Chapter One

Introduction

In this chapter fundamental theories for micromagnetics will be outlined, with emphasis on the Landau-Lifshitz-Gilbert (LLG) equation and components of the effective field. Micromagnetic simulation is directly based on these theories.

1.1 Motivation

Magnetism and its applications present good cases in point for the technological trends of our time: devices get smaller, speeds increase, and complex dynamic systems can be better understood with the rapidly soaring capability of computer-based modeling. Improvements in nanoscale lithography and development of ultrathin multilayer films have opened new possibilities for novel magnetic devices [1, 2]. These exploratory developments rely heavily on fundamental research using creative experimental techniques, as well as high performance numerical simulations.

This thesis focuses on micromagnetic simulations and experimental comparisons. The experiments are being performed in the ultrafast microscopy laboratories led by Prof. Freeman at the University of Alberta [3, 4, 5, 6]. In the following parts of

this chapter, the theories of micromagnetics will be outlined, especially those in two dimensional systems, because ultrathin films exhibit 2D properties. However, for more advanced studies a complete 3D model in multi-layer systems will be required, and this goes beyond what will be contained in this work.

The material being investigated in this work is $Ni_{80}Fe_{20}$ Permalloy. It is highly magnetized with a saturation magnetization >800(emu/cm^3), but its coercivity is so low that a small external field is able to flip the sample's magnetization. It also has other good properties such as high permeability and weak anisotropy. Benefiting from all these advantages, Peamalloy is used in more and more applications.

1.2 Theoretical background

1.2.1 Classical model

1.2.1.1 Equation of motion

Fundamental dynamics of magnetic materials are, to the first order approximation, governed by the following classical relation: the time rate of change of a magnetic moment $\vec{m}$ is proportional to the torque applied on this momentum [7, 8],

$$\frac{d\vec{\boldsymbol{m}}}{dt} = -g_0(\vec{\boldsymbol{m}} \times \vec{\boldsymbol{H}}) \tag{1.1}$$

where $\vec{\boldsymbol{H}}$ is the magnetic field applied on $\vec{\boldsymbol{m}}$, and $g_0 = 1.761 \times 10^{11} (\mathrm{s^{-1}T^{-1}})$ is the electron's gyromagnetic ratio (see Eq.(1.14) for definition details).

It is important to note that Eq.(1.1) is a classical equation, while micromagnetic phenomena involve underlying quantum mechanical considerations. An equivalence relation supports the validity of the classical equation by showing that it has exactly the same form as what is derived from quantum mechanics.

Quantum mechanics states that the time evolution of the expectation value of a spin $\vec{\boldsymbol{S}}$ obeys Schrödinger's equation [9],

$$i\hbar \frac{d}{dt} < \vec{\boldsymbol{S}} > (t) = < [\vec{\boldsymbol{S}}, \hat{\boldsymbol{H}}(t)] > \tag{1.2}$$

where $\hat{\boldsymbol{H}}(t)$ is the spin system's Hamiltonian. If the spin is in a time-dependent external field $\vec{\boldsymbol{B}}$, the Hamiltonian is of Zeeman type:

$$\hat{\boldsymbol{H}} = -\frac{g m_B}{\hbar} \vec{\boldsymbol{S}} \cdot \vec{\boldsymbol{B}} \tag{1.3}$$

where g is the gyromagnetic splitting factor, g=2.0x1.001159657 for free electrons; $m_B \equiv e\hbar / 2m_e$=9.2741×10^{-24}(J/T) is the Bohr magneton; $\hbar = 1.0546 \times 10^{-34} (\mathrm{J \cdot s})$ is the reduced Planck constant. The x-component of Eq.(1.2)'s right-hand-side is:

$$\begin{aligned} \left[S_x, \hat{\boldsymbol{H}}(t) \right] &= -\frac{g m_B}{\hbar} \left[S_x, S_x B_x(t) + S_y B_y(t) + S_z B_z(t) \right] \\ &= -\frac{g m_B}{\hbar} \left(B_y(t)[S_x, S_y] + B_z(t)[S_x, S_z] \right) \end{aligned} \tag{1.4}$$

and according to the commutation rules

$$[S_x, S_y] = i\hbar S_z$$
$$[S_y, S_z] = i\hbar S_x \tag{1.5}$$
$$[S_z, S_x] = i\hbar S_y$$

one obtains

$$\left[S_x, \hat{\boldsymbol{H}}(t)\right] = -\frac{g m_B}{\hbar} i\hbar \left(B_y(t) S_z - B_z(t) S_y\right) \tag{1.6}$$

The y/z-components have a similar expression and finally the equation of the spin's motion reads

$$\frac{d}{dt} < \vec{\boldsymbol{S}} > (t) = \frac{g m_B}{\hbar} < \vec{\boldsymbol{S}} > (t) \times \vec{\boldsymbol{B}}(t) \tag{1.7}$$

On the other hand, consider a classical angular momentum generated by an orbiting electron (equivalent to a current loop) and the following relation holds [9],

$$\vec{\boldsymbol{m}} = \frac{q_e}{2m_e} \vec{\boldsymbol{l}} \tag{1.8}$$

where $\vec{\boldsymbol{m}}$ is the current loop's dipole moment, $\vec{\boldsymbol{l}}$ is the angular momentum, and q_e and m_e are the electron's charge and mass, respectively.

In quantum mechanics, spin $\vec{\boldsymbol{s}}$ and angular momentum $\vec{\boldsymbol{l}}$ are usually treated in the same manner [9]; a similar geometric relation with Eq.(1.8) holds for spin, and we can write

$$\vec{\boldsymbol{m}} = g < \vec{\boldsymbol{s}} > \tag{1.9}$$

where g, the gyromagnetic ratio, is equal to

$$g = \frac{g q_e}{2m_e} = \frac{g m_B}{\hbar} < 0 \tag{1.10}$$

Substituting Eq.(1.9) into Eq.(1.7) one obtains

$$\frac{d}{dt}\vec{\boldsymbol{m}}(t) = g\left[\vec{\boldsymbol{m}}(t)\times\vec{\boldsymbol{B}}(t)\right] \tag{1.11}$$

Now, defining the magnetization as the total dipole moment per unit volume

$$\vec{\boldsymbol{M}} = \frac{\sum\vec{\boldsymbol{m}}}{\text{unit volume}} \tag{1.12}$$

it follows that (in the SI system of units)

$$\frac{d}{dt}\vec{\boldsymbol{M}}(t) = m_0 g\left[\vec{\boldsymbol{M}}(t)\times\vec{\boldsymbol{H}}(t)\right] \tag{1.13}$$

where $\vec{\boldsymbol{H}} = \vec{\boldsymbol{B}} / m_0$ is the magnetic field (not Hamiltonian $\hat{\boldsymbol{H}}$).

Conventionally people define

$$g_0 = m_0 \frac{g|m_B|}{\hbar} = -m_0 g \tag{1.14}$$

In the SI system, $g_0 = 1.761\times10^{11}(\mathrm{s^{-1}T^{-1}})$

Now we can write the equation governing magnetization motion

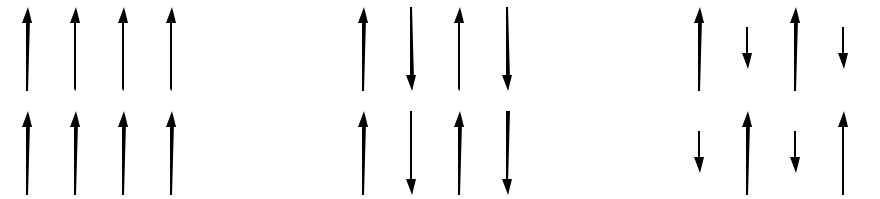
$$\frac{d}{dt}\vec{\boldsymbol{M}}(t) = -g_0\left[\vec{\boldsymbol{M}}(t)\times\vec{\boldsymbol{H}}(t)\right] \tag{1.15}$$

which is Eq.(1.1).

In conclusion, the equivalence between dipole moment and spin (Eq.(1.9)), and the analogy between Eq.(1.7) and Eq.(1.15) suggests that the classical equation is obeyed exactly, whatever the time dependence of the magnetic field is.

1.2.1.2 Magnetic order and ferromagnetic phenomena

Magnetic materials have different types of "magnetic order", such as ferromagnetism, antiferromagnetism, and ferrimagnetism [10], illustrated in Fig.(1.1). In this thesis, the focus is restricted to ferromagnetic materials.



(a) ferromagnetism (b) antiferromagnetism (c) ferrimagnetism

Fig.(1.1) Types of magnetic orders. (a), ferromagnetism: magnetic dipoles at each site point toward the same direction; (b), antiferromagnetism: neighboring dipoles align antiparellel and yield zero total magnetization; (c), ferrimagnetism: neighboring dipoles align antiparellel with different strength, making a net magnetization.

The atomic-scale magnetic dipoles underlying ferromagnetism are associated with the orbital and spin degrees of freedom of the electrons in the material [11]. The exchange interaction between dipoles, which is quantum mechanical in nature, is the basis of magnetic ordering. Ferromagnetism occurs when a saving in electrostatic Coulomb energy is enabled by parallel electron spin alignment (the spin dependence introduced via the Pauli principle). Below a critical temperature (the Curie temperature), a macro-scale magnetic dipole moment is formed.

In thin films, one can easily observe ferromagnetic "domains" [12, 13]. All magnetic dipoles have the same orientation within one domain, and can be treated as a big dipole moment (or "macrospin"). Magnetic properties of the whole film are then

determined by two aspects: the properties of individual domains and the motion of domain walls. Magnetic quantities of single domain structures are quite stable and controllable, and these structures have been used in many technological applications.

