that the magnetisation of a minimum energy structure should fulfil. These equations named after him

$$\boldsymbol{m} \times \frac{\partial \boldsymbol{m}}{\partial \boldsymbol{n}} = 0, \tag{1.15}$$

where $\boldsymbol{n}$ is the vector normal to the surface and

$$\boldsymbol{m} \times \boldsymbol{H}_{\text{eff}} = 0, \tag{1.16}$$

where $\boldsymbol{H}_{\text{eff}}$ is the total effective field given by

$$\boldsymbol{H}_{\text{eff}} = \boldsymbol{H}_{\text{e}} + \boldsymbol{H}_{\text{a}} + \boldsymbol{H}_{\text{z}} + \boldsymbol{H}_{\text{d}}. \tag{1.17}$$

Thus when the energy total energy functional is minimised, equation (1.16) shows that the resulting magnetisation must lie parallel to the effective field, where as equation (1.15) tells us that the magnetisation at the material edge should be parallel to $\partial \boldsymbol{m}/\partial \boldsymbol{n}$. The effective field itself consists of functional derivatives of the energy components described previously. Two strategies are then available for obtaining minimum energy solutions: 1) minimise the total energy in (1.14), using (1.18) as the basis of a graient field; or 2) solve the Landau, Lifshitz, Gilbert equation (Brown, 1963)

$$\frac{d\boldsymbol{M}}{dt} = \gamma \boldsymbol{M} \times \boldsymbol{H}_{\text{eff}} - \frac{\lambda}{M_{\text{s}}^2} \boldsymbol{M} \times \boldsymbol{M} \times \boldsymbol{H}_{\text{eff}}, \tag{1.18}$$

where γ is the gyromagnetic ratio, λ is a damping parameter and $\boldsymbol{M} = M_{\text{s}}\boldsymbol{m}$. Equation (1.18) is useful for calculating the dynamics of a micromagnetic simulation, however in practise energy minimisation techniques tend to reach equilibrium solutions in fewer steps.

Micromagnetics is a computationally challenging problem. Calculation of the total effective field is necessary for each step of a simulation be it an energy minimisation method or direct solution of the Landau, Lifshitz, Gilbert equation. The difficulty is even greater in the field of paleomagnetism where there is no control over the samples obtained in the field. Calculating realistic simulations for large irregular grains in realistic times is difficult. The finite element method is used to solve this problem, which allows approximation of irregular geometries using tetrahedral[1] meshes. It turns out that the finite element method is quite amenable to parallelisation, and so it is used in this thesis to model realistically large natural grains.

1.5 Summary

This section has given an overview of how magnetic domain structures are formed in a geological sample. The signal measured by paleomagnetists directly results from the domain structures found within ferromagnetic materials. These were compared against the other common types of magnetic materials, namely diamagnetic and paramagnetic materials and it was shown that ferromagnetic materials have the property of hysteresis (or memory). Next the reason why domain structures form was explored *i.e.* the interplay between demagnetising, exchange and anisotropy energy. Finally a brief outline of micromagnetic simulations is presented, this is explored in much more depth in chapters 2 and 3. It should be noted that the discussion above has not included mention of the magneto-elastic component of the effective field or thermal effects. Inclusion of these phenomena is beyond the scope of this thesis and is non-trivial with regards to efficient parallel implementations.

[1] Three dimensional analogues of triangles.

Bibliography

William Fuller Brown. *Micromagnetics*. Number 18. Interscience Publishers, 1963.

DJ Dunlop. Rock magnetism: fundamentals and frontiers, 2001.

IM Gelfand and SV Fomin. *Calculus of variations. Revised English edition translated and edited by Richard A. Silverman*. Prentice-Hall Inc., Englewood Cliffs, NJ, 1963.

K Keimpema, H De Raedt, and J Th M De Hosson. Electron holography image simulation of nanoparticles. *Journal of Computational and Theoretical Nanoscience*, 3(3):362–374, 2006.

Charles Kittel. Physical theory of ferromagnetic domains. *Reviews of modern Physics*, 21(4):541, 1949.

Nicola A Spaldin. Magnetic Materials, August 2010.

J Stöhr and HC Siegmann. *Magnetism: From Fundamentals to Nanoscale Dynamics*. Springer, 2006.

David Tabor. *Gases, Liquids, and Solids and Other States of Matter*. Cambridge Univ Pr, November 1991.

L Tauxe, R Butler, and SK Banerjee. *Essentials of paleomagnetism*. University of California, 2009.

Chapter 2

The Finite Element Method

2.1 Introduction

Calculation of the demagnetising field consists of solving Poisson's equation over a region of space. Techniques for solving such differential equations can be broken down in to two camps: finite difference methods and finite element/volume methods. Finite difference methods make use of local truncations of Taylor approximations of the governing equations - the so called 'strong form'. Consequently they impose local structure on the mesh (Peiró and Sherwin, 2005). In the case of cuboidal meshes, however, finite difference methods can leverage Fast Fourier Transform calculations, resulting in solutions with $\mathcal{O}(n \log n)$ computational time (n being the number of mesh vertices) (Strang, 2007). Unfortunately, finite difference methods are not suitable for complex geometries. Finite element/volume methods, on the other hand make use of local evaluations of integral forms of the governing equations (Peiró and Sherwin, 2005) and this is the approach that will be explored here.

In this chapter's description, Galerkin's approach of constructing a finite element discretisation is used. In short Galerkin's method makes the ansatz that it is possible to represent the solution of a partial differential equation (PDE) by a linear combination of functions taken from some function space (Brenner and Scott, 2008). A two dimensional example is presented since (at least conceptually) the transition from two to three dimensions is straight forward. Poisson's

equation is solved using a sample mesh with Dirichlet boundary conditions. The complete outline to solving a PDE follows :

1. derive the weak form of the differential equation,
2. discretise the mesh,
3. select test and trial functions,
4. form the (sparse) stiffness matrix A,
5. form the right hand side vector of known values $\boldsymbol{b}$,
6. solve the (sparse) linear system $A\boldsymbol{u} = \boldsymbol{f}$ (with $\boldsymbol{u}$ the vector of unknowns).

This procedure is found in several texts. Strang (2007) provides a good overview of the method. Ottosen et al. (1992) gives a solid introduction with many practical examples. This chapter builds primarily on Davies (1980) and Logg et al. (2012). Brenner and Scott (2008) and Strang and Fix (1973) provide a much more rigorous and mathematical overview of the finite element method.

2.2 Derivation of the Weak Form

Poisson's equation is very important in the physical sciences. It is usually written

$$\nabla^2\phi(\boldsymbol{x}) = f(\boldsymbol{x}), \tag{2.1}$$

namely that the second derivative of the scalar quantity ϕ at the spacial point $\boldsymbol{x}$ is equal to some known scalar value f at $\boldsymbol{x}$ (sometimes referred to as the force term). Equation (2.1) holds over some region, denoted Ω with boundary $\partial\Omega$ and is referred to as the 'strong form'.

Additional information is required to solve (2.1) in the form of boundary conditions. Dirichlet conditions specify the value of ϕ on the boundary, for example $\phi(\boldsymbol{x}) = g_D(\boldsymbol{x})$ with $\boldsymbol{x} \in \partial\Omega$. Neumann conditions on the other hand, specify the value of the first derivative of ϕ along the surface with respect to the surface normal $\hat{n}$, for example $\frac{\partial\phi}{\partial\hat{n}} = g_N(\boldsymbol{x})$ with $\boldsymbol{x} \in \partial\Omega$.

An important feature of equation (2.1) is that it must hold everywhere, since $\boldsymbol{x}$ ranges over every point in the region Ω. In order to derive a weak form, equation (2.1) is multiplied by a function $v(\boldsymbol{x})$. This funcion may be chosen to be whatever is convenient for the solution of the problem. In this example, $v(\boldsymbol{x})$ will be chosen (see below) to be a linear combination of 'hat functions' (defined below) which are non-zero over a small finite region. The integral of the subsequent product is then taken over the whole region Ω,

$$\int_\Omega v(\boldsymbol{x})\nabla^2\phi(\boldsymbol{x})\ dV = \int_\Omega v(\boldsymbol{x})f(\boldsymbol{x})\ dV \tag{2.2}$$

By application of Green's identity (B.1) this becomes

$$-\int_\Omega \nabla v(\boldsymbol{x})\cdot\nabla\phi(\boldsymbol{x})\ dV + \int_\Omega \nabla\cdot(v(\boldsymbol{x})\nabla\phi(\boldsymbol{x}))\ dV = \int_\Omega v(\boldsymbol{x})f(\boldsymbol{x})\ dV \tag{2.3}$$

and by application of the divergence theorem (B.2) the following form is derived

$$-\int_\Omega \nabla v(\boldsymbol{x})\cdot\nabla\phi(\boldsymbol{x})\ dV + \int_{\partial\Omega} v(\boldsymbol{x})\nabla\phi(\boldsymbol{x})\cdot\hat{n}\ dS = \int_\Omega v(\boldsymbol{x})f(\boldsymbol{x})\ dV \tag{2.4}$$

Applying boundary conditions will change the surface term in equation (2.4). For example, upon application of pure Dirichlet conditions, the surface term will disappear since the test functions comprising $v(\boldsymbol{x})$ will be chosen such that they are exactly zero on the boundary (since the exact value of ϕ is known on the boundary). For Neumann boundary conditions the surface term moves to the right hand side of (2.4) and is incorporated in to the force term.

If pure Neumann conditions are defined with respect to the surface normal, then equation (2.1) is only unique up to some additive constant. This can be seen by adding a constant to a solution of (2.1)

$$\nabla^2(\phi + k) = \nabla^2\phi + \nabla^2 k = \nabla^2\phi + 0 = \nabla^2\phi$$

furthermore for the boundary condition

$$\nabla (\phi + k) \cdot \hat{n} = \nabla \phi \cdot \hat{n} + \nabla k \cdot \hat{n} = \nabla \phi \cdot \hat{n} + 0 = \nabla \phi \cdot \hat{n}$$

The treatment presented here illustrates how the solution of (2.1) with pure Dirichlet boundary conditions is computed. However the pure Neumann problem has relevance when solving the demagnetising field using the hybrid FEM-BEM approach (Fredkin and Koehler, 1990).

2.3 Mesh Discretisation, Test and Trial Functions

In order to construct a linear system amenable to solution on a computer the region Ω is discretised in to subregions. Several strategies are possible, but this example assumes a planar region discretised in to triangles - since they are the two dimensional analogue tetrahedral elements. Figure 2.2 shows a region discretised in to 19 elements, with 16 nodes. The symbol N is used to represent the set of mesh nodes and the symbol E is used to represent the set of mesh elements.

This chapter assumes that interpolating basis functions are linear. Using interpolating functions of higher degree is conceptually straight forward but becomes more complicated notationally and when implementing code.

Definition 1. Neighbour elements of a node.

The symbol $N_e : \mathbb{Z} \to \mathbb{P}_3(\mathbb{Z})$ represents the neighbour elements function for some node i in a triangular mesh. It is the collection of unordered triples consisting of three integers containing i. This may be written as

$$N_e(i) = \{\{i, j_1, k_1\}, \{i, j_2, k_2\}, \cdots, \{i, j_n, k_n\}\} \tag{2.5}$$

Definition 2. Facet function.

The symbol $\psi_{i,j,k} : \mathbb{R}^2 \to \mathbb{R}$ represents the facet function over the triangle $\{(n_i^x, n_i^y), (n_j^x, n_j^y), (n_k^x, n_k^y)\}$ (for brevity henceforth referred to as the triangle

$\{i, j, k\}$). It has the property that

$$\psi_{i,j,k}\left(n_i^x, n_i^y\right) = 1 \tag{2.6}$$
$$\psi_{i,j,k}\left(n_j^x, n_j^y\right) = 0 \tag{2.7}$$
$$\psi_{i,j,k}\left(n_k^x, n_k^y\right) = 0 \tag{2.8}$$

The expression for $\psi_{i,j,k}$ over the triangle $\{i, j, k\}$ is given by the following equation for a plane in three dimensions

$$\psi_{i,j,k}(x, y) = \frac{c_c}{c_\psi} - \frac{c_x}{c_\psi}x - \frac{c_y}{c_\psi}y \tag{2.9}$$

with

$$c_c = \begin{vmatrix} n_i^x & n_j^x & n_k^x \\ n_i^y & n_j^y & n_k^y \\ 1 & 0 & 0 \end{vmatrix} \quad c_x = \begin{vmatrix} 1 & 1 & 1 \\ n_i^y & n_j^y & n_k^y \\ 1 & 0 & 0 \end{vmatrix} \quad c_y = \begin{vmatrix} n_i^x & n_j^x & n_k^x \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{vmatrix} \quad c_\psi = \begin{vmatrix} n_i^x & n_j^x & n_k^x \\ n_i^y & n_j^y & n_k^y \\ 1 & 1 & 1 \end{vmatrix} \tag{2.10}$$

The value for $\psi_{i,j,k}$ outside the triangle $\{i, j, k\}$ is zero everywhere. As an example consider calculating $\psi_{1,2,3}$ for triangle $n_1 = (0, 0)$, $n_2 = (1, 0)$ and $n_3 = (0, 1)$ as illustrated in figure 2.1, then $\psi_{1,2,3}(x, y) = 1 - x - y$. It is simple to extend this definition to higher dimensions such as three dimensional tetrahedra. Appendix C presents mathematica code to calculate facet functions in three dimensions.

Definition 3. Hat function.

The symbol $\alpha_i : \mathbb{R}^2 \to \mathbb{R}$ represents the hat function about node i. Diagrammatically it is the cone with polygonal base consisting of the neighbour elements of node i with unit height (see figure2.2). The hat function is then the sum of all the facet functions over $N_e(i)$

$$\alpha_i(x, y) = \sum_{\{i,j,k\} \in N_e(i)} \psi_{i,j,k}(x, y) \tag{2.11}$$

In the previous section, equation (2.1) was multiplied by the function $v(\boldsymbol{x})$ in order to derive the weak form (2.4). In order to derive a discretised version of

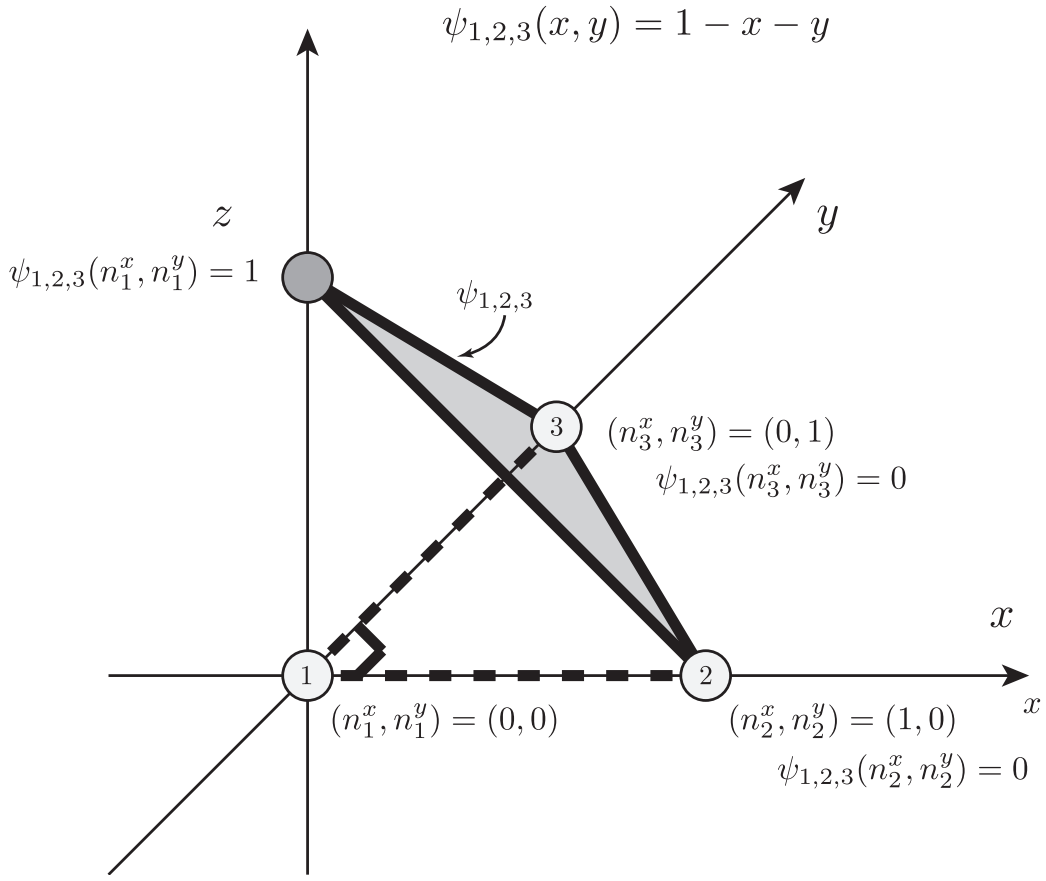


Figure 2.1: Example facet function $\psi_{1,2,3}$ for a triangle with node coordinates $(n_1^x = 0, n_1^y = 1), (n_2^x = 1, n_2^y = 0), (n_3^x = 0, n_3^y = 1)$. As can be seen the resulting plane passes through the points $(0,0,1), (1,0,0)$ and $(0,1,0)$ with (x,y) coordinates within the triangle varying linearly. $\psi_{1,2,3}$ is defined to be zero outside the triangle. The equation of the plane given by $\psi_{1,2,3}(x,y) = 1 - x - y$ evaluates to $\psi_{1,2,3}(0,0) = 0$, $\psi_{1,2,3}(1,0) = 0$ and $\psi_{1,2,3}(0,1) = 0$ as expected.

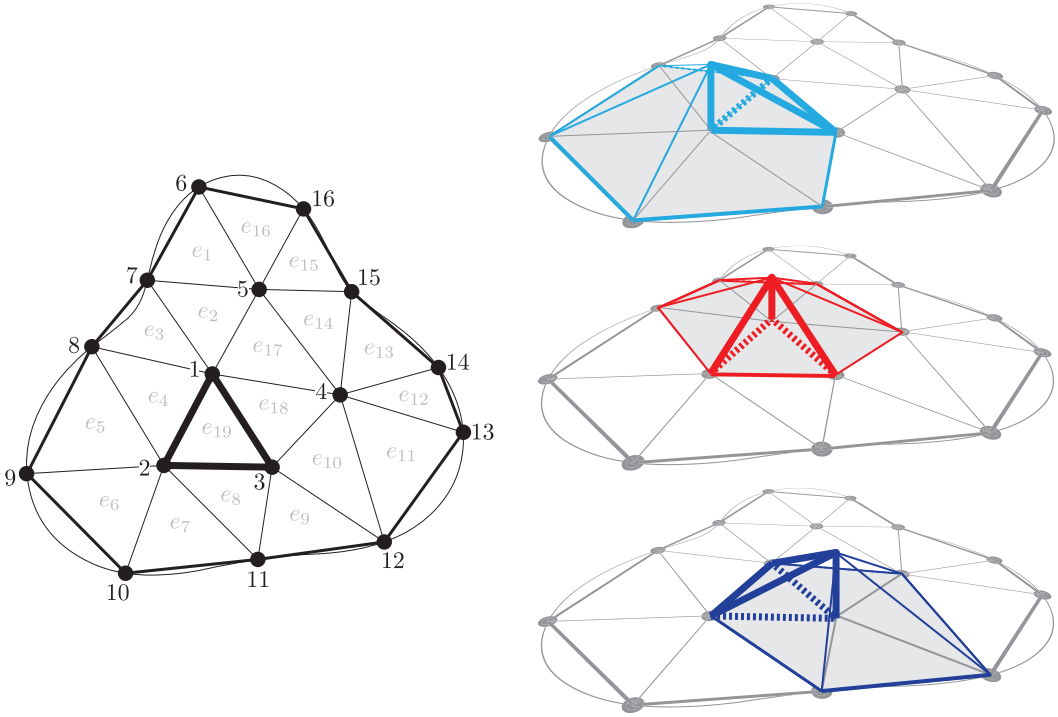


Figure 2.2: A two dimensional planar mesh (left); nodes are numbered and elements are denoted light gray. For any given node, a two dimensional 'hat' function may be constructed about it (right). Note that three facet functions are defined over the element e_{19}; these facet functions belong to hat functions defined at each of the three nodes of e_{19}, this observation is important when considering assembly of the stiffness matrix.

the problem, this function is represented as a linear combination of hat functions (denoted $\bar{v}$). These are then referred to as test functions,

$$v(\boldsymbol{x}) \approx \bar{v}(x, y) = \sum_{i \in N} c_i \alpha_i(x, y), \tag{2.12}$$

where the coefficients c_i represent some known value.

Likewise the solution may be approximated using the trial function $\bar{\phi}$, which is once more represented using a linear combination of hat functions.

$$\phi(\boldsymbol{x}) \approx \bar{\phi}(x, y) = \sum_{i \in N} u_i \alpha_i(x, y) \tag{2.13}$$

Here the same basis functions, *i.e.* the hat functions of definition 3, are used for both test and trial functions; if this is the case then they are referred to as *shape functions*. Note, it is not necessary for basis functions of test and trial functions to come from the same function space.

2.4 Constructing a Linear System

In order to solve the Poisson equation, the weak form of equation (2.4) is discretised. The region Ω is approximated by an assembly of triangles, $\Omega' = \bigcup_{e \in E}$. Furthermore, since shape functions are defined over each triangle, (2.4) may be approximated by substituting (2.12) and (2.13)

$$-\int_{\Omega'} \nabla \bar{v} \cdot \nabla \bar{\phi} \, dV = \int_{\Omega'} \bar{v} f \, dV \tag{2.14}$$

Component-wise, in two dimensions, this equation may then be written

$$-\int_{\Omega'} \sum_{i \in N} u_j \left(\partial_x \alpha_i \partial_x \alpha_j + \partial_y \alpha_i \partial_y \alpha_j \right) \, dV = \int_{\Omega'} \alpha_j f_j \, dV. \tag{2.15}$$

The indices i and j vary over the nodes of the triangulation of Ω'. The symbols ∂_x and ∂_y represent partial derivatives with respect to x and y respectively. For each unknown u_j in equation (2.15) there exists an equation of $|N|$ variables in i

(in practise most of these entries will evaluate to zero). For example, when $j = 1$ the following equation will arise

$$-u_1 \int (\partial_x \alpha_1 \partial_x \alpha_1 + \partial_y \alpha_1 \partial_y \alpha_1) + \cdots + \left(\partial_x \alpha_1 \partial_x \alpha_{|N|} + \partial_y \alpha_1 \partial_y \alpha_{|N|}\right)\, dV = \int \alpha_1 f_1 \, dV$$

Thus a linear system $A\boldsymbol{u} = \boldsymbol{f}$ may be constructed, where A is a matrix (also known as the stiffness matrix). The vector $\boldsymbol{u}$ contains unknowns and $\boldsymbol{f}$ is a vector of known values incorporating the right hand side of (2.1).

2.4.1 Stiffness Matrix Sparsity

Since the integrals in equation (2.15) are over sums, the sums may be moved through the integral sign. Therefore each integral will take place over the neighbour elements of a given node $N_e(i)$ or $N_e(j)$. However, hat functions are defined to be locally compact (they are zero everywhere except on some finite domain). This means that only nodes that share elements will appear as non-zero in the resulting stiffness matrix. For example figure 2.3 illustrates two hat functions that do not share elements (centred on nodes 2 and 4). Defining $\partial_x \alpha_i \partial_x \alpha_j + \partial_y \alpha_i \partial_y \alpha_j \equiv D(\alpha_i, \alpha_j)$ and expanding for $j = 2$ gives

$$\begin{aligned} -u_2 \Bigg[&\int_{\Omega'} D(\alpha_1, \alpha_2)\, dV + \int_{\Omega'} D(\alpha_2, \alpha_2)\, dV \\ &+ \int_{\Omega'} D(\alpha_3, \alpha_2)\, dV + \int_{\Omega'} D(\alpha_4, \alpha_2)\, dV \\ &+ \cdots \\ &+ \int_{\Omega'} D(\alpha_{16}, \alpha_2)\, dV \Bigg] = \int_{\Omega'} \alpha_2 f_2 \, dV \end{aligned}$$

Considering only the integral corresponding for $i = 4$

$$\int_{\Omega'} D(\alpha_4, \alpha_2)\, dV = \int_{\Omega'} (\partial_x \alpha_4 \partial_x \alpha_2 + \partial_y \alpha_4 \partial_y \alpha_2)\, dV$$

it can be seen that this integral must evaluate to zero, since by figure 2.3 and definition 2 the only region of Ω' in which these functions overlap evaluates to

zero. Hence it can be seen that if two shape functions do not share a domain of integration then their contribution to the stiffness matrix must be zero.

Considering node 2 it can be seen by a similar argument that contributions from nodes 4, 5, 6, 7, 12, 13, 14, 15 and 16 will evaluate to zero in the resulting stiffness matrix - this is over half the nodes of the mesh. Consequently the stiffness matrix A is sparse and most entries evaluate to zero.

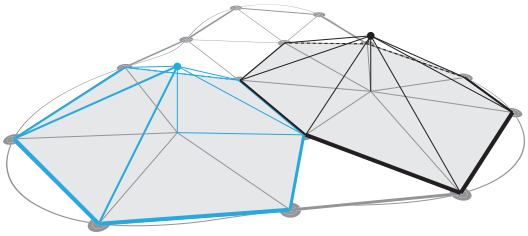


Figure 2.3: Two shape functions found in figure 2.2. Since neither the blue, α_2, nor the black, α_4, shape functions share an element, the corresponding entries in the stiffness matrix: $a_{24} = a_{42} = 0$.

2.4.2 Element-wise Construction of A

It was seen that shape functions that share elements result in non-zero entries in the resulting stiffness matrix. This observation hints at a more efficient way of constructing the stiffness matrix A.

Consider some element in $e \in \Omega'$ with local node indices e_p, e_q and e_r. The maximum number of hat functions defined over element e must be 3 since a triangle has three nodes, each one defining a hat function. Let these functions be denoted α_{e_p}, α_{e_q} and α_{e_r}. Then the 9 integrals

$$\int_e D(\alpha_{e_p}, \alpha_{e_p})\, dV \qquad \int_e D(\alpha_{e_p}, \alpha_{e_q})\, dV \qquad \int_e D(\alpha_{e_p}, \alpha_{e_r})\, dV$$
$$\int_e D(\alpha_{e_q}\alpha_{e_p})\, dV \qquad \int_e D(\alpha_{e_q}, \alpha_{e_q})\, dV \qquad \int_e D(\alpha_{e_q}, \alpha_{e_r})\, dV$$
$$\int_e D(\alpha_{e_r}, \alpha_{e_p})\, dV \qquad \int_e D(\alpha_{e_r}, \alpha_{e_q})\, dV \qquad \int_e D(\alpha_{e_r}, \alpha_{e_r})\, dV$$

must evaluate to non-zero since over e since: α_p, α_q and α_r are themselves non-zero and greater than or equal to 1.

Since each hat function consists of a sum of facet functions it is possible to construct a local '*element stiffness*' matrix for e for only the facet functions defined over the element e.

$$\begin{pmatrix} \int_e D(\psi_{e_p,e_q,e_r}, \psi_{e_p,e_q,e_r})\, dV & \int_e D(\psi_{e_p,e_q,e_r}, \psi_{e_q,e_r,e_p})\, dV & \int_e D(\psi_{e_p,e_q,e_r}, \psi_{e_r,e_p,e_q})\, dV \\ \int_e D(\psi_{e_q,e_r,e_p}, \psi_{e_p,e_q,e_r})\, dV & \int_e D(\psi_{e_q,e_r,e_p}, \psi_{e_q,e_r,e_p})\, dV & \int_e D(\psi_{e_p,e_q,e_r}, \psi_{e_r,e_p,e_q})\, dV \\ \int_e D(\psi_{e_r,e_p,e_q}, \psi_{e_p,e_q,e_r})\, dV & \int_e D(\psi_{e_r,e_p,e_q}, \psi_{e_q,e_r,e_p})\, dV & \int_e D(\psi_{e_r,e_p,e_q}, \psi_{e_r,e_p,e_q})\, dV \end{pmatrix} \tag{2.16}$$

with

$$\int_e D(\psi_{e_p,e_q,e_r}, \psi_{e_q,e_r,e_p})\, dV \equiv \int_e D(\alpha_p, \alpha_q)\, dV$$

Each entry in (2.16) corresponds to some global mesh index. In order to construct the global stiffness matrix A, each entry in (2.16) is added to the corresponding global position in A as per figure 2.4.

By repeatedly evaluating and inserting local stiffness matrices it is possible to construct the complete stiffness matrix A. Figure 2.5 illustrates the final sparsity pattern when constructing the mesh element by element, as can be seen the vast majority of the resulting stiffness matrix consists of zero/empty values.

Construction by elements is considerably more efficient than construction by nodes since naively node-wise construction would result in $O(|N|^2)$ evaluations most of which would be zero. However construction by element takes $O(|E|)$ resulting in a linear number of calculations as the size of the mesh increases. This also indicates that the sparse matrix's size grows linearly with the number of mesh nodes/elements.

2.4.3 Boundary Conditions

When applying Dirichlet boundary conditions some mesh nodes, *i.e.* the nodes in $\partial\Omega$ are given exact values according to g_D. Consequently it is no longer necessary to solve ϕ for boundary nodes since the solution is already known at these points.

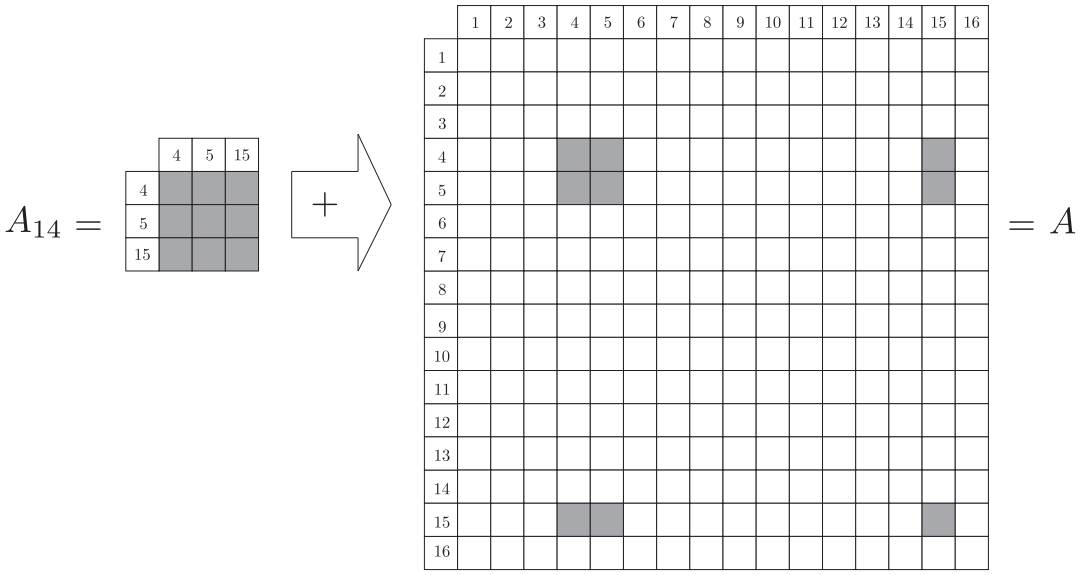


Figure 2.4: The local stiffness matrix A_{14} is calculated using (2.16), each of the corresponding local element indices are 'inserted' in to the global stiffness matrix A. Insertion is simply the procedure of adding the values of A_{14} to A at global node indices 4, 5 and 15.

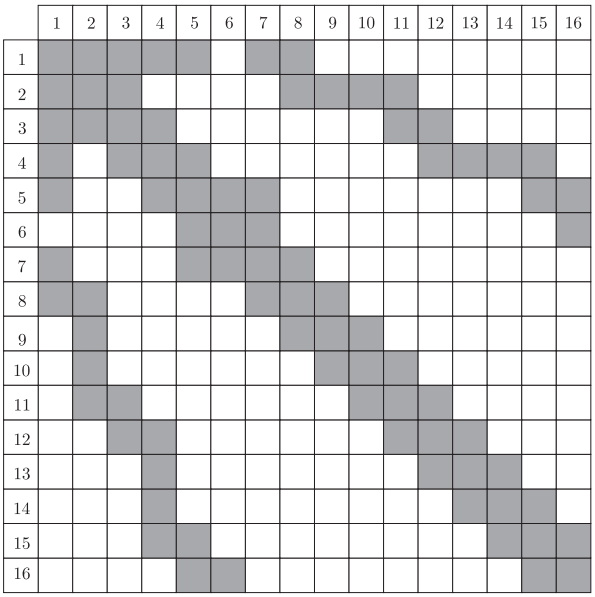


Figure 2.5: The sparsity pattern for the stiffness matrix resulting from the mesh found in figure 2.2.

The manner by which this is usually encoded in the stiffness matrix A and the forcing vector $\boldsymbol{b}$ is: for some fixed node index I on which a Dirichlet boundary condition holds

- set the Ith rows and columns of A as

$$\begin{aligned} a_{Ij} &= 0 \\ a_{jI} &= 0 \\ a_{II} &= 1 \end{aligned} \tag{2.17}$$

- set the b_Ith entry of the right hand side vector to the corresponding Dirichlet value

$$b_I = g_D(\boldsymbol{x}_I) \tag{2.18}$$

For example, if applying pure Dirichlet boundary conditions to the problem mesh of figure 2.2 then nodes 6 to 16 are points where g_D applies. Hence rows 6 to 16 of the associated stiffness matrix (see figure 2.2) are set to zero everywhere, except the diagonal (where they are set to one). Furthermore, components 6 to 16 in vector $\boldsymbol{b}$ are set to the Dirichlet values, thus we have the system outlined in figure 2.6.

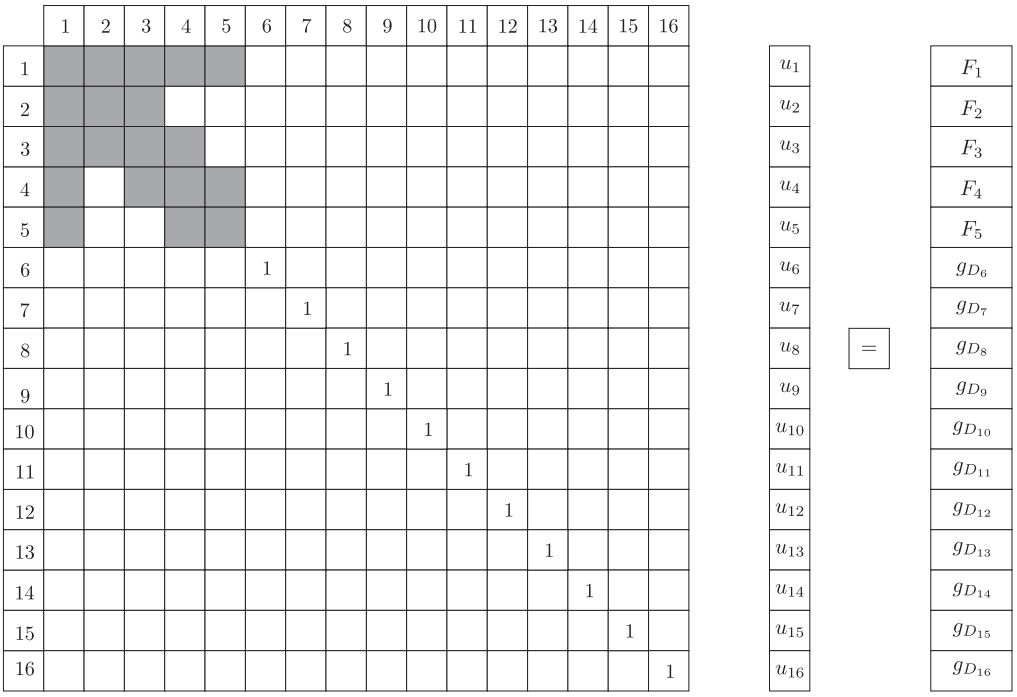


Figure 2.6: The linear system $A\boldsymbol{x} = \boldsymbol{b}$ resulting from the application of pure Dirichlet boundary conditions to the mesh found in figure 2.2.

