Imperial College London

Department of Earth Science and Engineering

MSc in Applied Computational Science and Engineering

Independent Research Project
Project Plan

Monte Carlo simulator for the exploration of magnetic nanostructures

by

Yuhang Zhang

Email: yuhang.zhang121@imperial.ac.uk
GitHub username: acse-yz11721
Repository: https://github.com/ese-msc-2021/irp-yz11721

Supervisors:

Dr. Marijan Beg
Dr. Samuel Holt

September 2022

Acknowledge

I would like to express my deepest appreciation to Dr. Marijan Beg, who offered me many suggestions and guided me through the personal and group meeting sessions. I am also thankful to my colleague Haoyu Wu for sharing his thoughts with me. Lastly, I want to thank my mother for encouraging and supporting me during the project.

Github **repo**: `https://github.com/ese-msc-2021/irp-yz11721/blob/main/reports/yz11721-final-report.pdf`

Abstract

The computational model of magnetic materials is now an indispensable tool in understanding the behaviour of magnetic nanomaterials. It enables scientists to study the nanomagnetic systems in unprecedented detail, assist in explaining and designing experiments, etc. Recent research on magnetic skyrmion has attracted attention and broadened the prospects for applying next-generation storage devices. Although scientists have studied the stability of magnetic skyrmions on the model using dynamic micromagnetic methods, this method does not know how to simulate temperature. This project proposes a Python library that allows users to model magnetic materials using a micromagnetic model. In addition, the model is able to accurately predict the micromagnetic model's lowest energy state at various temperatures.

Keywords— micromagnetics, finite differences, nanomaterials, Monte Carlo, skyrmions.

1 Introduction and Objectives

The simulation of magnetic materials has a long history. Although atomistic models of magnetic materials were first developed in 1925 to investigate the phase transition problem in a ferromagnet [1], it is still extensively used in studying the processes governing the complex behaviour of magnetic nanomaterials. Heisenberg model is one instance of them, in which we equal the magnetic material with a lattice of atomistic spins. Each spin processes a local magnetic moment, which can be seen as a unit vector in $\mathbb{R}^3$. The atomistic model can directly interact with the micromagnetic model; for example, we need to create a more extensive scale micromagnetic model. The atomistic model allows us to calculate material properties atomistically. After that, we can use the micromagnetic model more efficiently [2].

While it is not trivial to simulate a system containing thousands of spins for atomistic models [3]. Increasing the size of the system leads to more significant CPU and memory consumption. To overcome this, today micromagnetic model is extensively used, in which we assume that atomic spins are uniformly distributed in an infinitesimal volume. Therefore, the magnetisation field can be discretised into a mesh of magnetic moments, and the whole magnetic material is approximate a continuum [2].

Next, the dynamic spin model is state-of-the-art in computational magnetism. For example, developing dynamic spin models governed by Landau-Lifshitz-Gilbert (LLG) equation enables us to study the time domain behaviour of magnetic materials or reversal processes for a system [3].

Recent research on magnetic skyrmion proves it has the most potential for next-generation storage devices. While understanding the skyrmions' thermal and magnetic stability is still one of the challenges [4]. The dynamic spin model is extensively used to obtain dynamic information about the magnetic properties of a system. However, the dynamic micromagnetic method may not be feasible to "relax" the system to find its lowest energy states above absolute zero. Moreover, researchers currently have no easy-to-use and related open-source packages.

The objective of this study is to develop a Python library which allows users to simulate magnetic nanomaterials using micromagnetic models. The Metropolis Monte Carlo algorithm is used to predict the lowest energy state of micromagnetic models with high accuracy and simulate the temperature effects. The users are required to provide geometry information and material parameters. After that, the software is able to perform energy computation, find the system's lowest energy state and visualise the system's evolution by creating a gif.

1.1 Micromagnetic model

In this work, the numerical implementation of a micromagnetic model is used. The micromagnetic model aims to determine the spatial distribution of the magnetisation $\mathbf{M}(\mathbf{r}, t)$ at a given temperature. Using the finite-differences method, we discretise the material as a continuous vector field $\mathbf{M}$, called magnetisation, where the fundamental components of the continuous vector field are atomic magnetic moments in a cubic lattice [3], as shown in Fig. 1.