In numerical simulations, since most samples have a uniform distribution of magnetic dipoles, we can divide the sample into finite size grids, with the same magnitude of dipole moment for each element. Then the moment within one element can be expressed in spherical coordinates (shown in Fig.(1.2)):

$$\vec{\boldsymbol{M}}\left(\vec{\boldsymbol{r}}\right) = M_s V(\sin q \cos f, \sin q \sin f, \cos f) \tag{1.16}$$

where M_s is the saturation magnetization, usually with density dimension emu/cm^3, and V is the volume of the element. Eq.(1.16) greatly simplifies the linearization of Eq.(1.15) and forms the basis of micromagnetic simulation.

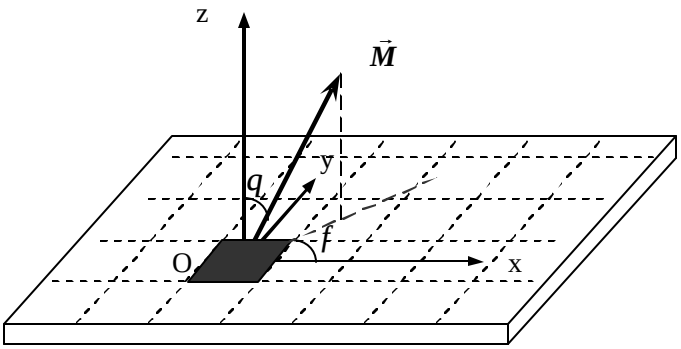


Fig.(1.2) Spherical coordinates. The thin film is divided by equal size elements, so that the magnitude of magnetization remains a constant.

1.2.2 Magnetization dynamics and LLG equation

1.2.2.1 The LLG equation

The Landau-Lifshitz-Gilbert (LLG) equation describes the governing mechanisms of motion of magnetization. It is developed from Eq.(1.15) by introducing a damping term which does not allow the length of the magnetization vector to change, consistent with the underlying assumption of ferromagnetism. One way to do so is replacing the field $\vec{\boldsymbol{H}}$ by an effective field $\vec{\boldsymbol{H}}_{\text{eff}}$ with an Ohmic type dissipation term [8, 10]:

$$\vec{\boldsymbol{H}}_{\text{eff}} = \vec{\boldsymbol{H}} - \alpha \frac{1}{g_0 M_s} \frac{d\vec{\boldsymbol{M}}}{dt} \tag{1.17}$$

where M_s is the saturation magnetization and α is a phenomenological damping parameter. Inserting Eq.(1.17) into Eq.(1.15) yields

$$\frac{d\vec{\boldsymbol{M}}(t)}{dt} = -g_0 \left[\vec{\boldsymbol{M}}(t) \times \vec{\boldsymbol{H}}(t) \right] + \frac{\alpha}{M_s} \left[\vec{\boldsymbol{M}}(t) \times \frac{d\vec{\boldsymbol{M}}(t)}{dt} \right] \tag{1.18}$$

Eq.(1.18) is known as Landau-Lifshitz-Gilbert (LLG) equation of magnetization motion. The effect of damping is illustrated in Fig.(1.3). The second term on the right-hand-side of Eq.(1.18) provides a frictional force that causes the magnetization to spiral down until completely aligned with the actual field $\vec{\boldsymbol{H}}$ after a long enough time.

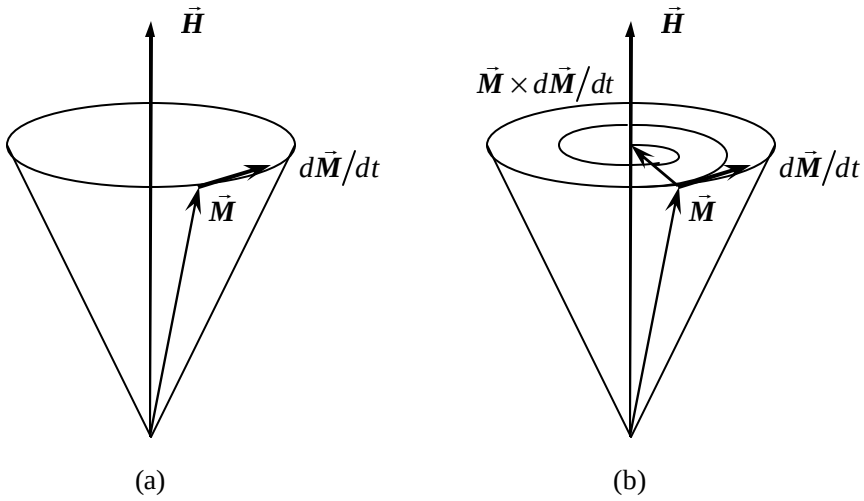


Fig.(1.3) Magnetization precession.
(a) no damping
(b) damping

Eq.(1.18) has an equivalent form that is easier to handle for numerical modeling:

$$(1+\alpha^2)\frac{d\vec{M}(t)}{dt} = -\gamma_0[\vec{M}(t)\times\vec{H}(t)] - \frac{\alpha\gamma_0}{M_s}\left\{\vec{M}(t)\times[\vec{M}(t)\times\vec{H}(t)]\right\} \quad (1.19)$$

For our treatment of ferromagnetic thin films, we will assume the thickness of the film to be uniform. When the plane is divided into equally sized rectangular cells, the volume of the cells is the same and the total magnetic moment of each cell has equal magnitude (the saturation magnetization, M_s, times the volume of the cell, V). In spherical coordinates shown in Fig.(1.2), one only needs to know the evolution of angles θ and ϕ to get the whole dynamics of the magnetization. This removes one degree of freedom in comparison to 3-component Cartesian coordinates (where, of course, the three components are not independent given M_s a constant, *i.e.*, $M_x^2 + M_y^2 + M_z^2 = M_s^2$, but this constraint is not used because the computation is more complicated than that in spherical coordinates). Eq.(1.19)

can be rewritten as

$$\begin{aligned} \frac{dq}{dt} &= a H_q + H_f \\ \sin q \frac{df}{dt} &= -H_q + a H_f \end{aligned} \tag{1.20}$$

where H_q, H_f are the effective field's $\hat{q}$, $\hat{f}$ components, respectively. t, which is a dimensionless time related to the real time t by

$$t = \frac{1+a^2}{g_0 M_s} t \tag{1.21}$$

Taking Permalloy as an example, $a = 0.008$ and M_s=860(Oe), then the factor equals about 66(ps), which means every integer dimensionless time step in the simulation corresponds to 66 picoseconds.

1.2.2.2 The effective field

Eq.(1.20) has a concise form, but it is not so easy to figure out precisely what the angular components of the effective field should be. The effective field has various sources such as the external field, anisotropy, exchange, demagnetizing fields, and thermal fluctuations. They are discussed separately in the following subsections.

1.2.2.2.1 External (Zeeman) term

The mapping between the unit vectors of a Cartesian coordinate system and those of a spherical coordinate system are:

$$\begin{aligned}\vec{\boldsymbol{i}}_r &= \sin q\cos f\,\vec{\boldsymbol{i}}_x + \sin q\cos f\,\vec{\boldsymbol{i}}_y + \cos q\,\vec{\boldsymbol{i}}_z \\ \vec{\boldsymbol{i}}_q &= \cos q\cos f\,\vec{\boldsymbol{i}}_x + \cos q\sin f\,\vec{\boldsymbol{i}}_y - \sin q\,\vec{\boldsymbol{i}}_z \\ \vec{\boldsymbol{i}}_f &= -\sin f\,\vec{\boldsymbol{i}}_x + \cos f\,\vec{\boldsymbol{i}}_y\end{aligned} \tag{1.22}$$

Assuming the external field acting at the lattice site (*i*, *j*) with magnetization vector $\vec{\boldsymbol{M}} = M_i\,\vec{\boldsymbol{i}}_r$ (for uniform thin films, $M_{ij} = M_S$=constant)

$$\vec{\boldsymbol{H}}^{(\mathrm{ext})} = H_x\vec{\boldsymbol{i}}_x + H_y\vec{\boldsymbol{i}}_y + H_z\vec{\boldsymbol{i}}_z \tag{1.23}$$

the $\hat{q}$ and $\hat{f}$ components of the field are

$$\begin{aligned}H_q^{(\mathrm{ext})} &= \vec{\boldsymbol{H}}^{(\mathrm{ext})}\cdot\vec{\boldsymbol{i}}_q = H_x\cos q\cos f + H_y\cos q\sin f - H_z\sin q \\ H_f^{(\mathrm{ext})} &= \vec{\boldsymbol{H}}^{(\mathrm{ext})}\cdot i_f = -H_x\sin f + H_y\cos f\end{aligned} \tag{1.24}$$

1.2.2.2.2 Uniaxial anisotropy term

Assume the local axis of anisotropy has an arbitrary direction in space and is specified in spherical coordinates as $\vec{\boldsymbol{i}}_{r_0}$, with angular coordinates (q_0, f_0). The magnetization vector at this site is $\vec{\boldsymbol{M}} = M_s\vec{\boldsymbol{i}}_r$, with angular coordinates (q, f). Define K_u as the uniaxial anisotropy constant and the anisotropy energy density can be written as [8, 11]

$$\begin{aligned}W_{\mathrm{ans}} &= K_u\left[1-(\vec{\boldsymbol{i}}_r\cdot\vec{\boldsymbol{i}}_{r_0})^2\right] \\ &= K_u\left\{1-\left[\cos q\cos q_0 + \sin q\sin q_0\cos(f-f_0)\right]^2\right\}\end{aligned} \tag{1.25}$$

There is a simple way to obtain equivalent field $\vec{\boldsymbol{H}}$ from the energy density.