2.4.4 Integration Over Triangles

An important detail that ties together evaluation of (2.16) and and the facet functions defined in 2 is that evaluating the integrals in (2.16) becomes complicated by the fact that bounds for the double integrals (which is what the $\int_e \bullet \, dV$'s become) are not known. There are two options to resolve this - one is to calculate

the integrals bounds in situ, the other is to transform the element e to some reference element with known bounds of integration. The latter option is described here, however the former option is also commonly found in finite element code.

The reference element in this example is the triangle with coordinates $(0,0)$, $(1,0)$ and $(0,1)$ shown in figure 2.7. The affine transformation that maps coordinates of the reference triangle on to an element of the mesh with coordinates (x_1, y_1), (x_2, y_2) and (x_3, y_3) is given by

$$\boldsymbol{x}' = \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} x_3 \\ y_3 \end{pmatrix} \tag{2.19}$$

The integrals in (2.16) may then be evaluated using

$$\int_0^1 \int_0^{1-x} D(\psi_{e_p,e_q,e_r}, \psi_{e_{p'},e_{q'},e_{r'}})(x', y') \; |J(\boldsymbol{x'})| \; dy \; dx \tag{2.20}$$

where $J(\boldsymbol{x}')$ is the Jacobian matrix of the transformation (2.19) and $e_{p'}$, $e_{q'}$ and $e_{r'}$ represent some cyclic permutation of the local node indices e_p, e_q and e_r for an element e.

$$J(\boldsymbol{x}') = \begin{pmatrix} \frac{\partial x'}{\partial x} & \frac{\partial x'}{\partial y} \\ \frac{\partial y'}{\partial x} & \frac{\partial y'}{\partial y} \end{pmatrix} \tag{2.21}$$

2.5 Solving The System

In order to solve the system $A\boldsymbol{u} = \boldsymbol{f}$ two options are available. The first is to take the system $A\boldsymbol{u} = \boldsymbol{f}$ and work with it directly. For example, it is possible to factor the matrix A in to a product of two matrices L and U with the properties that:

- L is a unit lower-diagonal matrix, *i.e.* only elements *below* the main diagonal are non-zero and entries *on* the main diagonal are 1,
- U is an upper-diagonal matrix, *i.e* only elements on or *above* the main diagonal are non-zero.

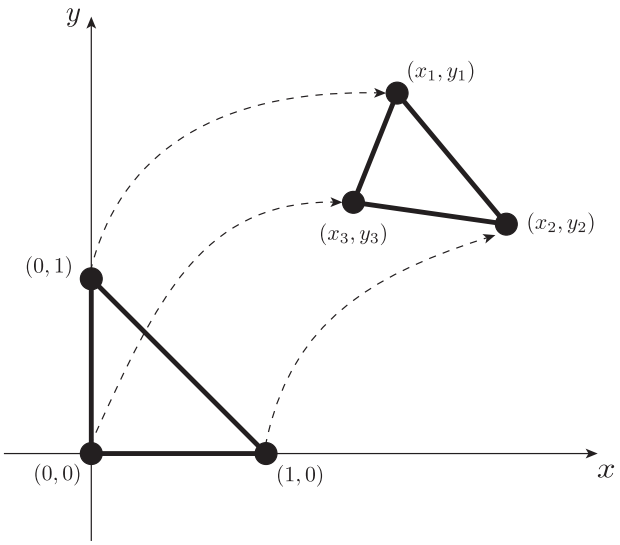


Figure 2.7: The reference element in two dimensions. Points of the reference element are mapped to a corresponding element in the finite element space. This allows integrals of equation 2.16 to be evaluated.

Techniques to factorise A in to the product L and U are usually based on some modification of Gaussian elimination, for example Crout's algorithm and Doolittle's algorithm. As such they take $O(n^3)$ arithmetic operations (where n is the number of rows/colums of matrix $\boldsymbol{A}$) to complete . Solution of the system proceeds as follows (Press, 2007)

1. solve the system $L\boldsymbol{y} = \boldsymbol{f}$ by forward substitution using

$$y_0 = \frac{f_0}{l_{00}}$$
$$y_i = \frac{1}{l_{ii}} \left[f_i \sum_{j=0}^{i-1} l_{ij} y_j \right] \tag{2.22}$$

2. solve the system $U\boldsymbol{u} = \boldsymbol{y}$ by backwards substitution using

$$x_{N-1} = \frac{y_{N-1}}{b_{N-1,N-1}}$$
$$x_i = \frac{1}{u_{ii}}\left[y_i - \sum_{j=i+1} u_{ij}x_j\right] \tag{2.23}$$

The triangular nature of the sums in equations (2.22) and (2.23) suggests a running time of $O(n^2)$. Hence once the factorisation $A = LU$ has been computed and the cost of $O(n^3)$ has been incurred, subsequent solutions for changing right hand sides of $A\boldsymbol{u} = \boldsymbol{f}$ are much more efficient.

2.5.1 Iterative Solvers

It has been observed previously that matrices that arise from application of finite element methods are in general sparse. While LU-factorisation is a good method for solving linear systems, it is mainly useful when the matrix A is dense. The sparsity structure is not guaranteed to be preserved after factorisation and for extremely large matrices storage becomes an issue. Two problems arise: 1) how should the linear system $A\boldsymbol{u} = \boldsymbol{b}$ be stored, 2) how can the system be solved?

In order to answer the second question, it is possible to consider an alternative problem, namely minimisation of

$$f(\boldsymbol{u}) = \frac{1}{2}\boldsymbol{u}^T A\boldsymbol{u} - \boldsymbol{b}\boldsymbol{x} + c \tag{2.24}$$

Such minima occur when $f'(\boldsymbol{u}) = 0$ and when the derivative of the quadratic form (2.24) is taken with respect to $\boldsymbol{u}$ (2.25), the linear system $A\boldsymbol{u} = \boldsymbol{b}$ appears (given symmetric A).

$$f'(\boldsymbol{u}) = \frac{1}{2}\left(A^T + A\right)\boldsymbol{u} - \boldsymbol{b} \tag{2.25}$$

Steepest Descent

In order to build a solution method out of (2.24), consider starting at some point $\boldsymbol{u}_0$ and attempting to travel *down* the path of steepest descent, -$f'(\boldsymbol{u})$, from that

$\boldsymbol{u}_0$. Then the next point, $\boldsymbol{u}_1$, is given by

$$\boldsymbol{u}_i = \boldsymbol{u}_{i-1} - \alpha f'(\boldsymbol{u}_i) \tag{2.26}$$

where α is some scalar controlling the amount to move in the steepest descent direction, that is usually found by performing a line search in the direction of steepest descent. A quantity called the *residual* is defined

$$\boldsymbol{r}_i = \boldsymbol{b} - A\boldsymbol{u}_i \tag{2.27}$$

This quantity gives the difference between the proposed solution $\boldsymbol{u}_i$ and the actual solution - which is unknown but has the property $\boldsymbol{b} - A\boldsymbol{u} = 0$. The important observation about the residual is that it is the negative of (2.25) (for symmetric equations).

When the residual (2.27) is near enough to zero, the method can halt and the final $\boldsymbol{u}_i$ is the vector minimising 2.24 and thus solving $A\boldsymbol{x} = \boldsymbol{b}$. The pseudo code (alg. 1) outlines the steepest descent method, with the optimisation that only one matrix vector multiplication is performed (Shewchuk, 1994).

Algorithm 1 Solution of the system $A\boldsymbol{u} = \boldsymbol{b}$ by the steepest descent method.

```
1  function SteepestDescent(A, u, b, ExitDiff, MaxI, RsclFrq)
2    while ExitDiff < |r_i| AND i ≤ MaxI do
3      if i mod RsclFrq = 0 then
4        r_i ← b - Ax_i
5      end if
6      t_i ← Ar_i
7      α_i ← (r_i·r_i)/(r_i·t_i)
8      r_{i+1} = r_i - α_i t_i
9      x_{i+1} = x_i + α_i r_i
10   end while
11 end function
```

Algorithm (alg. 1) is useful since it will form the basis of a subsequent energy minimising scheme for micromagnetic simulations called the *Hubert minimiser*

(see chapter 3). Furthermore it is important to note that the calculation is dominated by matrix vector multiplication and so optimising this operation becomes critical for efficiently solving linear systems of the form $A\boldsymbol{u} = \boldsymbol{b}$. By taking conjugate directions (*i.e.* mutually orthogonal search directions with respect to the matrix A) (alg. 1) becomes the Conjugate Gradient method (Saad, 2003) - a scheme requiring fewer iterations when solving linear systems (as long as they are symmetrical and positive definite).

2.5.2 Sparse Matrix Formats

Meshes consisting of large numbers of elements result in very large finite element matrices. When using first order Lagrange elements the size of this matrix would consist of n^2 elements where n is the number of mesh vertices. This storage requirement is very large, for example a mesh consisting of $100,000$ vertices would result in a matrix of approximately 80GB, assuming matrix entries are stored in double precision (typically of size 8 bytes each). Since most stiffness matrix entries evaluate to zero, it is possible to avoid storing these quantities. The typical sparse storage scheme is the Compressed Sparse Row/Column (CSR/CSC) scheme. CSR is described here, however CSC is similar except that the matrix is read column-wise.

Consider the sparse matrix depicted in figure 2.8. In order store this matrix in CSR format, three arrays are required `values`, `columnIndex` and `rowPointer`. The `values` array stores all of the non-zero entries in the matrix; the `columnIndex` array will store the column index for each of the values in `values`. The `rowPointer` array contains an index in to the `values` array indicating the element that starts each row. Indices that are next to each other in `rowPointer` may then be used to compute the indices of `values` that occupy a row, then for row i matrix non-zeros will consist of entries in `values` between `rowPointer[i]` and `rowPointer[i+1]-1`. This means that the `rowPointer` array contains the same number of entries as there are rows in the matrix plus one - this last value holds the total number of non-zero entries of the matrix.

Implementing matrix-vector multiplications are fairly straight forward for sparse matrix schemes using CSR. An example algorithm is outlined in (alg.

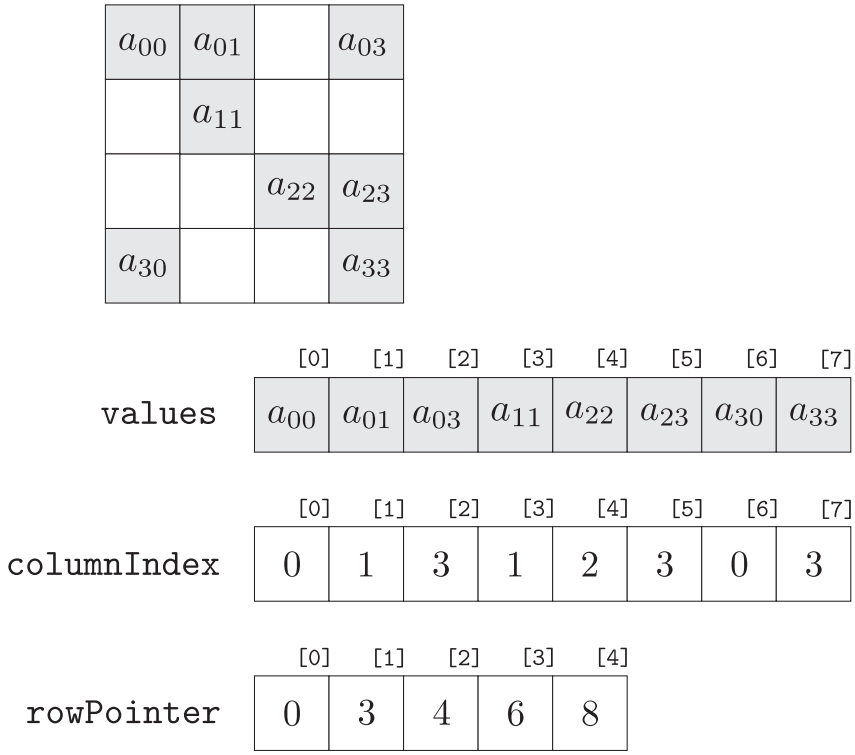


Figure 2.8: Example of a matrix stored in CSR format. Three arrays are required for such storage: 1) `values` to hold the non-zero matrix entries, 2) `columnIndex` to hold the column to which a `values` entry belongs and 3) `rowPointer` to hold indices in to the `values` array indicating the start and end non-zeros of a row.

2). The advantage of sparse storage is twofold. 1) the storage requirements are reduced drastically, 2) matrix vector operations do not require superfluous multiplications by zero to be performed. This means that matrix vector multiplication becomes linear in the number of matrix non-zero entries.

Algorithm 2 A function to multiply the vector `v` by the matrix `A` in CSR format. The result is stored in output vector `u`.

```
1 function MatrixVectorMultiply(A, v, u)
2   u[:]  = 0
3   for i = 0 to nrows-1 do
4     for j=rowPointer[i] to rowPointer[i]-1 do
5       u[i] ← u[i] + values[j] * v[columnIndex[j]]
6     end for
7   end for
8 end function
```

2.5.3 Bandwidth Minimisation

The matrix sparsity pattern such as that illustrated in figure 2.5 is suboptimal. For example, off diagonal elements lead to poor caching behaviour when performing matrix-vector multiplications. Consider the matrix vector multiplication $\boldsymbol{y} = A\boldsymbol{x}$. If A is in CSR format then off diagonal elements are stored contiguously in memory. The processor may then cache row values for the multiplication. However the indexing of the vectors is not efficient since these data structures are stored in a dense format. This means that the processor cannot make full use of spatial locality for caching of vector values.

Furthermore off diagonal elements have a detrimental effect on parallelisation, since the non-contiguous vectors will mean that when communication happens lots of small sections of the vectors will be send between processors. This results in latency dominated communication.

In order to alleviate these problems it should be noted that the sparsity structure of the stiffness matrix depends strongly upon the labelling of mesh vertices.

The Cuthill-McKee algorithm (Cuthill and McKee, 1969) is a method of relabelling the rows and columns of a matrix such that the sparsity pattern is diagonally dominant. The procedure begins by constructing a graph structure based on the elements of the matrix to be 'bandwidth-minimised'. For first order Lagrangian elements, this is exactly the input matrix. Next a modified breadth first search algorithm is performed, starting with a vertex of minimum degree (*i.e.* the vertex connected to the fewest neighbours); this vertex is labelled 1. The algorithm then proceeds in a series of 'levels' each level being one edge deeper from the starting vertex. The vertices of each level are labelled according to their degree. Figure 2.9 outlines the four stages of relabelling the graph (see figure 2.2) corresponding to the stiffness matrix (in figure 2.5). As can be seen the levels correspond to the breadth-first search front and nodes are numbered in degree order (smallest first). For completeness, the sparse matrix in figure 2.5 looks like figure 2.10 after bandwidth minimisation.

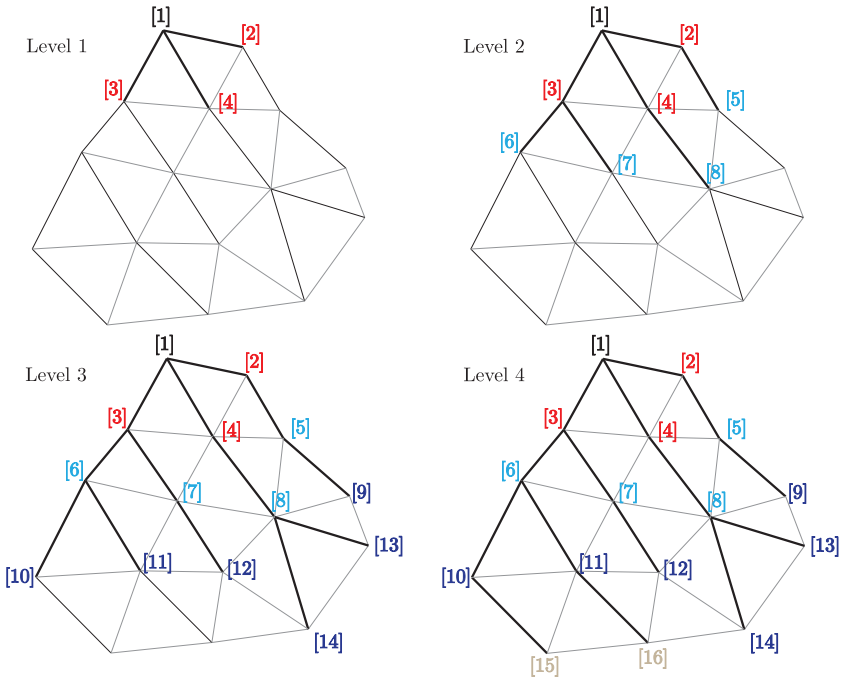


Figure 2.9: Progress of the Cuthill-Mckee bandwidth minimisation algorithm on a small two dimensional mesh. The algorithm proceeds in four main iterations or 'levels', with each level the result of a breadth-first tree traversal.

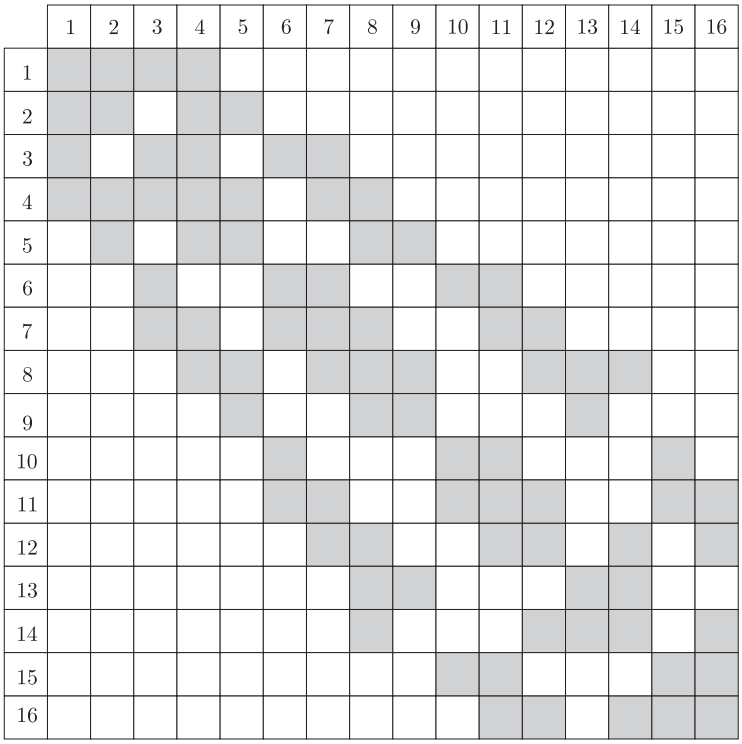


Figure 2.10: Bandwidth minimised stiffness matrix resulting from the an application of the Cuthill-Mckee algorithm. Notice how the matrix is 'bunched up' along the diagonal (compared to figure 2.5).

2.6 Summary

This chapter has attempted to sketch a program to solve Poisson's equation by discretising the spatial region over which said equation is defined. The simplest case of finite elements have been presented - that of first order Lagrangian elements. After the discretisation of Poisson's, a basic strategy for the solution of the linear system $A\boldsymbol{u} = \boldsymbol{b}$ was outlined. This highlights an important feature of iterative schemes in general - the dominance of matrix vector multiplications. Furthermore, this scheme becomes the basis of a future energy minimisation strategy. Sparse matrices and bandwidth minimisation are then presented as techniques to speed up the matrix-vector computation that dominates solution of linear systems.

Bibliography

Susanne C Brenner and Ridgway Scott. *The mathematical theory of finite element methods*, volume 15. Springer, 2008.

Elizabeth Cuthill and James McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, pages 157–172. ACM, 1969.

Alan J Davies. *The finite element method*. Clarendon Press, 1980.

D R Fredkin and T R Koehler. Hybrid Method for Computing Demagnetizing Fields. *IEEE Transactions on Magnetics*, 26(2):415–417, March 1990.

A Logg, KA Mardal, and GN Wells. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

Niels Saabye Ottosen, Hans Petersson, and Niels Saabye. *Introduction to the finite element method*. Prentice Hall Internationa,, 1992.

Joaquim Peiró and Spencer Sherwin. Finite difference, finite element and finite volume methods for partial differential equations. In *Handbook of materials modeling*, pages 2415–2446. Springer, 2005.

William H Press. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

Yousef Saad. *Iterative methods for sparse linear systems*. Siam, 2003.

Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain, 1994.

Gilbert Strang. *Computational science and engineering*. Wellesley-Cambridge Press Wellesley, 2007.

Gilbert Strang and George J Fix. *An analysis of the finite element method*, volume 212. Prentice-Hall Englewood Cliffs, NJ, 1973.

Chapter 3

A Micromagnetics Code Using FEniCS

This section presents MicroMag: a finite element code for micromagnetic simulations using FEniCS. In order to understand how MicroMag is put together, a sample code to solve the Poisson problem outlined in chapter 2 is presented. This is then followed by an overview of how the different energy contributions to the effective field, energy and energy gradient are calculated. After this each effective field component is examined in detail, and implementation issues are discussed. Being able to calculate the effective field (along with energy and energy gradient) is at the heart of micromagnetic modelling, however a method for finding minimum energy configuration methods is needed. MicroMag supports both direct solution of the Landau, Lifshitz, Gilbert (LLG) equation and energy minimisation. It also supports two energy minimisation methods: Conjugate Gradient and Hubert. The implementation details of these are also discussed below.

3.1 A Brief Overview of FEniCS

FEniCS (Logg et al., 2012) is a finite element framework designed to allow users to implement solutions to partial differential equations (PDEs) by specifying variational forms. It allows users to specify their problem in a high level mathematical language called the Unified Form Language (UFL) Alnæs et al. (2014). A form

compiler then reads UFL code and generates efficient C/C++ code. FEniCS also provides a full suite of Python interfaces that allow finite element codes to be written in very few lines. The user is able to embed UFL in their Python code, the form compiler is then called transparently for any given variational form. It should be noted that the form compiler is called only once for each form - this results in a one-off cost for generating and compiling variational forms.

Chapter 2 outlined the steps to discretise and solve a differential equation with boundary conditions specified over a domain. Most finite element codes have a similar structure. An example code for the problem 3.1

$$\nabla^2\phi = 6 \quad \text{with} \;\; g_{\mathrm{D}}(\boldsymbol{x}) = 0 \;\; \text{on} \;\; \boldsymbol{x} \in \partial\Omega \tag{3.1}$$

where Ω is the unit square is shown in listing 3.1, along with the solution in figure 3.1.

```
1  from dolfin import *
2
3  # Create mesh and define function space
4  mesh = UnitSquareMesh(50, 50)
5
6  # Function space of linear  Lagrangian  functions  as in (def. 3)
7  V = FunctionSpace(mesh, 'Lagrange', 1)
8
9  # Define boundary conditions
10 u0 = Constant('0')
11
12 def u0_boundary(x, on_boundary):
13     return on_boundary
14
15 bc = DirichletBC(V, u0, u0_boundary)
16
17 # Define variational problem
18 u = TrialFunction(V)                    # equivalent to (eqn. 2.13)
19 v = TestFunction(V)                     # equivalent to (eqn. 2.12)
20
21 a = inner(nabla_grad(u), nabla_grad(v))*dx  # l.h.s of (eqn. 2.4)
22 L = Constant(6.0)*v*dx                     # r.h.s of (eqn. 2.4)
23
```

```
24 # Assemble the matrix
25 A = assemble(a)          # Construct matrix as in section 2.4
26
27 # Assemble the f vector
28 f = assemble(L)          # Construct vector as in section 2.4
29
30 # PETSc solver  object (Conjugate Gradient) sans preconditioning.
31 solver = PETScKrylovSolver('cg', 'none')
32 solver.set_operator(A)
33
34 # Apply boundary conditions to matrix A, as in section 2.4.3
35 bc.apply(A)
36
37
38 # Apply boundary conditions to vector u, as in section 2.4.3
39 bc.apply(f)
40
41 # Compute solution Au = f
42 u = Function(V)
43 solver.solve(u.vector(), f)
44
45 # Write solution to file in VTK format
46 file = File('u.pvd')
47 file << u
```

Listing 3.1: A complete code to solve the Poisson equation problem with Dirichlet boundary conditions in FEniCS

The code in listing 3.1 begins with pulling in references from FEniCS' dolfin interface. This allows the Python code access to all of the FEniCS infrastructure. Line 4 makes use the dolfin function (UnitSquareMesh) to define a two dimensional mesh. This mesh is composed of triangular elements that fill the unit square with 50 elements in the x direction and 50 elements in the y direction.

Section 2.3 in chapter 2 outlined the basis functions that make up first order Lagrangian elements of two dimensions. These are defined on line 7, where the dimensionality of the domain is given by `mesh` and first order Lagrangian elements are requested.

Lines 10-15 define boundary conditions where `u0` is the Dirichlet condition

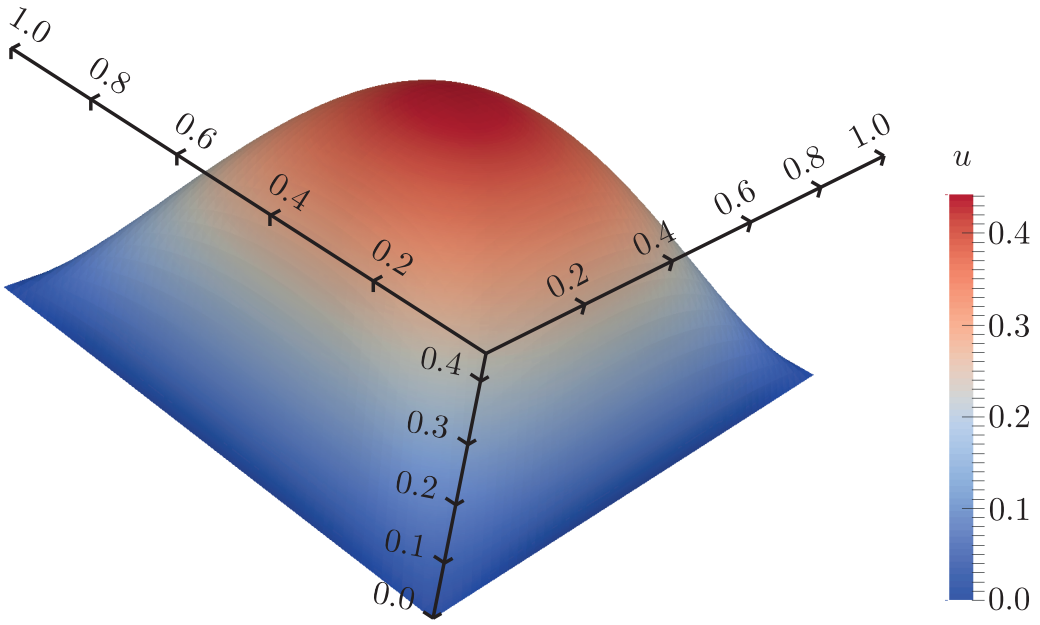


Figure 3.1: Example numerical solution of a Poisson problem solved using the FEniCS finite element framework. The code is shown in listing 3.1. This example is a solution to the Poisson problem $\nabla^2 u = 6$. The problem is posed over a square region with an edge length of 1, where the boundary condition requires that $u = 0$ on the boundary.

that the solution becomes zero on the boundary. Lines 12-13 define a function that is called whenever a point is on the boundary and some action is to be taken (in this case it just returns true for a point on the boundary) and line 15 defines a boundary value object over the function space `V`, with value `u0` and boundary given by the `u0_boundary` function.

Lines 18 and 19 define test and trial function objects as defined by equations (2.12) and (2.13) respectively. At this point in the execution they do not have numerical values, however they can be used symbolically in variational forms. Lines 21 and 22 are such variational forms and they represent the left hand side (line 21) and right hand side (line 22) of the variational form of (2.4).

Actual numerical calculations happen on lines 25 and 38. Here the `assemble` function takes a form consisting of the symbolic representations that have been described previously and calls the UFL compiler to generate C/C++ code. The generated code is then compiled and executed to construct a matrix or vector (depending on the number of free indices of the form). The expression in line 21 is converted in to a sparse matrix via (2.16) with the construction details outlined in section 2.4.2.

Since FEniCS' assembly process will produce sparse matrix vector systems, it is best to solve such systems using iterative methods to fully take advantage of sparse matrix representations. Line 31 explicitly requests a conjugate gradient solver provided by the PETSc linear algebra library(Balay et al., 2014a,b, 1997), and line 32 assigns the matrix to the conjugate gradient solver.

The last part of the problem set up is to apply boundary conditions. Line 35 applies boundary conditions to the matrix - this has the effect of 'knocking out' rows and columns corresponding to boundary nodes in the stiffness matrix as outlined in section 2.4.3. Line 39 applies the boundary condition values to the right hand side vector, basically replacing entries corresponding to boundary vertices with the Dirichlet values (this is again illustrated in 2.4.3).

All that remains after this is to allocate storage for the solution (line 43) and to execute the solver. Lines 46 and 47 will output the solution to a Paraview (Kitware, 2015) file.

3.1.1 Degrees of Freedom

The FEniCS finite element package provides a uniform framework for dealing with finite elements other than the standard linear elements discussed in chapter 2. When using Lagrangian elements the definition of hat functions means that the values defined over an element depend on the element vertices. This need not be the case however. In order to deal with this, FEniCS introduces the concept of degrees of freedom. These are the integral values associated with an element, but not necessarily associated with the vertices. For example, when using Nédélec elements (Anjam and Valdman, 2014), degrees of freedom are associated with the edges (facets in three dimensions).

In practise, degrees of freedom are just indices for the components of the matrix and vectors that result from a finite element discretisation. Since the goal of FEniCS is to abstract the solution of PDEs to writing integral forms, it is difficult to directly manipulate the vectors and matrices that FEniCS produces. In order to get around this, a `dof_manager` is provided in MicroMag. This class keeps a store of indices for parts of vectors associated with components local to a process. So if an operation needs to be performed at each degree of freedom (which are the same as mesh vertices for Lagrangian elements), the `dof_manager` can be used to index each of these values. An example of where this comes in useful is when the magnetisation vector $\boldsymbol{m}$ needs to be normalised - this is a point wise operation where vector $\boldsymbol{m}_i$, at each degree of freedom i needs to be accessed.

3.2 Effective Field Components - the Big Picture

Calculating the effective field, energy and energy gradient comprises the bulk of the work performed by micromagnetic simulations. Expressions for the effective field and energy gradient components are derived from expressions of the energy. The relationship between effective field, energy gradient and energy is outlined in figure 3.2.

Starting from the 'top down', the expression for the energy of a component is taken (box {1} in figure 3.2). This is in the form of a volume integral of

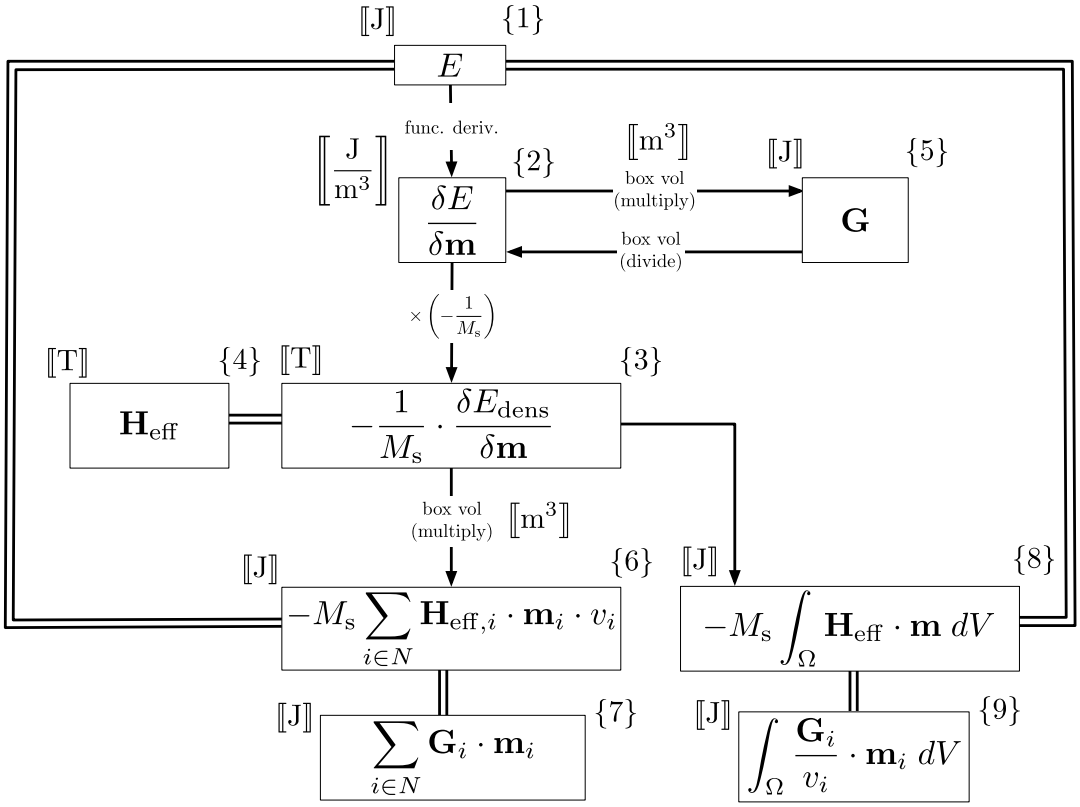


Figure 3.2: Schematic illustrating how energy, energy gradient and effective field calculations are related to each other. Each component box in the figure has associated with it a label (in braces) along with the unit for the quantity that is calculated (in double square brackets). As an example for reading the graph, conversion of an energy gradient (box {5}) to a an effective field (box {4}) requires division by box volume and then multiplication by the $-1/M_\mathrm{s}$. The double lines illustrate quantities that are equal to each other.

energy density; (3.2) shows the basic form where $\mathcal{L}$ is some functional of position $\boldsymbol{r}$, magnetisation $\boldsymbol{m}$ and magnetisation gradient $\nabla\boldsymbol{m}$. It is then necessary to understand how the energy changes with respect to how the magnetisation varies. This is achieved taking the functional derivative of (3.2) and multiplying by $-1/M_\mathrm{s}$ (Gilbert, 2004); shown by boxes {2}, {3} and {4} in the figure.

$$E\left(\boldsymbol{m}\right) = \int_\Omega \mathcal{L}(\boldsymbol{r}, \boldsymbol{m}\left(\boldsymbol{r}\right), \nabla\boldsymbol{m}\left(\boldsymbol{r}\right))\, dV \tag{3.2}$$

Taking only the functional derivative of (3.2) leaves an energy density gradient. When performing energy minimisation (described below) an energy gradient is required (box {5} in the figure) and so at each point of space the expression for the energy density gradient must be multiplied by a volume. In the case of a discretised mesh this is a volume contribution associated with a mesh vertex. To calculate the energy gradient on a discrete mesh, $\delta E/\delta\boldsymbol{m}$ is evaluated at each mesh vertex and then multiplied by a *box volume*. The box volume itself consists of one quarter the sum of element volumes incident to a vertex. This is because each element has four vertices and the volume of an element is averaged between those vertices; (3.3) defines the box volume v_i at mesh vertex i, where $N_\mathrm{e}(i)$ is the set of elements incident to node i (see definition 1).

$$v_i = \frac{1}{4} \sum_{e \in N_\mathrm{e}(i)} \mathrm{volume}(e) \tag{3.3}$$

The box volume is an important quantity as it allows effective fields to be converted, after appropriate scaling by $-M_\mathrm{s}$, to energy gradients (and vice versa). In the next section an integral expression for box volume is derived so that the variational form notation in FEniCS, corresponding to lines 22 and 23 in listing 3.1, can be used to calculate box volumes.