At zero temperature ($T = 0$), the magnitude of the magnetisation $M_s = |\mathbf{M}|$, called saturation magnetisation, is isotropic, meaning that is a constant everywhere inside the continuous vector field. In addition to this, magnetisation $\mathbf{M}(\mathbf{r}, t)$ is a function of both space $\mathbf{r}$ and time t. The normalised magnetisation $\mathbf{m}(\mathbf{r}, t)$ is more commonly used to describe the magnetisation. It is written as:

$$\mathbf{m}(\mathbf{r}, t) = \frac{\mathbf{M}(\mathbf{r}, t)}{M_s} \tag{1}$$

In which, $\mathbf{r} = (x, y, z)$ represents the position of a single atomic magnetic moment, $\mathbf{m}(\mathbf{r}) = (m_x, m_y, m_z)$ and $|\mathbf{m}| = 1$.

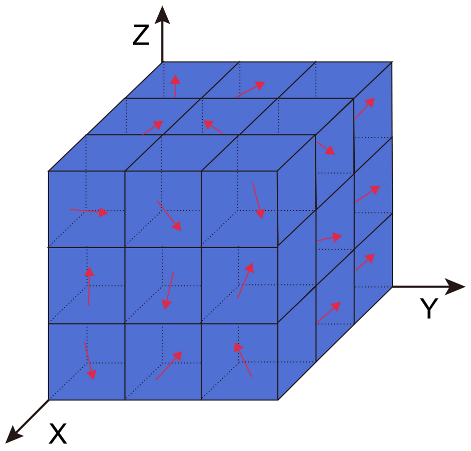


Figure 1: Discretised magnetisation field in the three spatial dimensions, the red arrow in each discretisation cell represents a single atomic magnetic moment $\mathbf{M}(\mathbf{r}_i, t_i)$ at at time t_i.

1.2 Interactions

Exchange energy The exchange energy is treated as the dominant term in the continuous model, which is produced due to the symmetry of the electron wavefunction [5]. Exchange interaction describes that at sufficiently low temperatures, individual neighbouring spins' magnetic moments are parallel or antiparallel to each other in magnetic materials. When the neighbouring spins are aligned parallel, the material is considered as ferromagnets. On top of that, if neighbouring spins are aligned parallel antiparallel, the material is considered as antiferromagnets [6]. The exchange energy of the continuous model is performed over the volume of the sample. The exchange energy is defined with the equation:

$$E_{\text{ex}} = \int_V A\mathbf{m} \cdot \nabla^2 \mathbf{m} \, \mathrm{d}V \tag{2}$$

In which, A is the exchange parameter in J m^{-1}, meaning that the exchan g eenergy of eachspin is not related to its spatial location in the model. Instead, it depends on the relative positions of the spin and its nearest-neighbour spins.

Dzyaloshinskii-Moriya interaction energy Dzyaloshinski discovered that the chiral interactions might occur for some materials with an antisymmetric exchange [7]. Then, Moriya developed an extension of superexchange theory that enable us to understand chiral interactions [8]. In recognition of their remarkable contributions, we named this interaction Dzyaloshinski–Moriya Interaction (DMI). There are two types of DMI: bulk DMI and interfacial DMI. We used the bulk DMI, which is defined as:

$$E_{\text{dmi}} = \int_V D\mathbf{m} \cdot (\nabla \times \mathbf{m}) \mathrm{d}V \tag{3}$$

In which, D represents the DMI constant in J m^{-2}.

Anitrosopy energy In this work, we use the simplest form of anisotropy energy called Uniaxial anitrosopy energy. Regarding some materials in real, atomic magnetic moments in them have a preference in one particular direction, $\mathbf{u}$, which is often referred to as the easy axis. The uniaxial anisotropy energy of the micromagnetics can be written in the form,

$$E_{\text{anis}} = \int_V K_u \left[1 - (\mathbf{m} \cdot u)^2\right] \mathrm{d}V \tag{4}$$

In which, K_u is the uniaxial anisotropy constant in $\mathrm{J\ m^{-3}}$.

Zeeman energy The Zeeman energy describes the interaction between the magnetisation $\mathbf{M}$ and external magnetic fields $\mathbf{H}$. It is written as:

$$E_{\text{zeeman}} = \int_V -\mu_0 M_s \mathbf{H} \cdot \mathbf{m} \mathrm{d}V \tag{5}$$

1.3 Monte-Carlo Algorithm

One use of the Metropolis Monte-Carlo (MMC) algorithm is to simulate different magnetic models such as the Ising model [9], Heisenberg model [10], etc. In our case, the continuous micromagnetic model, we equate a magnetic material with a cubic lattice of particle spins, where the spins are uniformly distributed in a lattice. $\boldsymbol{r} = (x, y, z)$ represents the position of each particle spin.