Assume that $\vec{\boldsymbol{M}}$ rotates by a small amount $(\Delta q, \Delta f)$. The change in energy density is then given by

$$\begin{aligned}\Delta W &= -\vec{\boldsymbol{H}} \cdot \Delta\vec{\boldsymbol{M}} \\ &= -(H_r \vec{\boldsymbol{i}}_r + H_q \vec{\boldsymbol{i}}_q + H_f \vec{\boldsymbol{i}}_f) \cdot (M_s \Delta q\, \vec{\boldsymbol{i}}_q + M_s \sin q\, \Delta f\, \vec{\boldsymbol{i}}_f) \\ &= -M_s H_q \Delta q - M_s H_f \sin q\, \Delta f\end{aligned} \tag{1.26}$$

consequently,

$$\begin{aligned}H_q &= -\frac{1}{M_s}\frac{\partial W}{\partial q} \\ H_f &= -\frac{1}{M_s \sin q}\frac{\partial W}{\partial f}\end{aligned} \tag{1.27}$$

Eq.(1.27) are quite general and give components of the equivalent field in terms of corresponding energy density. For uniaxial anisotropy, these components are

$$\begin{aligned}H_q^{(\text{ans})} &= -\frac{K_u}{M_s}\{\sin 2q\,[\cos^2 q_0 - \sin^2 q_0 \cos^2(f - f_0)] \\ &\quad - \cos 2q \sin 2q_0 \cos(f - f_0)\} \\ H_f^{(\text{ans})} &= -\frac{K_u}{M_s}[\sin q \sin^2 q_0 \sin 2(f - f_0) \\ &\quad + \cos q \sin 2q_0 \sin(f - f_0)]\end{aligned} \tag{1.28}$$

Eq.(1.28) looks complicated, because the spatial orientations of the anisotropy axes are presumed to be arbitrarily distributed in the material. Nevertheless, in single crystals the anisotropy direction is uniform across the specimen, and even in polycrystalline thin films used in applications, a certain anisotropy direction can be induced by applying some external magnetic field during the course of film growth [4, 12, 13]. For example, in our patterned Permalloy films, the anisotropy direction lies in-plane and points to the y-axis (see Fig.(1.1) for reference), which is defined as the “easy axis”, and the x-axis is consequently called the “hard axis”. In this case, $q_0 = f_0 = 90°$ holds everywhere in the sample, and gives

$$H_q^{(ans)} = \frac{K_u}{M_s}\sin 2q \sin^2 f$$
$$H_f^{(ans)} = \frac{K_u}{M_s}\sin q \sin 2f \tag{1.29}$$

1.2.2.2.3 Exchange interaction

First, the exchange energy density in terms of orientations of coupled moments should be expressed. Suppose $\vec{\boldsymbol{M}}$ and $\vec{\boldsymbol{M}}_1$ are two neighboring moments on the lattice with distance d between them, and let A_x represent the macroscopic exchange stiffness coefficient [8, 10]. The exchange energy density for this dipole pair is then written as [8]

$$W_{\text{xhg}} = \frac{2A_x}{d^2}\left(1 - \frac{\vec{\boldsymbol{M}}}{\left|\vec{\boldsymbol{M}}\right|}\cdot\frac{\vec{\boldsymbol{M}}_1}{\left|\vec{\boldsymbol{M}}_1\right|}\right) \tag{1.30}$$

In 2D simulations on thin film samples, all sites have the same magnitude of magnetization, thus $\left|\vec{\boldsymbol{M}}\right| = \left|\vec{\boldsymbol{M}}_1\right| = M_s$. In terms of spherical coordinates, Eq.(1.30) is written as

$$\begin{aligned} W_{\text{xhg}} &= \frac{2A_x}{d^2}\left(1 - \vec{\boldsymbol{i}}_r \cdot \vec{\boldsymbol{i}}_{r_1}\right) \\ &= \frac{2A_x}{d^2}\left[1 - \cos q \cos q_1 - \sin q \sin q_1 \cos(f - f_1)\right] \end{aligned} \tag{1.31}$$

Using Eq.(1.27), the effective exchange field on $\vec{\boldsymbol{M}}$ as a result of interaction with $\vec{\boldsymbol{M}}_1$ is

$$\begin{aligned} H_q^{(\text{xhg})} &= -\frac{2A_x}{M_s d^2}\left[\sin q \cos q_1 - \cos q \sin q_1 \cos(f - f_1)\right] \\ H_f^{(\text{xhg})} &= -\frac{2A_x}{M_s d^2}\sin q_1 \sin(f - f_1) \end{aligned} \tag{1.32}$$

In the numerical modeling, only the nearest neighboring sites are included in the calculations of the exchange term [8, 10], consistent with the short-range nature of the exchange interaction, *e.g.*, in the Heisenberg Hamiltonian which sums only over nearest neighbor spin pairs.

1.2.2.2.4 Demagnetizing field

Demagnetization is the classical dipole-dipole magnetic interaction. In the thin film, suppose one dipole moment $\vec{m}$ is located at the origin (0,0) and another dipole, $\vec{m}_1$, has the coordinates $(x_1 = l\cos y\,, y_1 = l\sin y\,)$ as shown in Fig.(1.4). $\vec{m}_1$ applies a field on $\vec{m}$ with the tendency to flip it towards the opposite orientation, such that the total magnetization of this two dipole system would vanish. The accumulation of all of these dipolar interactions across a specimen is called the "demagnetizing field".

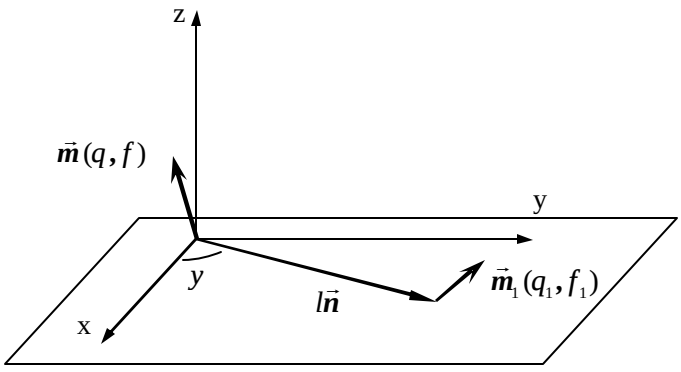


Fig.(1.4) Demagnetizing field between two magnetic dipoles.

We define $\vec{\boldsymbol{n}} = \cos y\ \vec{\boldsymbol{i}}_x + \sin y\ \vec{\boldsymbol{i}}_y$, the unit vector along the line connecting the origin to the point (x_1, y_1). The demagnetizing field at the origin is [7, 8]

$$\vec{\boldsymbol{H}}^{(\mathrm{dmag})} = \frac{3\vec{\boldsymbol{n}}\left(\vec{\boldsymbol{n}} \cdot \vec{\boldsymbol{m}}_1\right) - \vec{\boldsymbol{m}}_1}{l^3} \tag{1.33}$$

Using Eq.(1.22), we obtain the field's angular components:

$$\begin{aligned} H_q^{(\mathrm{dmag})} &= \vec{\boldsymbol{H}}^{(\mathrm{dmag})} \cdot \vec{\boldsymbol{i}}_q = \frac{\left|\vec{\boldsymbol{m}}_1\right|}{l^3}\left[3(\vec{\boldsymbol{n}} \cdot \vec{\boldsymbol{i}}_{r_1})(\vec{\boldsymbol{n}} \cdot \vec{\boldsymbol{i}}_q) - (\vec{\boldsymbol{i}}_{r_1} \cdot \vec{\boldsymbol{i}}_q)\right] \\ H_f^{(\mathrm{dmag})} &= \vec{\boldsymbol{H}}^{(\mathrm{dmag})} \cdot \vec{\boldsymbol{i}}_f = \frac{\left|\vec{\boldsymbol{m}}_1\right|}{l^3}\left[3(\vec{\boldsymbol{n}} \cdot \vec{\boldsymbol{i}}_{r_1})(\vec{\boldsymbol{n}} \cdot \vec{\boldsymbol{i}}_f) - (\vec{\boldsymbol{i}}_{r_1} \cdot \vec{\boldsymbol{i}}_f)\right] \end{aligned} \tag{1.34}$$

or equivalently,

$$\begin{aligned} H_q^{(\mathrm{dmag})} &= \frac{hd^2 M_s}{l^3}\{\sin q \cos q_1 \\ &\quad + \cos q \sin q_1[3\cos(f - y)\cos(f_1 - y) - \cos(f - f_1)]\} \\ H_f^{(\mathrm{dmag})} &= \frac{hd^2 M_s}{l^3}\sin q_1[\sin(f - f_1) - 3\sin(f - y)\cos(f_1 - y)] \end{aligned} \tag{1.35}$$

The total demagnetizing field on a dipole moment in the film's lattice is the sum of the demagnetizing field from all the other dipoles. The strength of $\vec{\boldsymbol{H}}^{(\mathrm{dmag})}$ fades quickly with distance as $1/l^3$, while the number of dipoles within this distance increases with the factor l^2, so the sum of dipole-dipole interaction drops with the factor $1/l$ when the calculation range grows. In early works [8], the demagnetizing term was not calculated over the entire sample in order to shorten the runtime of simulation programs. The current standard is to employ a series of FFT-based algorithms [10, 14, 15]. The basic idea is to transform Eq.(1.33) into Fourier space, and all calculations are performed through fast Fourier transformations (FFT). The calculations then become linear, leading to faster execution speed.

1.2.2.2.5 Thermal fluctuation

The micromagnetic dynamics described by the LLG equation (Eq.(1.18)) is completely deterministic. Starting from a certain initial state, the micromagnetic simulation always results in unique numerical outputs, which is obviously not the case. A lot of work has been done on the stochastic thermal effect [16], which is believed to be the key factor that causes random fluctuations in magnetic samples.