3.3 Box Volume

This section presents a way to calculate (3.3) in FEniCS. In order to do this it is necessary to prove the result in (3.5).

$$\int_{\Omega} \alpha_i \, dV = \frac{1}{4} \sum_{e \in N_e(i)} \text{vol}(e) \tag{3.4}$$

From definition 3 in chapter 2, the left hand side of the above equation is equal to.

$$\int_{\Omega} \alpha_i \, dV = \sum_{\{i, e_{j2}, e_{j3}, e_{j4}\} \in N_e(i)} \int_{e_j} \psi_{i, e_{j2}, e_{j3}, e_{j4}}(x, y, z) \, dV \tag{3.5}$$

The schematic figure 3.3 shows the situation with $n = |N_e(i)|$ elements: e_1 to e_n incident to vertex i. Consider performing the integral in (3.5) for any one of the $\psi_{i, e_{j2}, e_{j3}, e_{j4}}(x, y, z)$ under the summation. Without loss of generality and to make subsequent notation simpler consider a mapping

$$\begin{aligned} i &\to 1 \text{ with coordinates } (x_1, y_1, z_1) \\ e_{j2} &\to 2 \text{ with coordinates } (x_2, y_2, z_2) \\ e_{j3} &\to 3 \text{ with coordinates } (x_3, y_3, z_3) \\ e_{j4} &\to 4 \text{ with coordinates } (x_4, y_4, z_4) \end{aligned} \tag{3.6}$$

This then allows the jth integral of (3.5) to be written as

$$\int_{e_j} \psi_{i, e_{j2}, e_{j3}, e_{j4}}(x, y, z) \, dV = \int_0^{1-x'-y'} \int_0^{1-x'} \int_0^1 \psi_{1,2,3,4}(x', y', z') \, |J(x', y', z')| \, dz' \, dy' \, dx' \tag{3.7}$$

Where x', y' and z' are defined according to the affine transformation from the reference element (2.19) and $J(x', y', z')$ is the Jacobian matrix of the transform

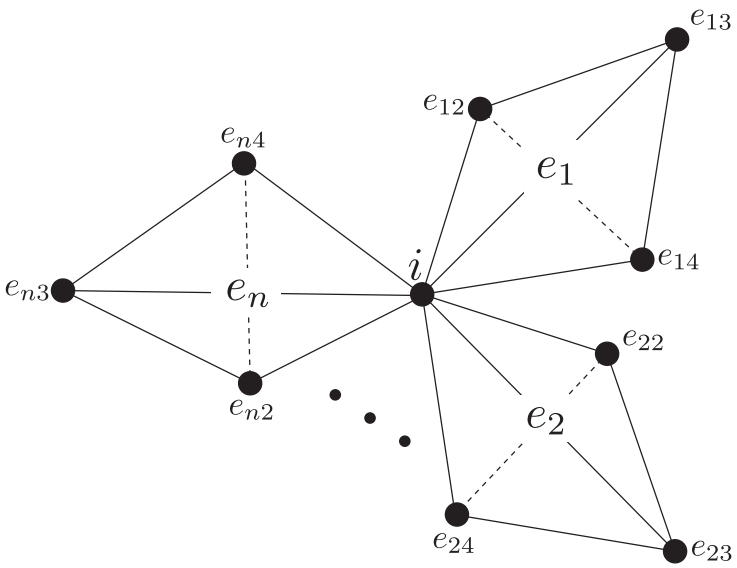


Figure 3.3: Schematic representation of the contributions of each facet function to the hat function about a mesh vertex i. Thus for some vertex i, there are n elements incident to that vertex. In the diagram $n = |N_e(i)|$; elements are indexed by a single variable, whereas element vertices are indexed by two index variables (one for the element and one for the local vertex with an element).

(2.21) (in three dimensions). Evaluating this integral results in

$$\int_{e_j} \psi_{i,e_{j2},e_{j3},e_{j4}}(x,y,z)\, dV = \frac{1}{4}\left(\frac{1}{3!}\begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix}\right) = \frac{1}{4}\mathrm{vol}(e_j) \tag{3.8}$$

Where the quantity inside the large brackets is the volume of a tetrahedron in determinant form (Zwillinger, 2002) as required for (3.5) to hold. Since there is a way to write the box volume as an integral form, the FEniCS code in listing 3.2 may be used to calculate box volumes. Note the similarity to (3.5) on line 13, the `assemble` function on line 16 performs the numerical integration and summation.

```
1  from dolfin import *
2
3  # Create mesh and define function space
4  mesh = UnitCubeMesh(1, 1, 1)
5
6  # Function space of linear  Lagrangian  functions  as in (def. 3)
7  V = FunctionSpace(mesh, 'Lagrange', 1)
8
9  # Define test fuction from function space.
10 psi = TestFunction(V)
11
12 # Define an expression for box volume.
13 box_vol_expr = psi*dx
14
15 # Assemble box volumes in to a matrix
16 box_volume = assemble(box_vol_expr)
17
18 print box_volume.get_local()
```

Listing 3.2: Example code to calculate box volumes for vertices in FEniCS.

3.4 Energy, Energy Gradient and Effective Field

In this section expressions are presented to calculate anisotropy, exchange, demagnetising and external components for the effective field. The idea is to follow boxes {1} to {4} in figure 3.2. This will give expressions for effective fields that are then implemented in code. Once effective fields have been calculated, energy and energy gradients can be calculated at relatively small computational cost. This is because calculation of energy gradients require a simple point-wise multiplication of box volumes and scalar constants, following {4}, {3}, {2} and {5} in figure 3.2; whereas calculations of energy require taking integrals, following {4}, {3} and {8} in figure 3.2.

3.4.1 Cubic Anisotropy

Deriving an expression for the cubic anisotropy effective field $\boldsymbol{H}_{\mathrm{a}}$ begins from the definition of the energy (1.12). It is usual to drop the component of the anisotropy corresponding to the K_2 value. Furthermore if direction cosines of magnetisation are projected on to the standard Cartesian basis, then an expression for the anisotropy energy may be written as

$$E_{\mathrm{a}} = K_1 \int_{\Omega} (m_1^2 m_2^2 + m_1^2 m_3^2 + m_2^2 m_3^2)\, dV \tag{3.9}$$

for a non-uniform magnetisation over a region Ω. Here K_1 is the first anisotropy constant and $\boldsymbol{m} = (m_1, m_2, m_3)$. Performing steps {2} and {3} in figure 3.2

results in

$$
\begin{aligned}
\boldsymbol{H}_{\mathrm{a}} &= -\frac{1}{M_{\mathrm{s}}}\frac{\delta E_{\mathrm{a}}}{\delta \boldsymbol{m}} \\
&= -\frac{K_1}{M_{\mathrm{s}}}\left(\frac{\partial\left(m_1^2m_2^2+m_1^2m_3^2+m_2^2m_3^2\right)}{\partial \boldsymbol{m}} - \nabla\cdot\frac{\partial\left(m_1^2m_2^2+m_1^2m_3^2+m_2^2m_3^2\right)}{\partial\nabla\boldsymbol{m}}\right)^{0} \\
&= -\frac{K_1}{M_{\mathrm{s}}}\left(\frac{\partial\left(m_1^2m_2^2+m_1^2m_3^2+m_2^2m_3^2\right)}{\partial \boldsymbol{m}}\right) \\
&= -\frac{2K_1}{M_{\mathrm{s}}}\begin{pmatrix} m_1(m_2^2+m_3^2) \\ m_2(m_1^2+m_3^2) \\ m_3(m_1^2+m_2^2)\end{pmatrix}
\end{aligned}
\tag{3.10}
$$

The expression given in (3.10) assumes that the direction cosines of the magnetisation are the standard Cartesian basis. However this need not be the case. By simply rotating the basis, it is possible to specify anisotropy with respect to arbitrary orientations. If this is the case then (3.10) becomes

$$
\boldsymbol{H}_{\mathrm{a}} = -\frac{2K_1}{M_{\mathrm{s}}}\begin{pmatrix} \boldsymbol{m}\cdot R\hat{e}_1\left[\left(\boldsymbol{m}\cdot R\hat{e}_2\right)^2+\left(\boldsymbol{m}\cdot R\hat{e}_3\right)^2\right] \\ \boldsymbol{m}\cdot R\hat{e}_2\left[\left(\boldsymbol{m}\cdot R\hat{e}_1\right)^2+\left(\boldsymbol{m}\cdot R\hat{e}_3\right)^2\right] \\ \boldsymbol{m}\cdot R\hat{e}_3\left[\left(\boldsymbol{m}\cdot R\hat{e}_1\right)^2+\left(\boldsymbol{m}\cdot R\hat{e}_2\right)^2\right]\end{pmatrix}
\tag{3.11}
$$

where $\hat{e}_1$, $\hat{e}_2$ and $\hat{e}_3$ are the standard Cartesian basis vectors and R is a rotation matrix.

It should be noted that (3.11) is for convenience only. It allows a user to specify different anisotropy axes within code without having to create multiple copies of the same mesh (each with a different orientation).

Implementation Issues

Implementation-wise, anisotropy is one of the simplest energy contributions to calculate. This is because there is an explicit expression to evaluate for each mesh vertex that depends only on the magnetisation at that vertex. In MicroMag (3.11) is evaluated for each degree of freedom (corresponding to a mesh vertex) in listing 3.3.

```
1  void
2  anis_t::perform (
3      std::shared_ptr< dolfin::Function > m,
4      std::shared_ptr< dolfin::Function > Ha
5  ) {
6
7    // Zero the output vector.
8    Ha->vector()->zero();
9
10   // Retrieve the degrees of freedom indices.
11   dof_list_t &dof_list = _dof_manager.dof_list();
12
13   // Retrieve vector components in to local storage.
14   m->vector()->get_local(&_m[0], dof_list.size(), &dof_list[0]);
15   Ha->vector()->get_local(&_Ha[0], dof_list.size(),
         &dof_list[0]);
16
17   // For each triple in local array _m.
18   for (size_t i = 0; i < _m.size(); i += 3) {
19
20     // Retrieve magnetisatoin.
21     double mx = _m[i+0];
22     double my = _m[i+1];
23     double mz = _m[i+2];
24
25     // Standard cartesian basis.
26     double i1=1.0, j1=0.0, k1=0.0;
27     double i2=0.0, j2=1.0, k2=0.0;
28     double i3=0.0, j3=0.0, k3=0.0;
29
30     // Anisotropy constant.
31     double K1=-1.24E4;
32
33     // Saturation magnetisation.
34     double Ms = 4.8E5
35
36     double m_dot_e1, m_dot_e2, m_dot_e3;
37
38     double A, B, C;
```

```
39
40      double Ha_x = 0.0;
41      double Ha_y = 0.0;
42      double Ha_z = 0.0;
43
44      // Calculate dot product m.e1
45      m_dot_e1 = mx*i1 + my*j1 + mz*k1;
46
47      // Calcualte dot product m.e2
48      m_dot_e2 = mx*i2 + my*j2 + mz*k2;
49
50      // Calculate dot product m.e3
51      m_dot_e3 = mx*i3 + my*j3 + mz*k3;
52
53      A = m_dot_e1 * m_dot_e2;
54      B = m_dot_e1 * m_dot_e3;
55      C = m_dot_e2 * m_dot_e3;
56
57      // Calculate anisotropy effective field.
58      Ha_x += -2.0*K1 / Ms * (
59          A*(i1*m_dot_e2 + i2*m_dot_e1) +
60          B*(i1*m_dot_e3 + i3*m_dot_e1) +
61          C*(i2*m_dot_e3 + i3*m_dot_e2)
62      );
63
64      Ha_y += -2.0*K1 / Ms * (
65          A*(j1*m_dot_e2 + j2*m_dot_e1) +
66          B*(j1*m_dot_e3 + j3*m_dot_e1) +
67          C*(j2*m_dot_e3 + j3*m_dot_e2)
68      );
69
70      Ha_z += -2.0*K1 / Ms * (
71          A*(k1*m_dot_e2 + k2*m_dot_e1) +
72          B*(k1*m_dot_e3 + k3*m_dot_e1) +
73          C*(k2*m_dot_e3 + k3*m_dot_e2)
74      );
75
76      _Ha[i+0] = Ha_x;
77      _Ha[i+1] = Ha_y;
```

```
78      _Ha[i+2] = Ha_z;
79
80    }
81
82    // Write back vector to storage.
83    Ha->vector()->set(&_Ha[0], dof_list.size(), &dof_list[0]);
84
85    // Synchronization point.
86    Ha->vector()->apply("insert");
87  }
```

Listing 3.3: The perform function in the `anis_t` object will compute pointwise the anisotropy effective field for an input magnetisation `m`. Output will be written to `Ha`. Both `m` and `Ha` are FEniCS data types and lines 7-15 illustrate how the `dof_manager` is used to populate local arrays with vector component values. The main `for` loop (lines 18-80) calculates the anisotropy field by (3.11). Finally, the local array for the anisotropy field is written back to FEniCS on line 83. Line 84 synchronises local sections of the array between processes.

The code in listing 3.3 outlines how the basic anisotropy calculation in MicroMag is performed. The important parts are contained in lines 58-74, where the anisotropy field is calculated. For efficiency the dot products in (3.11) are pre calculated, as are multiplications of those dot products (the `A`, `B` and `C` values).

In order to calculate the energy gradient the following formula is used

$$(\boldsymbol{G}_a)_i = -M_\mathrm{s}(\boldsymbol{H}_\mathrm{a})_i v_i \tag{3.12}$$

in figure 3.2 stepping back through boxes {4}, {3}, {2} and {5}. Note that this is again a pointwise operation and proceeds in a manner similar to listing 3.3. In order to calculate the energy, the expression in box {8} of figure 3.2 may be used. The FEniCS code for this is shown in listing 3.4.

```
1 from dolfin import *
2
3 # ... Mesh and effective field defined above ...
4
5 energy_expr = Constant(-1.0*Ms) * inner(m, Ha) * dx
```

```
energy = assemble(energy_expr)
```

Listing 3.4: Example to calculate anisotropy energy using FEniCS. The example assumes that the anisotropy energy stored in `Ha` has already been calculated for a given magnetisation `m`.

3.4.2 Exchange

Deriving an expression for the exchange effective field follows in a similar manner to that outlined in the previous section for anisotropy. The exchange energy is defined as

$$E_{\mathrm{e}} = \frac{A}{L^2}\int_{\Omega} (\nabla \boldsymbol{m})^2 \; dV \tag{3.13}$$

The quantity L is a scaling factor that depends on the length scale at which the exchange energy is calculated, for example if using meters then $L^2 = 1$ and so the scaling factor has no effect when using standard S.I units. If the length scale is in micrometres then the length scale *does* have an effect *i.e.* $L^2 = 1 \times 10^{-12}$. The effective anisotropy field is calculated by

$$\begin{aligned}
\boldsymbol{H}_{\mathrm{e}} &= -\frac{1}{M_{\mathrm{s}}}\frac{\delta E_e}{\delta \boldsymbol{m}} = -\frac{A}{M_{\mathrm{s}}L^2}\left(\cancelto{0}{\frac{\partial(\nabla \boldsymbol{m})^2}{\partial \boldsymbol{m}}} - \nabla \cdot \frac{\partial\,(\nabla \boldsymbol{m})^2}{\partial \nabla \boldsymbol{m}}\right) \\
&= \frac{2A}{M_{\mathrm{s}}L^2}\nabla \cdot \nabla \boldsymbol{m} \\
&= \frac{2A}{M_{\mathrm{s}}L^2}\nabla^2 \boldsymbol{m}
\end{aligned} \tag{3.14}$$

Note that units are consistent since A/L^2 has units of J/m^3 as expected and $\boldsymbol{H}_{\mathrm{e}}$ has units of tesla.

Implementation Issues

Evaluating (3.14) is not as straight forward as evaluating the anisotropy. It is still an explicit formula in the sense that the exchange effective field is some direct function of the magnetisation. However this function now involves derivatives in

the form of the Laplacian operator. This means that for a given mesh vertex, in order to calculate the exchange field, it is necessary to know the magnetisations of connected vertices. Fortunately when constructing the stiffness matrix using (2.16), it is precisely the discretised Laplacian operator that is being constructed. This means that it is possible to compute $\nabla^2 \boldsymbol{m}$ with just a single matrix vector multiplication (the matrix being the stiffness matrix of the laplacian). However, the stiffness matrix is evaluated via integrals over elements and so each entry in the stiffness matrix is a factor of *box volume* too large. Referencing figure 3.2, the starting point for the exchange is the energy graient $\boldsymbol{G}$ (box $\{5\}$). From this point one only to work backwards through the diagram to calculate the effective field and then the energy.

Parallelisation

Since the exchange relies on a single matrix vector multiplication (by the stiffness matrix) that is sparse and distributed, parallelisation of the exchange is straight forward and delegated to the linear algebra back end used by FEniCS.

3.4.3 Demagnetising Field

The demagnetising energy is defined as

$$E_\mathrm{d} = -\frac{M_\mathrm{s}}{2} \int_\Omega \boldsymbol{H}_\mathrm{d} \cdot \boldsymbol{m} \, dV. \tag{3.15}$$

By taking the functional derivative and multiplying by $-1/M_\mathrm{s}$, the expression for the demagnetising field $\boldsymbol{H}_\mathrm{d}$ can be shown to satisfy steps $\{1\}$ to $\{4\}$ in figure 3.2 (Gilbert, 2004). In order to actually calculate $\boldsymbol{H}_\mathrm{d}$, Maxwell's equations are used

$$\boldsymbol{B} = \mu_0 \left(\boldsymbol{H} + \boldsymbol{M} \right) \tag{3.16}$$

$$\nabla \cdot \boldsymbol{B} = 0 \tag{3.17}$$

$$\nabla \times \boldsymbol{H} = 0 \tag{3.18}$$

Where $\boldsymbol{H}$ is the demagnetising field due to magnetostatic charges, $\boldsymbol{B}$ is the induction field and $\boldsymbol{M}$ is the magnetisation field of the material. Equation (3.16)

shows that the total induction field is a combination of the demagnetising field and the magnetisation within a sample,(3.17) is Gauss' law and (3.18) is Ampere's law in a current free region (Grant and Phillips, 2013). By taking the divergence of (3.16) and making use of (3.17) it is possible to write

$$\mu_0 M_\mathrm{s} \nabla \cdot \boldsymbol{m} = -\mu_0 \nabla \cdot \boldsymbol{H} \tag{3.19}$$

Since the demagnetising field is conservative according to (3.18), it is possible to write $\mu_0 \boldsymbol{H} = -\nabla\phi$, *i.e.* $\boldsymbol{H}$ is the gradient of a scalar potential field ϕ. Putting all this together

$$\nabla^2 \phi = \mu_0 M_\mathrm{s} \nabla \cdot \boldsymbol{m} \tag{3.20}$$

For the material region Ω, the magnetisation is a vector field of unit length. However outside the material, $\boldsymbol{m}$ is zero. In this formulation, the scalar potential vanishes at infinity.

Implementation Issues

In order to solve (3.20), it is necessary to somehow deal with the boundary condition that the scalar potential disappears to zero at infinity. In order to achieve this, a spatial transform method is used. Using this method, a section of the mesh is marked as representing all of space up to infinity. Then a transform is applied that stretches the shape functions defined over elements in the *mapped region* (Imhoff et al., 1990,b; Brunotte et al., 1992). This has the effect of distorting the shape functions defined over finite elements. Care should be taken to make sure that the order of the mapping that distorts the shape functions is at least that of the decay of the scalar potential (*i.e.* $1/|\boldsymbol{r}|^2$) (Abert et al., 2013b,a).

The mapping used in MicroMag is based on the spherical mapping developed in (Imhoff et al., 1990,b). A mesh is constructed with three sub-meshes: a material region Ω_mat, surrounded by a space region Ω_umap, that is in turn surrounded by the mapped region Ω_map as seen in figure 3.4.

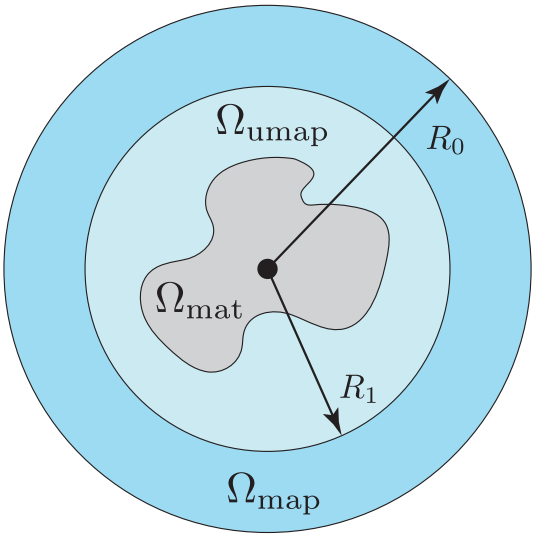


Figure 3.4: Schematic representation of the spherical transform. The material region Ω_{mat} is encapsulated within a space region Ω_{umap} of radius R_1; the mapping is not applied in this region. The Ω_{umap} region is then encapsulated in a sphere of radius R_0 representing infinity. The annulus Ω_{map} of width $R_0 - R_1$ is the region in which the spherical transform is applied. Note that if the material region is spherical it is not necessary to have the additional R_1 region; also if Ω_{mat} is not spherical then it is desirable to make the inner sphere fit as tightly as possible to the material boundary since this results in a smaller volume to mesh.

The mapping applied to Ω_{map} is given by (3.21)

$$\boldsymbol{x}' = \left(\frac{R_1}{|\boldsymbol{x}|} \sqrt{\frac{R_0 - R_1}{R_0 - |\boldsymbol{x}|}} \right) \boldsymbol{x}. \tag{3.21}$$

This takes points from 'mesh space' $\boldsymbol{x}$ to points in 'mapped space' $\boldsymbol{x}$. Note that as $|\boldsymbol{x}| \to R_1$ the mapped point $\boldsymbol{x}'$ becomes $\boldsymbol{x}$ and as $|\boldsymbol{x}| \to R_0$ the mapped point $\boldsymbol{x}'$ approaches ∞. In the mapped region the weak form (2.4) becomes

$$\int_{\Omega_{\text{map}}} J^{-1} \nabla v(\boldsymbol{x}) \cdot J^{-1} \nabla \phi(\boldsymbol{x}) \, |J| \; dV = 0 \tag{3.22}$$

where J is the inverse Jacobian matrix of the transform (3.21) and $|J|$ is the determinant of the Jacobian matrix. The right hand side is equal to zero, since this equation only applies in the mapped space region Ω_{map}. The material region Ω_{mat} and the unmapped region Ω_{umap} are represented by (3.23) and (3.24) respectively.

$$\int_{\Omega_{\text{mat}}} \nabla v(\boldsymbol{x}) \cdot \nabla \phi(\boldsymbol{x}) \, dV = \int_{\Omega_{\text{mat}}} \mu_0 M_{\text{s}} \nabla \cdot \boldsymbol{m} \, dV \tag{3.23}$$

$$\int_{\Omega_{\text{umap}}} \nabla v(\boldsymbol{x}) \cdot \nabla \phi(\boldsymbol{x}) \, dV = 0 \tag{3.24}$$

Listing 3.5 illustrates how to solve for the scalar potential in FEniCS.

```
1  from dolfin import *
2  from SphericalMapping import *
3
4  R1 = 0.05
5  R0 = 0.07
6
7  mapping = SphericalMapping(R1, R0) # Calculates eqn 3.21
8
9  method = 'cg'
10 precon = 'none'
11 degree = 1
12
```

```
13 fin = HDF5File(mpi_comm_world(), 'sphere.h5', 'r')
14
15 mesh = Mesh()
16 fin.read(mesh, 'mesh', False)
17
18 meshfn = MeshFunction('size_t', mesh)
19 fin.read(meshfn, 'mesh')
20
21 V  = FunctionSpace(mesh, 'CG', degree)
22
23 dx = Measure('dx')[meshfn]
24
25 sv = TestFunction(V)
26 su = TrialFunction(V)
27
28 def boundary(x, on_boundary):
29     return on_boundary
30
31 bc = DirichletBC(V, Constant(0), boundary)
32
33 m_space = Constant((0,0,0))
34 m       = Constant((1,0,0))
35
36 phi = Function(V)
37
38 invJ = mapping.invJ() # Inverse Jacobian matrix from 3.21
39 detJ = mapping.detJ() # Jacobian from 3.21
40
41 a = inner(     grad(su),     grad(sv))     *dx(1)+\ # Eqn. 3.23
42     inner(     grad(su),     grad(sv))     *dx(2)+\ # Eqn. 3.24
43     inner(invJ*grad(su),invJ*grad(sv))*detJ*dx(3)   # Eqn. 3.22
44
45 L   = inner(m      ,grad(sv))*dx(1)+\ # Eqn. 3.23
46       inner(m_space,grad(sv))*dx(2)+\ # Eqn. 3.24
47       inner(m_space,grad(sv))*dx(3)   # Eqn. 3.22
48
49 A = assemble(a)
50 solver = PETScKrylovSolver(method, precon)
51 solver.set_operator(A)
```

```
52 bc.apply(A)
53
54 b = assemble(L)
55 bc.apply(b)
56
57 solver.solve(phi.vector(), b)
58
59 File('phi.xdmf') << phi
```

Listing 3.5: An example of solving for the magnetic scalar potential using FEniCS. The example involves a mesh consisting of three sub-meshes: 1) material, 2) unmapped space 3) mapped space.

Line 7 is an external object containing code to return expressions for the inverse Jacobian matrix and Jacobian for mapping (3.21). Line 18 introduces the idea of a `MeshFunction` - these are functions defined over the elements of the mesh; in MicroMag they are used to mark sections of the mesh associated with material, unmapped space and mapped space sub-meshes. In listing 3.1 sub-mesh 1 is the material, sub-mesh 2 is the unmapped space and sub-mesh 3 represents mapped space. The important difference between the listing above and listing 3.1 appears on line 41-43; this is where the spherical mapping is applied to the part of the mesh associated with mapped space (line 43). Assembly of the force vector is also split in to three parts, note that submeshes 2 and 3 take the value `m_space` which is just defined as the zero vector (line 33).

Parallelisation

The previous effective field components were simple to parallelise. The anisotropy is a point-wise operation, each calculation relating to a vertex is independent. Calculation of the exchange is fundamentally just single matrix vector multiplication - this is a well understood problem and straight forward to parallelise. Parallelising the calculation of the scalar potential (and so the demagnetising field) requires solution of the system described previously. The strategy of calculating the scalar potential as a pure finite element problem via the spatial transform method means that again parallelisation is straight forward and can be implemented directly in FEniCS. Figure 3.5 shows schematically how the

problem is split between boundaries. It can be seen that communication occurs between adjacent sections of the mesh - a naive implementation would require each dipole-dipole interaction to be calculated resulting in an expensive all to all communication (where each independent process would need to communicate with every other process).

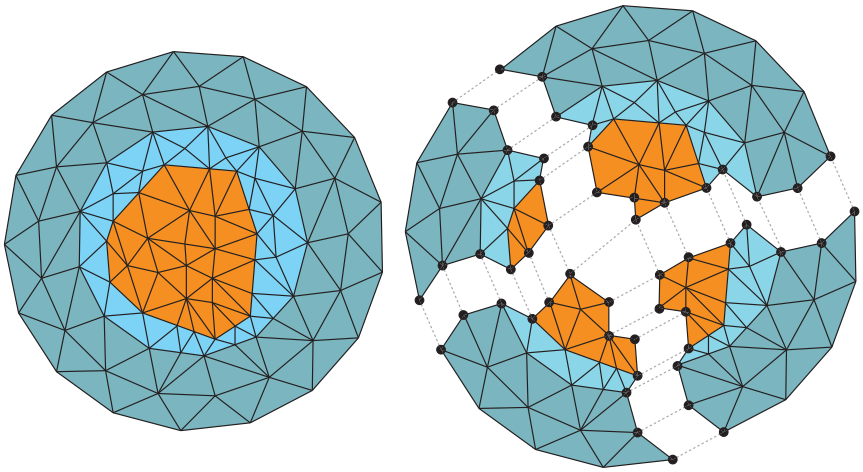


Figure 3.5: Schematic of the parallel decomposition of a mesh (with sub-meshes) between four processes. The start mesh (left) is split up in to approximately four pieces (right). Calculation of matrix vector multiplications when solving the system $A\boldsymbol{u} = \boldsymbol{b}$ are performed locally except for shared vertices called 'halo points' (the black vertices) which must communicate data between neighbour mesh pieces.

3.5 Energy Minimisation

In order to calculate minimum energy configurations, MicroMag provides two possible minimisation methods. One is a conjugate gradient method implemented in the Toolkit for Advanced Optimisation (TAO) (Munson et al.). The other method is an energy minimisation technique based on a steepest descent method called the *Hubert minimiser*. Creating minimisers is fairly straight forward and requires the user to provide a class that implements the following functions

- `init_m` set the initial magnetisation configuration,
- `minimise` perform the minimisation,
- `get_m` retrieve the solution after minimisation.

Effective field calculators are implemented in Python. This means that a mechanism to call these calculators is needed since they are not directly accessible from C/C++. The decision to have the user specify minimisers in C/C++ stems from the fact that when using third party libraries (such as TAO and later Sundials) a Python interface may not be available.

3.5.1 The Callback Mechanism

Each minimisation class requires a reference to a `MinimiserCallback` object. This is a reference to the Python object that will be responsible for calculating the energy and energy gradient. Listing 3.6 illustrates a minimal example of how such a class may be implemented.

```
1  #include "MinimiserCallback.h"
2  #include "MinimiserParameters.h"
3
4  class SampleMinimiser
5  {
6    public:
7
8      void init_m(VectorObject &input) {
9        // Code to initialise internal representations
10       // from VectorObject.
11     }
12
13     void minimise() {
14       // Code to perform minimisation.
15     }
16
17     void get_m(VectorObject &output) {
18       // Code to copy internal representations back
19       // to VectorObject.
20     }
```

```
21
22      void set_callback(MinimiserCallback &callback)
23      {
24        this->_callback = &callback;
25      }
26
27      void perform_callback()
28      {
29        if (!_callback) {
30          // Code to report error to user.
31        } else {
32          MinimiserParameters params;
33
34          // Code to set minimiser parameters.
35
36          _callback->call(params);
37
38          // Code to process minimiser parameters output
39          // after callback completes.
40      }
41
42    private:
43      MinimiserCallback *callback;
44  };
```

Listing 3.6: A skeleton implementation of a minimiser class.

Lines 8-11, 13-15 and 17-19 are the implementation of the `init_m`, `minimise` and `get_m` methods highlighted previously. The `init_m` and `get_m` can take as parameters any representation of a vector (the placeholder `VectorObject` is used in the example), however within MicroMag they are usually pointers to FEniCS' own data types.

The `set_callback` method on lines 22-25 is a way to assign callbacks to the minimiser (as shown in 3.7). The `perform_callback` method actually executes the callback. The suggested implementation first checks whether a callback has been assigned and reports the error to the calling class if this is not the case. A `MinimiserParameters` class is used to pass data from the minimiser to the callback and vice-versa to retrieve the callback's result. Line 43 is a pointer to

the callback object itself.

The code in 3.7 shows a minimal implementation of a callback.

```
1  class SampleCallback(MinimiserCallback):
2
3      def __init__(self):
4
5          super(SampleCallback, self).__init__()
6
7          # Code to initialise the class
8
9      def call(self, params):
10
11         # Code to perform the callback. Params is the
12         # same object that was passed from c/c++
```

Listing 3.7: A skeleton implementation of a callback.

The callback must inherit from `MinimiserCallback` (line 1). This is a Python class that is generated by SWIG. It is also important to explicitly call the constructor of `MinimiserCallback` from the constructor of `SampleCallback` (line 5). In order for the callback to execute, it is also necessary to provide the `call` function (line 9). This function takes a parameter called `params` which is the object passed on line 36 of listing 3.6. It is this object that minimiser and callback use to communicate data between each other.

Finally SWIG is used to generate a Python wrapper for `SampleMinimiser`, so that it may also be called from MicroMag scripts. Putting all this together listing 3.8 illustrates what a complete minimiser code in Python may look like.

```
1  # Code to define vectors, matrices data etc.
2  m = VectorObject()
3
4  # Create the minimiser callback.
5  callback = SampleCallback()
6
7  # Create the minimiser.
8  minimiser = SampleMinimiser()
9  minimiser.set_callback(callback)
10
```

```
11 # Perform a minimisation.
12 minimiser.init_m(m)
13 minimiser.minimise()
14 minimiser.get_m(m)
```

Listing 3.8: A sample script of a minimisation problem.

Line 2 creates a vector (in the sample the placeholder `VectorObject` is used) and lines 5 and 8 create the callback and minimiser respectively. Line 9 associates the callback with the minimiser. In order to perform the minimisation, a minimiser is given an initial configuration (line 12) and then told to minimise (line 13). Once complete, the solution is copied back. MicroMag only assumes that the minimiser exposes initialise, execute and retrieve routines - this provides the possibility to chain together different minimisers and solvers since the implementation of time steppers (such as those used to solve the Landau, Lifshitz, Gilbert equation) is the same.

3.5.2 TAO Minimiser

The Toolkit for Advanced Optimisation (TAO) is used by the `TaoMinimiser` object to find minimum energy magnetisation configurations via the conjugate gradient method. It conforms to to the design described previously. Internally it minimises the magnetisation in spherical polar coordinates rather than Cartesian since this automatically enforces the requirement that $\boldsymbol{m}$ be of unit length. The only additional points to note about the `TaoMinimiser` are then

- the `init_m` method converts the input vector field with Cartesian components to a vector field with spherical polar components,
- the `get_m` method retrieves a vector field with Cartesian components corresponding to the internal representation (with spherical polar components).

It is also important to note that prior to executing the callback (corresponding to line 36 of listing 3.6), the `TaoMinimiser` must convert spherical polar field to a Cartesian field. This is all performed transparently as far as clients of the `TaoMinimiser` class are concerned.

3.5.3 Hubert Minimiser

Like the TAO minimiser, the Hubert minimiser (Fabian, 2014) maintains an internal representation of the magnetisation with spherical polar components. However the actual minimisation is similar to the descent method presented in algorithm 1. *I.e.* the magnetisation $\boldsymbol{m}_{t+1}$ is chosen to be a linear combination of the old magnetisation and the gradient. In the Hubert minimiser the α value, corresponding to lines 7-9 of algorithm 1, is varied dynamically instead of being the minimum of a line search. The algorithm for the Hubert minimiser is shown in algorithm 3.

The Hubert minimiser proceeds as a succession of *creep phases*. Within each creep phase a 'good value' of α is maintained, as long as the energy of successive solutions is moving downhill. The creep phase is shown between lines 10 and 37 of algorithm 3. While the α value is 'good' (*i.e.* resulting in a downward direction of the energy) the `creepCount` is incremented and energy, energy gradient and magnetisation are updated to new values (lines 30-36). In this state the Hubert minimiser also tests to see whether the gradient is too flat, if it is then the routine exits with a valid solution. If the creep phase managed to creep forward enough times, with an existing α value, then the loop denoted by ‡ terminates and α increases by $\Delta\alpha$. At this point the algorithm decides that it has been progressing well enough and will try to accelerate the α value.