MMC algorithm is a Markov chain Monte Carlo method. At each iteration, we choose a candidate spin and rotate it. Following this, the algorithm has an acceptance ratio which depends on transition probability w_{ij}. It can be written as [10]:

$$w_{ij} = \begin{cases} \exp(-\Delta E/k_b T) & \text{when } \Delta E > 0 \\ 1 & \text{when } \Delta E \leq 0 \end{cases} \tag{6}$$

In which, the energy difference of the spin system $\Delta E = E_{\text{new}} - E_{\text{old}}$, k_b represents the Boltzmann constant and T represents the Kelvin temperature. A simple example of the use of this method is shown in Algorithm 1.

Algorithm 1 colour black Monte Carlo method.

```
1:  Initialise the system.
2:  Choose the simulation times $N$.
3:  Choose $S_m$ as an initial position.
4:  for $i = 1$ to $N$ do
5:      Produce a new position $S_w$ using random method.
6:      if the energy difference$\Delta E <= 0$ then
7:          $S_{m+1} = S_w$
8:      else
9:          Let $r$ be a uniform random variable on (0, 1)
10:         if $\exp(-\Delta E/k_b T) >= r$ then
11:             $S_{m+1} = S_w$
12:         else
13:             $S_{m+1} = S_m$
14:         end if
15:     end if
16:     $S_m = S_{m+1}$
17: end for
```

1.4 Spin update strategy

The MMC requires iteratively generating massive random samples to find the lowest energy state of the micromagnetic model [10]. Thus, an effective method to update the spin is critial for the success of Monte Carlo simulation. We found that the process of updating spin is quite similar to a random walk on the sphere [11]. On top of that, the new spin state from a given state has to include all the spin orientations. Following this, this work tried two spin update methods in both the Sphere and Cartesian coordinate systems.

Spherical coordinates This method ensures a point (x, y, z) is on the sphere for any (θ, ϕ). In spherical coordinates, r is the radius, θ represents the azimuthal angle over the range $[0, 2\pi]$, ϕ represents the polar angle over the range$[0, \pi]$.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \sin(\theta) \cdot \cos(\varphi) \\ \sin(\theta) \cdot \sin(\varphi) \\ \cos(\theta) \end{bmatrix} \tag{7}$$

A commonly misuse of spherical coordinate is to generate uniformly distributed numbers u, v (both uniform on the interval $[0, 1]$), then let $\theta = 2\pi u$ and $\phi = \pi$v to generate a uniform distribution of θ and ϕ. This incorrect strategy will cause points to cluster around the poles and sparse around the equator. Therefore, to distribute points uniformly according to the surface shape of a sphere, we consider assigning $\phi = \cos^{-1}(2v - 1)$ and $\theta = 2\pi u$.

Cartesian coordinates Using Cartesian coordinates, we first assigning $\mu_0 = (x, y, z)$ as a given state, then generating $\mu = (x_0, x_1, x_2)$ from a uniform distribution on $(-1, 1)$ and norm $= \sqrt{x_0^2 + x_1^2 + x_2^2} = 1$. The equation to update the spin is written as:

$$\mu_1 = \mu_\Delta \cdot \varepsilon + \mu_0 \tag{8}$$

In which ε represents the step width. ε is equal to 0.1 in this work. The trial spin μ_1 must then be normalised.

The program has been tested both in a version which uses polar coordinates for which the spin state was stored like $S_i = (\theta, \phi)$ and in one using Cartesian coordinates $S_i = (x, y, z)$. The result indicates that the polar coordinates version has better performance (up to 5% faster execution) and thus has been used for most simulations.

2 Code Metadata

2.1 Environment for Development

This project was written in Python 3.9 on a Mac computer. The package **NumPy** [12] was used for producing simulations on the continuous vector field. **Ubermag** [13], an existing micromagnetic python package, is used to validate the micromagnetic model in this work. In addition, **imageio** and **base64** are used to visualise the micromagnetic model's evolutionary process. **json** and **os** were used to access the normalised vector field $\mathbf{m}$. The project was pushed to GitHub after completing it. A file called README.md contains detailed instructions on how to run and test the whole program. After that, the jupyter notebooks were used to familiarise the user with several basic usages of this python library.