A simple model is to introduce a stochastic thermal magnetic field $\vec{\boldsymbol{H}}_{\text{th}}$, and add it to the effective field $\vec{\boldsymbol{H}}_{\text{eff}}$. This treatment covers all kinds of thermal interactions such as phonons, conduction electrons, nuclear spins, damping dissipation, *etc.* The system has infinite degrees of freedom, in principle; thus, the thermal field can be assumed to be a Gaussian-distributed random process. In simulation codes, an array of Gaussian random numbers with mean 0 and standard deviation 1 is generated (denoted by *rand_gauss*). The Cartesian components of the thermal field is then given by

$$H_{\text{th}}^{(x,y,z)} = rand\_gauss^{(x,y,z)} \sqrt{\frac{2a\,k_B T}{g\,M_s V \Delta t}} \tag{1.36}$$

where k_B is the Boltzmann's constant, T is the temperature, V is the volume of the cell, and Δt is the time interval between two integration steps.

Chapter Two

Micromagnetic Simulation Model

In this chapter, the algorithms and code structures of the simulation are presented. I upgraded the simulation code from FORTRAN77 to FORTRAN90, because the latter version takes advantage of modern language features such as dynamic memory and object-oriented programming (OOP). I show benchmarking tests to demonstrate the success of both simulation codes. Finally, various visualization schemes are presented to display the simulated results.

2.1 Simulation model

2.1.1 Coordinate system

As mentioned in Chapter One, the x-axis and y-axis are not set equivalently. The magnetocrystalline anisotropy, mainly generated by an external field applied during the film growth, makes one direction the easy axis and the other the hard axis. The easy axis is usually chosen to be the longer side of the rectangle, or the long axis of an elliptical sample. The y-axis is the sample's easy axis then, as shown in Fig(2.1). Because by this setting, the primary anisotropy direction in spherical coordinates can be written as (90^0, 90^0), and the effective anisotropy

field is described by Eq.(1.29), which is a simplified form. Otherwise, if the x-axis is set as the easy axis, the anisotropy direction is (90^0, 0^0), requiring different equations to be used in the simulation.

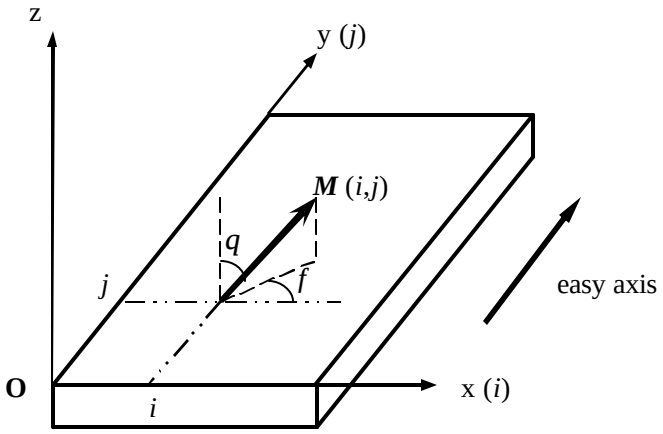


Fig.(2.1) Coordinate system used in the simulation. Integer *i*, *j* are indices of the 2D lattice for iterative computations across the whole sample. Note: (1), the easy axis of the magnetic structure is set in the y-direction, see discussions in the following paragraph. (2), the shape of magnetic structures is not necessarily rectangular; see discussions on the "mask" in the next section.

2.1.2 The "mask"

A five-component array mask(–2:2, *i*, *j*) is used to control the simulation. The index *i* varies from 0 to nxmax+1, and *j* varies from 0 to nymax+1, where "nxmax" and "nymax" are the cells number of the lattice in the x and y direction.

The simulation is performed over the whole sample, with *i* from 1 to nxmax and *j* from 1 to nymax (a little "smaller" than the mask). If a cell (*i, j*) lies inside the magnetic structure, we assign mask(0, *i, j*)=1; otherwise if it is outside the magnetic structure, we set mask(0, *i, j*)=0. This component of the mask array is multiplied with relevant magnetic quantities so that the magnetic sample and non-magnetic areas are distinguished, and the electromagnetic boundary conditions are met simultaneously.

The other four components of the mask array are used to control the calculation on the exchange interaction in a similar way as mask(0, *i, j*), see the next section and the subroutine "*derivs*" in Appendix I for details.

Generally speaking, any non-uniform or localized calculation can be treated in the same way, just by adding more components to the mask array, as shown in Fig.(2.2). For example, if a small recording head is put close to the magnetic sample, the external field is non-uniform across the surface (localized excitation). Another example is in magneto-optic recording, switching is assisted by localized heating, and the thermal fluctuation must be calculated in a non-uniform way. We can add two or more components to the mask array to take such effects into account. This capability (to add "customized features") is what most strongly justifies continuing to work on our own codes rather than converting to commercial or publically available packages.

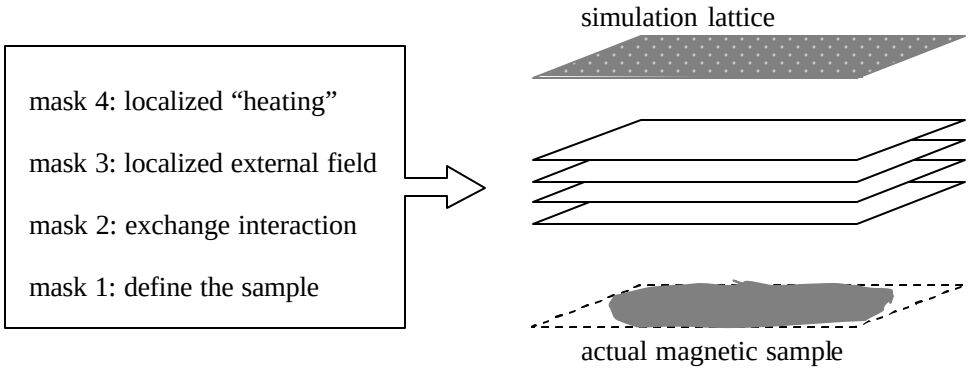


Fig.(2.2) Schematic layout of the mask array. Only mask1 and mask2 are actually used in current simulations; mask3 and mask4 are under development for future projects.

2.1.3 Calculation of different terms of the effective field

As discussed in section 1.2.2.1, the spherical coordinates are used in the simulation to calculate the effective field. External field and demagnetizing field are calculated in Cartesian coordinates first and then converted to spherical coordinates using Eq.(1.22).

Exchange interaction: Direct calculation in the spherical coordinates is performed using Eq.(1.32). Only nearest neighboring cells are taken into account [10, 11], *i.e.* the exchange term for the cell (i, j) consists of four pieces, from the cells (i, j+1), (i+1, j), (i–1, j), (i, j–1), respectively. The “+1” and “–1” are implemented by assigning mask(–2, i, j)= –1, mask(–1, i, j)= –1, mask(1, i, j)=1

and mask(2, *i*, *j*)=1. At sample edges where the number of interacting cells reduces, one or two components of the mask array above is assigned zero to meet the boundary conditions. At non-magnetic sites, all four components are zero so that no exchange term is calculated. See the subroutine "*derivs*" in Appendix I for details.

Demagnetizing field: Eq.(1.35) is the general formula to calculate the demagnetizing field, and we use the FFT based algorithm [12, 13] to implement the calculation in the Cartesian coordinates. Two subroutines "*DZFFT2D*" and "*ZDFFT2D*" from SGI/Cray Scientific Library (SCSL) are used to do the transformations. In cases when the thermal fluctuation term is involved in the simulation, the calculation is inserted in the demagnetizing term [13, 16]. See the subroutine *"hdem"* in Appendix I for details.

Anisotropy term: We consider only uniaxial anisotropy, and use Eq.(1.29) to calculate it [1]. Since this equation is pretty simple, we use either spherical or Cartesian coordinates to calculate it. See the function *"hanis"* in Appendix I for details.

External field: The contribution from uniform external fields is calculated in the Cartesian coordinates, and then added to the demagnetizing term before they are converted into spherical coordinates together. See the subroutine *"hfun"* in Appendix I for details.

[1] See more discussion on the formula of anisotropy calculation in section 2.4.3.

2.1.4 The ODE integrator

The summation of different terms above (*i.e.* the total effective field) is substituted into Eq.(1.20) and by integrating this differential equation the movement of the magnetization is obtained. The integrator that we normally use in the FORTRAN77 code is the "RKSUITE" package released by R.W. Brankin *et.al* [17]. In section 2.3 I present my work on an upgraded code written in FORTRAN90, where an integrator using Cash Karp's embedded Runge-Kutta algorithm [18] is used. "RKSUITE" also has a newer version written in FORTRAN90 [19], and to plug this into my upgraded simulation code is one of my future tasks.

2.2 Simulation code structure

In this section the basic elements of the micromagnetic simulation are discussed. The computing and network services (CNS) of University of Alberta hosts five SGI Origin parallel supercomputers. Our group uses one of them, "Aurora", which has 44 processors and 11.75Gb RAM. Users submit and monitor their programs on Aurora through the Portable Batch System (PBS). PBS also enables auto-resumption of the jobs in case a running job is suspended by the system.

I worked on more than one version of the simulation codes. They are written in either FORTRAN77 or FORTRAN90, or they use different kernel integrators. However, the basic ideas are the same, sharing the schematic structure in Fig.(2.3)).

The main program "*sim2d*" works as a universal driver. Its purposes are to: (1), communicate with Aurora through PBS to update the simulation's status; (2), collect necessary information through certain subroutines and functions about the geometry of the sample, the initial magnetic configurations, random numbers, *etc.*, and call the integration routines to let the job go; (3), output the simulated results (both temporal curves and spatial images), and store the whole workspace in specific files at the end of each iteration, so that if the job is terminated due to computer shut-down, it would be able to resume.