It is possible a specific α value may be too large and results in a $\boldsymbol{m}$-configuration that results in an energy increase (line 19). In this scenario, creeping is terminated and α is reduced (lines 20, 21 respectively). It is important to make sure that α does not become too small in order to avoid numerical errors so lines 22-29 begin the α-correction phase. In this phase the magnetisation is perturbed and the whole minimiser begins again (from line 2). Failure to result in a better energy minimum after a fixed number of restarts results in a failure. However if α-correction is successful then the algorithm proceeds as usual.

The minimiser maintains a trail of energy values that it has calculated in `eTrail`. This allows the algorithm maintain a history of energy values when deciding to exit rather than the simpler difference between two states. This is

because, even though the algorithm only accepts $\boldsymbol{m}$-configurations in the 'downward' direction, every energy state is stored in the trail.

Figure 3.6 outlines the three phases of the Hubert minimiser algorithm described above.

Algorithm 3 Hubert Minimiser. The symbols $\mathbf{m}_t$, $\mathbf{G}_t$ and E_t refer to the magnetisation, energy gradient and energy at iteration step t respectively. The α value is the same as that found in lines 7-9 of algorithm 1, α_s a scaling factor set by the user, α_min is the smallest acceptable value for α and $\Delta\alpha$ is the amount by which α is increased/decreased as the algorithm progresses. The $\| \bullet \|$ symbol is the ℓ^2 norm and the $| \bullet |$ symbol is absolute value. The $\oplus\oplus$ symbol looks like the prefix version of the '++' operator in C++; its meaning should be interpreted as: "return the value of `nstart` and increment its value by 1, if the new value is equal to the last index of `eTrail`, then set `nstart` to zero".

```
1  function HubertMinimiser
2    while exit is FALSE † do
3      if totalRestart is TRUE then
4        Calculate G_t and E_t from m_t
5        eTrail[⊕⊕nstart] ← E_t
6        α ← 1.0
7        totalRestart ← FALSE
8      end if
9      creepCount ← 0
10     while creepCount < maxCreep ‡ do
11       m_{t+1} ← m_t - α α_s G_t
12       Calculate G_{t+1} and E_{t+1} from m_{t+1}
13       eTrail[⊕⊕nstart] ← E_{t+1}
14       G^2 ← ||G_{t+1}||^2
15       ΔE = |eTrail[nstart] - E_{t+1}| / eTrail.size
16       if ΔE ≈ 0 then
17         exit ← TRUE, break loop ‡
18       end if
```

Algorithm 3 Hubert Minimiser (continued)

```
        if E_t < E_{t+1} then
          creepCount ← 0
          α ← α / Δα²
          if α < α_min then
            resetCount ← resetCount + 1
            perturb m_t
            if resetCount > resetMax then
              exit ← TRUE, break loop ‡
            end if
            totalRestart ← TRUE, break loop ‡
          end if
        else
          creepCount ← creepCount + 1
          E_t ← E_{t+1}, m_t ← m_{t+1}, G_t ← G_{t+1}
          if G² ≈ 0 then
            exit ← TRUE, break loop ‡
          end if
        end if
      end while‡
      α ← α Δα
      resetCount ← 0
    end while†
end function
```

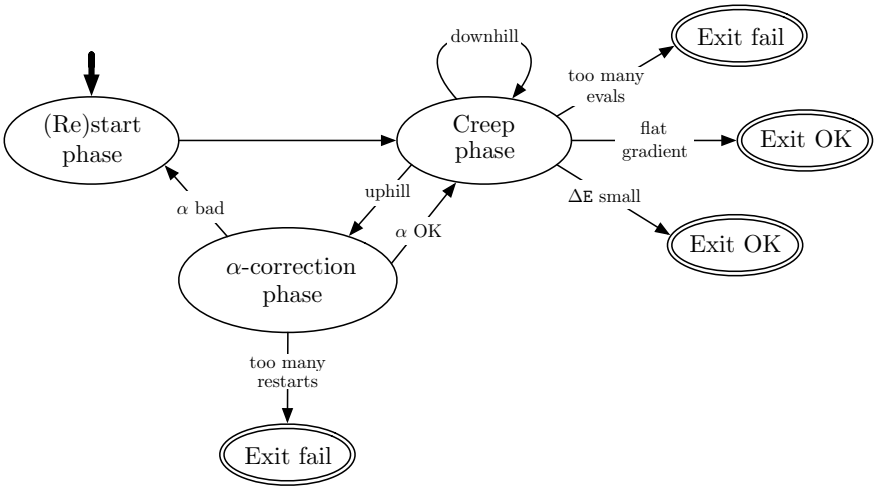


Figure 3.6: The phases of the Hubert minimisation algorithm. Arrows show transitions between phases and double lines show exit points in the algorithm. The algorithm starts in the *restart phase* (indicated by the thick arrow) corresponding to lines 3-8 in algorithm 3. The algorithm then proceeds to the *creep phase* - lines 10-19 & lines 30-39. If values for α are too small, there is an *α-correction phase* corresponding to lines 22-29 which then feeds back in to the *restart phase*.

3.6 Time Steppers

The Landau, Lifshitz, Gilbert equation is discussed in section 1. Since it is non-linear it requires a suitable time stepping method. MicroMag uses the Sundials time stepping library (Hindmarsh et al., 2005) to perform time integration of the LLG. Calling the library is similar to the way in which the TAO library and the the Hubert minimiser are called. A backend implementation is written in C/C++ and the callback mechanism described in section 3.5.1 allows the time stepping code to call routines to calculate effective field components.

3.7 MicroMag Design - The Big Picture

In this section, the major components of MicroMag have been described in some detail. However it is useful to take an overview of the architecture. MicroMag

is designed to be modular and easily extendable, with some components written in Python and some in C/C++. Those parts of MicroMag written in C/C++ are generally for point-wise operations - these components are much faster when implemented in a compiled language. The C/C++ implementations are callable from Python via use of the Simplified Wrapper and Interface Generator (SWIG) (Beazley et al., 1996). This approach gives the speed of C/C++ with the simplicity of Python.

Figure 3.7 illustrates the design of MicroMag. It shows core objects and how they relate to each other. At the heart of most scripts is the `Model` object. This is a controller object that allows the user to assemble micromagnetic models. Its basic functions are:

- allow the user to specify sub-mesh information and properties,
- register effective field solvers,
- register micromagnetic solvers (energy minimisation or LLG).

Once the Model has been specified, it may be thought of as a black box - give it a magnetisation as input $\boldsymbol{m}$ and it will calculate the minimum energy configuration as output.

The `CubicAnisotropyPointwise`, `DemagSphericalMapping` and `ExchangeMatrixVector` classes are written in Python. Each class has an interface recognised by the `Model` class and in the course of calculating a solution some methods of each will be called several times. These methods are:

- `perform` is a method that takes as input a magnetisation and calculates the effective field, energy and energy gradient - which are assumed to be internal to the class,
- `H` is a method that returns the calculated effective field,
- `get_H` is a method that takes as input a `dolfin.Function` object, this object will be populated with the effective field calculated by the class,
- `EGrad` is a method that returns the calculated energy gradient field,

- `get_EGrad` is a is a method that takes as input a `dolfin.Function` object, this object will be populated with the gradient field calculated by the class,
- `energy` is a method that returns the energy calculated by the class.

And they must be implemented by classes that wish to be part of the effective field calculation.

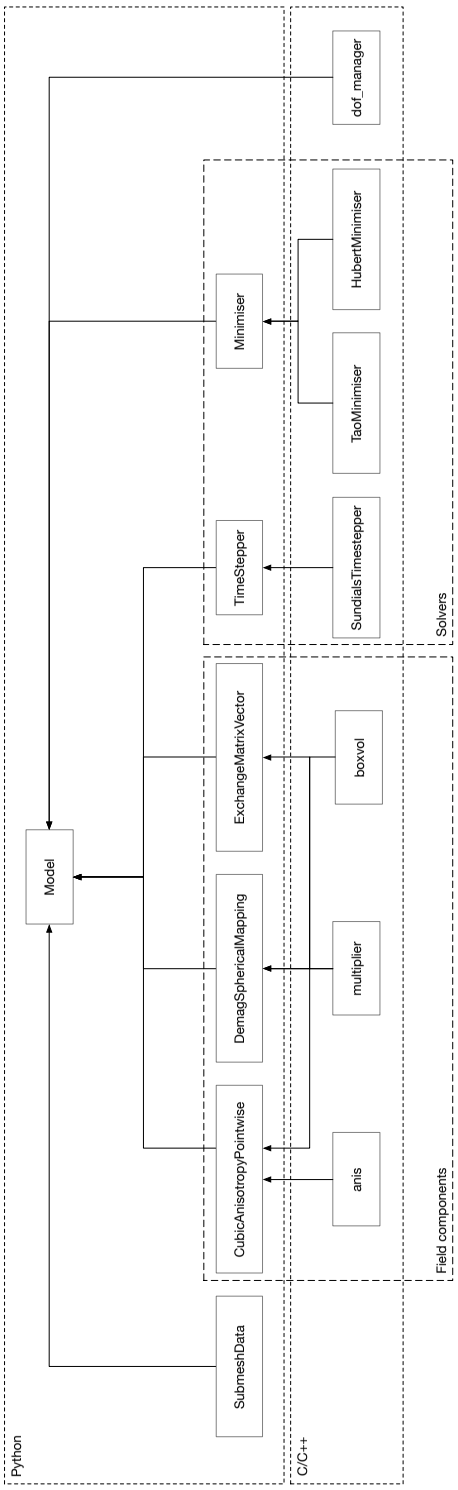


Figure 3.7: The overall design schematic of MicroMag. Arrows represent the idea that objects are components of another object, for example a `DemagSphericalMapping` object is a component of a `Model` object. The fine dotted lines show which language a given object is implemented in and the dashed lines group together field component calculators and different types of solvers.

3.8 Summary

This chapter has taken a look at the implementation details of MicroMag. It has outlined a strategy convert the definitions of the effective field component energies outlined in chapter 1 in to expressions of effective field, energy and energy gradient and how those quantities may be computed. Since the focus of micromagnetics is to find configurations of the magnetisation that minimise the effective field energy, a mechanism to fit energy calculations in to MicroMag were discussed. This is results in the callback mechanism that allows MicroMag to make use of eternal libraries such as TAO and Sundials. Furthermore a minimisation method called the *Hubert minimiser* based on a gradient descent idea was presented in detail. Finally an overview of MicroMag was presented to give an idea of the major software components fit together.

Bibliography

Claas Abert, Lukas Exl, Florian Bruckner, Andre Drews, and Dieter Suess. magnum.fe: A micromagnetic finite-element simulation code based on FEniCS. *Journal of Magnetism and Magnetic Materials*, 345:29–35, November 2013a.

Claas Abert, Lukas Exl, Gunnar Selke, Andre Drews, and Thomas Schrefl. Numerical methods for the stray-field calculation: A comparison of recently developed algorithms. *Journal of Magnetism and Magnetic Materials*, 326:176–185, January 2013b.

Martin S Alnæs, Anders Logg, Kristian B Ølgaard, Marie E Rognes, and Garth N Wells. Unified form language: A domain-specific language for weak formulations of partial differential equations. *ACM Transactions on Mathematical Software (TOMS)*, 40(2):9, 2014.

Immanuel Anjam and Jan Valdman. Fast MATLAB assembly of FEM matrices in 2D and 3D: Edge elements. *arXiv.org*, September 2014.

Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, and Hong Zhang. PETSc Web page. `http://www.mcs.anl.gov/petsc`, 2014a. URL `http://www.mcs.anl.gov/petsc`.

Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.5, Argonne National Laboratory, 2014b.

David M Beazley et al. Swig: An easy to use tool for integrating scripting languages with c and c++. In *Proceedings of the 4th USENIX Tcl/Tk workshop*, pages 129–139, 1996.

X Brunotte, G Meunier, and J F Imhoff. Finite-Element Modeling of Unbounded Problems Using Transformations - a Rigorous, Powerful and Easy Solution. *Ieee Transactions on Magnetics*, 28(2):1663–1666, March 1992.

Karl Fabian. Hubert minimiser, 2014. Personal communication.

T L Gilbert. A phenomenological theory of damping in ferromagnetic materials. *Ieee Transactions on Magnetics*, 40(6):3443–3449, November 2004.

Ian S Grant and William Robert Phillips. *Electromagnetism*. John Wiley & Sons, 2013.

Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.

J F Imhoff, G Meunier, X Brunotte, and J C Sabonnadiere. An original solution for unbounded electromagnetic 2D- and 3D-problems throughout the finite element method. *Magnetics, IEEE Transactions on*, 26(5):1659–1661, 1990a.

J F Imhoff, G Meunier, and J C Sabonnadiere. Finite-Element Modeling of Open Boundary-Problems. *Ieee Transactions on Magnetics*, 26(2):588–591, March 1990b.

Kitware. Paraview, 2015. URL `http://www.paraview.org`. Accessed: May 2015.

A Logg, KA Mardal, and GN Wells. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

Todd Munson, Jason Sarich, Stefan Wild, Steven Benson, and Lois Curfman McInnes. Tao 2.0 users manual. Technical Report ANL/MCS-TM-322, Mathematics and Computer Science Division, Argonne National Laboratory. URL `http://www.mcs.anl.gov/tao`. Accessed: May 2015.

Daniel Zwillinger. *CRC standard mathematical tables and formulae*. CRC press, 2002.

Chapter 4

Testing and Performance Scaling

This section outlines the testing and performance characteristics of MicroMag. In the first section the testing procedure is examined, where a set of known inputs and expected outputs are compared against values computed by MicroMag. Such testing takes place primarily at the object level and so for each object in figure 3.7 there is a corresponding unit test in MicroMag. The second section examines the scaling characteristics of MicroMag, where the execution times for the critical sections of code (namely the field component calculations) are compared against the number of processors used - allowing users to approximately gauge the number of processes required for some problem size.

4.1 Testing

The following section discusses the testing performed on the MicroMag code. Tests for the individual field components are broken down in to two parts: verification against existing results and verification of the energy gradient. Verification against existing results are either analytical or come from some other data source (discussed below); these are implemented in Mathematica, which is then used to generate the expected data in units tests. Verification of the energy gradient proceeds as follows:

1. for some given magnetisation, $\boldsymbol{m}$, calculate the energy $E(\boldsymbol{m})$, corresponding to the effective field component being tested,

2. pick some (random) mesh vertex and apply a small perturbation Δm_x to the x component of the magnetisation (call this new field $\boldsymbol{m}_\Delta = \boldsymbol{m} + \langle \Delta m_x, 0, 0 \rangle$),

3. calculate the energy for the perturbed magnetisation $E(\boldsymbol{m}_\Delta)$,

4. calculate the finite difference gradient given by (4.1)

$$G_{\mathrm{fd}}(\boldsymbol{m}) \approx \frac{E(\boldsymbol{m}_\Delta) - E(\boldsymbol{m})}{\Delta m_x}, \tag{4.1}$$

   where G_{fd} is the finite difference approximation for the gradient.

It is expected that as Δm_x is reduced, the error between the finite element approximation and the calculated gradient for the given component should approach zero.

4.1.1 Demagnetising Field

This section presents testing results for the calculation of the magnetic scalar potential using MicroMag against two analytical calculations: a sphere and a cuboid slab. The scalar potential in a uniformly magnetised sphere (along the x axis) is given by Jackson and Jackson (1962)

$$\phi_{\mathrm{a}}(r,\theta) = \begin{cases} \frac{M_{\mathrm{s}}\mu_0 r}{3}\cos(\theta) & -a < r < a \\ \frac{M_{\mathrm{s}}\mu_0 a^3}{3r^2}\cos(\theta) & r \geq a \\ -\frac{M_{\mathrm{s}}\mu_0 a^3}{3r^2}\cos(\theta) & r \leq -a \end{cases}, \tag{4.2}$$

where a is the radius of the sphere, M_{s} is the saturation magnetisation and (r, θ) are the polar coordinates of a test charge in the xy-plane. For the uniformly

magnetised slab, the scalar potential is given by Ravaud and Lemarquand (2009)

$$\phi_{\mathrm{a}}(x,y,z) = \frac{M_{\mathrm{s}}\mu_0}{4\pi}\sum_{i,j,k=1}^{2}(-1)^{i+j+k}\Bigg(L_{\mathrm{y},j} + (z-L_{\mathrm{z},k})\tan^{-1}\left(\frac{y-L_{\mathrm{y},j}}{z-L_{\mathrm{z},k}}\right)$$
$$+ (x-L_{\mathrm{x},i})\log\left(y-L_{\mathrm{y},j}+\sqrt{(x-L_{\mathrm{x},i})^2+(y-L_{\mathrm{y},j})^2+(z-L_{\mathrm{z},k})^2}\right)$$
$$- (z-L_{\mathrm{z},k})\tan^{-1}\left(\frac{(x-L_{\mathrm{x},i})\,(y-L_{\mathrm{y},j})}{(z-L_{\mathrm{z},k})\sqrt{(x-L_{\mathrm{x},i})^2+(y-L_{\mathrm{y},j})^2+(z-L_{\mathrm{z},k})^2}}\right)$$
$$+ (y-L_{\mathrm{y},j})\log\left(x-L_{\mathrm{x},i}+\sqrt{(x-L_{\mathrm{x},i})^2+(y-L_{\mathrm{y},j})^2+(z-L_{\mathrm{z},k})^2}\right)\Bigg), \tag{4.3}$$

where M_{s} is the saturation magnetisation, μ_0 is the permeability of free space and (x,y,z) are the Cartesian coordinates of a test charge. The dimensions of the slab are given by the 'vectors' $(L_{\mathrm{x},1}, L_{\mathrm{x},2})$, $(L_{\mathrm{y},1}, L_{\mathrm{y},2})$ and $(L_{\mathrm{z},1}, L_{\mathrm{z},2})$.

In order to verify the code, the above analytic equations, ϕ_{a}, are calculated for the two geometries: a sphere of radius 1 unit and and a cuboid slab with $x_{\mathrm{length}} = 5$, $y_{\mathrm{width}} = 10$ and $z_{\mathrm{width}} = 20$ units. An approximate solution is calculated using MicroMag for the geometries called ϕ_{mm}. The output of the analytic and approximate scalar potential is then sampled at the same point for each geometry as seen in figure 4.1 producing two sample vectors. Finally the ℓ^2 norm is calculated between calculated between analytic and and approximate results using (4.4)

$$\mathrm{norm} = \left(\sum_{i=0}^{N}\left(\phi_{\mathrm{a}}(x_0+i\Delta x) - \phi_{\mathrm{mm}}(x_0+i\Delta x)\right)^2\right)^{1/2}, \tag{4.4}$$

where i is the integer index corresponding to a sample point in figure 4.1 and N is the total number of sample points. The geometries are outlined in table below

Geometry id	**No. of Sphere elements**	**No. of slab elements**
1	1,375	1,307
2	1,700	1,872
3	2,396	3,088
4	3,394	4,497
5	5,242	6,522
6	10,470	9,988
7	16,108	18,837
8	39,454	36,854
9	140,009	103,134
10	1,152,402	450,261
11	9,950,973	9,616,645

Table 4.1: Number of elements within each test mesh used for the demagnetising field. The Geometry id is corresponds to the sphere id in figure 4.2a, the slab id in figure 4.2b and the id values in figure 4.2c below.

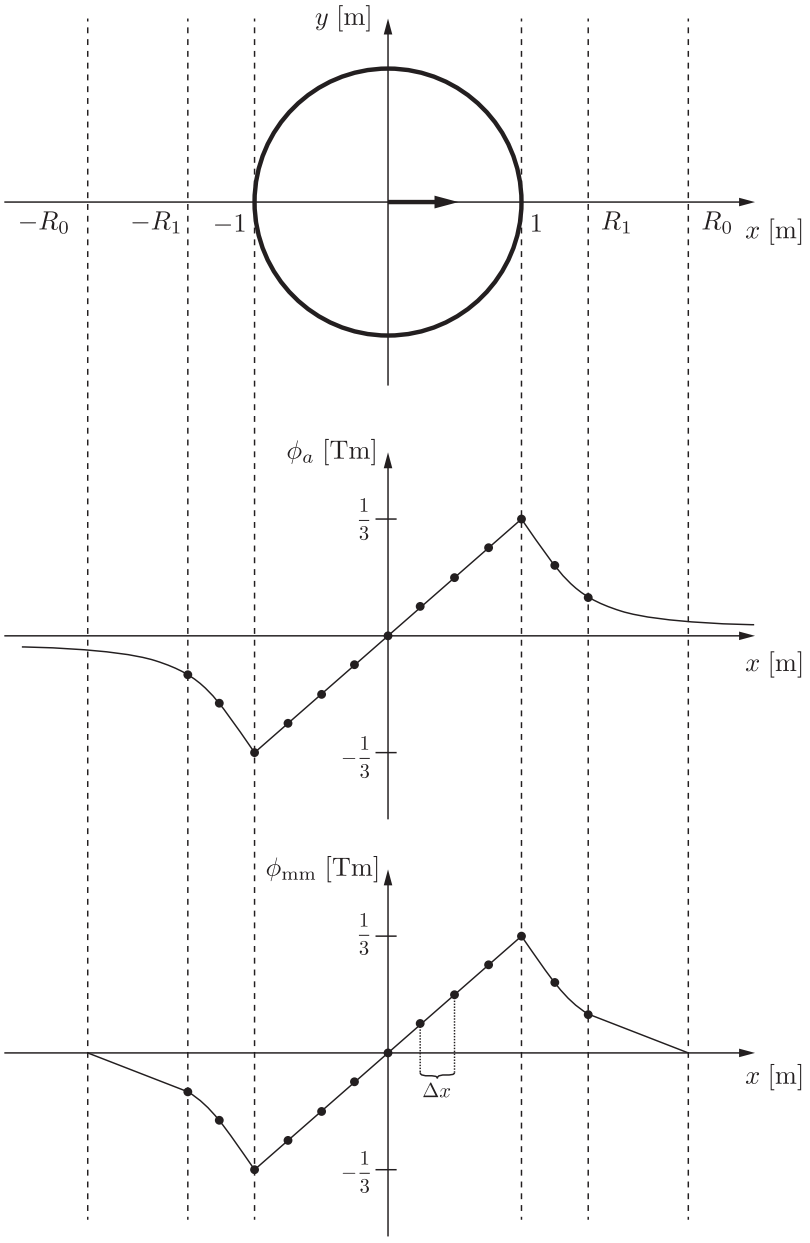
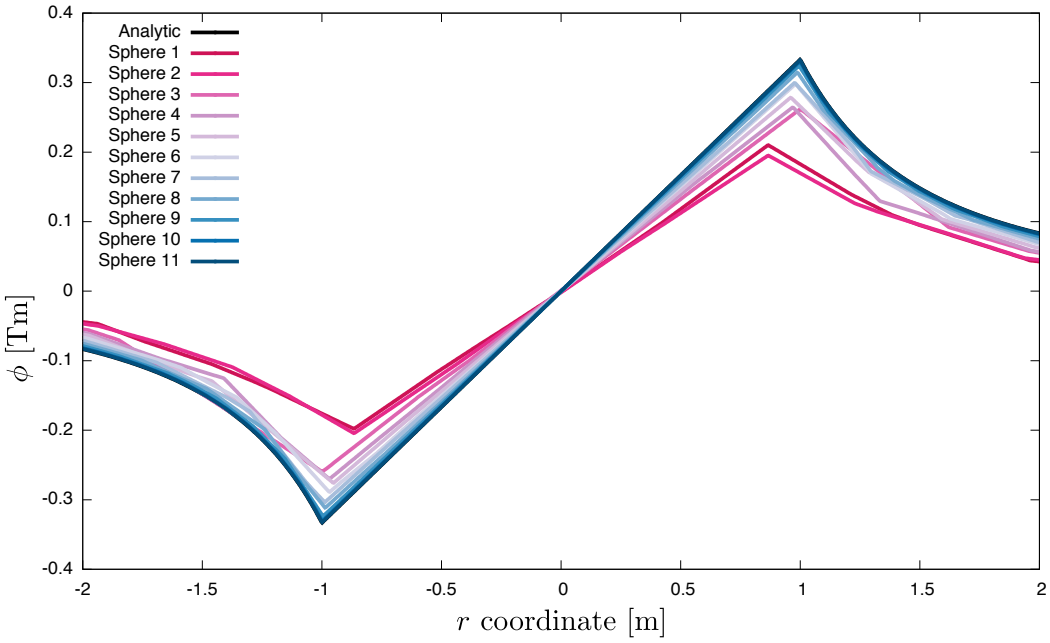


Figure 4.1: Testing the scalar potential for the demagnetising field in a sphere. The top axes show a schematic of a spherical geometry having a unit radius with a uniform magnetisation along the $\langle 1, 0, 0 \rangle$ axis. The centre and bottom axes show the analytical scalar potential and the potential calculated by MicroMag sampled along the x-axis (depicted by dots), for a unit saturation magnetisation value. Δx is the sample distance in (4.4), R_1 and R_0 values correspond to the unmapped space and mapped space radii respecively, this is described in 3.4.3. It should be noted that between R_1 and R_0 the scalar potential asymptotically approaches to zero for the analytic solution, but for the calculated solution it linearly vanishes to exactly zero (at R_0) - this is a result of the mapping described in 3.4.3.

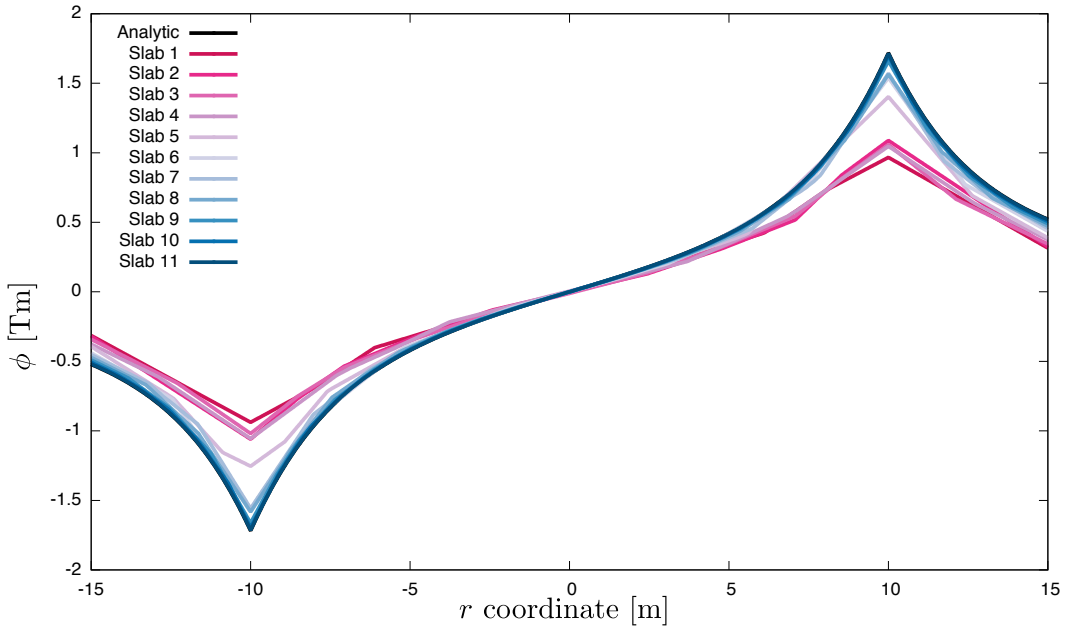
Results

The graphs shown in figure 4.2 demonstrate how the calculation of the scalar potential for a spherical and cuboid slab geometry change with respect to decreasing element size. It can be seen that, as expected, a decrease in element size (resulting in an increase in the number of elements) results in smaller and smaller values for the ℓ^2 norm measurement. Thus as the element density increases, so does the accuracy of the scalar potential calculation.

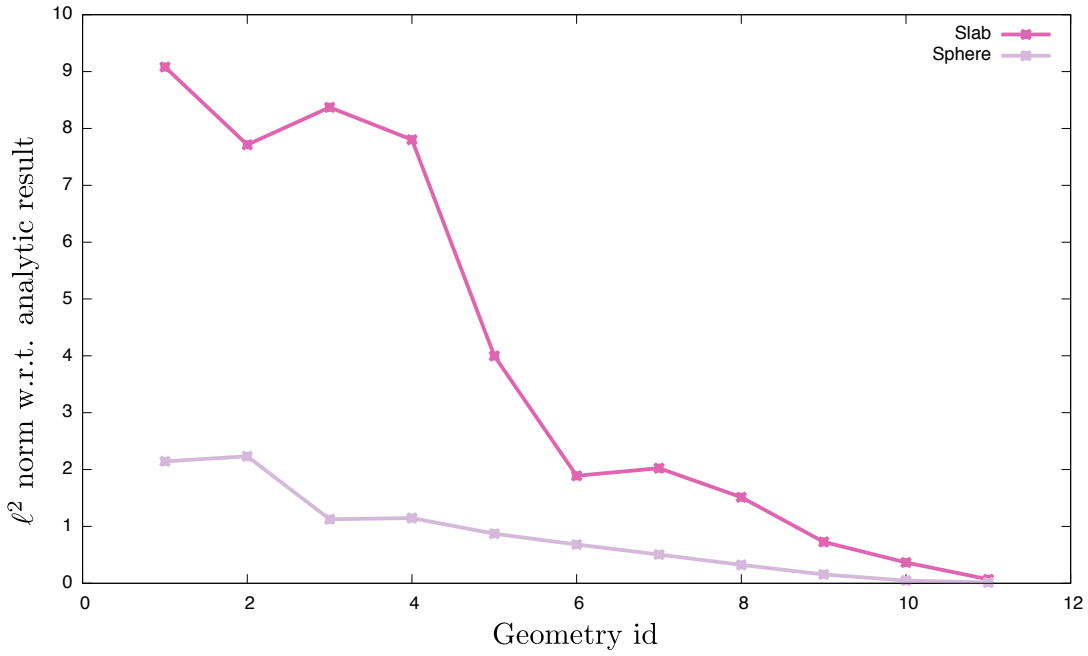


(a)

Figure 4.2



(b)



(c)

Figure 4.2: The demagnetising potentials for a sphere and a slab are shown for two test geometries. Figure (a) and (b) show the scalar potential sampled along the x and z axes respectively for uniform magnetisations along the given directions. It can be seen that as the number of elements is increased, the calculated value for the scalar potential approaches the analytic results presented in equations (4.2) and (4.3). Figure (c) quantifies this difference by calculating the ℓ^2 norm of the vector formed by sampling the analytic and calculated potentials at corresponding points.

The graph in figure 4.3 illustrates how the demagnetising energy and energy gradient become closer when performing the finite difference test (described above) for four sample points. Again it can be observed that the relative error decreases as the perturbation in the x component is decreased until the final point which exhibits a slight wobble.

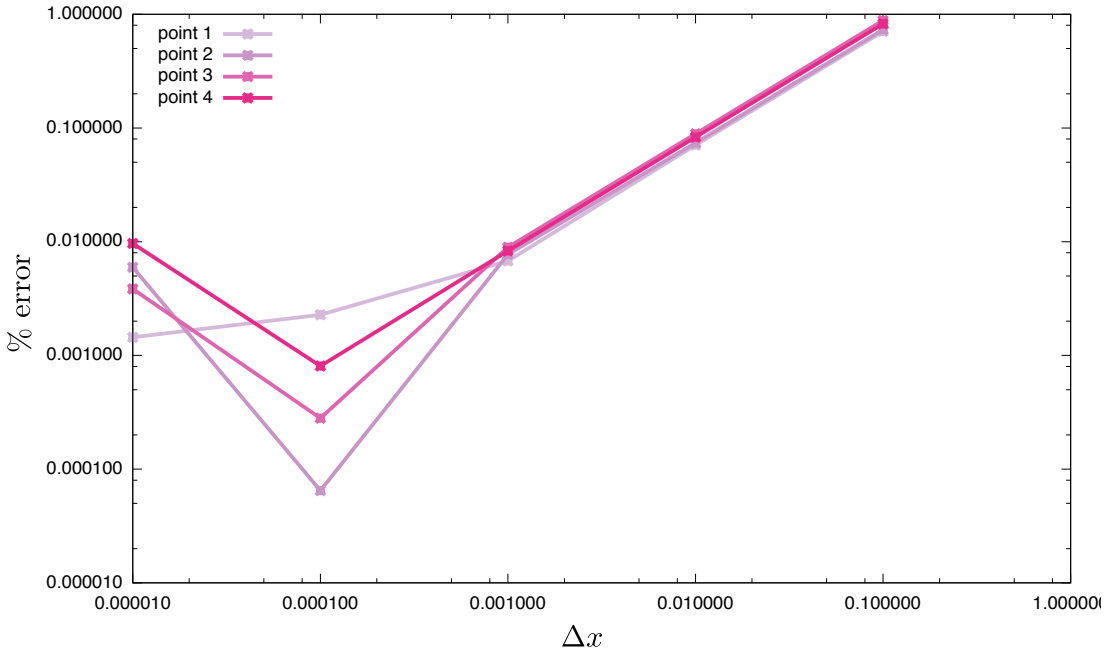
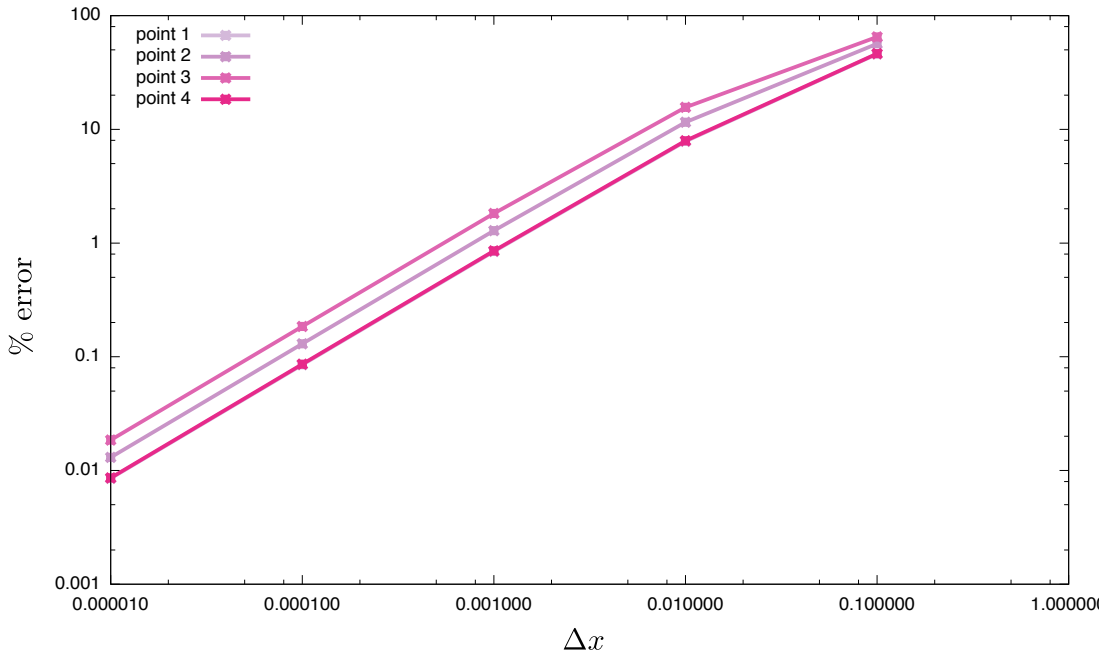


Figure 4.3: Results for the finite difference test of the demagnetising energy gradient for four randomly selected vertices.