2.2 Module Overview

The main functions are inside the continious_model folder, while the detailed directory tree of the GitHub repository is shown as follows:

```
root
├── continuous_model
│   ├── __init__.py
│   ├── energy.py
│   ├── field.py
│   ├── mesh.py
│   ├── plot_model.py
│   ├── simulator.py
│   └── tools.py
├── iterations
├── normalised_m
├── pics
├── plots
└── tests
```

```
│
│   ├── test_energy.py
│   └── test_simulator.py
├── save_plot.ipynb
├── tutorial.ipynb
└── visualization.ipynb
```

The continuous_model folder contains all the required modules to create a micromagnetic model and run the Monte Carlo simulation. normalised_m lists several initial normalised unit vectors $\mathbf{m}$. iterations folder contains normalised unit vector $\mathbf{m}$ at different Monte Carlo different steps. After that, save_plot.ipynb is employed to plot the magnetisation vector fields and save them to VTK files in the plots folder. Following this, there is visualisation.ipynb to visualise the magnetisation vector field, and a gif is saved in the pics folder.

3 Methodology

3.1 Simulation process

In our Monte Carlo simulation, We first establish a three-dimensional continuous micromagnetic model to represent a piece of magnetic material. After that, we follow a flow chart in Fig. 2 to find the lowest energy state of the micromagnetic model. There are several essential properties of energy terms, such as effective field $\mathbf{H}_{\text{eff}}$ or energy density $w(\mathrm{m})$ needs to be determined during the Monte Carlo simulation. The equation to calculate $\mathbf{H}_{\text{eff}}$ was written as:

$$\mathbf{H}_{\text{eff}} = -\frac{1}{\mu_0 M_s}\frac{\delta E[\mathbf{m}]}{\delta \mathbf{m}} \tag{9}$$

In which $E[\mathbf{m}]$ represents the total energy functional of the system, and it is defined as:

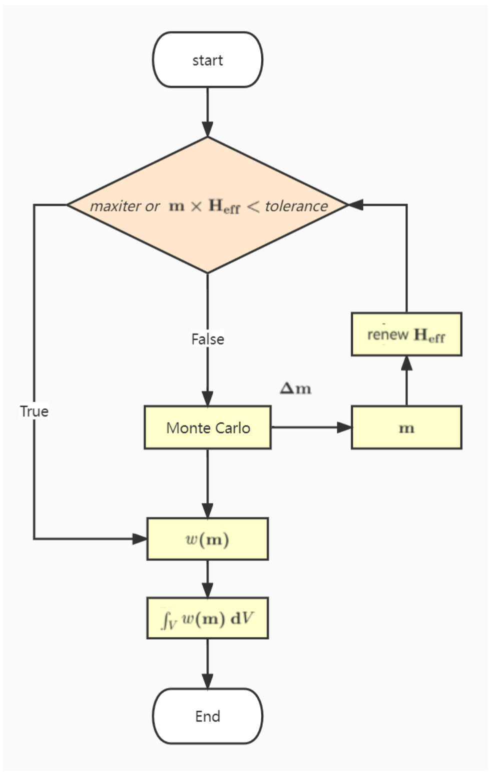

start
maxiter or $\mathbf{m} \times \mathbf{H}_{\text{eff}} <$ tolerance
False
True
Monte Carlo
$\Delta\mathbf{m}$
$\mathbf{m}$
renew $\mathbf{H}_{\text{eff}}$
$w(\mathbf{m})$
$\int_V w(\mathbf{m})\,\mathrm{d}V$
End

Figure 2: The procedure of a oversimplified Monte Carlo simulator. Starting from the a initial unit vector $\mathbf{m}$, after numbers of Monte Carlo steps, the final unit vector $\mathbf{m}$ and the system's total energy at this time is saved and calculated separately.

$$E = \int \left[w_{\text{ex}} + w_{\text{dmi}} + w_{\text{z}} + w_{\text{d}} + w_{\text{a}}\right] \mathrm{d}V \tag{10}$$

In which, $\mathrm{d}V = \mathrm{d}x\mathrm{d}y\mathrm{d}z$, which represents the volume of a single cell in discretised mesh. w_{ex} represents the exchange energy density, w_{dmi} represents the dmi energy density, w_{d} represents the demagnetisation energy density w_{a} represents the Uniaxial anisotropy energy density and w_{z} represents the Zeeman energy density. The total energy of the micromagnetic model is the integration of these energy density terms over the volume of the magnetic material sample.