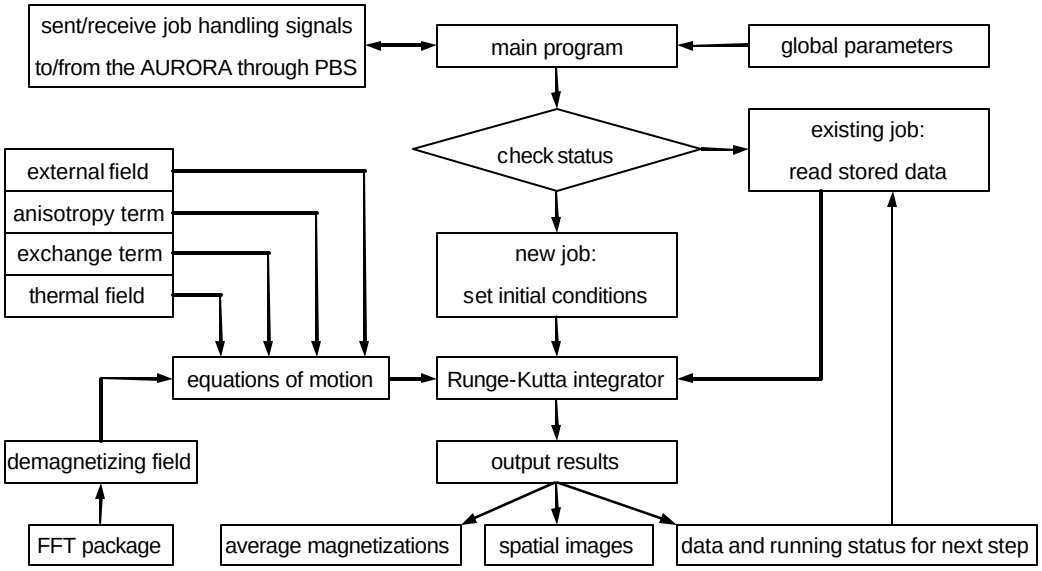


Fig.(2.3) Programming structure of the simulation codes

2.3 Upgrade from FORTRAN77 to FORTRAN90

The FORTRAN90 version doesn't change the basic algorithm of the simulation. Below I discuss two major modifications that improve the performance.

2.3.1 Code modularization

In the old FORTRAN77 code, the main program contains many definition statements for variables and parameters, most of which are either global or local quantities used by the subroutines or functions other than the main program itself. In FORTRAN90, this programming style might even be rejected by the compiler. I have to put local variables into specific subroutines or functions and encapsulate them. Only global variables and explicit parameters can be visible to other parts of the program. Global settings are put into a "MODULE" structure, which is a new feature of FORTRAN90 capable of replacing the "INCLUDE" files in almost every FORTRAN77 program. This module is then quoted by every subroutine and function using the "USE" statement. Fig.(2.4) gives a comparison of the programming style between two codes.

comparing objects	**FORTRAN77**	**FORTRAN90**
global settings	"INCLUDE globals.inc"	"USE globals"
security	unsafe: *e.g.*, the file " include.inc" is not compiled, making potential errors on variable definition	safe: *e.g.*, the usage of "IMPLICIT NONE"
Readability	bad: with "COMMON" statements	good: data encapsulation

Fig.(2.4) Different features of FORTRAN77 and FORTRAN90

2.3.2 Implementation of the integrator

The FORTRAN77 code uses a Runge-Kutta package which contains obsolete features like "COMMON", "EQUIVALENCE" statements. I proceeded to make another integrator based on Cash Karp's embedded Runge-Kutta algorithm [18]. It is in the FORTRAN90 programming style, and has better conformity with other parts of the program. The performance of FORTRAN90 integrator is discussed in the following section.

2.4 The code benchmarking

Comparison with experiment is the ultimate means whereby a model and its numerical implementation can be validated[2]. On the other hand, various mature micromagnetic simulation codes now exist and have been tested on a series of standard problems (*e.g.* SP1~SP4 from the National Institute for Standards and Technology). Therefore, it is feasible to perform initial benchmarking tests against some of these other programs.

2.4.1 Description of the test problems

Our test problems are chosen from Roger Koch's [20]. In these problems, the sample is a 400(nm) long, 200(nm) wide rectangle with thickness 6.25(nm). It is

[2] Some simple idealized analytical models are also used to validate the simulation results [8, 16, 30], but they are not covered in this thesis.

divided into cubic cells with linear dimension 6.25(nm) to make a grid of 64x32x1 cells. Material parameters used are saturated magnetization $4pM_s = 10(\text{kGauss})$, exchange constant A=10^{-6}(erg/cm) and damping constant $a = 0.01$. To calculate an initial state, we start with all spins pointing in the negative y-direction (see Fig.(2.1) for reference). There is a field of 100(Gauss) in the negative y-direction and 100(Gauss) in the x-direction. The sample has a 5(ns) period to equilibrate. Then it is left in zero external fields for another 5(ns) to equilibrate. With this initial state, a zero risetime DC magnetic field pulse of 0, 50, 100, 150 and 200(Gauss) is applied along y-direction and 0, 50, 100, 150 and 200(Gauss) along x-direction for 2(ns). Then the sample is allowed to equilibrate in zero fields for 3(ns).

Comparison between Roger Koch's results and ours has been done previously. In our group's existing results, complete sets of pictures have been shown as a success of the FORTRAN77 codes [13]. My benchmarking tests results, from both the FORTRAN77 and upgraded FORTRAN90 code, are compared with a commercial package from M. R. Scheinfein [3].

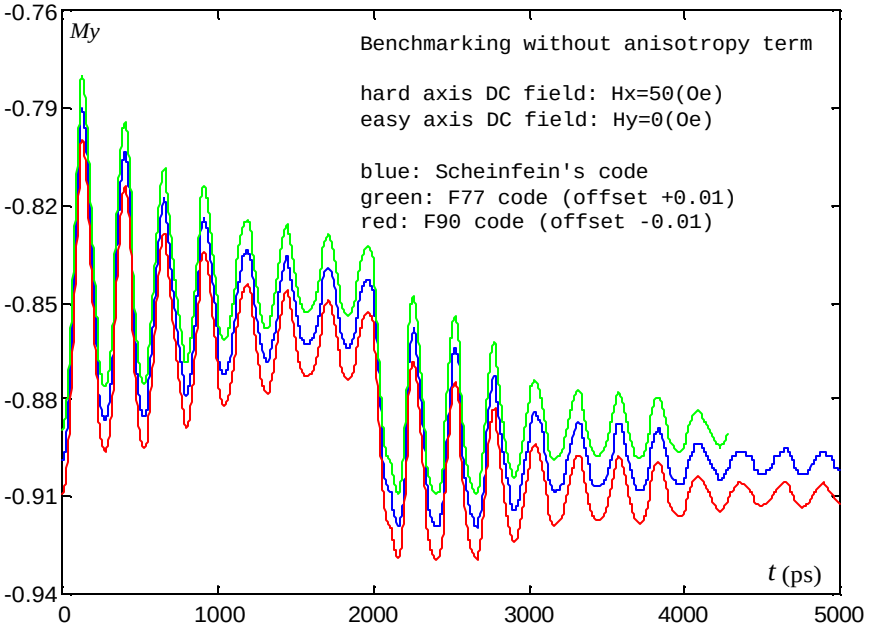
2.4.2 Benchmarking without anisotropy term

The standard problem described above does not include the anisotropy term, so I first switch off the anisotropy term in the simulations. One of the benchmarking results is presented in Fig.(2.5), where the DC magnetic field applied in x-direction is 50(Oe) and y-direction is 0(Oe). Fig.(2.5b) gives the relative deviations between the Scheinfein's code and our F77/F90 codes, which are both

[3] Online information: http://llgmicro.home.mindspring.com/

well controlled. It also shows that the F77 code exhibits a better error property than the F90 counterpart, because the F77 RKSUITE ODE solver, as a successful package, involves specialized routines to optimize the step-size and control the error, while my Cash-Carp algorithm integrator in F90 code uses a much simpler scheme to do so. Nevertheless, as long as the relative errors remain in the order of ~0.1% as shown in Fig.(2.5b), the local error property would not become a problem.

(a)



(b)

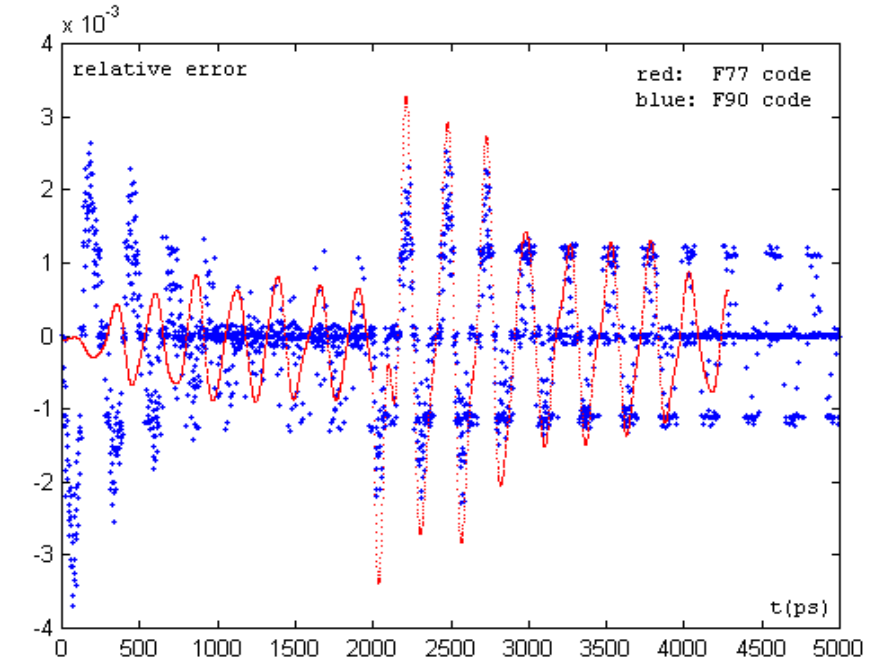


Fig.(2.5) (*color*) Benchmarking result when anisotropy term is ignored. (a), The temporal evolution of magnetization simulated by the FORTRAN77 and FORTRAN90 codes match the result from Scheinfein's code so perfectly that I have to put small offset on the y-axis data to make them visible. (b), relative deviations between our codes and Scheinfein's, based on a cubic spline interpolation.