4.1.2 Exchange and Anisotropy Field

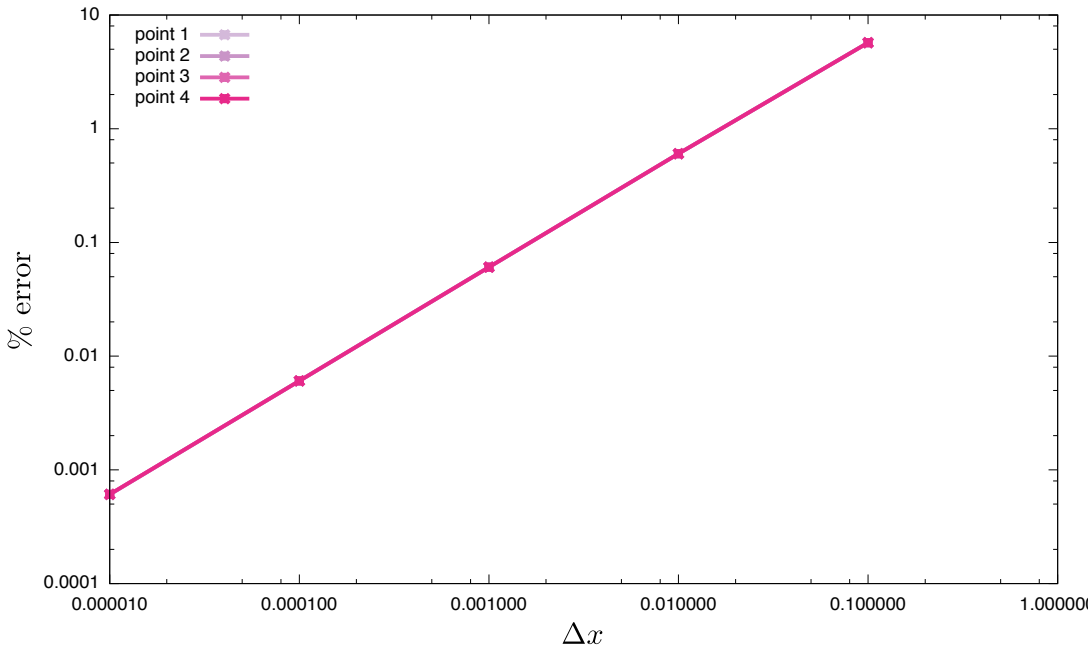
In order to verify the correctness of the exchange field, the discretised Laplacian is calculated in Mathematica (see appendix C). This results in the Laplacian matrix which is then applied to a vector field defined over a small test volume. The resulting exchange field is then used to verify that the output for the exchange field calculation in MicroMag is correct. A similar method is used for the anisotropy field, where expressions for the anisotropy energy and energy gradient are first calculated using Mathematica. The output of these worksheets are then incorporated in to unit tests.

Results for the finite difference test are presented in figure 4.4. Again the graphs illustrate that as the perturbation in the magnetisation is reduced, the finite difference gradient approaches the calculated finite difference gradient.



(a)

Figure 4.4



(b)

Figure 4.4: Finite difference tests for the exchange (a) and anisotropy (b) energy gradient for four randomly selected vertices.

4.1.3 Additional Tests

A number of other tests are incorporated in to MicroMag and form part of the unit testing framework. These are described briefly in the table below.

Test	Description
Box volume	verify that calculated box volumes match expected box volumes.
LLG timestepper	verify that the exchange calculator and an initial random field becomes uniform. verify that the anisotropy calculator and and initial uniform field aligns with an easy axis.
Energy solvers	verify that the exchange calculator and an initial random field becomes uniform. verify that the anisotropy calculator and and initial uniform field aligns with an easy axis.
Effective field calculators	verify that for a given magnetisation configuration, the calculated effective field matches expected values.

Table 4.2: Additional MicroMag tests.

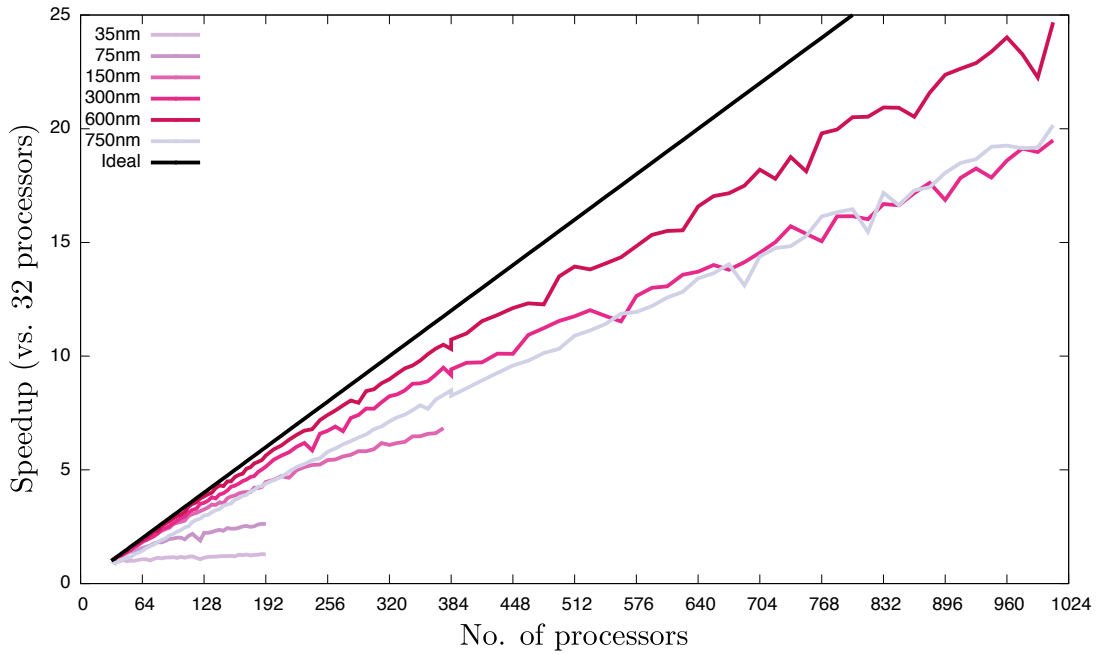
4.2 Performance

This section presents parallel performance scaling results for MicroMag. The primary focus is on the calculation of the effective field and in particular the demagnetising field since it is this calculation that dominates a micromagnetic computation. Scaling results are performed by calculating the average running time of an effective field calculation for a given problem size on a given number of processors. Calculations are split in to three runs. Within each run, the effective field calculation is performed six times, however the first calculation is always disregarded since the first call also includes an additional setup cost unseen in subsequent solver calls. Thus three runs are executed, with five timings made for each effective field component calculation. These times are then averaged to get the final execution time for an effective field component. The problem sizes are illustrated in the table below.

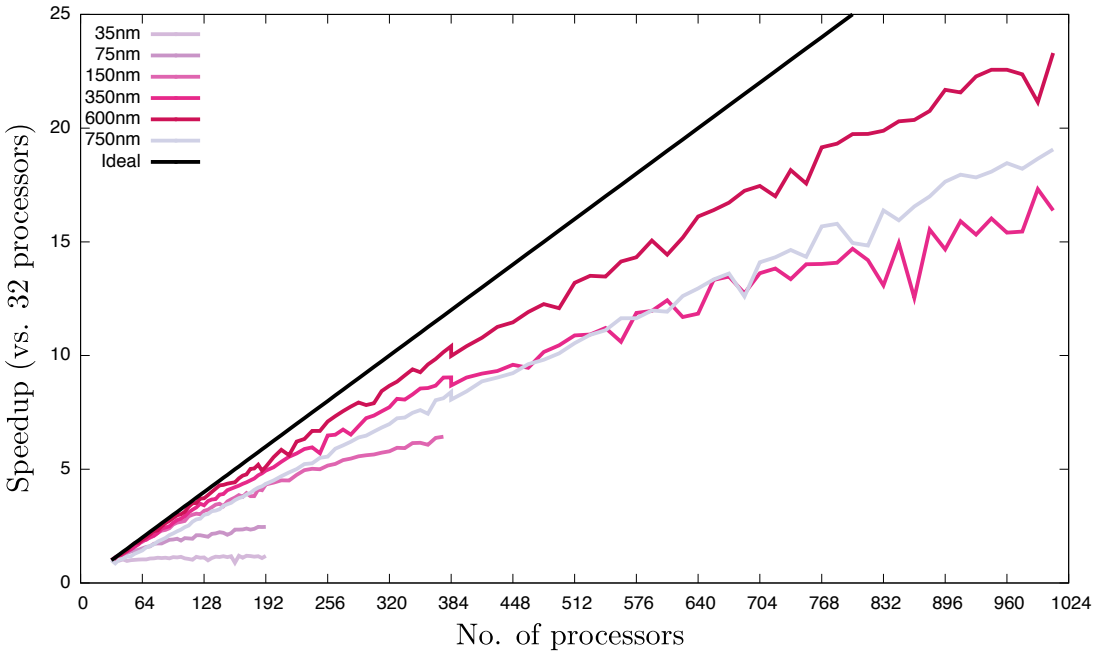
Radius	**No. of elements**
35nm	27,071
75nm	119,743
150nm	677,377
350nm	6,322,827
600nm	33,011,463
750nm	45,941,599

Table 4.3: Models used for scaling results. Each model consists of a sphere with given radius. The total number of elements per model is also shown.

Speedup results are shown for the demagnetising, exchange and anisotropy field calculations in the graphs below. The timings were computed on the ARCHER UK National Supercomputing Service (`http://www.archer.ac.uk`). This architecture consists of 4920 compute nodes each with two 12 core processors (thus there are a total of 24 compute cores per node). Scaling is with respect to 32 processors, since this spans two nodes - the final model consisting of a 700nm radius sphere does not fit on a single node and so for comparison purposes the value of 32 processors as a baseline is used.

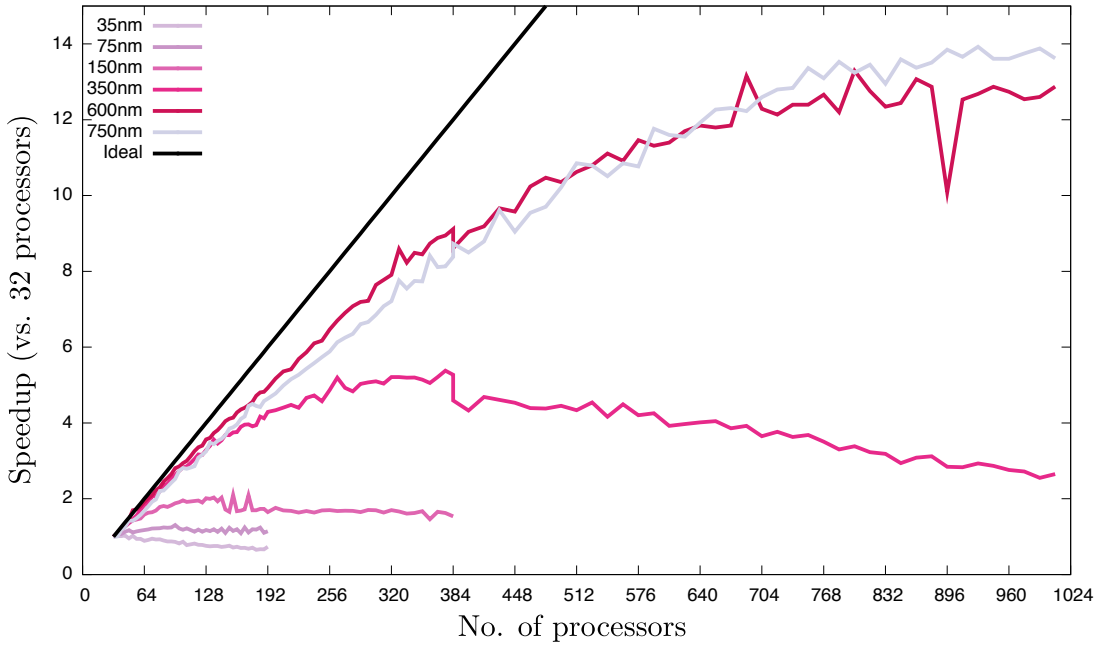


(a) Anisotropy field scaling



(b) Exchange field scaling

Figure 4.5



(c) Demagnetising field scaling

Figure 4.5: Scaling results for the calculation of the anisotropy (a), exchange (b) and demagnetising (c) fields.

The scaling graphs in figure 4.5 demonstrate that as the number of processors is increased, the performance of each respective calculation increases. Ideal scaling is defined as the case where speedup is equal to the number of cores being used (since the base line shown in figure 4.5 is 32 then ideal speedup is no of cores divided by 32). Scaling graphs 4.5a and 4.5b for the anisotropy and exchange calculations all show near ideal performance especially for large problem sizes. This is to be expected since for the anisotropy calculation, each field value at the node is independent of each other value. Likewise for the exchange, the computation consists of a single matrix vector multiplication. Actual speedups are not precisely ideal. For example when considering the anisotropy calculation at 256 processors a speedup of 8 is expected. However smaller actual speedups are observed (the best speedup being for the 600nm sphere with a value of 7.41). This slightly non-ideal performance is most likely due to the need for synchronisation of computations prior to combining values together in the effective field.

The computation of the effective field is dominated by the demagnetising field calculation. For example when using the 750nm radius sphere with 32 processors, the anisotropy calculation takes 6.2 seconds to complete where as the demagnetising calculation takes 38.3 seconds. Thus anisotropy and exchange computations start from much lower base than the demagnetising field, resulting in small gains regardless of large speedups. The graph in 4.5c shows speedups for the demagnetising field. These graphs demonstrate classic speedup curves, with an initial linear region then a tailing off of performance before a peak (the 350nm curve is particularly striking in this regard). Beyond the peak, the computation is dominated by the cost of communication between cores and in particular between nodes - 35nm, 75nm, 150nm and 350nm meshes all show how performance degrades beyond some optimal number of processors. It can be seen however that increasing the problem size shifts the point at which performance degrades to the right and thus it becomes feasible to use larger number of processors for larger and larger models.

This behaviour may be understood in the context of a simple model of the computations and communications taking place. Figure 4.6 shows an $n \times n$ problem split among p processors with local communication - the same communication pattern found in the demagnetising field calculation. Each circle represents a computation and each blue line represents communication between neighbouring processors. The total number of computations performed in parallel is n^2/p since those n^2 computations are divided among the p processors. In general any given processor must communicate with four of its neighbours in two communication phases, a sending phase and a receiving phase. Thus the cost of communication between a process and its neighbours is $\sqrt{n^2/p}$. Added to each communication is the cost of sending a message of length zero - analogous to the idea of turning on a tap and waiting for a given amount of time before the water begins to flow. This cost is the latency and is modelled by adding a constant L to the communication. The expected speedup for this model may then be estimated by

$$S(n,p) = \frac{n^2}{n^2/p + 8\sqrt{n^2/p} + L}, \tag{4.5}$$

where S is the speedup versus a single processor, n represents the size of the problem and p is the number of processors. This model shows behaviour similar to the actual scaling behaviour as can be seen in figure 4.7.

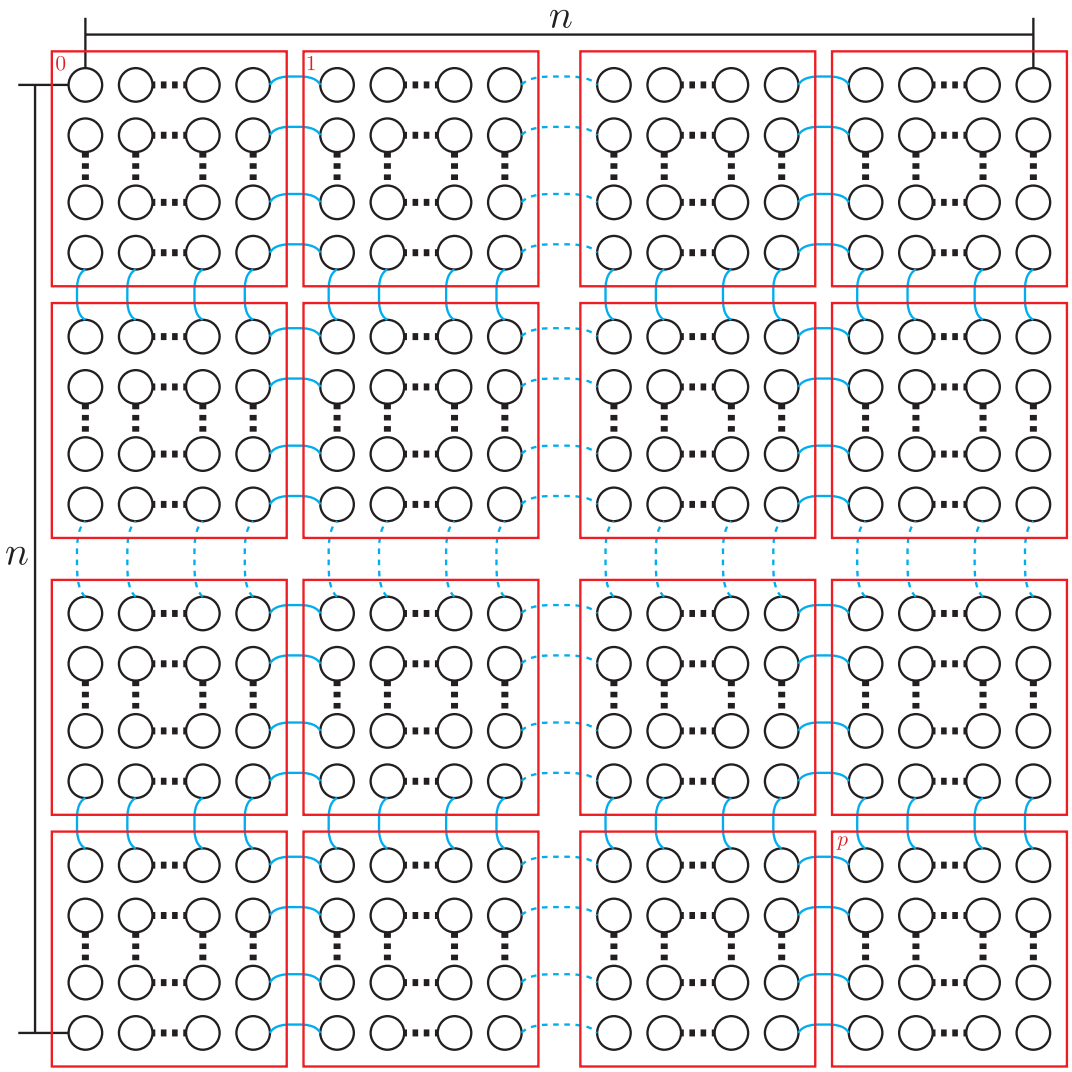
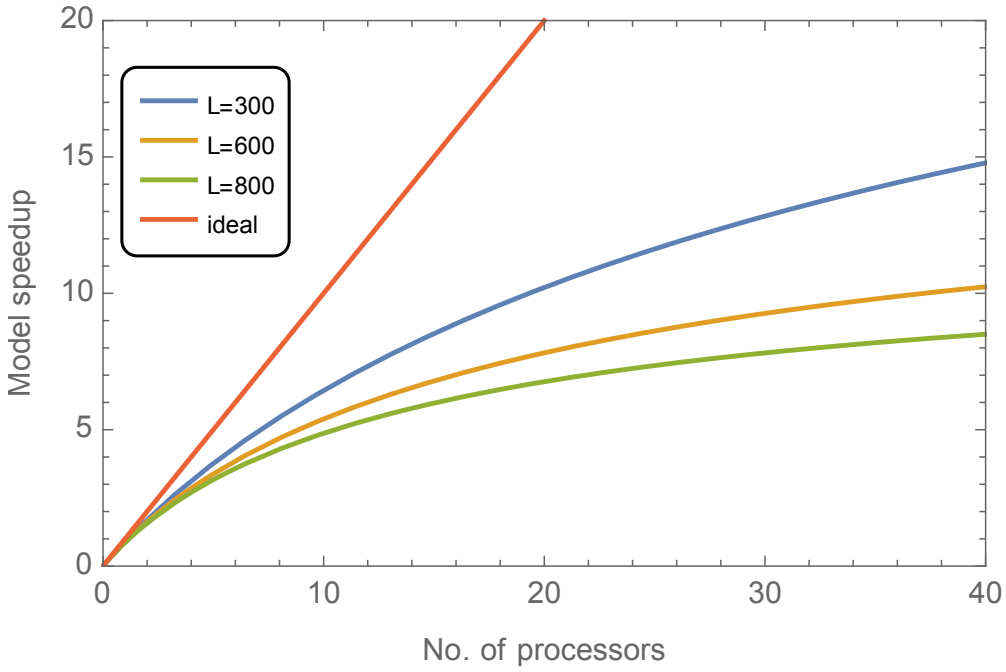
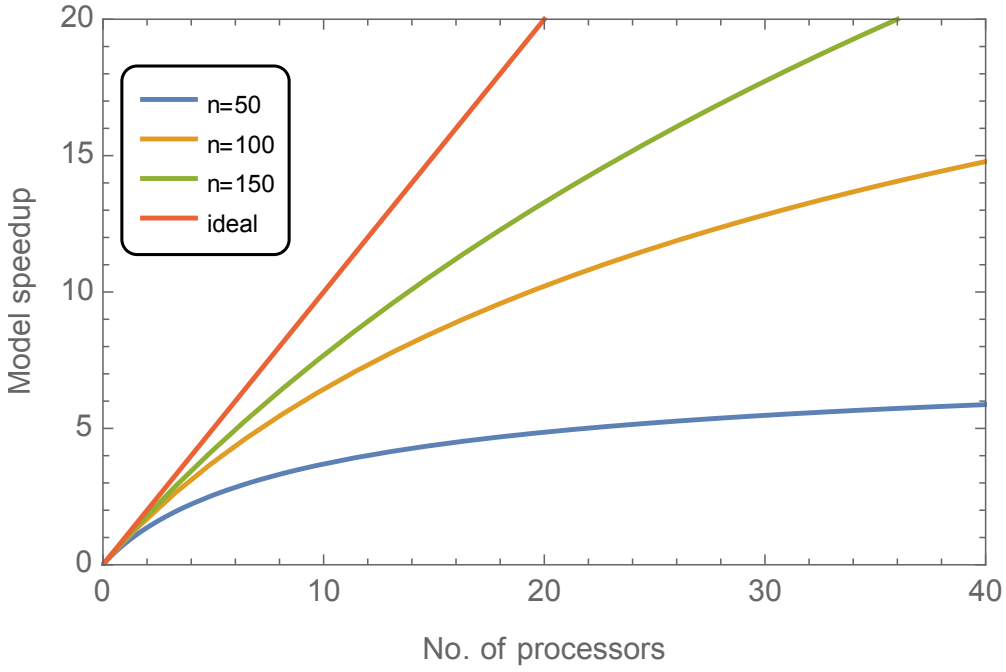


Figure 4.6: A simple model of scaling behaviour. This example considers a two dimensional mesh consisting of n^2 nodes divided among p processors. The blue lines indicates local communication that occurs between adjacent nodes - each communication consisting of a send and receive phase.



(a)



(b)

Figure 4.7: Scaling behaviour based on the asymptotic model in equation (4.5). Figure a shows the effect of increasing latency values for a fixed problem size; where as figure b shows the effect of increasing the problem size given some fixed latency value L. As can be seen the general shape of the curves is similar to the most significant scaling curves shown in figure 4.5c.

4.3 Summary

This chapter presented an overview of how the components of MicroMag are tested. In particular the demagnetising, anisotropy and exchange fields were examined. Finally, the performance of MicroMag was examined again with emphasis on the components of the effective field computation.

Bibliography

John David Jackson and John D Jackson. *Classical electrodynamics*, volume 3. Wiley New York etc., 1962.

Romain Ravaud and Guy Lemarquand. Magnetic Field Produced by a Parallelepipedic Magnet of Various and Uniform Polarization. *Progress In Electromagnetics Research*, 98:207–219, 2009.

Chapter 5

Domain Structure, From Nano to Micro Scale

5.1 Introduction

The magnetisation structures found in ferromagnetic materials are able to respond to relatively low external magnetic fields. Such materials can then retain these structures over geologically significant time scales. It is important for us to understand these domain structures since they record information about the conditions in which the magnetisation occured. For example, paleomagnetic core samples provide strong evidence that the Earth's crust is divided into several moving pieces called tectonic plates; since to a first approximation and accounting for geomagnetic reversals, core samples found in the field do not have a uniform magnetisation.

Understanding the stability and formation of domain structures is extremely important. This chapter presents a size-hysteresis study that examines the evolution of domain structure as a function of grain size in magnetite - a geologically important mineral. There are two main questions to address: 1) how stable are multi-domain structures in geologically significant grain sizes and 2) how do these domain structures evolve as the size of a geometry is changed? In order to answer these questions two grain shapes, a sphere and a cuboctahedron, have been chosen. The size of these grains is then increased (from the nanometre scale up to

the micrometer scale) and then reduced back to the original size. By seeding the initial magnetisation for a given sample with the solution of a previous sample, changes in domain structure are then observed.

5.1.1 Background

Mathematical modelling of large multi-domain grains is an extremely difficult task. One of the primary difficulties is calculation of the demagnetising field - a quantity that depends not only on grain geometry but also on the current magnetic configuration. This was first noted by Merrill (1977) and pursued by Dunlop (1983) and by Dunlop and Xu (1994); Xu and Dunlop (1994) when attempting to construct a theory of thermoremanent magnetisation in multi-domain grains. While these were valiant attempts at understanding how multi-domain structures develop, they made the assumption of laminar layers of magnetisation. This gives rise to a one dimensional description of domain structures, since it is assumed that each layer is uniformly magnetised.

On the other hand, one of the first unconstrained micromagnetic models was developed by Williams and Dunlop (1989). Such models make use of finite difference schemes to compute the demagnetising field. These methods were enhanced by using the Fast Fourier Transform (FFT) method (Fabian et al., 1996) with the addition of running FFT models in parallel processors by (Wright et al., 1997). Although finite difference schemes are useful for modelling cuboidal geometries, a micromagnetic model for arbitrary geometries was developed by Williams et al. (2006). This method uses the boundary element method (Fredkin and Koehler, 1990) to calculate the demagnetising field.

The problems with the methods outlined above is that the material is either constrained to be cuboidal, or must make use of the boundary element method which results in a dense matrix-vector system for solution. Such models are therefore limited in the maximum grain size that can be modelled. For example Williams et al. (2006) examine particles in the range 30nm to 200nm.

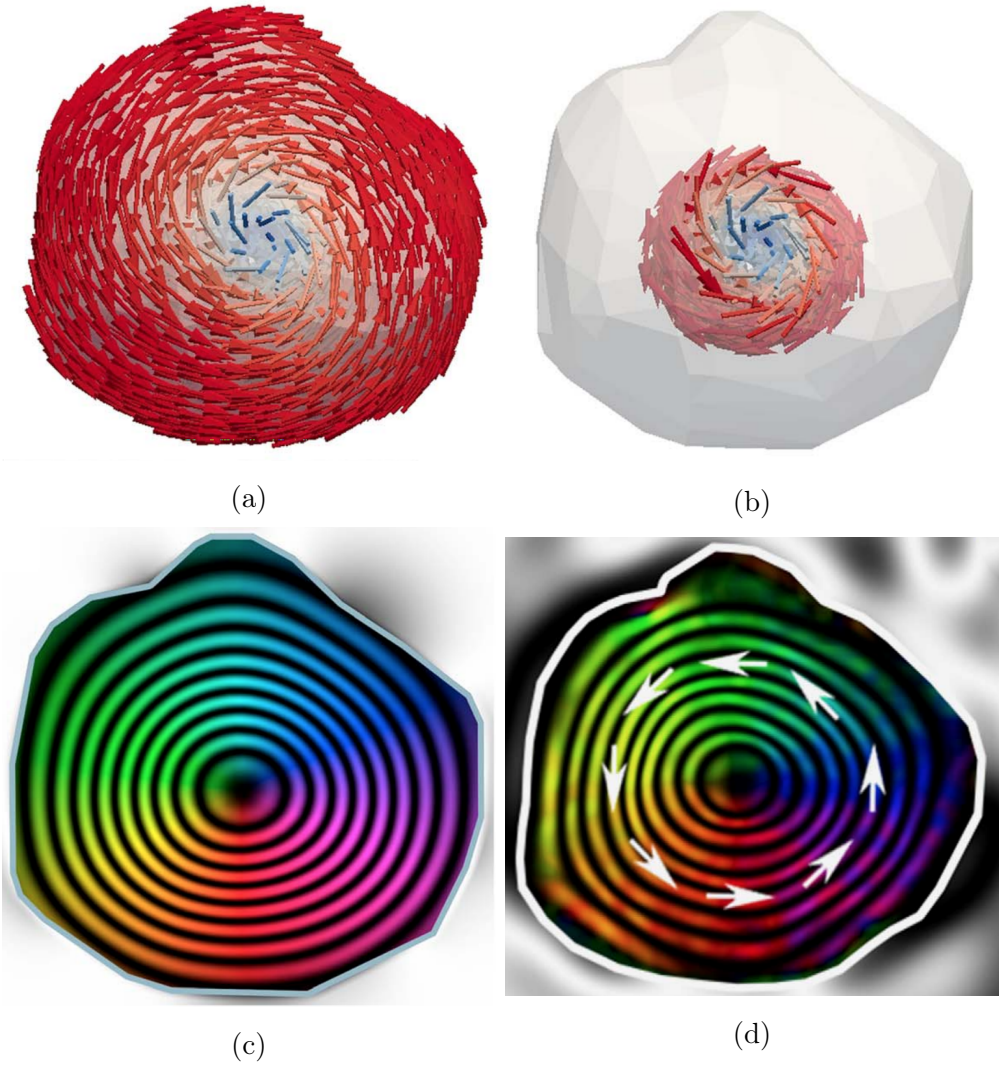
There are several techniques that may be used to experimentally image domain structures in magnetic materials. Once such method is the Bitter technique,

which uses a colloidal suspension of magnetic particles (usually magnetite) applied to a specially prepared surface (Hubert and Schäfer, 1998). Such ferrofluids then interact with the stray field produced by the magnetisation of the particle. Particle sizes that may be visualised in this way are usually large, on the order of 10μm however Geiß et al. (1995) managed to obtain Bitter pattern images for grains as small as 0.5μm (500nm).

The magnetic force microscope method maps out the stray field due to the magnetic structure within a grain. It consists of a magnetic needle that traverses a prepared surface, this needle experiences a force due to the stray field. One of the first such experiments were performed by Williams et al. (1992), who used the technique to produce a three dimensional map of the domain structure over a 20μm^2 section of magnetite.

Magneto-optical methods such as the Kerr and Faraday techniques, rely on the interaction of polarised light with the electron structure of the atoms comprising a ferromagnetic crystal (Hubert and Schäfer, 1998). Once more these techniques have been used to produce images of domain structures in magnetite on the order of 10μm, as observed in Appel et al. (1990) for example.

More recently, Almeida et al. (2014b), in collaboration with the author, have produced images of domain structures in small (on the order of 200nm) grains of magnetite using off-axis electron holography. Almeida et al. (2014a) examined the stability of domain structures in magnetite as a function of temperature. In this study the small grain shown in figure 5.1d was heated from room to temperature to 700°C and then cooled back down to room temperature; with domain structures observed at 100°C intervals. The study found that the pseudo-single domain vortex structure was robust and maintained its fidelity in the presence of heating/cooling (Almeida et al., 2014a).



(a) (b)

(c) (d)

Figure 5.1: Simulated and observed holography. Image (a) shows the micromagnetic simulation for a 200nm pseudo-single domain grain of magnetite, with the core highlighted in (b). The simulated holography image (c) shows a close correspondence between the observed structure (d). Images (a), (b) and (c) are taken from Almeida et al. (2014a), image (d) is taken from Almeida et al. (2014b).

In order to calculate the simulated holography image in figure 5.1c a three dimensional model was built of the grain using Blender (Blender community, 2014), as seen in figure 5.2b. Trelis (csimsoft, 2015) was then used to produce the mesh in figure 5.2a. The generation of degenerate elements (discussed below) was

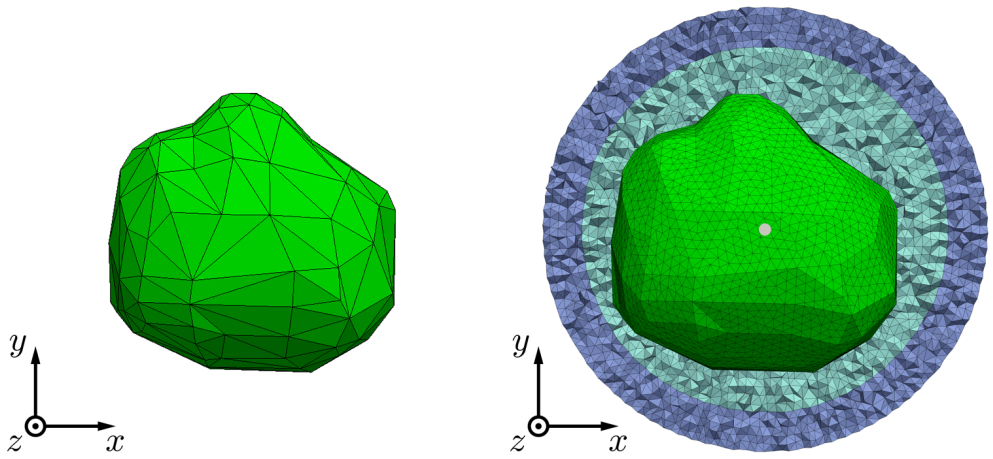
avoided by manually adjusting the blender model and inner and outer spherical shells were added in Trelis.

Micromagnetic calculations were run using the code described in chapter 3 to produce the models seen in in figures 5.1a and 5.1b. A magnetic vector potential $\boldsymbol{A}$, was then calculated from, the results of the micromagnetic calculation, using a technique similar to the computation of the scalar potential in chapter 3.

By integrating the z component of the vector potential, the in-plane phase shift was calculated according to (5.1)

$$H^{ij} = \cos\left(c\sum_{k=1}^{n}\boldsymbol{A}_z^{ijk}\right), \tag{5.1}$$

where H is the phase shift and c is an amplification factor (Almeida et al., 2014a). The contours in figure 5.1c were then coloured according to the direction of the in-plane averaged mangetic induction field.



(a) (b)

Figure 5.2: Meshes for simulated holography used in Almeida et al. (2014b,a). The mesh in figure (a) was generated using the Blender three-dimensional modelling package (Blender community, 2014). This model was then read in to Trelis (csimsoft, 2015) and inner and outer shells were generated. This resulted in the mesh in figure (b) was generated.