Due to demagnetisation energy is a trivial energy term, while it requires a massive computational complexity and the Monte Carlo performance cannot be guaranteed. Several researches show that it is an acceptable choice that we do not include demagnetisation energy term in the simulation [6]. Therefore, the supported interactions in this work do not include demagnetisation energy term. Following this, we define tolerance and maximum iteration as our termination conditions. In every Monte Carlo step, we calculate $\mathbf{m} \times \mathbf{H}_{\text{eff}}$ and count iteration number i. if the terminate condition is met, we stop and save the current normalised magnetisation field $\mathbf{m}$. After that, we integrate the energy density over the volume of the magnetic material sample to calculate its total energy.

Iron germanide (FeGe) is the selected magnetic material in this project, for its electron spins are able to show a skyrmion lattice arrangement being considered for use in next-generation generation storage devices. The derived constants for FeGe are shown in Table. 1.

Table 1: Table of derived constants for FeGe

FeGe properties	Value	Unit
The local magnetic moments of Fe	1.16	μ_B
The local magnetic moments of Ge	-0.086	μ_B
Unit cell size a	4.7	Å
ordering temperature T_c	278.7	K
saturation magnetisationM_s	384	kAm^{-1}
exchange parameter$A$	8.78	pJm^{-1}
DMI material parameter D	1.58	mJm^{-2}

3.2 Technical details of the Monte Carlo algorithm

In most cases, the micromagnetic models have a combination of more than one term; for example, a model contains exchange, DMI, and Zeeman energy. When 'relax' this model using the Monte Carlo algorithm, the ordinary approach (Method 1) to update a trial spin in each Monte Carlo step is to recalculate the total energy for the whole unit vector field $\mathbf{m}$. However, it would occupy an extensively amount of CPU and RAM resources. We use **cProfile** (a python package) to understand which functions take the most CPU time. We found that the Laplacian operator $\nabla^2\mathbf{m}$ from the exchange energy term and Curl operator $\nabla \times \mathbf{m}$ from the energy term take up most of the total CPU time cost. Therefore, we have optimised the Laplacian and Curl operators to accelerate the Monte Carlo simulation process (Method 2). The Python code for optimising the Laplace operator was written as:

```python
def laplace_iter(self, spin_loc, old_laplace_component, old_m_padding):
        x, y, z = spin_loc[0], spin_loc[1], spin_loc[2]


        new_laplace_operator_x = old_laplace_component[0].copy()
        new_laplace_operator_y = old_laplace_component[1].copy()
        new_laplace_operator_z = old_laplace_component[2].copy()

        if (x == (self.nx-1) or 0) or (y == (self.ny-1) or 0) or (z == (self.nz
    -1) or 0):
            m_padding = np.pad(self.array, ((1, 1), (1, 1),
                        (1, 1), (0, 0)), 'edge')

        else:
            m_padding = old_m_padding.copy()
            m_padding[x+1, y+1, z+1] = self.array[x, y, z]

        new_laplace_operator_x[:, y, z] = (1 / self.a_x ** 2) * (
                                    m_padding[2:, y+1, z+1] -
                                    2 * m_padding[1:-1, y+1, z+1] +
                                    m_padding[:-2, y+1, z+1])

        new_laplace_operator_y[x, :, z] = (1 / self.a_y ** 2) * (
                                    m_padding[x+1, 2:, z+1] -
                                    2 * m_padding[x+1, 1:-1, z+1] +
                                    m_padding[x+1, :-2, z+1])

        new_laplace_operator_z[x, y, :] = (1 / self.a_z ** 2) * (
                                    m_padding[x+1, y+1, 2:] -
                                    2 * m_padding[x+1, y+1, 1:-1] +
                                    m_padding[x+1, y+1, :-2])

        laplace_component = (new_laplace_operator_x, new_laplace_operator_y,
                             new_laplace_operator_z)

        return laplace_component, m_padding
```

In which, spin_loc represents the location of the trial spin in a discretised mesh. The Laplacian of magnetisation $\nabla^2\mathbf{m} = \text{new\_laplace\_operator\_x} + \text{new\_laplace\_operator\_y} + \text{new\_laplace\_operator\_z}$. old_m_padding represents a mesh that we have extended it in x, y, z directions by adding more discretisation cells.