2.4.3 Benchmarking with corrected anisotropy term

When uniaxial anisotropy term in the old FORTRAN77 code is inserted to do the benchmarking, disagreement arises, as shown in Fig.(2.6). The problem lays in the formula, where an easy-axis-favored magnetic field is phenomenologically treated as the anisotropy term. While the true anisotropy field should be calculated in 3D space, as Eq.(1.28-29) have shown. I modified the formula so that the anisotropy term is calculated in spherical coordinates, as desired in Eq.(1.29). The success of this correction is shown in Fig.(2.7), where the FORTRAN90 code matches Scheinfein's perfectly again.

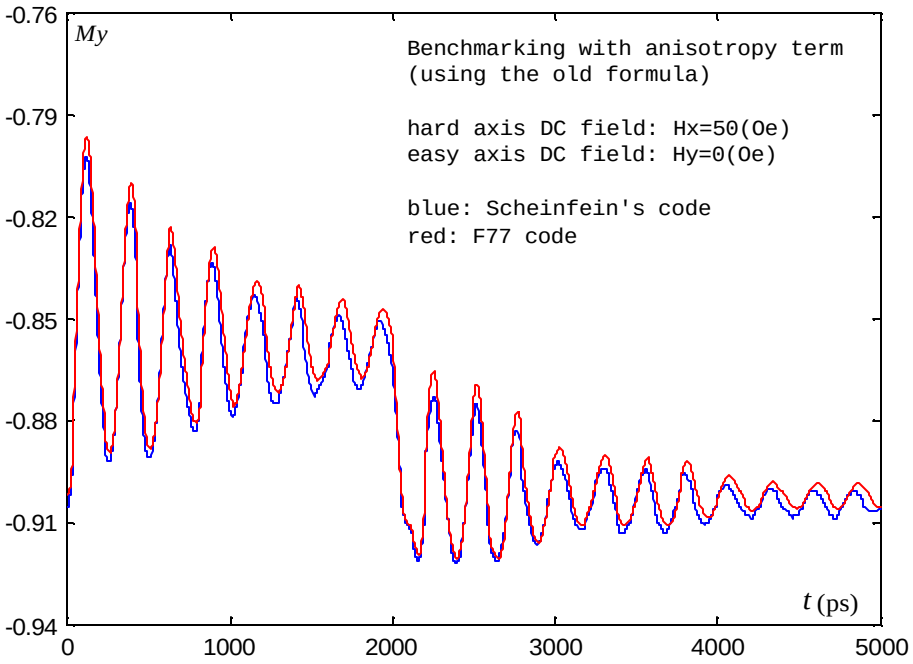


Fig.(2.6) (*color*) Benchmarking result when anisotropy term is included and old formula is used. The disagreement is accumulating and becomes significant.

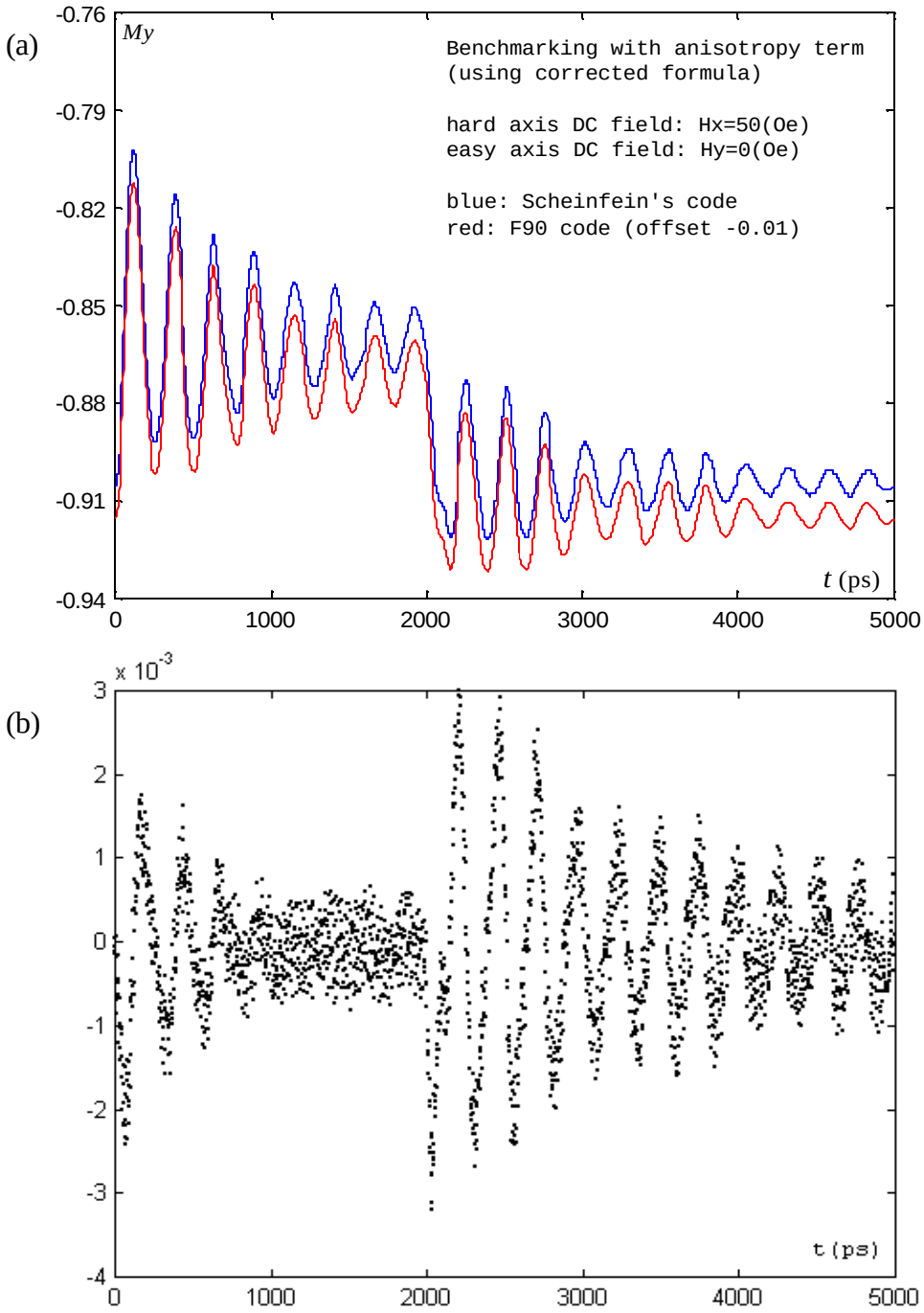


Fig.(2.7) (*color*) Benchmarking result when anisotropy term is included and the corrected anisotropy formula is used. (a), average magnetization along easy axis, where offset has to be employed to make two curves visible. (b), relative deviations as a function of time computed by cubic spline interpolation.

2.5 Visualization of results

I made a graphic user interface (GUI) to display the simulated images. It is written in Matlab graphic user interface design environment (GUIDE), and shown as a regular Matlab figure, see Fig.(2.8).

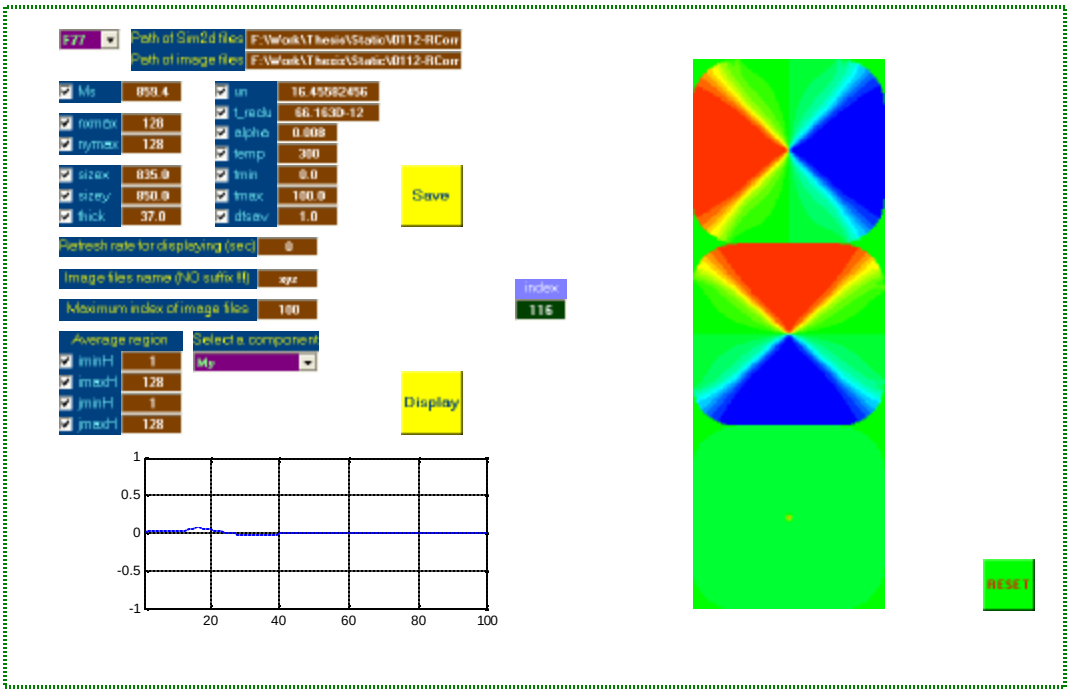


Fig.(2.8) The GUI.