5.2 Materials and Methods

Tables (5.1 & 5.2) give a summary of the geometries used for the size hysteresis study. Spheres are denoted by their radius in nano meters (nm) where as cuboctahedra are denoted by an equivalent volume radius; that is a 100nm cuboctahedron has a volume equivalent to a sphere of 100nm radius. Cuboctahedra are generated by scaling a 'standard cuboctahedron' with vertices $(\pm 1, \pm 1, 0)$, $(\pm 1, 0, \pm 1)$ and $(0, \pm 1, \pm 1)$ this results in a cuboctahedron with edge length $\sqrt{2}$nm as seen in figure 5.3. The scale factor S in (5.2) is then applied to the standard cuboctahedron to produce a geometry with volume equivalent to a sphere of radius r. Scale factors are calculated according to

$$S = \sqrt[3]{\frac{\pi}{5}} r. \tag{5.2}$$

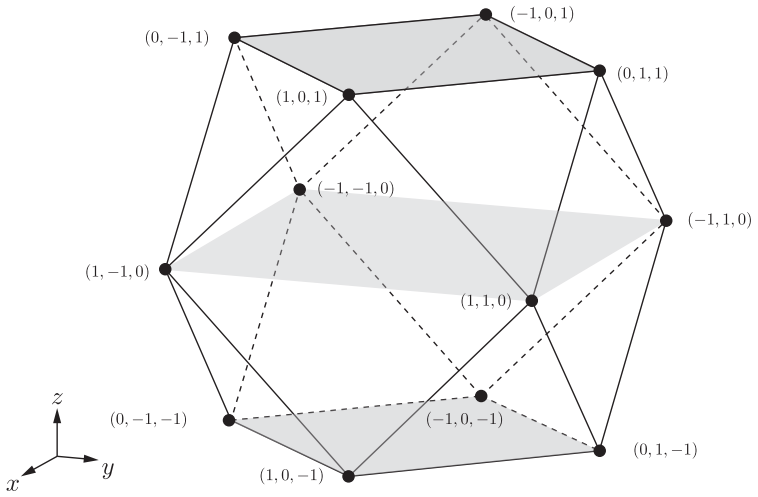


Figure 5.3: The standard cuboctahedra used when generating cuboctahedral meshes. The three planes (in grey) represent the planes $z = 1$ (top), $z = 0$ (middle) and $z = -1$ bottom. The vertices for this geometry are scaled by the scale factor to arrive at a cuboctahedron with volume equivalent to a sphere of a given radius.

5.2.1 Mesh Generation

It is desirable to generate 'good meshes' for the spherical and cuboctahedral geometries described above. These meshes should have the following properties (which are explained in more detail below):

1. mesh elements should not be degenerate,
2. mesh vertices between regions should be conforming,
3. the meshing algorithm should try to respect the element size specified by the user as much as possible.

Long and thin elements or elements containing coplanar vertices are said to be degenerate. Such elements result in numerical instabilities when constructing a stiffness matrix (as outlined in section 2.4.2). Therefore the meshing algorithm needs to be able to identify degenerate meshes and adapt the construction of the mesh to eliminate them. This is usually done by local refinement, where the local area around a degenerate element is subdivided and vertices in that neighbourhood are repositioned.

As explained in section 3.4.3, the mesh needs to specify three sub-regions Ω_{mat}, Ω_{umap} and Ω_{map} (see figure 3.4). When sub-meshes are produced for these sub-regions they must be conforming, *i.e.* they must share the same vertices and those vertices must lie on a specific boundary region. This puts a restriction on the mesh generation and increases the difficulty of eliminating degenerate elements, since vertices on the boundary between two sub-regions cannot be arbitrarily moved (*i.e* any movement of these vertices must ensure that they remain on the boundary region).

The meshing software Trelis (csimsoft, 2015) is used to generate the meshes of the geometries described previously. This software tool allows the user to specify a three-dimensional model along with mesh requirements such as element shape, approximate element size, geometry boundaries and sub-mesh regions. Listing 5.1 provides an example of a typical use case, with a cubic material region and concentric spherical shells.

```
1  create brick  x 0.080
2  create sphere r 0.090
3  create sphere r 0.200
4
5  subtract volume 1 from volume 2 keep
6  subtract volume 2 from volume 3 keep
7  delete volume 2, 3
8  compress
9
10 merge   volume 1 2
11 imprint volume 1 2
12
13 merge   volume 2 3
14 imprint volume 2 3
15
16 volume all scheme tetmesh
17 volume 1 size 0.007
18 volume 2 size 0.008
19 volume 3 size 0.009
20
21 mesh volume all
22
23 block 1 volume 1
24 block 2 volume 2
25 block 3 volume 3
```

Listing 5.1: Example Trelis script. Line 1 creates a cube of length 80nm (with base units in micons) and lines 2 & 3 create the inner and outer sphere respectively. Line 5 performs a set-wise subtraction of the cubic volume from the first sphere, keeping the original geometries and line 6 subtracts the inner sphere from the outer sphere. Lines 7 & 8 delete the original spheres and line 8 performs a renumbering to eliminate volume-index gaps (due to deletions). Lines 10-14 specify boundaries between cube and inner spherical shell and the inner/outer shell - mesh vertices will conform to these boundaries. Lines 16-19 specify the type of elements to use when creating a mesh, along with element sizes for each volume. Lines 23-25 associate index numbers with the volumes, these correspond to sub-meshes in Micromag.

Trelis defines the size of an element as being approximately equal to an edge length and it attempts to keep generated elements as equilateral as possible. Consequently Trelis was found to generate good meshes, but with the disadvantage of requiring an external proprietary tool.

The exchange length of magnetite is approximately 9nm; and element sizes in the range 7nm to 10nm were used. This range is a balance between achieving good resolution in meshes and difficulties that arise when generating large meshes. Such difficulties occur when attempting to tessellate a large space space with very small elements resulting long mesh generation times and degenerate tetrahedra. It was found that partitioning the initial geometry description in Trelis into sub-pieces alleviated much of these problems, however this then introduces the problem of having to explicitly define boundaries between sub-pieces.

The demagnetising field calculation is described in detail above (sec. 3.4.3), but briefly it is solved for all space using a spatial transformation due to Imhoff et al. (1990) and Brunotte et al. (1992). This means that it is necessary to mesh two additional regions of space along with the material. These are referred to as the 'mapped region' and the 'unmapped region'. The mapped region, applies the spatial transformation outlined in (Imhoff et al., 1990) to calculate the demagnetising potential up to infinity. The calculation for the unmapped region is similar, only no spatial transformation is applied. These extra submesh regions allow the problem to be parallelised (see section 3.4.3) however they introduce additional constraints on mesh generation, since element vertices must be conforming across adjacent regions. Unlike the submesh associated with the material region, space-region submeshes (both mapped and unmapped) are not constrained to be within a particular size range. However, the majority of the meshes generated for this study do use a common element size. The main reasons for this are that 1) resolving the space region in high detail produces a more accurate estimation of the demagnetising potential and 2) large differences between element sizes of adjacent mesh regions can cause Trelis to produce elements larger than the desired size range. In the cases where element sizes between material and space regions differ, such differences are kept relatively small in order to maintain high resolution of space regions, but to minimise distortion of material submesh elements.

	element size (nm)	
Radius (nm)	Material	Mapped
15	7.0	7.0
25	7.0	7.0
35	7.5	7.5
45	7.5	7.5
55	7.5	7.5
75	7.5	7.5
100	7.5	7.5
125	7.5	7.5
150	8.0	8.0
200	9.0	9.0
250	9.0	9.0
350	9.0	9.0
450	9.0	9.0
600	9.0	9.0
750	10.0	10.0

Table 5.1: Summary of spherical geometries with associated element sizes used for material and mapped region. Note that spherical geometries do not require the unmapped region.

		element size (nm)		
Radius (nm)	Scale factor	Material	Mapped	Unmapped
15	0.0128	7.0	7.0	7.0
25	0.0214	7.0	7.0	7.0
35	0.0299	7.0	7.0	7.0
45	0.0385	7.0	7.0	7.0
55	0.0471	8.0	8.0	8.0
75	0.0642	8.0	8.0	8.0
100	0.0856	8.0	8.0	8.0
125	0.1071	8.0	8.0	8.0
150	0.1285	8.0	8.0	8.0
200	0.1713	8.0	8.0	8.0
250	0.2141	9.0	9.0	9.0
350	0.2998	9.0	9.0	9.0
450	0.3854	9.0	9.0	9.0
600	0.5139	9.0	9.0	9.0
750	0.6424	10.0	20.0	20.0
1000	0.8565	10.0	20.0	30.0
1350	1.1563	10.0	30.0	40.0

Table 5.2: Summary of cuboctahedral geometries. The scale factor is the value used to scale a cuboctahedral 'standard mesh' to a mesh with equivalent spherical volume given by the radius value according to (5.2). Element sizes used for material, mapped and unmapped mesh regions are also shown.

5.2.2 Running Models

Micromagnetic models are run on the ARCHER supercomputer service (ARCHER). This service consists of a Cray XC30 supercomputer with 4920 compute nodes. Each node consists of two 12-core, 2.7 GHz Intel Ivy Bridge processors - giving a total of 118080 processor cores. This results in a theoretical maximum peak performance of 2×10^{14} instructions per second. A key feature of ARCHER is Cray's Aries interconnect - a high bandwidth, low latency network configured in

a Dragonfly topology (Kim et al., 2008; Alverson et al., 2012).

Performance curves given in 2.4.3 are used as a rough guide to select the numbers of processors used to calculate solutions for each of the meshes. Actual numbers of processors used for particular problems are selected to be within approximately 50% parallel efficiency[1], for example when running 600nm models the number of cores used was $24 \times 7 = 408$ resulting in a parallel efficiency of approximately 48%.

Size hysteresis for the models outlined in (tbls. 5.1 & 5.2) are performed in sequence with the output of one model being the input of the next. The initial starting magnetisation for the smallest meshes is chosen to be slightly off the $\langle 1, 1, 1 \rangle$ direction, the result of this calculation is then the initial magnetisation for the next mesh. Since mesh sizes vary in size, it is necessary to interpolate the result of one model to another. In order to achieve this, a mesh vertex from a destination mesh is scaled to a point in the source mesh. This destination point is not guaranteed to correspond with a vertex in the source mesh and so the actual value of the magnetisation is linearly interpolated from the source mesh value and then normalised to be of unit length.

The Hubert minimiser described in section 3.5.3 is used to calculate minimum energy domain states for the magnetisation in each model. The energy energy minimisation parameters are kept the same for each calculation. Parameters used in the Hubert energy minimiser are outlined in the table below.

Parameter	Value	Description
α	0.001	Multiplier amount for traversal in a gradient direction.
$\Delta\alpha$	2.6	The amount to travel in a gradient direction.
ΔE	1×10^{-9}	The energy difference for exit.
G^2	1×10^{-11}	The gradient magnitude difference for exit.

Table 5.3: Parameters used in the Hubert energy minimiser (see section 3.5.3 for further details).

[1]Parallel efficiency is the fraction of the total effort in performing a calculation that is not lost in communication and other computational overheads. It is defined as $E = {}^{S}/_{p}$, where E is parallel efficiency, S is the speedup and p is the number of processors/cores used.

Material Parameters

Micromagnetic modelling assumes that the material parameters for magnetite remain constant. The material parameters used are presented in (tbl. 5.4).

Parameter	Value	Description
M_s	4.8×10^5 ($^{A}/_{m}$)	Saturation magnetisation.
A	1.34×10^{-11} ($^{J}/_{m}$)	Exchange constant
$K1$	-1.24×10^4 ($^{J}/_{m^3}$)	Anisotropy constant.

Table 5.4: Material parameters for magnetite at room temperature (25°C) taken from Pauthenet and Bochirol (1951) and Heider and Williams (1988).

5.3 Results

Results are obtained by micromagnetic modelling using the MicroMag code described in this thesis and are in the form of HDF5 (HDF) files along with accompanying XDMF (Balay et al., 2011) metadata files. These files are read and processed using Paraview (Kitware, 2015). The image in figure 5.4 illustrates a typical micromagnetic solution. It is of a vortex domain state in a 100nm radius cuboctahedra (coloured by the normalised anisotropy energy described below); the remainder of this thesis refers to model sizes in terms of radii of equivalent spherical volume and the statement "100nm radius cuboctahedra" is understood to mean a cuboctahedron with volume equivalent to a sphere of 100nm.

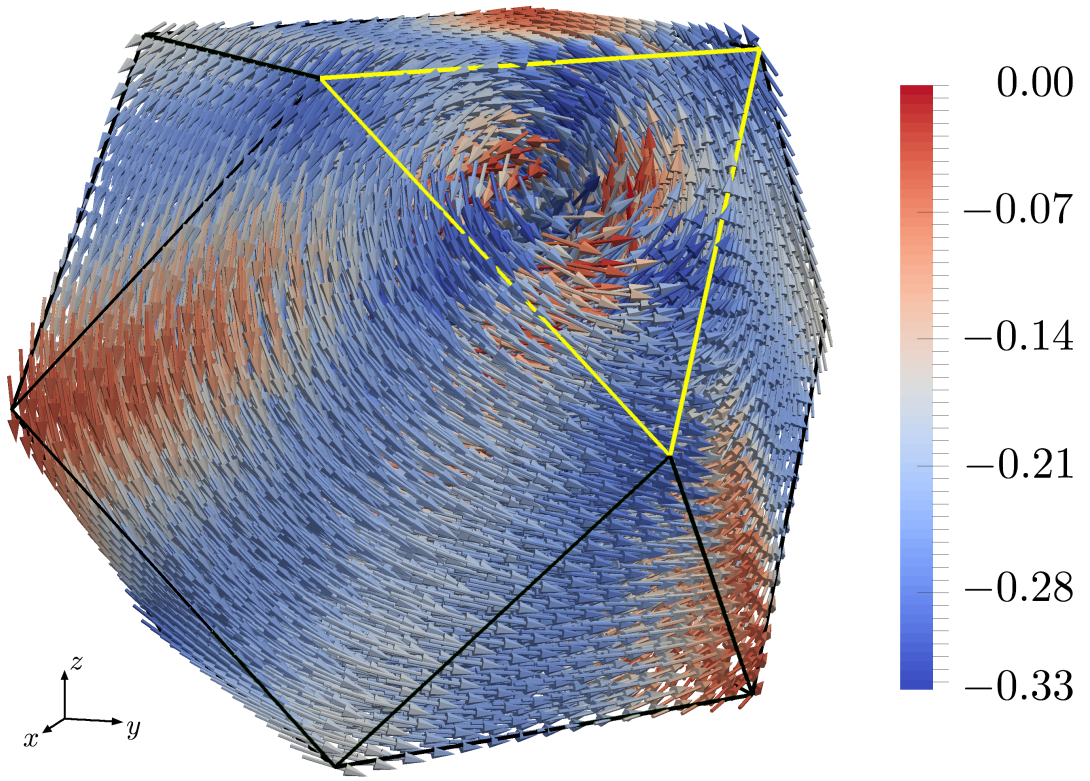


Figure 5.4: Vortex core pseudo single domain (PSD) structure in a 100nm cuboctahedron aligned along the $\langle 1, 1, 1\rangle$ direction. The size of the cuboctahedron is equivalent to the volume of a sphere with 100nm radius. The vectors have been coloured by the normalised anisotropy energy which is given by (5.3). Note how the vortex core produces three red spots on the triangle oriented in the $\{1, 1, 1\}$ plane (highlighted yellow) as the vectors in the core rotate through three hard axes.

The image in figure 5.4 illustrates immediately the difficulties of visualising micromagnetic data and three dimensional vector fields in general. Even if fully interactive three dimensional models are available, extracting meaningful information from a vector field is not straight forward. This thesis primarily uses two ways to present results. Firstly slices are taken along the crystallographic planes $\{1, 1, 0\}$ and $\{1, 1, 1\}$. These planes are then coloured according to normalised

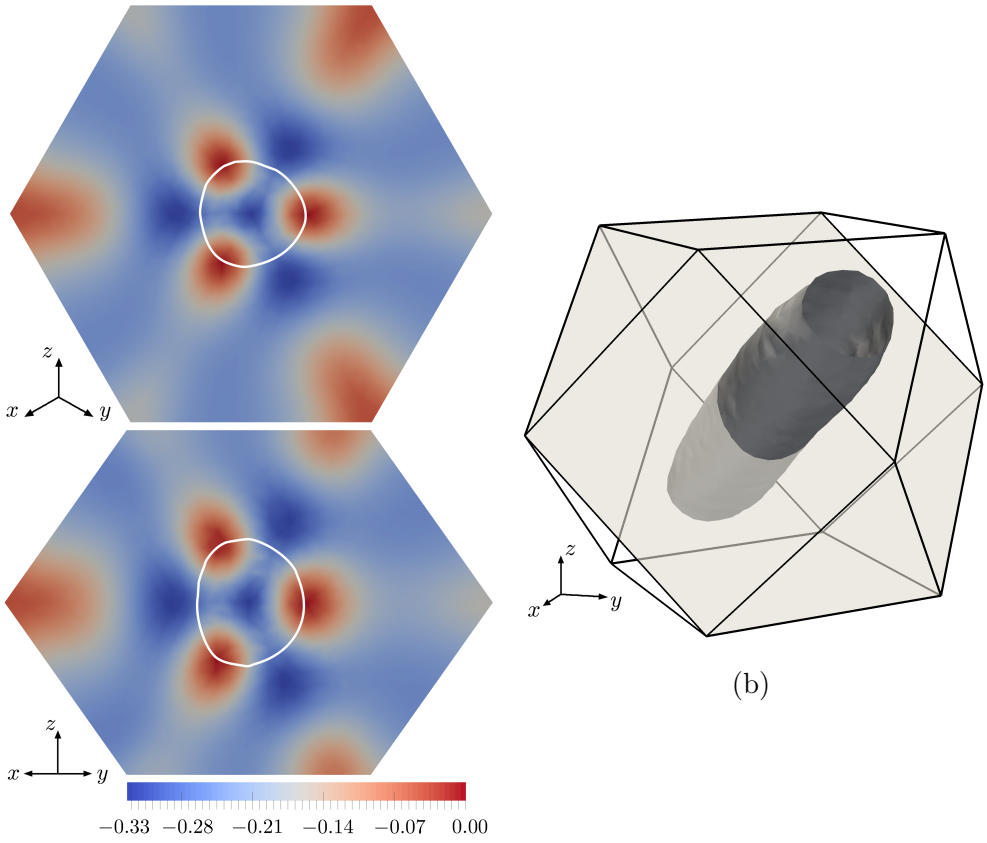
anisotropy energy

$$E_{\mathrm{a}} = -\frac{1}{2}\left(1 - \left(m_x^4 + m_y^4 + m_z^4\right)\right), \tag{5.3}$$

where m_x, m_y and m_z are the Cartesian components of the magnetisation field and $-1/3 \leq E_{\mathrm{a}} \leq 0$. Thus a value of zero for E_{a} corresponds to an energy maximum and a vector lying along a crystallographic hard axis for magnetite (*e.g.* a $\langle 1,0,0\rangle$ direction), whereas a value of $-1/3$ corresponds to an energy minimum and a vector lying along a crystallographic easy axis (*e.g* a $\langle 1,1,1\rangle$ direction). The second way of visualising domain structures used in this thesis is to look at helicity isosurfaces. The helicity is given by

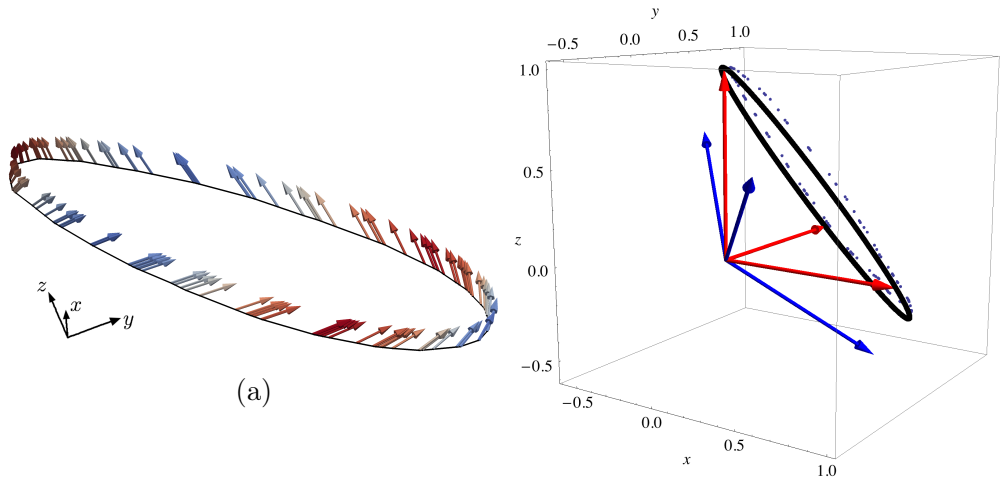
$$H = \boldsymbol{m} \cdot (\nabla \times \boldsymbol{m}) \tag{5.4}$$

where $\boldsymbol{m}$ is the magnetisation. Informally helicity may be thought of as the amount of local twisting in the magnetisation vector field. This means that regions of high helicity correspond to domain structures such as domain walls or vortex cores. The images in figure 5.5 illustrate the normalised anisotropy energy in the $\{1,1,1\}$ and $\{1,1,0\}$ planes, along with the helicity isosurface for the 100nm cube of figure 5.4. It is interesting to note how the vortex core is highlighted by the presence of three red dots in the anisotropy plane images. This feature occurs because the vectors that correspond to the vortex core (as seen in figure 5.6a) approximately trace out a cone with a base in the $\{1,1,1\}$ plane (see figure 5.6b). Since this cone contains three hard axes for magnetite, the magnetisation vectors pass through these hard axes resulting in three high anisotropy energy regions, given by the three red dots. These observations are confirmed by Witt et al. (2005) in their study of domain structures in magnetosomes.



(a) (b)

Figure 5.5: Anisotropy and helicity information for the 100nm cuboctahedron. Subfigure (a) shows anisotropy slices for the 100nm cuboctahedron in the $\{1, 1, 1\}$ plane (top) and the $\{1, 1, 0\}$ plane (bottom). The colouring is by normalised anisotropy energy (5.3); the intersection between the anisotropy planes and the helicity surface of subfigure (b) is shown as a white outline. Subfigure (b) shows a helicity isosurface for the 100nm cuboctahedron. The orientation of the geometry is the same as that found in figure 5.4, however this time the vortex core is easily identifiable as the tubular structure aligned along the $\{1, 1, 1\}$ plane (highlighted by shading).



(a) (b)

Figure 5.6: Change in magnetisation along helicity contour. Subfigure (a) illustrates how the magnetisation vectors along the helicity contour, the white loop in figure 5.5a, change when plotted in the $\{1, 1, 1\}$ plane (see also Witt et al. (2005) figure 11). It can be seen that these vectors precess about the $(1, 1, 1)$ vector and trace out a cone. Subfigure (b) plots the end points of these vectors (blue dots) and they can be seen to trace out the circle of a cone; the black circle shows the base of a cone passing exactly through the hard axes: $\langle 1, 0, 0\rangle$, $\langle 0, 1, 0\rangle$ and $\langle 0, 0, 1\rangle$ (shown as red arrows). The blue dots almost match the black circle (the closeness of the match depends on the choice of helicity) and so the magnetisation along the helicity contour passes approximately through each of the three hard axes. This gives rise to the three (high anisotropy energy) red dots seen in figure 5.5a. The three easy axes: $\langle -1, 1, 1\rangle$, $\langle 1, -1, 1\rangle$ and $\langle 1, 1, -1\rangle$ are also shown in subfigure (b) as blue arrows. The end points of these vectors are some distance away from the black circle and blue dots. This again corresponds to what is observed in figure 5.5a since the white loop does not pass through any dark blue coloured regions (of low anisotropy energy).

5.3.1 Size Hysteresis, Ascending

Results for the size hysteresis described in section 5.2 are presented in figure 5.7. In the case of both cuboctahedra and spherical models the initial state is uniformly magnetised along the $\langle 1,1,1\rangle$ axis. This can be seen by the uniform dark blue anisotropy energy surfaces for both models. As the model size increases, the domain state develops in to a flower state for the cuboctahedral grain at around 55nm radius - with the 75nm radius grain developing more distinct flowering at the corners of both the anisotropy planes. For spherical geometries, the 55nm radius point shows a sudden change in the domain structure to something resembling a vortex state as shown by the three spots in the anisotropy images.

By the 100nm point both cuboctahedra and spheres have well developed vortex cores. However, spherical vortex cores are not exactly $\langle 1,1,1\rangle$ aligned (unlike cuboctahedra). This remains the case until 200nm for spheres, at which point the vortex core aligns to the easy axis. This transition can be seen most clearly by observing the $\langle 1,1,1\rangle$ and $\langle 1,0,0\rangle$ orientated helicity surfaces - the 7^{th} and 9^{th} columns of (figs. 5.7c & 5.7d) - which show the tube of the core rotating from an almost $\langle 1,1,0\rangle$ orientation to $\langle 1,1,1\rangle$. In the same size range (100nm to 200nm), cuboctahedra vortex cores show no change in orientation. However the round, tubular character of the helicity isosurface gives way to a structure that resembles a twisted triangular prism. The spherical helicity isosurfaces also show some evidence of becoming more triangular (column 9 of 5.7d) by 200nm.

The anisotropy slices are also interesting when considering the 100nm to 200nm size range. For spheres in particular it can be seen that the 100nm sphere shows four red spots in the anisotropy slices, with the left and rightmost spots smeared. As the vortex core becomes $\langle 1,1,1\rangle$ aligned this gives way to the expected three red spots which appear to tighten up as size increases and flattening of the helicity becomes more pronounced. This tightening up of the three red spots is also observed in the cuboctahedral grains but to a lesser extent and corresponds to the vortex core transitioning from a tubular structure to the twisted triangular structures observed.

From 250nm to 600nm, the core shows little development for spherical geometries with regards to helicity. The flattening of the isosurface becomes more

pronounced and the red spots in the anisotropy surfaces seem to become tighter. At this point light blue spokes of higher energy anisotropy regions emanate from the red spots outward towards the grain surface. These regions separate successively larger dark blue regions of low anisotropy. The clearest example is the 600nm sphere which hints towards the possible final domain structure with dark blue domains and lighter blue walls.

For the same size range (250nm to 600nm) the cuboctahedra show greater development in the vortex core. It can be seen that at the ends of the isosurface fins begin to emerge (350nm). Whereas the centre of the isosurface becomes more triangular. This corresponds to the tightening of the red spots in the anisotropy slices - since the more triangular the core becomes the less distance the magnetisation vectors have to rotate through the hard axes. By 600nm the fins are greatly pronounced and resemble propellers oriented at 45° to each other, as seen most clearly in column 7 of 5.7b. Again, a nascent domain structure is present in the 600nm cuboctahedra, seen most clearly in the $\langle 1, 1, 1 \rangle$ anisotropy slice. This is similar to what is observed for spheres but the light blue spokes fan out at a greater rate and the dark blue domains are not as pronounced. Furthermore the corners show tight regions of high anisotropy which are likely controlled by the geometry of the grain.

For spheres the transition from 600nm to 750nm is dramatic. Both the anisotropy slices and the helicity isosurface show evidence of complex domain structure. The anisotropy images in particular show evidence of dark blue domains with magnetisation vectors oriented along easy axes especially in the centre as well as possible closure domains developing near the surface.

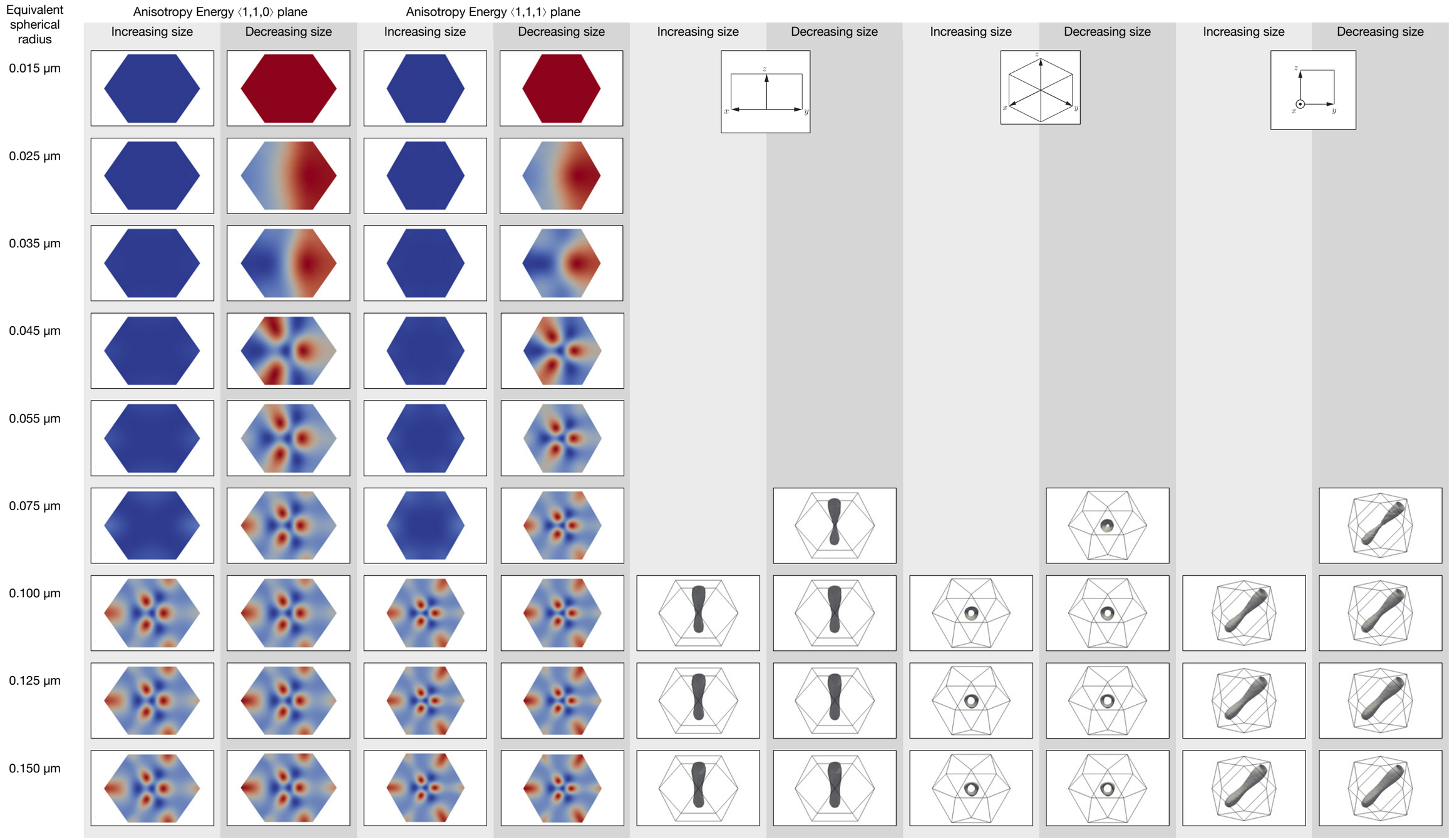
For cuboctahedra the transition to a multi-domain state is more gradual and the domain structure that is hinted at in the 600nm particle continues to develop and become more defined. This is particularly evident in the anisotropy images, where the gradual broadening out of the light blue spokes become tighter. Furthermore the development of closure domain-like structures become more distinct in the corners - particularly in the $\langle 1, 1, 1 \rangle$ anisotropy slice. It is likely that this domain structure will continue to become more and more refined as the size of the grain is increased. Unfortunately, due to the constraints of available computer resources and time, it was not possible to model larger grain sizes in this thesis.

However the transition from PSD to MD structure is clear. Since the 1350nm slice is the largest grain that was simulated, this is examined in further detail below.

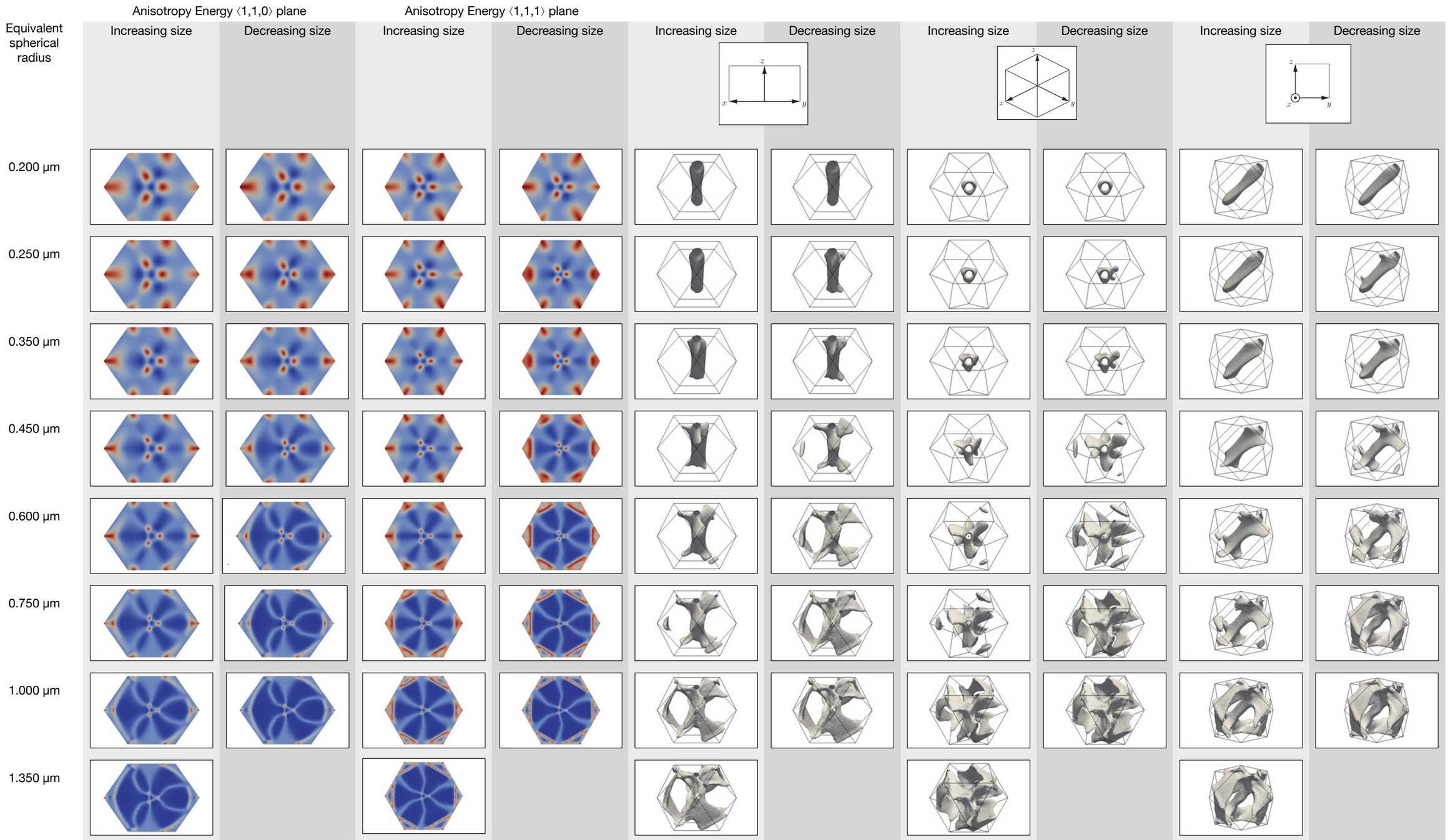
5.3.2 Size Hysteresis, Descending

As the grain size is reduced, there is evidence of that the complex structures found in larger grains persists for both the sphere and the cuboctahedra. In spheres, for example, the complex structure observed at 750nm persists until the size is reduced to 250nn at which point there is once more a sudden transition to a vortex state. The complex structure observed in the 1350nm cuboctahedron disappears more gradually until it becomes a vortex structure at 350nm, albeit still showing fins at the tube ends when viewing the isosurfaces.