We pass the trial spin location, Laplacian of individual components of the unit vector field $\mathbf{m}$ from the previous iteration and the previous magnetisation field padding(including boundary condition) to the laplace_iter function. Updating $\text{new\_laplace\_operator\_x}[:, \text{y}, \text{z}]$, $\text{new\_laplace\_operator\_y}[\text{x}, :, \text{z}]$ and $\text{new\_laplace\_operator\_z}[\text{x}, \text{y}, :]$, respectively, meaning that we only update discretisation cells that are on the same axis as trial cell.

Experiment results in Fig. 3 shown that Method 2 is generally faster than Method 1 on different meshes and consumes less memory. When simulating larger systems, Method 2's advantages are more obvious.

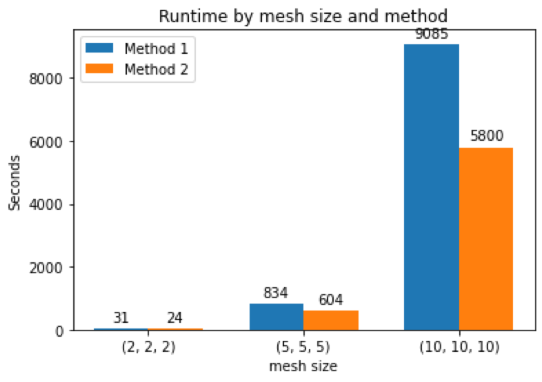


Figure 3: Performance Comparison between for different methods.

4 Result

4.1 Code design

We define a micromagnetic system modelling a thin-filme FeGe cuboid, 150 nm long, 150nm wide and 10 nm thick. This micromagnetic system contains exchange, DMI, and Zeeman energy. Material parameters of the simulated material are listed in Table.1. In addition, we apply an external magnetic field $\mathbf{B} = (0, 0, 0.1)$ T in the positive z-direction to the system. In which, $\mathbf{B} = \mu_0\mathbf{H}$, μ_0 represents the magnetic constant and $\mu_0 = 4\pi \times 10^{-7}\text{N}/\text{A}^2$.

In practice, we first need to obtain geometry information and material parameters to create a magnetic system. Following this, we pass those parameters to a finite-difference mesh called my_cool_mesh, which contains cell size, number of cells, etc. Following this, we use a numpy array of size (30, 30, 2, 3) to represent the initial unit vector $\mathbf{m}$. We know that the emergence of skyrmionic state configuration strongly depends on the initial magnetisation configuration. Therefore, we choose a helical configuration as our initial magnetisation configuration. When coordinates of the cells in the mesh satisfy $(x - x_0)^2 + (y - y_0)^2 <= (20 \times 10^{-9}\mathrm{m})^2$, the vectors in the cells are all assigned to $(0,\ 0.1, -1))/(0.1^2 + (-1)^2)^{1/2}$, otherwise, they are assigned to $(0,\ 0.1,\ 1)/(0.1^2 + (1)^2)^{1/2}$. Following this, we create a magnetisation field object $\mathrm{my\_cool\_m}$ and implement a Monte Carlo simulation to find this model's 'relaxed' state. After that, the user may consider plotting magnetisation fields from a specific plane and create a gif to visualise the results of the micromagnetic model.

In the next few lines, we present the code to run a Monte Carlo simulation for the micromagnetic system we mentioned above.

```python
# external pacakges that we need
import discretisedfield as df
import micromagneticmodel as mm
import oommfc as oc
import matplotlib.pyplot as plt
import numpy as np
# my own package
from continuous_model import *

# specify several parameters
Ms = 3.84e5
mu0 = 4 * np.pi * 1e-7
A = 8.78e-12
D = 1.58e-3
B = 0.1
H = [0, 0, B/mu0]

initial_m = np.load('./normalised_m/m_is_skyrmion.npy')

# call discretised mesh class to
# create a Rectangular discretised mesh object
nx = initial_m.shape[0]
ny = initial_m.shape[1]
nz = initial_m.shape[2]
my_cool_mesh = RectangularMesh(nx=nx, ny=ny, nz=nz, units=5e-9)

# call m_Field class to create a magnetisation field object
my_cool_m = m_Field(my_cool_mesh, Ms, initial_m)

# call Simulator class to create a simulator
# object for a sample
my_cool_simulator = Simulator(A, D, H, 0, 0, 0)
# "relax" the system, minimise the energy of the system
m_final, E_final = my_cool_simulator.compute_minimum(my_cool_m, 0.001, 15000000)
```