On the GUI board, we can manually assign values for the key parameters used in the simulation, so that we don't need to open those FORTRAN files and change them one by one. Another important usage is that the time-evolution of the magnetization state can be continuously displayed like a movie. Also, the average value of any magnetization component can be plotted as a function of time. These features are bundled into one window and give great convenience for data analysis.

Two schemes are commonly employed to display color maps showing the distribution of magnetization. The first is to display the x/y/z-components of the magnetization vector separately. Since the vector's magnitude is a constant M_s, we can use normalized magnetizaiton M_x/M_s, M_y/M_s, and M_z/M_s. To get color maps, we use a color bar from -1 to +1, represented by blue to red, respectively (see Fig.(2.9a)). The second scheme is for those problems in which the magnetization is mostly confined in-plane, and the f angle is the only variable determining the orientation of the magnetization. Another color bar is used to represent angles from 0 to $2p$ (see Fig.(2.9b)). In this thesis, both methods will be used to display simulation results.

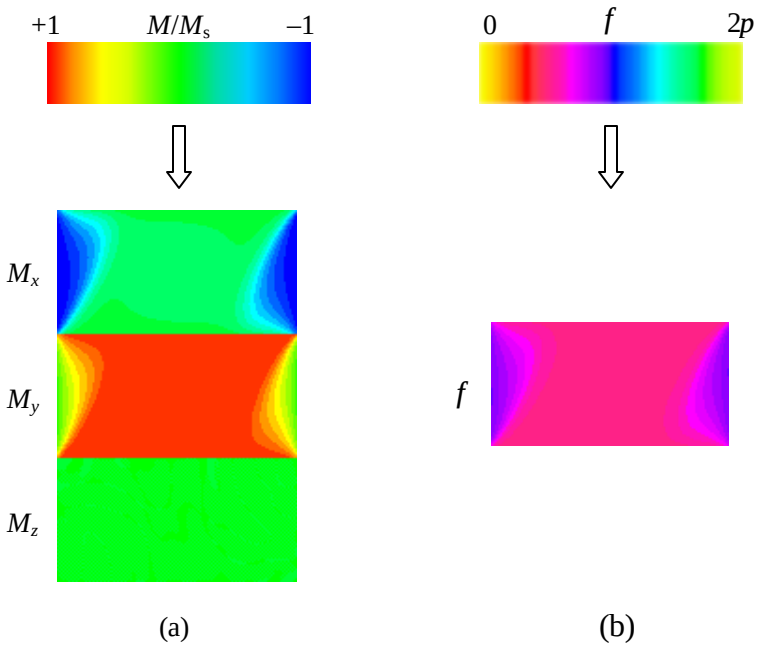


Fig.(2.9) (*color*) Two schemes to display the magnetization. (a), display three components using a color bar from –1 to +1; (b), display the f angles using a color bar from 0 to $2p$ (the reference color wheel is shown in Appendix IV).

The GUI consists of a number of Matlab functions. Some of them are provided by Matlab itself, and I wrote the others – for example, "*RGBshow.m*" converts data file to colored image in Red-Green-Blue format; "*vectormap.m*" displays a vector map for in-plane magnetization distribution. The Matlab codes are included in Appendix III.

Chapter Three

Quasi-static problems

Equilibrium states of ferromagnetic patterned particles are studied in this chapter. These particles are of sub-micrometer size typically, with thickness 10-50(nm). Different quasi-static states will be presented, with comparisons between experimental measurements and simulated images. Selection of damping constant in the simulation will be discussed. Energy hierarchy is considered as the key to understand the different stabilities of these states. Furthermore, magnetic interactions between neighboring particles will be discussed. Simulation results confirm the existence of this long-range coupling, and show that the forming of quasi-static states, to some extent, is a probabilistic phenomenon.

3.1 Introduction

The first step towards investigating the dynamic behaviors of a magnetic sample is to study it under static conditions. Experimentally, the technique of off-axis electron holography offers the highest spatial resolution for quantitative magnetic imaging. It achieves magnetic sensitivity by simultaneously measuring the amplitude and the phase shift of the electron wave passing through the sample [21, 22, 23, 24]. Magnetic induction at one site can be derived from a certain form of

integration of the phase shift along the direction of the incident beam[25]. Thus, quantitative information of magnetization at very fine spatial resolution can be obtained [22].

It is important to compare measured data with numerical results computed by micromagnetic simulations. Specific comparisons between quantitative electron holography and numerical simulation have only been attempted a very limited number of times [23]. The problems discussed in this chapter are aimed at just this.

3.2 Equilibrium magnetic states

3.2.1 Introduction to magnetic states

Consider a magnetic thin film fictitiously divided into a large number of finite elements, each of which can be treated as a magnetic "macrospin". Both the external field and the magnetic coupling with other parts of the sample contribute to the macrospin's energy. The total energy of the sample is the sum (or integration) over all macrospins. Any magnetization distribution of the sample yields one total energy value, and we can call this configuration a magnetization state [9, 38]. Fig.(3.1) shows several states; they are not necessarily real, because some of them are simply useful to initialize a simulation problem.

An infinite number of magnetic states – real or unreal – construct a continuous energy "landscape". An equilibrium state that is observed in practice has to at

least stay in a local energy minimum, where small perturbations around the configuration increase the total energy and consequent dissipating motions tend to return the stable state. If the energy change is large enough, the energy barrier between two equilibrium states may be overcome and phase transitions may occur. Effects of energy will be discussed more intensively with a semi-quantitative approach in section 3.2.6.

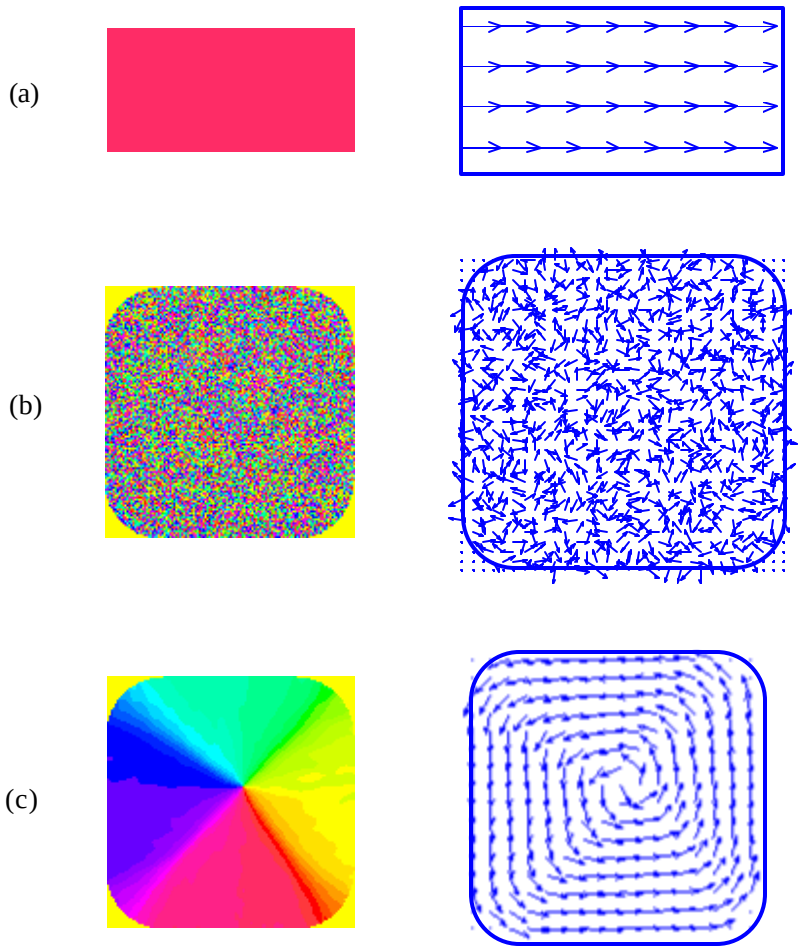
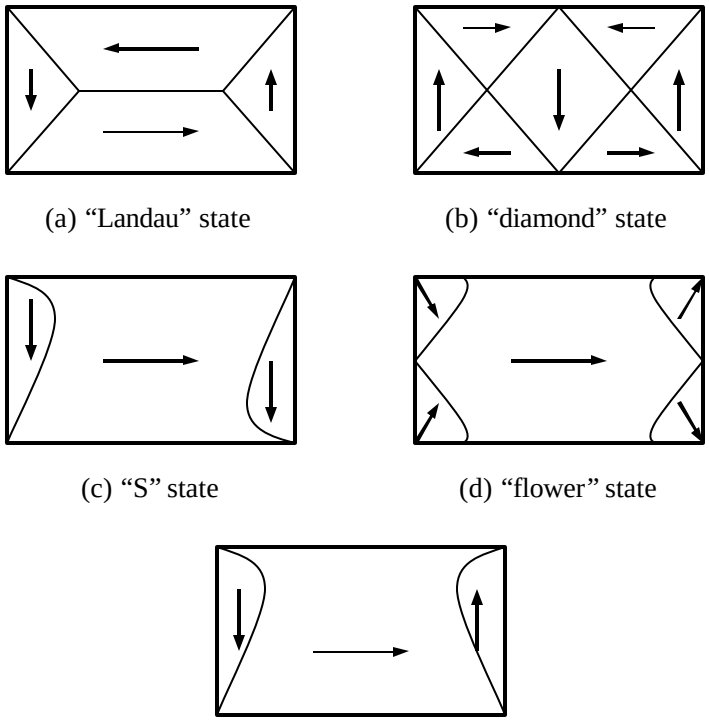


Fig.(3.1) (*color*) Examples of magnetic states plotted with Matlab. The left column shows the color-map profiles of the in-plane angle, and the right column shows corresponding vector maps. (a), uniform distribution; (b), random in-plane distribution; (c), a vortex state.