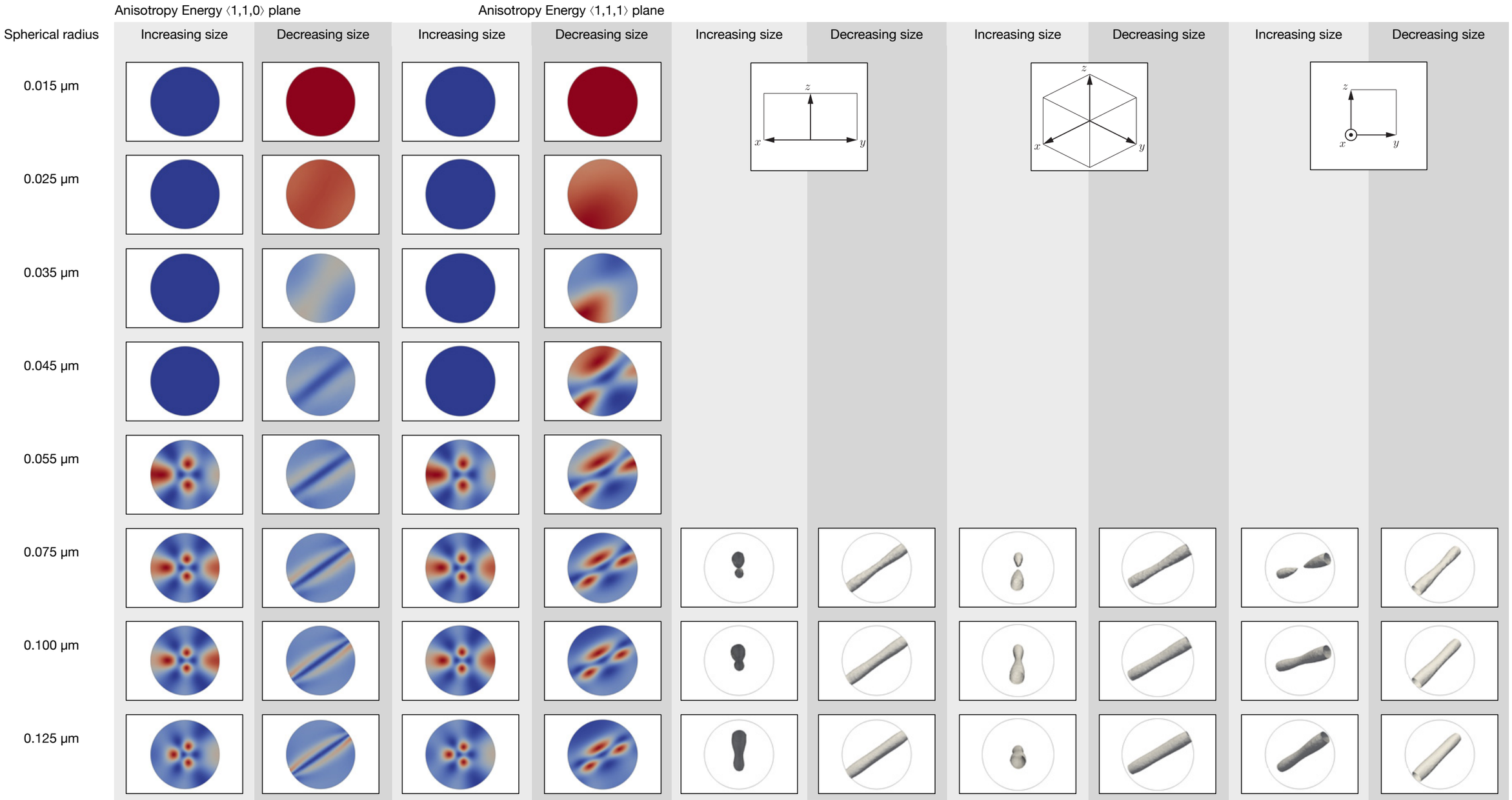
The vortex state finally collapses in to a flower state by about 35nm for both cuboctahedra and spheres (though this arguably happens for spheres at the 45nm point). The final state for both spheres and cuboctahedra being uniform along the $\langle 0, 0, 1\rangle$ direction for spheres and $\langle 0, 0, -1\rangle$ for cuboctahedra. This is clearly not a physically correct solution, since it is expected that the final domain state be uniformly magnetised and directed along one of the easy axis. However, this phenomenon could possibly be due to the energy minimisation method being used; since in order to become uniformly magnetised the vortex core must unwind in to a flower state which must then close up and then rotate. It is possible then that the solution is caught in some weakly metastable state and energy minimisation is unable to overcome some (possibly very small) energy barrier. The possibility of exact alignment along the hard axes was examined by taking an exact uniform initial field aligned in each of the hard directions for the 15nm grain. However in all cases, the energy minimiser evolved to a uniform solution along one of the easy axes as expected.


Equivalent spherical radius
Anisotropy Energy ⟨1,1,0⟩ plane
Anisotropy Energy ⟨1,1,1⟩ plane
Increasing size
Decreasing size
Increasing size
Decreasing size
Increasing size
Decreasing size
Increasing size
Decreasing size
Increasing size
Decreasing size
0.015 µm
0.025 µm
0.035 µm
0.045 µm
0.055 µm
0.075 µm
0.100 µm
0.125 µm
0.150 µm
z
x
y

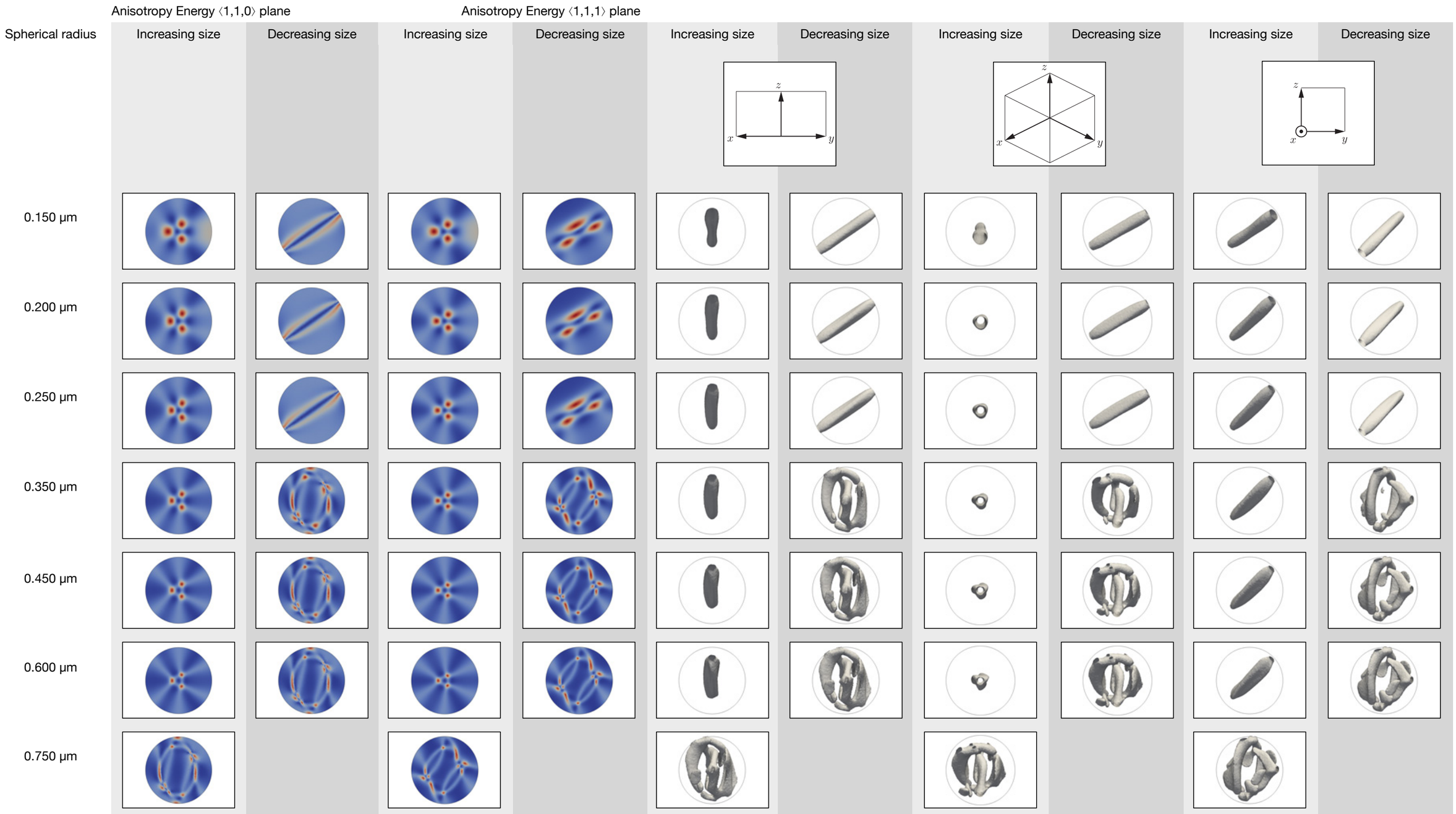
(a)


Anisotropy Energy ⟨1,1,0⟩ plane
Anisotropy Energy ⟨1,1,1⟩ plane
Equivalent spherical radius
Increasing size
Decreasing size
Increasing size
Decreasing size
Increasing size
Decreasing size
Increasing size
Decreasing size
Increasing size
Decreasing size
z
x
y
0.200 µm
0.250 µm
0.350 µm
0.450 µm
0.600 µm
0.750 µm
1.000 µm
1.350 µm

(b)


Anisotropy Energy ⟨1,1,0⟩ plane
Anisotropy Energy ⟨1,1,1⟩ plane
Spherical radius
Increasing size
Decreasing size
Increasing size
Decreasing size
Increasing size
Decreasing size
Increasing size
Decreasing size
Increasing size
Decreasing size
0.015 µm
0.025 µm
0.035 µm
0.045 µm
0.055 µm
0.075 µm
0.100 µm
0.125 µm
z
x
y

(c)


Anisotropy Energy ⟨1,1,0⟩ plane
Anisotropy Energy ⟨1,1,1⟩ plane
Spherical radius
Increasing size
Decreasing size
Increasing size
Decreasing size
Increasing size
Decreasing size
Increasing size
Decreasing size
Increasing size
Decreasing size
z
x
y
0.150 µm
0.200 µm
0.250 µm
0.350 µm
0.450 µm
0.600 µm
0.750 µm

(d)

Figure 5.7: Results for size hysteresis of cuboctahedra and spheres. Figures (a), (b), (c) and (d) present the results of calculations described above. The images are split in to two sets of two: (figs. a & b) are cuboctahedra, (figs. c & d) are spheres. All results are presented in tabular form with inreasing sizes in even number columns (light grey) and decreasing sizes in odd number columns (dark grey). The first two columns show anisotropy energy slices in the $\{1, 1, 0\}$ plane, the next two columns show anisotropy energy slices in the $\{1, 1, 1\}$ plane. The last six columns show three sets of helicity isosurfaces for three different perspectives depicted by the axes. Some solutions do not have an isosurface depicted, this is because the solution is in a uniform or flowering state or because the selected helicity value is not small enough to pick out the isosurface.

5.3.3 The 1350nm Grain

The images shown in figure 5.10 are large versions of the anisotropy slices for the 1350nm cuboctahedral grain shown in 5.7a and 5.7b. In order to examine these slices in more detail the angles between adjacent domains and the widths of domain walls is approximated. The structures that appear to be closure domains in figure 5.10b are not included as it is believed that those structures are not yet completely formed, as indicated by a light blue colouring in their centres.

It is thought that the domains that appear in figure 5.10 are Bloch type body domains rotating through an angle of 71.5°. Figure 5.8 shows the magnetisation along a line through a domain wall from three different perspectives for the $\{1, 1, 1\}$ plane. It can be seen that there is significant amount of rotation through the plane as would be expected in a Bloch wall.

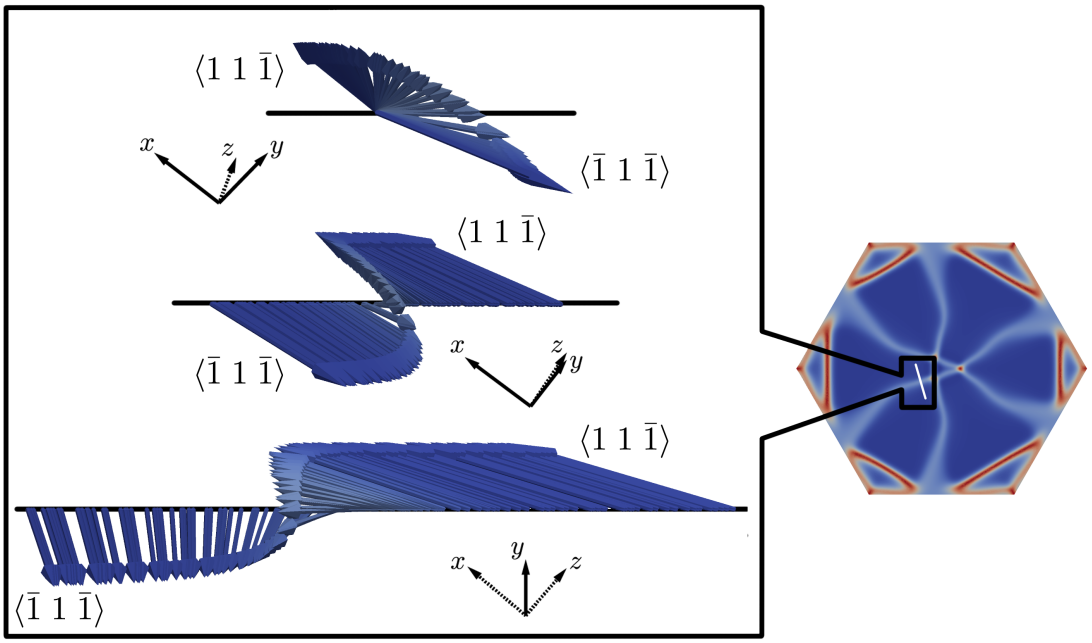


Figure 5.8: Block wall in the $\{1, 1, 1\}$ plane. The white line on the right indicates the region in which the magnetisation is sampled. The three inset images show the magnetisation along this line with respect to the plane (indicated by a black line) - in each of these three images, the plane is being viewed head on. It can be seen that there is significant rotation through the plane, along with some degree of rotation within the plane as the magnetisation swings from one easy axis to another.

In order to estimate wall widths, angle of rotation is taken along one of the white lines indicated in figure 5.10b. This results in the sigmoid graph illustrated in figure 5.9. The linear part of the graph corresponds to the region in which the magnetisation vectors are rotating the quickest and therefore define the width of the domain wall. By fitting a line along the linear region of the sigmoid and projecting to the maximum/minimum angles the wall width is estimated (Lilley, 1950; Hubert and Schäfer, 1998).

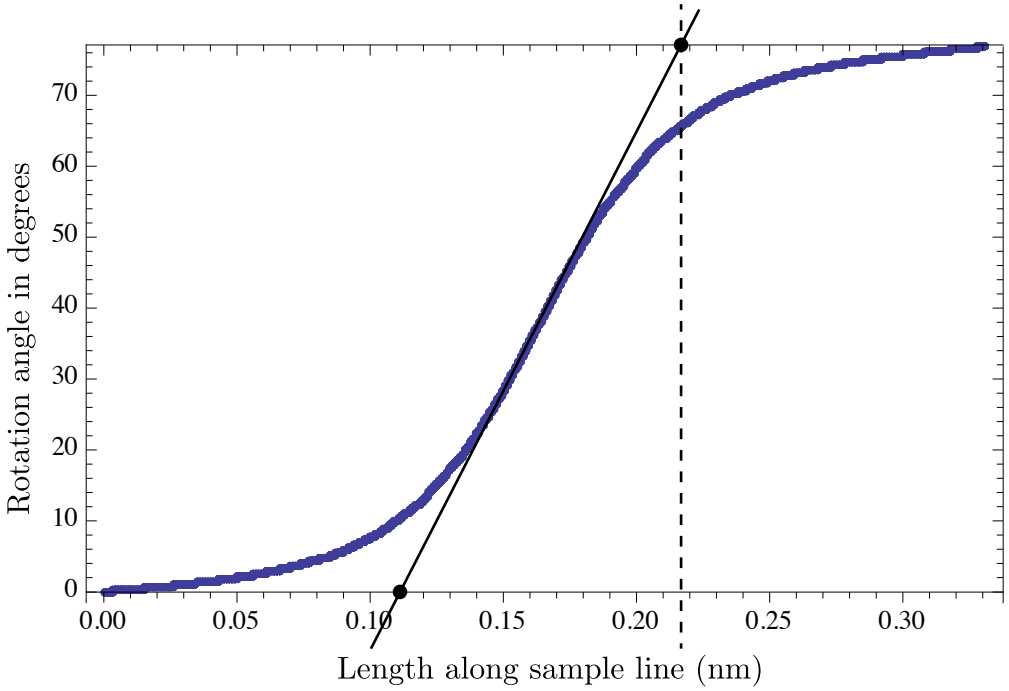
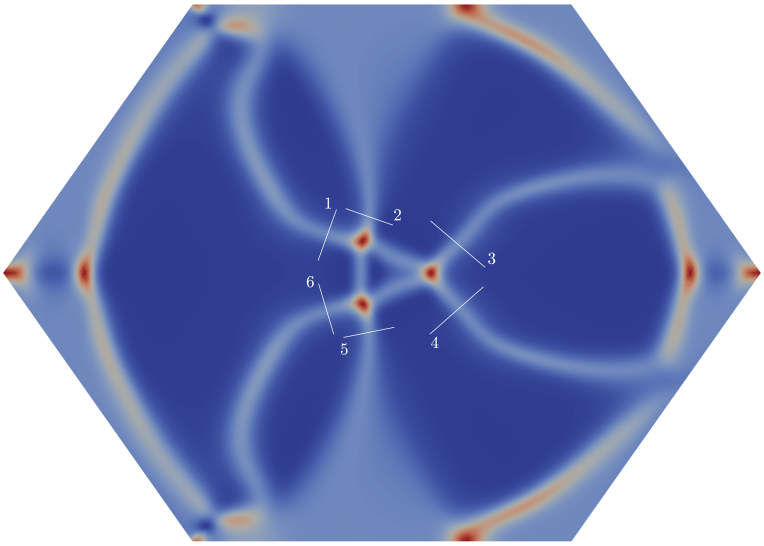
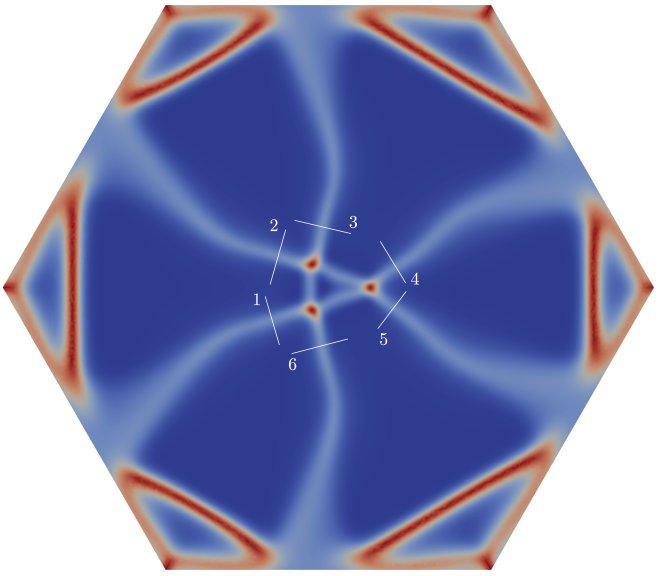


Figure 5.9: Estimating the wall width. The graph shows the anisotropy energy plotted along the 1 & 6 sample line of figure 5.10b. The arrows indicate where the end of the graph knees are taken, the upper bound is the subtracted from the lower to estimate the wall width.

Since it is expected that the magnetisation within domains is directed along one of the easy directions, the angles between domains should be one of 70.5°, 109.5° or 180° - since these are the only angles available between vectors directed along the diagonals of a cube. Table 5.5 below shows the angles between the domains indicated in (figs. 5.10a & 5.10b).



(a) Anisotropy energy in the $\{1, 1, 0\}$ plane.



(b) Anisotropy energy in the $\{1, 1, 1\}$ plane.

Figure 5.10: Anisotropy energy values for the the 1350nm equivalent spherical radius cuboctahedra in two planes. The model is believed to be entering the multidomain state and shows evidence of complex domain structure, with domains (in dark blue lying along the easy axes) separated by domain walls (lighter blue lines). The numbers in each figure denote the walls in (tbl. 5.6) and angle through which each domain rotate, shown in (tbl. 5.5).

Domains	$\langle 1,1,1 \rangle$		$\langle 1,1,0, \rangle$	
	Angle	Error (%)	Angle	Error (%)
1 & 2	69.3858	1.5805	66.9114	5.0902
2 & 3	69.5517	1.3451	72.0852	2.2485
3 & 4	68.6776	2.5850	67.6204	4.0846
4 & 5	69.3390	1.6468	66.9285	5.0660
5 & 6	69.4185	1.5340	72.1096	2.2831
1 & 6	69.4962	1.4239	67.4104	4.3824

Table 5.5: Estimated domain angles in the $\{1,1,0\}$ and $\{1,1,1\}$ planes across domain walls corresponding to the lines in figure 5.10. The error value is the relative error of the measured angle with respect to the expected angle of 70.5°.

As can be seen from the above table, all domains observed are 71.5° walls. The errors in domain wall angle in both $\{1,1,0\}$ and $\{1,1,1\}$ planes is small in both cases.

Dunlop (2001) gives estimates of the expected width of the domain walls according to

$$\delta_w = \pi \sin\left(\frac{\phi}{2}\right)\left(\frac{A}{K}\right)^{\frac{1}{2}}, \tag{5.5}$$

where δ_w is the domain wall width, A is the anisotropy constant, $K = 1.64 \times 10^3 \mathrm{Jm}^{-3}$ and ϕ is the angle through which the magnetisation of the domain wall rotate. Using this an estimate for the domain wall width is 0.1643 nm for a wall rotating through 71°. The table 5.6 summarises the wall widths illustrated in (figs. 5.10a & 5.10b). As can be seen there is some variance in the errors. This could be partly due to the fact that estimating domain width from figure 5.9 is prone to some small errors since identification of the linear region relies on manually fitting the line - this could be eliminated by automatically identifying the linear region of the sigmoid. Another factor could be that the domain structure observed is not yet fully developed since further increases in size could result in thinner, more distinct domain walls. This is certainly the trend observed in 5.7a and 5.7b. It should also be noted that the model in Dunlop (2001) makes

assumptions about how magnetisation varies across domain walls (*i.e.* it is a one dimensional model where angles θ rotate in a single plane with $\theta \in [0, 180]$) along with assuming that the contribution from the magneto-static interaction is negligible for the internal domain walls.

Wall no.	$\langle 1,1,1 \rangle$		$\langle 1,1,0, \rangle$	
	Length (nm)	Error (%)	Length(nm)	Error (%)
1	0.1296	21.1476	0.1224	25.5236
2	0.1282	21.9492	0.1219	25.7877
3	0.1258	23.4594	0.0867	47.2181
4	0.1308	20.3761	0.1588	3.3637
5	0.1338	18.5600	0.1310	20.2890
6	0.1149	30.0917	0.1025	37.6105

Table 5.6: Estimated domain widths in the $\{1,1,0\}$ and $\{1,1,1\}$ planes along the lines indicated in figure 5.10. The error value is the relative error of the measured wall width with respect to an expected wall width of 0.1643 nm.

5.4 Conclusions

The results that have been presented above are an important addition to the understanding of how multi-domain structures evolve. The picture that emerges is that complex domain structures develop from a combination of the fins in the vortex core along with boundary domains emerging from the surface. This is particularly striking when considering the cuboctahedral grain. When comparing the largest grain size (the 1350nm cuboctahedron) against theoretical results, there is a close correspondence with what is predicted by theory. Large body domains aligned along the easy axes are clearly present and what appear to be the start of closure domains are also seen. Comparing the angles between adjacent domains gives good correspondence between what is expected and estimates of domain wall widths seem to be on the same order of magnitude as calculated in Dunlop (2001).

It would be valuable to run models for even greater grain sizes, particularly

in the case of cuboctahedra, in order to see how domain structure develops. It is believed that the domain walls should tighten and become more defined and the closure domains should become more distinct, since this is the trend observed in 5.7a and 5.7b.

Bibliography

Trevor P Almeida, Takeshi Kasama, Adrian R Muxworthy, Wyn Williams, Lesleis Nagy, and Rafal E Dunin-Borkowski. Observing thermomagnetic stability of nonideal magnetite particles: Good paleomagnetic recorders? *Geophysical Research Letters*, 41(20):7041–7047, 2014a.

Trevor P Almeida, Takeshi Kasama, Adrian R Muxworthy, Wyn Williams, Lesleis Nagy, Thomas W Hansen, Paul D Brown, and Rafal E Dunin-Borkowski. Visualized effect of oxidation on magnetic recording fidelity in pseudo-single-domain magnetite particles. *Nature communications*, 5, 2014b.

Bob Alverson, Edwin Froese, Larry Kaplan, and Duncan Roweth. Cray xc series network. *Cray Inc., White Paper WP-Aries 01-1112*, 2012.

E Appel, V Hoffmann, and HC Soffel. Magneto-optical kerr effect in (titano) magnetite, pyrrhotite and hematite. *Physics of the Earth and Planetary Interiors*, 65(1):36–42, 1990.

ARCHER. *ARCHER Supercomputing Service*. URL `http://www.archer.ac.uk/`. Accessed: May 2015.

S Balay et al. *XDMF: extensible data model and format*, 2011. URL `http://www.xdmf.org`. Accessed: May 2015.

Blender community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 2014. URL `http://www.blender.org`.

X Brunotte, G Meunier, and J F Imhoff. Finite-Element Modeling of Unbounded Problems Using Transformations - a Rigorous, Powerful and Easy Solution. *Ieee Transactions on Magnetics*, 28(2):1663–1666, March 1992.

csimsoft. *Trelis software Version 15.0.3*, 2015. URL `http://www.csimsoft.com`. Accessed: May 2015.

David J. Dunlop. On the demagnetizing energy and demagnetizing factor of a multidomain ferromagnetic cube. *Geophysical Research Letters*, 10(1):79–82, 1983.

David J Dunlop and Song Xu. Theory of partial thermoremanent magnetization in multidomain grains: 1. repeated identical barriers to wall motion (single microcoercivity). *Journal of Geophysical Research: Solid Earth (1978–2012)*, 99(B5):9005–9023, 1994.

DJ Dunlop. Rock magnetism: fundamentals and frontiers, 2001.

K. Fabian, A. Kirchner, W. Williams, F. Heider, T. Leibl, and A. Hubert. Three-dimensional micromagnetic calculations for magnetite using FFT. *Geophysical Journal International*, 124(1):89–104, 1996.

D R Fredkin and T R Koehler. Hybrid Method for Computing Demagnetizing Fields. *IEEE Transactions on Magnetics*, 26(2):415–417, March 1990.

CE Geiß, F Heider, and HC Soffel. Magnetic domain observations on magnetite and titanomaghemite grains (0.5/˜ m-10 pm). *Geophys. J. Int*, 1995.

HDF. *HDF5 user's guide*. URL `http://www.hdfgroup.org/HDF5/doc/UG/`. Accessed: May 2015.

F Heider and W Williams. Note on temperature-dependence of exchange constant in magnetite. *Geophys. Res. Lett.*, 15(2):184–187, 1988.

Alex Hubert and Rudolf Schäfer. *Magnetic domains: the analysis of magnetic microstructures*. Springer Science & Business Media, 1998.

J F Imhoff, G Meunier, X Brunotte, and J C Sabonnadiere. An original solution for unbounded electromagnetic 2D- and 3D-problems throughout the finite element method. *Magnetics, IEEE Transactions on*, 26(5):1659–1661, 1990.

John Kim, Wiliam J Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. In *ACM SIGARCH Computer Architecture News*, volume 36, pages 77–88. IEEE Computer Society, 2008.

Kitware. Paraview, 2015. URL `http://www.paraview.org`. Accessed: May 2015.

BA Lilley. Energies and widths of domain boundaries in ferromagnets. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(319), 1950.

Ronald T Merrill. The demagnetization field of multidomain grains. *J. Geomag. Geoelectr*, 29:285–292, 1977.

R Pauthenet and L Bochirol. Aimantation spontanée des ferrites. *J. Phys. Radium*, 12(3):249–251, 1951.

W Williams, V Hoffmann, F Heider, T Göddenhenrich, and C Heiden. Magnetic force microscopy imaging of domain walls in magnetite. *Geophysical Journal International*, 111(3):417–423, 1992.

Wyn Williams and David J Dunlop. Three-dimensional micromagnetic modelling of ferromagnetic domain structure. 1989.

Wyn Williams, Adrian R Muxworthy, and Greig A Paterson. Configurational anisotropy in single-domain and pseudosingle-domain grains of magnetite. *Journal of Geophysical Research: Solid Earth (1978–2012)*, 111(B12), 2006.

Anne Witt, Karl Fabian, and Ulrich Bleil. Three-dimensional micromagnetic calculations for naturally shaped magnetite: octahedra and magnetosomes. *Earth And Planetary Science Letters*, 233(3):311–324, 2005.

TM Wright, W Williams, and DJ Dunlop. An improved algorithm for micromagnetics. *Journal of Geophysical Research: Solid Earth (1978–2012)*, 102(B6): 12085–12094, 1997.

Song Xu and David J Dunlop. Theory of partial thermoremanent magnetization in multidomain grains: 2. effect of microcoercivity distribution and comparison with experiment. *Journal of Geophysical Research: Solid Earth (1978–2012)*, 99(B5):9025–9033, 1994.

Chapter 6

Thesis Conclusions and Future Work

This thesis has presented a micromagnetics code that is able to utilise many core resources available on modern high performance computing (HPC) infrastructures. It demonstrates good scaling and has been shown to model large naturally occurring grains. In particular chapter 5 has presented a study of the evolution of domain structures in magnetite up to the micron scale using unconstrained micromagnetic modelling. This work begins to fill the gap in our knowledge with respect to what happens in the transition from pseudo-single domain to multi-domain structures, as well as looking at the stability of these structures with respect to grain size. It is a first step in attempting to understand whether large, geologically significant, grains can be good recorders of magnetic information.

6.1 Other Micromagnetics Codes

Typically past approaches such as that found in Nmag (Fischbacher et al., 2007) and Merrill (first referenced in Williams et al. (2006), with publication forthcoming) have made use of the boundary element method to calculate the stray field. This method is known to be computationally difficult due to the resulting dense matrix-vector problem. The approach taken in this thesis makes use of a spatial transformation method described in chapter 3 that is more amenable

to parallelisation, however this imposes restrictions on the mesh requiring additional regions for mapped and unmapped space. An alternative method is to use boundary elements along with hierarchical matrix compression (Bebendorf, 2008). This technique results in sparse matrix-vector computations for the calculation of the stray field, without additional meshing. As of writing there exist no parallel implementations for boundary element method with h-matrix compression, however the BEM++ project (Smigaj et al., 2012) is working towards this goal, along with integration in to the FEniCS finite element framework used for this thesis.

Other micromagnetic software includes OOMMF (Donahue and Porter, 1999). Which is a mature finite difference software that is designed to run on shared memory parallel machines. The MicroMagnum software (MicroMagnum, 2015) takes a similar approach to OOMMF, in that it is a finite difference software. However it is optimised to use graphical processing units (GPU) for parallel computation of the stray field. The main advantages of the finite difference/regular grid approach is that there is a gain in computational speed by using the fast Fourier transform. Unfortunately this also restricts the geometries of materials modelled to ones that are naturally cuboidal. This is generally not a problem in the material science domain (the primary focus of OOMMF and MicroMagnum). However for more complex geometries such as those found to occur naturally, it is undesirable to approximate complex geometries, such as the cuboctahedra or spheres presented in chapter 5, with regular blocks - unless the number of blocks is very large (resulting in high resolution approximations of general geometries). However this defeats the advantage of the regular grid method in the first place! The Micromag code presented in this thesis is optimised to run large models at scale, taking advantage of both inherent parallelism in the FEM and its ability to approximate arbitrary geometries accurately.

6.2 Future Work

The work presented here does not include thermal effects. It would be valuable to understand the stability of domain structures with respect to fluctuations due to heating magnetisation. Furthermore, understanding energy barriers between

domain states would be useful as this would give an indication of the energies requires to transition from one domain structure to another. This may be used to aid our understanding of the thermal stability of geological samples. A useful extension of the model would be to include the nudged elastic band (NEB) method. This method allows the calculation of minimum energy paths between local energy minima.

Understanding how the characteristics of particles behave with respect to oxidation is another possible extension of the code presented. It should be easy to subdivide the material region into sub regions and to assign material parameters to each of those sub-regions. The difficulty of this method lies in assigning parameters to mesh vertices that correspond to the boundary between oxidised and non-oxidised states. Though programatically this this is fairly simple to do and some preliminary work suggests that solutions are robust with respect to variances in the parameters used to estimate these boundary parameters (Ge et al., 2014).

Additional energy terms are straight forward to incorporate in the code presented in this thesis since the approach is modular with each energy term implemented as a class. What is required is an implementation of a python class with functionality to calculate the a field and energy for a given magnetisation. The complete interface required for additional energy term calculators is described in section 3.7. The `Model` class is then able to call the new energy calculator and execute the necessary functionality provided by the user. Estimating the computational cost of each field calculation is more difficult. A field calculator could have excellent parallelisation characteristics (such as the anisotropy field calculation described in chapter 3), or it could be difficult calculation such as the computation of the demagnetising field. However one interesting direction could be to interleave calculation of each effective field component, thus introducing multiple levels of parallelism in the code since each effective field component is independent of the other.

The code presented here is a useful addition to the computational methods and techniques already available to paleomagnetists and material scientists. It demonstrates that micromagnetic simulations are a feasible problem for HPC architectures and by running large models we can gain an understanding in to

the nature of magnetism and magnetic materials that we have never had before. Micromag allows us to answer some very basic, but important, questions "how do domains form?" "what are the hysteresis properties of large naturally occurring grains?" "do such grains carry a significant paleomagnetic signal?" "what are the hysteresis properties of large, geologically realistic assemblies of grains?". And, with a little imagination, the author is confident that it can answer much much more.

Bibliography

Mario Bebendorf. *Hierarchical Matrices*. Springer-Verlag Berlin Heidelberg, 2008.

Mike Donahue and Don Porter. Oommf users guide, version 1.0, 1999. Interagency Report NISTIR 6376.

Thomas Fischbacher, Matteo Franchin, Giuliano Bordignon, and Hans Fangohr. A systematic approach to multiphysics extensions of finite-element-based micromagnetic simulations: Nmag. *Magnetics, IEEE Transactions on*, 43(6): 2896–2898, 2007.

Kunpeng Ge, Wyn Williams, Qingsong Liu, and Yongjae Yu. Effects of the core-shell structure on the magnetic properties of partially oxidized magnetite grains: experimental and micromagnetic investigations. *Geochemistry, Geophysics, Geosystems*, 15(5):2021–2038, 2014.

MicroMagnum. *MicroMagnum*, 2015. URL `http://micromagnum.informatik.uni-hamburg.de`. Accessed: Sept 2015.

Wojciech Smigaj, S Arridge, T Betcke, Joel Phillips, and Martin Schweiger. Solving boundary integral problems with bem++. *ACM Transactions on Mathematical Software*, 2012.

Wyn Williams, Adrian R Muxworthy, and Greig A Paterson. Configurational anisotropy in single-domain and pseudosingle-domain grains of magnetite. *Journal of Geophysical Research: Solid Earth (1978–2012)*, 111(B12), 2006.

Appendix A

Glossary of Symbols

The following is a brief outline of the symbols used though out this thesis.