4.2 Model validation

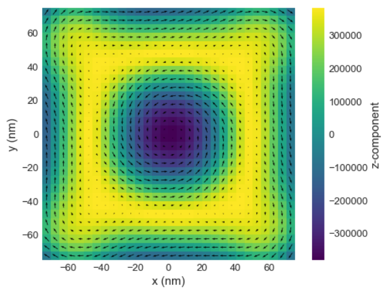
Next, we perform validation testing for the above example. We are comparing the final state of our unit vector $\mathbf{m}$ with that of a Python package Ubermag. Ubermag uses a popular micromagnetic calculator called OOMMF, which is capable of varying magnetisation in the three spatial dimensions. The way to do that is to minimise the energy using algorithms such as the conjugate gradient method or solve Landau-Lifshitz-Gilbert (LLG) equation [14].

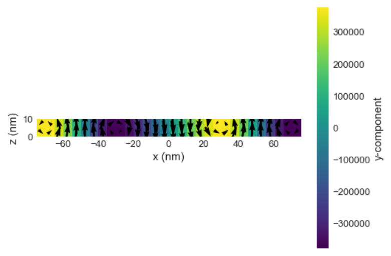
We assume that Ubermag can produce a known good unit vector $\mathbf{m}$. After that, we define tolerance and use $\mathrm{numpy.allclose}()$ function to perform an element-wise comparison between the arrays to find out which values are the same. We assigned the value of 1×10^{-8} to the relative tolerance RTOL and the value of 0.01 to the absolute tolerance ATOL. The equation to determine whether or not our result is valid is written as:

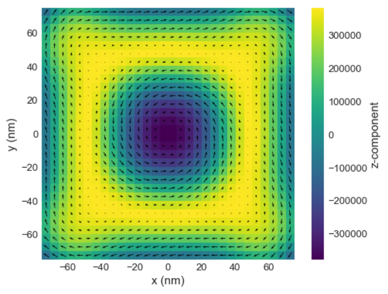
$$|\ \text{Known good value}\ -\ \text{Computed value}\ | <\ \text{RTOL}\ ^*\ |\ \text{Known good Value}\ | + \text{ATOL} \qquad (11)$$

After 1.5 billion Monte Carlo steps, we found that our result agrees well with the known good values from

Ubermag. In addition, we plot the xy magnetisation vector field at the top of the sample and the xz magnetisation vector field through the middle of the model, as shown in Fig. .4 So that it is more intuitive for us to compare our result and Ubermag's result.


(a) our program xy magnetization vector field


(b) our program xz magnetization vector field


(c) Ubermag xy magnetization vector field

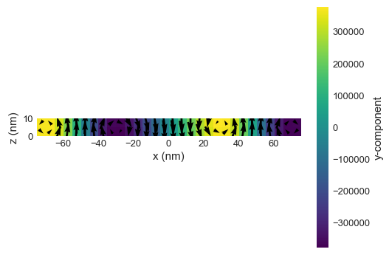
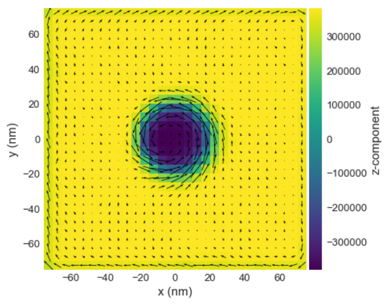

(d) Ubermag xz magnetization vector field

Figure 4: Magnetisation configurations of the lowest energy state from our program and Ubermage. In (b), (d), we plot the xy magnetization vector field from the upper surface of the sample. In (a), (c), we plot the xz magnetization vector field through the middle of the sample.

Moreover, we plot xy magnetisation vector field from the upper surface of the sample at temperature $T = 20$ K and $T = 40$ K, as shown in Fig. 5. We found that we spent more time or Monte Carlo steps at higher than temperature $T = 0$ case to "relax" the system and find its lowest energy state.


(a) $T = 20$ xy magnetization vector field

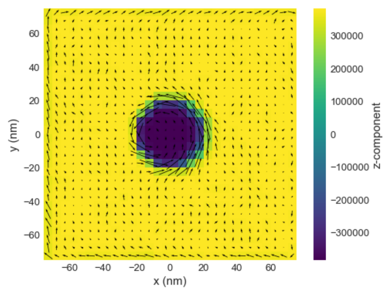

(b) $T = 40$ xy magnetization vector field

Figure 5: After 4.5 billion Monte Carlo steps, xy magnetization vector field from the upper surface of the sample at different temperature. In (a) Temperature $T = 20$. In (b), Temperature $T = 40$.