Symbol	**Description**
n	an integer (its role is usually context dependent).
$\mathbb{P}_n(S)$	the power set of S consisting of subsets of size n.
A	a (sparse) stiffness matrix resulting from the evaluation of some bilinear form.
$\boldsymbol{b}$	a right hand side (discretised) force vector.
$\boldsymbol{u}$	a vector of unknowns.
$\boldsymbol{x}$	a spatial point in $\mathbb{R}^2$ or $\mathbb{R}^3$.
Ω	a region of $\mathbb{R}^2$ or $\mathbb{R}^3$.
$\partial\Omega$	the boundary of Ω.
$\hat{n}$	the unit normal to the surface $\partial\Omega$.
$\frac{\partial\phi}{\partial\hat{n}}$	the directional derivative of the scalar field ϕ in the direction of $\hat{n}$.
$\phi(\boldsymbol{x})$	the unknown (scalar) function of the Poisson's equation.
$f(\boldsymbol{x})$	the right hand side (forcing) function of Poisson's equation.
$v(\boldsymbol{x})$	a test function defined on Ω.
$g_D(\boldsymbol{x})$	Dirichlet boundary condition value defined on $\boldsymbol{x} \in \partial\Omega$.
$g_N(\boldsymbol{x})$	Neumann boundary condition value defined on $\boldsymbol{x} \in \partial\Omega$.
$N_e(i)$	the neighbour elements of node index i (i.e. elements incident to node i).

n_i^j	the jth component of a mesh node at index i (for example n_2^x is the x component of the 2nd mesh node).
$\psi_{i,j,k}$	The shape function with mesh node coordinate indices i, j and k.
M_s	The saturation magnetisation constant.
K_1	The anisotropy constant.
A	The exchange constant (note that this is the same symbol used for the stiffness matrix above, however its use will be clear from context).
$\boldsymbol{G}_\mathrm{a}$	Anisotropy energy gradient.
$\boldsymbol{H}_\mathrm{a}$	Anisotropy field.
E_a	Anisotropy energy.
$\boldsymbol{G}_\mathrm{e}$	Exchange energy gradient.
$\boldsymbol{H}_\mathrm{e}$	Exchange field.
E_e	Exchange energy.
$\boldsymbol{G}_\mathrm{d}$	Demagnetising energy gradient.
$\boldsymbol{H}_\mathrm{d}$	Demagnetising field.
E_d	Demagnetising energy.
μ_0	Permeability of free space ($\mu_0 = 1.256637 \times 10^{-6}$ m Kg s^{-2} A^{-2})

In general matrices are written using upper case roman letters, lower case roman letters denoting individual elements. For example the matrix A consists of elements a_{ij} where i is the row index in to A and j is the column index in to A.

$$A = [a_{ij}]_{nm} \tag{A.1}$$

The n and m values denote the number of rows/columns of a matrix (though usually these are omitted).

Notationally, vectors may be thought of as matrices with a single row or column and elements have a single index.

$$\boldsymbol{b} = [b_i]_n \tag{A.2}$$

The n value denotes the number of components of the vector (though usually this is omitted).

Appendix B

Theorems and Results

Theorems and results used throughout this thesis are presented here without proof.

B.1 Vector Identities

For vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ and scalar α the following identities hold

$$\nabla \cdot (\alpha \boldsymbol{u}) \equiv \alpha \nabla \cdot \boldsymbol{u} + \boldsymbol{u} \nabla \alpha \cdot \tag{B.1}$$

B.2 Divergence Theorem

Given a vector field $\boldsymbol{u}$ defined on some region Ω with boundary $\partial\Omega$ the following holds

$$\int_{\Omega} \nabla \cdot \boldsymbol{u} \, dV = \int_{\partial\Omega} \boldsymbol{u} \cdot \hat{n} \, dS \tag{B.2}$$

where dV is the differential volume element and dS is the differential surface element.

B.3 Calculus of Variations

Consider the functional I given by

$$I\left[f_1(x_1,\cdots x_n),\cdots,f_m(x_1,\cdots,x_n)\right] = \int_\Omega \mathcal{L}(x_1,\cdots,x_n;f_1,\cdots f_m;f_{1,1},\cdots,f_{m,n};f_{1,11},f_{m,nn};\cdots;f_{1,1\cdots1}\cdots f_{m,n\cdots n})\,dV$$

where I is a functional depending on scalar value functions f_i with $1 \leq i \leq m$ and each function f_i in turn depends on the variables x_j with $1 \leq j \leq n$. Furthermore if

$$f_{i,\mu} = \frac{\partial f_i}{\partial x_\mu} \qquad f_{i,\mu_1\mu_2} = \frac{\partial^2 f_i}{\partial x_{\mu_1}\partial x_{\mu_2}} \qquad f_{i,\mu_1\mu_2\mu_3} = \frac{\partial^2 f_i}{\partial x_{\mu_1}\partial x_{\mu_2}\partial x_{\mu_3}} \qquad \cdots$$

then the functional derivative is given by

$$\frac{\delta\mathcal{L}}{\delta \boldsymbol{f}} = \frac{\partial\mathcal{L}}{\partial f_i} + \sum_{j=1}^{n}(-1)^j \frac{\partial^j}{\partial x_{\mu_1}\cdots\partial x_{\mu_j}}\left(\frac{\partial\mathcal{L}}{\partial f_{i,\mu_1\cdots\mu_n}}\right).$$

More specifically for a three component vector field, each component may be thought of as a function of three three spatial variables. Then assuming that the functional depends only up to first derivatives

$$I\left[m_1(x_1,x_2,x_3),m_2(x_1,x_2,x_3),m_3(x_1,x_2,x_3)\right] = \int_\Omega \mathcal{L}(x_1,x_2,x_3;m_1,m_2,m_3;m_{1,1},\cdots,m_{3,3})\,dV,$$

the functional derivative is given by

$$\begin{aligned}\frac{\delta\mathcal{L}}{\delta\boldsymbol{m}} &= \begin{pmatrix} \frac{\partial\mathcal{L}}{\partial m_1} - \left(\frac{\partial}{\partial x_1}\frac{\partial\mathcal{L}}{\partial m_{1,1}} + \frac{\partial}{\partial x_2}\frac{\partial\mathcal{L}}{\partial m_{1,2}} + \frac{\partial}{\partial x_3}\frac{\partial\mathcal{L}}{\partial m_{1,3}}\right) \\ \frac{\partial\mathcal{L}}{\partial m_2} - \left(\frac{\partial}{\partial x_1}\frac{\partial\mathcal{L}}{\partial m_{2,1}} + \frac{\partial}{\partial x_2}\frac{\partial\mathcal{L}}{\partial m_{2,2}} + \frac{\partial}{\partial x_3}\frac{\partial\mathcal{L}}{\partial m_{2,3}}\right) \\ \frac{\partial\mathcal{L}}{\partial m_3} - \left(\frac{\partial}{\partial x_1}\frac{\partial\mathcal{L}}{\partial m_{3,1}} + \frac{\partial}{\partial x_2}\frac{\partial\mathcal{L}}{\partial m_{3,2}} + \frac{\partial}{\partial x_3}\frac{\partial\mathcal{L}}{\partial m_{3,3}}\right) \end{pmatrix} \\ &= \begin{pmatrix} \frac{\partial\mathcal{L}}{\partial m_1} \\ \frac{\partial\mathcal{L}}{\partial m_2} \\ \frac{\partial\mathcal{L}}{\partial m_3} \end{pmatrix} - \nabla\cdot\begin{pmatrix} \frac{\partial\mathcal{L}}{\partial m_{1,1}} & \frac{\partial\mathcal{L}}{\partial m_{1,2}} & \frac{\partial\mathcal{L}}{\partial m_{1,3}} \\ \frac{\partial\mathcal{L}}{\partial m_{2,1}} & \frac{\partial\mathcal{L}}{\partial m_{2,2}} & \frac{\partial\mathcal{L}}{\partial m_{2,3}} \\ \frac{\partial\mathcal{L}}{\partial m_{3,1}} & \frac{\partial\mathcal{L}}{\partial m_{3,2}} & \frac{\partial\mathcal{L}}{\partial m_{3,3}} \end{pmatrix}.\end{aligned}$$

However the matrix part of the above expression is a derivative of the scalar $\mathcal{L}$ by a matrix and so may be written

$$\frac{\delta\mathcal{L}}{\delta\boldsymbol{m}} = \frac{\partial\mathcal{L}}{\partial\boldsymbol{m}} - \nabla\cdot\frac{\partial\mathcal{L}}{\partial A},$$

where

$$A = \begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{pmatrix} = \begin{pmatrix} \frac{\partial m_1}{\partial x_1} & \frac{\partial m_1}{\partial x_2} & \frac{\partial m_1}{\partial x_3} \\ \frac{\partial m_2}{\partial x_1} & \frac{\partial m_2}{\partial x_2} & \frac{\partial m_2}{\partial x_3} \\ \frac{\partial m_3}{\partial x_1} & \frac{\partial m_3}{\partial x_2} & \frac{\partial m_3}{\partial x_3} \end{pmatrix} = \nabla\boldsymbol{m}.$$

This means that the functional derivative is given by

$$\frac{\delta\mathcal{L}}{\delta\boldsymbol{m}} = \frac{\partial\mathcal{L}}{\partial\boldsymbol{m}} - \nabla\cdot\frac{\partial\mathcal{L}}{\partial\nabla\boldsymbol{m}},$$

Appendix C

Mathematica Code for Stiffness Matrix Assembly

What follows is a presentation of some Mathematica code to highlight aspects of the finite element method when using first order (linear) Lagrangian elements. In particular, the code here illustrates how to implement the most important aspect of the finite element method, namely discretisation of the Laplacian operator. It also illustrates how one can go about 'scaling up' the description presented in Chapter 2 to higher dimensions. Briefly the code presented here includes

- construction of facet functions,
- integration over general elements by mapping to the reference element,
- construction of the stiffness matrix

It should be noted that actual stiffness matrix construction is very inefficient since in this illustrative example dense matrices are used.

Construct a linear (1st order) continuous Galerkin interpolating function $\psi_{n_1,n_2,n_3,n_4}(x, y, z) \equiv \psi(x, y, z)$ for the tetrahedron defined by mesh vertices n_1, n_2, n_3 and n_4. Note: The value of the interpolation function is then defined to be:

$$\psi\left(n_1^x, n_1^y, n_1^z\right) = 1$$
$$\psi\left(n_2^x, n_2^y, n_2^z\right) = 0$$
$$\psi\left(n_3^x, n_3^y, n_3^z\right) = 0$$
$$\psi\left(n_4^x, n_4^y, n_4^z\right) = 0$$

In[87]:=
```
CGLinear[n1_, n2_, n3_, n4_] := Module[
  {Cx, Cy, Cz, Cψ, Cc},
  (* For the constant, don't knock out any rows. *)
  Cc = Det[{{n1[[1]], n2[[1]], n3[[1]], n4[[1]]},
            {n1[[2]], n2[[2]], n3[[2]], n4[[2]]},
            {n1[[3]], n2[[3]], n3[[3]], n4[[3]]},
            {1, 0, 0, 0}}];
  (* For x, knock out the 'x-row'. *)
  Cx = Det[{{1, 1, 1, 1},
            {n1[[2]], n2[[2]], n3[[2]], n4[[2]]},
            {n1[[3]], n2[[3]], n3[[3]], n4[[3]]},
            {1, 0, 0, 0}}];
  (* For y, knock out the 'y-row'. *)
  Cy = Det[{{n1[[1]], n2[[1]], n3[[1]], n4[[1]]},
            {1, 1, 1, 1},
            {n1[[3]], n2[[3]], n3[[3]], n4[[3]]},
            {1, 0, 0, 0}}];
  (* For z, knock out the 'z-row'. *)
  Cz = Det[{{n1[[1]], n2[[1]], n3[[1]], n4[[1]]},
            {n1[[2]], n2[[2]], n3[[2]], n4[[2]]},
            {1, 1, 1, 1},
            {1, 0, 0, 0}}];
  (* For ψ, knock out the 'ψ-row'. *)
  Cψ = Det[{{n1[[1]], n2[[1]], n3[[1]], n4[[1]]},
            {n1[[2]], n2[[2]], n3[[2]], n4[[2]]},
            {n1[[3]], n2[[3]], n3[[3]], n4[[3]]},
            {1, 1, 1, 1}}];
  Function[{x, y, z}, FCc/FCψ - FCx/FCψ x - FCy/FCψ y - FCz/FCψ z]
   /. {FCx -> Cx, FCy -> Cy, FCz -> Cz, FCψ -> Cψ, FCc -> Cc}
];
```

Example usage, the interpolating function over the reference tetrahedra evalues to 1 at the first vertex and zero at all others.

In[88]:=
```
ψ = CGLinear[{1, 0, 0}, {0, 1, 0}, {0, 0, 0}, {0, 0, 1}];
ψ[1, 0, 0]
ψ[0, 1, 0]
ψ[0, 0, 0]
ψ[0, 0, 1]
```

Out[89]= 1

Out[90]= 0

Out[91]= 0

Out[92]= 0

Integreate the function *fun* over the tetrahedra defined by *n1*, *n2*, *n3* and *n4*.

In[109]:=
```
IntegrateOverTetra[fun_, n1_, n2_, n3_, n4_] := Module[
  {A, M, x, y, z, JacM, DetJacM},

  (* Matrix for affine transformation to reference tetrahedra. *)
  A = {{n1[[1]] - n4[[1]], n2[[1]] - n4[[1]], n3[[1]] - n4[[1]]},
       {n1[[2]] - n4[[2]], n2[[2]] - n4[[2]], n3[[2]] - n4[[2]]},
       {n1[[3]] - n4[[3]], n2[[3]] - n4[[3]], n3[[3]] - n4[[3]]}};

  (*Affine transformation from reference tetrahedra. *)
  M[x_, y_, z_] = A.{x, y, z} + n4;

  (* Jacobian of transform from reference tetrahedra .*)
  JacM[x_, y_, z_] =
   {{D[M[x, y, z][[1]], x], D[M[x, y, z][[1]], y], D[M[x, y, z][[1]], z]},
    {D[M[x, y, z][[2]], x], D[M[x, y, z][[2]], y], D[M[x, y, z][[2]], z]},
    {D[M[x, y, z][[3]], x], D[M[x, y, z][[3]], y], D[M[x, y, z][[3]], z]}};

  (* Determinant of the Jacobian. *)
  DetJacM[x_, y_, z_] = Det[JacM[x, y, z]];

  (* Integrate function over reference tetrahedra. *)
  Integrate[fun[M[x, y, z][[1]], M[x, y, z][[2]], M[x, y, z][[3]]] *
    Abs[DetJacM[x, y, z]], {x, 0, 1}, {y, 0, 1 - x}, {z, 0, 1 - y - x}]

];
```

Example usage, the translated reference tetraheda has the same volume as the reference tetrahedra.

In[135]:=
```
fun = Function[{x, y, z}, 1];
IntegrateOverTetra[fun, {1, 1, 1}, {2, 1, 1}, {1, 2, 1}, {1, 1, 2}] ==
  1/6 Abs[Det[{{0, 0, 0, 1}, {1, 0, 0, 1}, {0, 1, 0, 1}, {0, 0, 1, 1}}]]
```

Out[136]=
```
True
```

Assemble the stiffness matrix/discretised Laplacian for an input mesh. VCL is a list of coordinates of vertices in the from

$$\text{VCL} = \begin{matrix} n_1^x & n_1^y & n_1^z \\ n_2^x & n_2^y & n_2^z \\ \vdots & \vdots & \vdots \\ n_{|N|}^x & n_{|N|}^y & n_{|N|}^z \end{matrix}$$

where as TIL is a list of 4-tuple indices in to VCL denoting the four vertices of a tetrahedron in the form

$$\text{TIL} = \begin{matrix} n_{e_1,1} & n_{e_1,2} & n_{e_1,3} & n_{e_1,4} \\ n_{e_2,1} & n_{e_2,2} & n_{e_2,3} & n_{e_2,4} \\ \vdots & \vdots & \vdots & \vdots \\ n_{e_{|E|},1} & n_{e_{|E|},2} & n_{e_{|E|},3} & n_{e_{|E|},4} \end{matrix}$$

```
AssembleStiffnessMatrix[VCL_, TIL_] := Module[
    {i, j,
     ψ1, ψ2, ψ3, ψ4,
     KStiff, A, NIDX,
     ψA11, ψA12, ψA13, ψA14,
     ψA21, ψA22, ψA23, ψA24,
     ψA31, ψA32, ψA33, ψA34,
     ψA41, ψA42, ψA43, ψA44,
     A11, A12, A13, A14,
     A21, A22, A23, A24,
     A31, A32, A33, A34,
     A41, A42, A43, A44,
     G1, G2, G3, G4},

    KStiff = Table[0, {i, Length[VCL]}, {j, Length[VCL]}];

    For[i = 1, i ≤ Length[TIL], i = i + 1,

     A = Table[0, {i, Length[VCL]}, {j, Length[VCL]}];

     (* Compute ψ functions for the element. *)
     NIDX = RotateLeft[TIL[[i]], 0];
     ψ1 = CGLinear[{VCL[[NIDX[[1]], 1]], VCL[[NIDX[[1]], 2]], VCL[[NIDX[[1]], 3]]},
       {VCL[[NIDX[[2]], 1]], VCL[[NIDX[[2]], 2]], VCL[[NIDX[[2]], 3]]},
       {VCL[[NIDX[[3]], 1]], VCL[[NIDX[[3]], 2]], VCL[[NIDX[[3]], 3]]},
       {VCL[[NIDX[[4]], 1]], VCL[[NIDX[[4]], 2]], VCL[[NIDX[[4]], 3]]}];

     NIDX = RotateLeft[TIL[[i]], 1];
     ψ2 = CGLinear[{VCL[[NIDX[[1]], 1]], VCL[[NIDX[[1]], 2]], VCL[[NIDX[[1]], 3]]},
       {VCL[[NIDX[[2]], 1]], VCL[[NIDX[[2]], 2]], VCL[[NIDX[[2]], 3]]},
       {VCL[[NIDX[[3]], 1]], VCL[[NIDX[[3]], 2]], VCL[[NIDX[[3]], 3]]},
       {VCL[[NIDX[[4]], 1]], VCL[[NIDX[[4]], 2]], VCL[[NIDX[[4]], 3]]}];

     NIDX = RotateLeft[TIL[[i]], 2];
     ψ3 = CGLinear[{VCL[[NIDX[[1]], 1]], VCL[[NIDX[[1]], 2]], VCL[[NIDX[[1]], 3]]},
       {VCL[[NIDX[[2]], 1]], VCL[[NIDX[[2]], 2]], VCL[[NIDX[[2]], 3]]},
       {VCL[[NIDX[[3]], 1]], VCL[[NIDX[[3]], 2]], VCL[[NIDX[[3]], 3]]},
       {VCL[[NIDX[[4]], 1]], VCL[[NIDX[[4]], 2]], VCL[[NIDX[[4]], 3]]}];

     NIDX = RotateLeft[TIL[[i]], 3];
     ψ4 = CGLinear[{VCL[[NIDX[[1]], 1]], VCL[[NIDX[[1]], 2]], VCL[[NIDX[[1]], 3]]},
       {VCL[[NIDX[[2]], 1]], VCL[[NIDX[[2]], 2]], VCL[[NIDX[[2]], 3]]},
       {VCL[[NIDX[[3]], 1]], VCL[[NIDX[[3]], 2]], VCL[[NIDX[[3]], 3]]},
       {VCL[[NIDX[[4]], 1]], VCL[[NIDX[[4]], 2]], VCL[[NIDX[[4]], 3]]}];

    (* Compute weak gradients of the ψ functions for each matrix entry. *)
     ψA11[x_, y_, z_] =
      D[ψ1[x, y, z], x] * D[ψ1[x, y, z], x] + D[ψ1[x, y, z], y] * D[ψ1[x, y, z], y] +
       D[ψ1[x, y, z], z] * D[ψ1[x, y, z], z];

     ψA12[x_, y_, z_] =
```

```
 D[ψ1[x, y, z], x] * D[ψ2[x, y, z], x] + D[ψ1[x, y, z], y] * D[ψ2[x, y, z], y] +
  D[ψ1[x, y, z], z] * D[ψ2[x, y, z], z];

ψA13[x_, y_, z_] =
 D[ψ1[x, y, z], x] * D[ψ3[x, y, z], x] + D[ψ1[x, y, z], y] * D[ψ3[x, y, z], y] +
  D[ψ1[x, y, z], z] * D[ψ3[x, y, z], z];

ψA14[x_, y_, z_] =
 D[ψ1[x, y, z], x] * D[ψ4[x, y, z], x] + D[ψ1[x, y, z], y] * D[ψ4[x, y, z], y] +
  D[ψ1[x, y, z], z] * D[ψ4[x, y, z], z];

ψA21[x_, y_, z_] =
 D[ψ2[x, y, z], x] * D[ψ1[x, y, z], x] + D[ψ2[x, y, z], y] * D[ψ1[x, y, z], y] +
  D[ψ2[x, y, z], z] * D[ψ1[x, y, z], z];

ψA22[x_, y_, z_] =
 D[ψ2[x, y, z], x] * D[ψ2[x, y, z], x] + D[ψ2[x, y, z], y] * D[ψ2[x, y, z], y] +
  D[ψ2[x, y, z], z] * D[ψ2[x, y, z], z];

ψA23[x_, y_, z_] =
 D[ψ2[x, y, z], x] * D[ψ3[x, y, z], x] + D[ψ2[x, y, z], y] * D[ψ3[x, y, z], y] +
  D[ψ2[x, y, z], z] * D[ψ3[x, y, z], z];

ψA24[x_, y_, z_] =
 D[ψ2[x, y, z], x] * D[ψ4[x, y, z], x] + D[ψ2[x, y, z], y] * D[ψ4[x, y, z], y] +
  D[ψ2[x, y, z], z] * D[ψ4[x, y, z], z];

ψA31[x_, y_, z_] =
 D[ψ3[x, y, z], x] * D[ψ1[x, y, z], x] + D[ψ3[x, y, z], y] * D[ψ1[x, y, z], y] +
  D[ψ3[x, y, z], z] * D[ψ1[x, y, z], z];

ψA32[x_, y_, z_] =
 D[ψ3[x, y, z], x] * D[ψ2[x, y, z], x] + D[ψ3[x, y, z], y] * D[ψ2[x, y, z], y] +
  D[ψ3[x, y, z], z] * D[ψ2[x, y, z], z];

ψA33[x_, y_, z_] =
 D[ψ3[x, y, z], x] * D[ψ3[x, y, z], x] + D[ψ3[x, y, z], y] * D[ψ3[x, y, z], y] +
  D[ψ3[x, y, z], z] * D[ψ3[x, y, z], z];

ψA34[x_, y_, z_] =
 D[ψ3[x, y, z], x] * D[ψ4[x, y, z], x] + D[ψ3[x, y, z], y] * D[ψ4[x, y, z], y] +
  D[ψ3[x, y, z], z] * D[ψ4[x, y, z], z];

ψA41[x_, y_, z_] =
 D[ψ4[x, y, z], x] * D[ψ1[x, y, z], x] + D[ψ4[x, y, z], y] * D[ψ1[x, y, z], y] +
  D[ψ4[x, y, z], z] * D[ψ1[x, y, z], z];

ψA42[x_, y_, z_] =
 D[ψ4[x, y, z], x] * D[ψ2[x, y, z], x] + D[ψ4[x, y, z], y] * D[ψ2[x, y, z], y] +
  D[ψ4[x, y, z], z] * D[ψ2[x, y, z], z];

ψA43[x_, y_, z_] =
 D[ψ4[x, y, z], x] * D[ψ3[x, y, z], x] + D[ψ4[x, y, z], y] * D[ψ3[x, y, z], y] +
  D[ψ4[x, y, z], z] * D[ψ3[x, y, z], z];

ψA44[x_, y_, z_] =
 D[ψ4[x, y, z], x] * D[ψ4[x, y, z], x] + D[ψ4[x, y, z], y] * D[ψ4[x, y, z], y] +
  D[ψ4[x, y, z], z] * D[ψ4[x, y, z], z];

(*Integrate the shape functions over the tetrahedra.*)
 A11 = IntegrateOverTetra[ψA11,
   {VCL[[TIL[[i, 1]], 1]], VCL[[TIL[[i, 1]], 2]], VCL[[TIL[[i, 1]], 3]]},
   {VCL[[TIL[[i, 2]], 1]], VCL[[TIL[[i, 2]], 2]], VCL[[TIL[[i, 2]], 3]]},
   {VCL[[TIL[[i, 3]], 1]], VCL[[TIL[[i, 3]], 2]], VCL[[TIL[[i, 3]], 3]]},
```

```
  {VCL[[TIL[[i, 4]], 1]], VCL[[TIL[[i, 4]], 2]], VCL[[TIL[[i, 4]], 3]]}];

A12 = IntegrateOverTetra[ψA12,
  {VCL[[TIL[[i, 1]], 1]], VCL[[TIL[[i, 1]], 2]], VCL[[TIL[[i, 1]], 3]]},
  {VCL[[TIL[[i, 2]], 1]], VCL[[TIL[[i, 2]], 2]], VCL[[TIL[[i, 2]], 3]]},
  {VCL[[TIL[[i, 3]], 1]], VCL[[TIL[[i, 3]], 2]], VCL[[TIL[[i, 3]], 3]]},
  {VCL[[TIL[[i, 4]], 1]], VCL[[TIL[[i, 4]], 2]], VCL[[TIL[[i, 4]], 3]]}];

A13 = IntegrateOverTetra[ψA13,
  {VCL[[TIL[[i, 1]], 1]], VCL[[TIL[[i, 1]], 2]], VCL[[TIL[[i, 1]], 3]]},
  {VCL[[TIL[[i, 2]], 1]], VCL[[TIL[[i, 2]], 2]], VCL[[TIL[[i, 2]], 3]]},
  {VCL[[TIL[[i, 3]], 1]], VCL[[TIL[[i, 3]], 2]], VCL[[TIL[[i, 3]], 3]]},
  {VCL[[TIL[[i, 4]], 1]], VCL[[TIL[[i, 4]], 2]], VCL[[TIL[[i, 4]], 3]]}];

A14 = IntegrateOverTetra[ψA14,
  {VCL[[TIL[[i, 1]], 1]], VCL[[TIL[[i, 1]], 2]], VCL[[TIL[[i, 1]], 3]]},
  {VCL[[TIL[[i, 2]], 1]], VCL[[TIL[[i, 2]], 2]], VCL[[TIL[[i, 2]], 3]]},
  {VCL[[TIL[[i, 3]], 1]], VCL[[TIL[[i, 3]], 2]], VCL[[TIL[[i, 3]], 3]]},
  {VCL[[TIL[[i, 4]], 1]], VCL[[TIL[[i, 4]], 2]], VCL[[TIL[[i, 4]], 3]]}];

A21 = IntegrateOverTetra[ψA21,
  {VCL[[TIL[[i, 1]], 1]], VCL[[TIL[[i, 1]], 2]], VCL[[TIL[[i, 1]], 3]]},
  {VCL[[TIL[[i, 2]], 1]], VCL[[TIL[[i, 2]], 2]], VCL[[TIL[[i, 2]], 3]]},
  {VCL[[TIL[[i, 3]], 1]], VCL[[TIL[[i, 3]], 2]], VCL[[TIL[[i, 3]], 3]]},
  {VCL[[TIL[[i, 4]], 1]], VCL[[TIL[[i, 4]], 2]], VCL[[TIL[[i, 4]], 3]]}];

A22 = IntegrateOverTetra[ψA22,
  {VCL[[TIL[[i, 1]], 1]], VCL[[TIL[[i, 1]], 2]], VCL[[TIL[[i, 1]], 3]]},
  {VCL[[TIL[[i, 2]], 1]], VCL[[TIL[[i, 2]], 2]], VCL[[TIL[[i, 2]], 3]]},
  {VCL[[TIL[[i, 3]], 1]], VCL[[TIL[[i, 3]], 2]], VCL[[TIL[[i, 3]], 3]]},
  {VCL[[TIL[[i, 4]], 1]], VCL[[TIL[[i, 4]], 2]], VCL[[TIL[[i, 4]], 3]]}];

A23 = IntegrateOverTetra[ψA23,
  {VCL[[TIL[[i, 1]], 1]], VCL[[TIL[[i, 1]], 2]], VCL[[TIL[[i, 1]], 3]]},
  {VCL[[TIL[[i, 2]], 1]], VCL[[TIL[[i, 2]], 2]], VCL[[TIL[[i, 2]], 3]]},
  {VCL[[TIL[[i, 3]], 1]], VCL[[TIL[[i, 3]], 2]], VCL[[TIL[[i, 3]], 3]]},
  {VCL[[TIL[[i, 4]], 1]], VCL[[TIL[[i, 4]], 2]], VCL[[TIL[[i, 4]], 3]]}];

A24 = IntegrateOverTetra[ψA24,
  {VCL[[TIL[[i, 1]], 1]], VCL[[TIL[[i, 1]], 2]], VCL[[TIL[[i, 1]], 3]]},
  {VCL[[TIL[[i, 2]], 1]], VCL[[TIL[[i, 2]], 2]], VCL[[TIL[[i, 2]], 3]]},
  {VCL[[TIL[[i, 3]], 1]], VCL[[TIL[[i, 3]], 2]], VCL[[TIL[[i, 3]], 3]]},
  {VCL[[TIL[[i, 4]], 1]], VCL[[TIL[[i, 4]], 2]], VCL[[TIL[[i, 4]], 3]]}];

A31 = IntegrateOverTetra[ψA31,
  {VCL[[TIL[[i, 1]], 1]], VCL[[TIL[[i, 1]], 2]], VCL[[TIL[[i, 1]], 3]]},
  {VCL[[TIL[[i, 2]], 1]], VCL[[TIL[[i, 2]], 2]], VCL[[TIL[[i, 2]], 3]]},
  {VCL[[TIL[[i, 3]], 1]], VCL[[TIL[[i, 3]], 2]], VCL[[TIL[[i, 3]], 3]]},
  {VCL[[TIL[[i, 4]], 1]], VCL[[TIL[[i, 4]], 2]], VCL[[TIL[[i, 4]], 3]]}];

A32 = IntegrateOverTetra[ψA32,
  {VCL[[TIL[[i, 1]], 1]], VCL[[TIL[[i, 1]], 2]], VCL[[TIL[[i, 1]], 3]]},
  {VCL[[TIL[[i, 2]], 1]], VCL[[TIL[[i, 2]], 2]], VCL[[TIL[[i, 2]], 3]]},
  {VCL[[TIL[[i, 3]], 1]], VCL[[TIL[[i, 3]], 2]], VCL[[TIL[[i, 3]], 3]]},
  {VCL[[TIL[[i, 4]], 1]], VCL[[TIL[[i, 4]], 2]], VCL[[TIL[[i, 4]], 3]]}];

A33 = IntegrateOverTetra[ψA33,
  {VCL[[TIL[[i, 1]], 1]], VCL[[TIL[[i, 1]], 2]], VCL[[TIL[[i, 1]], 3]]},
  {VCL[[TIL[[i, 2]], 1]], VCL[[TIL[[i, 2]], 2]], VCL[[TIL[[i, 2]], 3]]},
  {VCL[[TIL[[i, 3]], 1]], VCL[[TIL[[i, 3]], 2]], VCL[[TIL[[i, 3]], 3]]},
  {VCL[[TIL[[i, 4]], 1]], VCL[[TIL[[i, 4]], 2]], VCL[[TIL[[i, 4]], 3]]}];

A34 = IntegrateOverTetra[ψA34,
  {VCL[[TIL[[i, 1]], 1]], VCL[[TIL[[i, 1]], 2]], VCL[[TIL[[i, 1]], 3]]},
```

```
    {VCL[[TIL[[i, 2]], 1]], VCL[[TIL[[i, 2]], 2]], VCL[[TIL[[i, 2]], 3]]},
    {VCL[[TIL[[i, 3]], 1]], VCL[[TIL[[i, 3]], 2]], VCL[[TIL[[i, 3]], 3]]},
    {VCL[[TIL[[i, 4]], 1]], VCL[[TIL[[i, 4]], 2]], VCL[[TIL[[i, 4]], 3]]}];

 A41 = IntegrateOverTetra[ψA41,
    {VCL[[TIL[[i, 1]], 1]], VCL[[TIL[[i, 1]], 2]], VCL[[TIL[[i, 1]], 3]]},
    {VCL[[TIL[[i, 2]], 1]], VCL[[TIL[[i, 2]], 2]], VCL[[TIL[[i, 2]], 3]]},
    {VCL[[TIL[[i, 3]], 1]], VCL[[TIL[[i, 3]], 2]], VCL[[TIL[[i, 3]], 3]]},
    {VCL[[TIL[[i, 4]], 1]], VCL[[TIL[[i, 4]], 2]], VCL[[TIL[[i, 4]], 3]]}];

 A42 = IntegrateOverTetra[ψA42,
    {VCL[[TIL[[i, 1]], 1]], VCL[[TIL[[i, 1]], 2]], VCL[[TIL[[i, 1]], 3]]},
    {VCL[[TIL[[i, 2]], 1]], VCL[[TIL[[i, 2]], 2]], VCL[[TIL[[i, 2]], 3]]},
    {VCL[[TIL[[i, 3]], 1]], VCL[[TIL[[i, 3]], 2]], VCL[[TIL[[i, 3]], 3]]},
    {VCL[[TIL[[i, 4]], 1]], VCL[[TIL[[i, 4]], 2]], VCL[[TIL[[i, 4]], 3]]}];

 A43 = IntegrateOverTetra[ψA43,
    {VCL[[TIL[[i, 1]], 1]], VCL[[TIL[[i, 1]], 2]], VCL[[TIL[[i, 1]], 3]]},
    {VCL[[TIL[[i, 2]], 1]], VCL[[TIL[[i, 2]], 2]], VCL[[TIL[[i, 2]], 3]]},
    {VCL[[TIL[[i, 3]], 1]], VCL[[TIL[[i, 3]], 2]], VCL[[TIL[[i, 3]], 3]]},
    {VCL[[TIL[[i, 4]], 1]], VCL[[TIL[[i, 4]], 2]], VCL[[TIL[[i, 4]], 3]]}];

 A44 = IntegrateOverTetra[ψA44,
    {VCL[[TIL[[i, 1]], 1]], VCL[[TIL[[i, 1]], 2]], VCL[[TIL[[i, 1]], 3]]},
    {VCL[[TIL[[i, 2]], 1]], VCL[[TIL[[i, 2]], 2]], VCL[[TIL[[i, 2]], 3]]},
    {VCL[[TIL[[i, 3]], 1]], VCL[[TIL[[i, 3]], 2]], VCL[[TIL[[i, 3]], 3]]},
    {VCL[[TIL[[i, 4]], 1]], VCL[[TIL[[i, 4]], 2]], VCL[[TIL[[i, 4]], 3]]}];

 (*Populate local stiffness matrix.*)
 G1 = TIL[[i, 1]];
 G2 = TIL[[i, 2]];
 G3 = TIL[[i, 3]];
 G4 = TIL[[i, 4]];

 A[[G1, G1]] = A11;
 A[[G1, G2]] = A12;
 A[[G1, G3]] = A13;
 A[[G1, G4]] = A14;
 A[[G2, G1]] = A21;
 A[[G2, G2]] = A22;
 A[[G2, G3]] = A23;
 A[[G2, G4]] = A24;
 A[[G3, G1]] = A31;
 A[[G3, G2]] = A32;
 A[[G3, G3]] = A33;
 A[[G3, G4]] = A34;
 A[[G4, G1]] = A41;
 A[[G4, G2]] = A42;
 A[[G4, G3]] = A43;
 A[[G4, G4]] = A44;
 KStiff = KStiff + A;
];

 (*Return stffness matrix (in dense format).*)
 KStiff
];
```

Printed by Wolfram Mathematica Student Edition