5 Discussion and Conclusion

Considering the limited time available in this work, only one optimising strategy of Monte Carlo simulation is adopted. The way we implemented Monte Carlo in this project is still computationally expensive. This issue limits our thinking about how temperature quantitatively affects test samples in the future. Although no literature currently mentions how to accelerate the Monte Carlo simulator for a continuous micromagnetic model, some researchers have applied the Checkerboard algorithm to accelerate the Monte Carlo simulation on an Ising model. We might be able to follow up and apply that approach to the continuous micromagnetic model [9].

Moreover, our mesh only supports one crystallographic arrangement which is a simple cubic lattice. However, there are several other crystallographic arrangements, for example, centred-cubic arrangement, hexagonal arrangement, etc. This makes our simulator preclude its application to many other magnetic materials.

In summary, this project uses the finite-difference method to discretise the magnetic nanomaterials sample into a mesh of magnetic moments. After that, the sample can be approximated as a continuous magnetisation field. We introduce a Monte Carlo simulator to determine the lowest energy magnetisation state of the continuous micromagnetic model for magnetic materials. In addition, a cuboid was used to show us the skyrmionic texture and verify the correctness of the Monte Carlo simulator. We found that our program can produce skyrmionic equilibrium states and test results are consistent with that of an external open-sourced python package Ubermag. In addition, skyrmion configurations can be affected by the external temperature. Neither test case of $T = 20$ nor $T = 40$ has the skyrmion configurations present in the sample.

References

[1] Ernst Ising. *Beitrag zur theorie des ferro-und paramagnetismus*. PhD thesis, Grefe & Tiedemann, 1924.

[2] Richard FL Evans, Weijia J Fan, Phanwadee Chureemart, Thomas A Ostler, Matthew OA Ellis, and Roy W Chantrell. Atomistic spin model simulations of magnetic nanomaterials. *Journal of Physics: Condensed Matter*, 26(10):103202, 2014.

[3] Marc-Antonio Bisotti, David Cortés-Ortuño, Ryan A Pepper, Weiwei Wang, Marijan Beg, Thomas Kluyver, and Hans Fangohr. Fidimag–a finite difference atomistic and micromagnetic simulation package. *arXiv preprint arXiv:2002.04318*, 2020.

[4] Marijan Beg, Rebecca Carey, Weiwei Wang, David Cortés-Ortuño, Mark Vousden, Marc-Antonio Bisotti, Maximilian Albert, Dmitri Chernyshenko, Ondrej Hovorka, Robert L Stamps, et al. Ground state search, hysteretic behaviour and reversal mechanism of skyrmionic textures in confined helimagnetic nanostructures. *Scientific reports*, 5(1):1–14, 2015.

[5] Stephen Blundell. Magnetism in condensed matter, 2003.

[6] Marijan Beg. *Skyrmionic states in confined helimagnetic nanostructures*. PhD thesis, University of Southampton, 2016.

[7] Igor Dzyaloshinsky. A thermodynamic theory of "weak" ferromagnetism of antiferromagnetics. *Journal of physics and chemistry of solids*, 4(4):241–255, 1958.

[8] Tôru Moriya. Anisotropic superexchange interaction and weak ferromagnetism. *Physical review*, 120(1):91, 1960.

[9] Kun Yang, Yi-Fan Chen, Georgios Roumpos, Chris Colby, and John Anderson. High performance monte carlo simulation of ising model on tpu clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2019.

[10] Yehui Zhang, Bing Wang, Yilv Guo, Qiang Li, and Jinlan Wang. A universal framework for metropolis monte carlo simulation of magnetic curie temperature. *Computational Materials Science*, 197:110638, 2021.

[11] Kurt Binder, Dieter Heermann, Lyle Roelofs, A John Mallinckrodt, and Susan McKay. Monte carlo simulation in statistical physics. *Computers in Physics*, 7(2):156–157, 1993.

[12] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[13] Marijan Beg, Martin Lang, and Hans Fangohr. Ubermag: Toward more effective micromagnetic workflows. *IEEE Transactions on Magnetics*, 58(2):1–5, 2021.

[14] Marijan Beg, Ryan A Pepper, and Hans Fangohr. User interfaces for computational science: A domain specific language for oommf embedded in python. *AIP Advances*, 7(5):056025, 2017.