Intensive experiments and theoretical modeling on thin film's equilibrium states have been achieved through efforts of many research groups [23, 26, 27, 38]. There are several "favorite" states. Fig.(3.2) gives a list of them for a rectangular structure [9].



(a) "Landau" state (b) "diamond" state

(c) "S" state (d) "flower" state

(e) "C" state

Fig.(3.2) Some of the equilibrium states for a rectangular sample. Arrows outline the magnetization's orientation.

3.2.2 Flux-closure magnetic states

When no external field is applied, the equilibrium magnetic states are dictated by

minimizing the magnetic "self energy" of the thin film. Usually the "Landau" and "diamond" states are observed in remanence, both with some sort of vortex structure. I ran simulations on rectangular Permalloy thin films and successfully reproduced these flux-closure states, shown in Fig.(3.3), where the main geometric and magnetic parameters are also given.

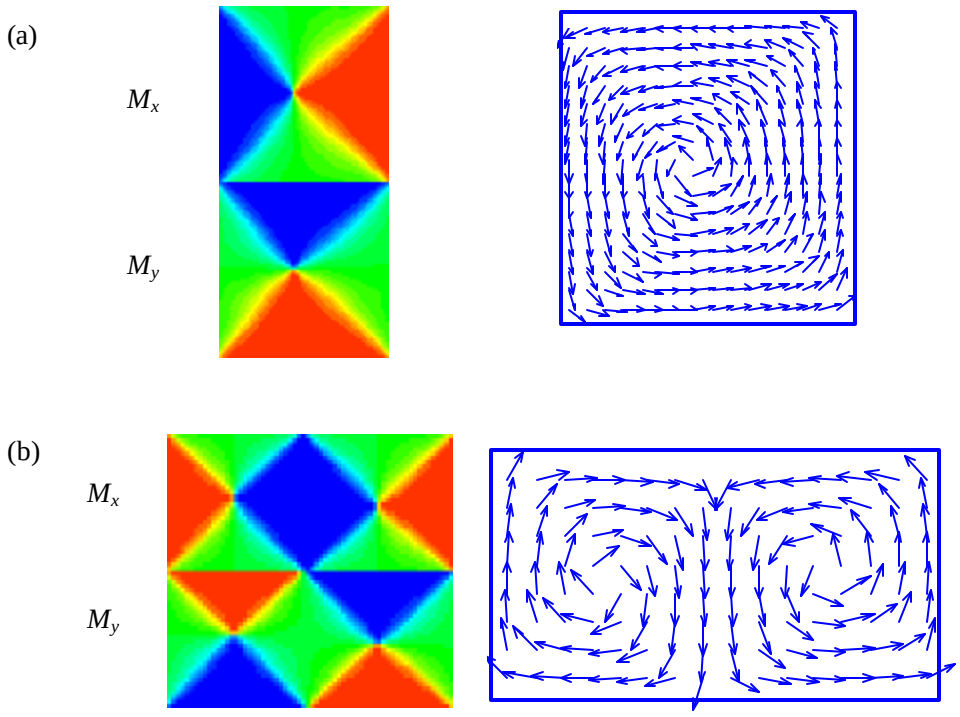


Fig.(3.3) (*color*) Simulated flux-closure structure of Permalloy rectangular platelets. Only M_x and M_y components are displayed because M_z is determined by the restriction $M_x^2+M_y^2+M_z^2=M_s^2$. (a), the "Landau" state of a 400x400x10(nm^3) particle. (b), the "diamond" state of a 500x1000x15(nm^3) particle. No external field is applied in both cases, and a random distribution is employed as the initial state. The damping constant is 0.008, as it is for Permalloy.

3.2.3 High remanence states: “flower” and “S”

External disturbance, such as a strong enough in-plane field, would break a film’s vortex structure and makes a fairly uniform magnetization distribution. Then the magnetic platelet can be treated as an information bit standing for “0” or “1”, which plays a key role in technologies of data storage. A large number of studies are based on this application.

Depending on the field’s direction, different equilibrium magnetic states will be obtained. If the field is exactly parallel to the easy axis, “macrospins” in the film will mostly align to this direction except for those close to the two ends. The demagnetization term forces these spins towards parallel to the short sides (*i.e.*, along the hard axis direction), and form some kind of small sub-domains. This state is named “flower” state after its appearance, see Fig.(3.4a).

Following situation occurs more generally, when the in-plane bias field is not parallel to the easy axis. The sub-domains in the “flower” state will then reassemble themselves with the easy-axis-symmetry broken, eventually leading to the “S” state, shown in Fig.(3.4b). The “S” configuration has higher stability, because a transverse bias field will break the “flower” state and reach the “S” state as the new equilibrium (the dynamic evolution is illustrated in Fig.(3.4c)).

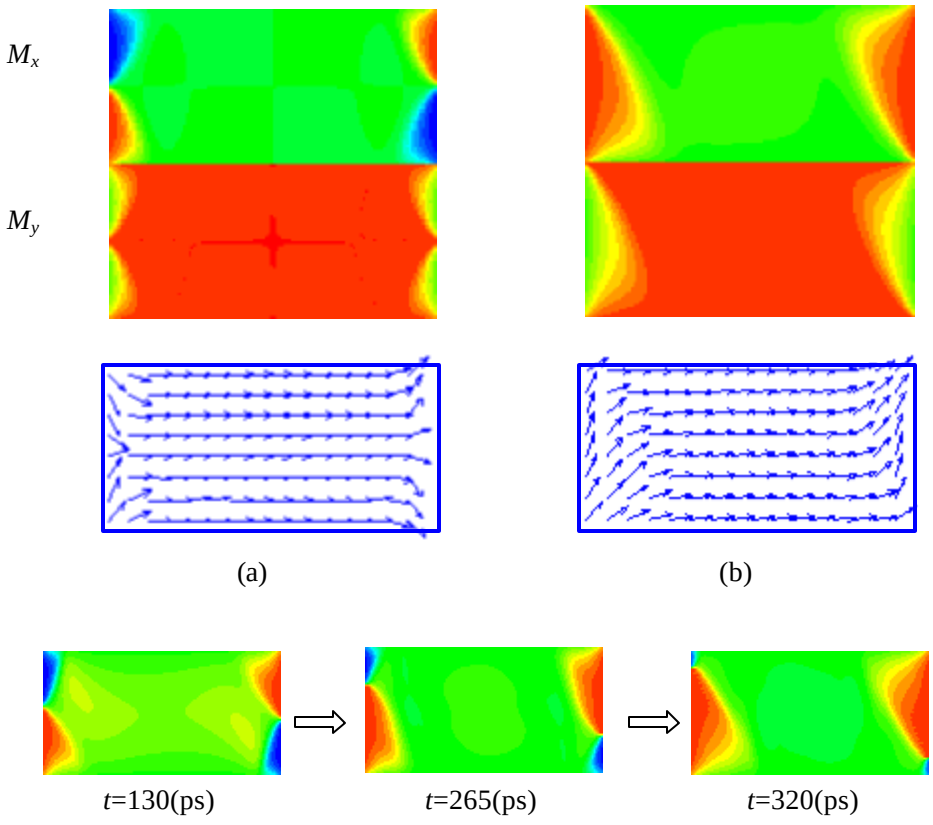


Fig.(3.4) (*color*) Simulated equilibrium states with an in-plane bias field. The initial state is a random distribution. The damping constant is set to be 0.008. The size of the sample is 500x1000x10(nm^3). (a), the "flower" state, $H_{x\text{-bias}}$=0 and $H_{y\text{-bias}}$=60(Oe); (b), the "S" state, $H_{x\text{-bias}}$=40(Oe) and $H_{y\text{-bias}}$=60(Oe); (c), dynamic evolution from the "flower" state to the "S" state, by setting the former as the initial state (t=0) and then applying a transverse field suddenly (*i.e.* $H_{x\text{-bias}}$=40(Oe) when t>0). Only M_x component images are shown to save space.

3.2.4 Distortion: the "C" state

I ran a series of simulations on a Permalloy microstructure of dimension 500x250x10(nm^3) to investigate how its quasi-static state depends on different bias field configurations. Three typical situations are shown in Fig.(3.5).

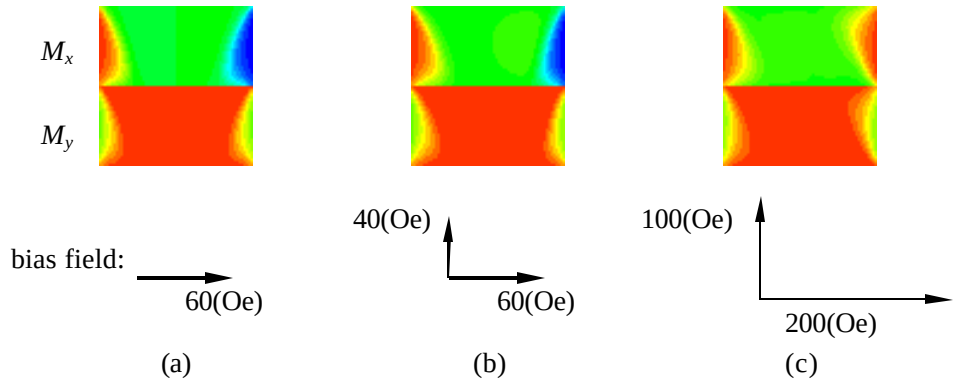
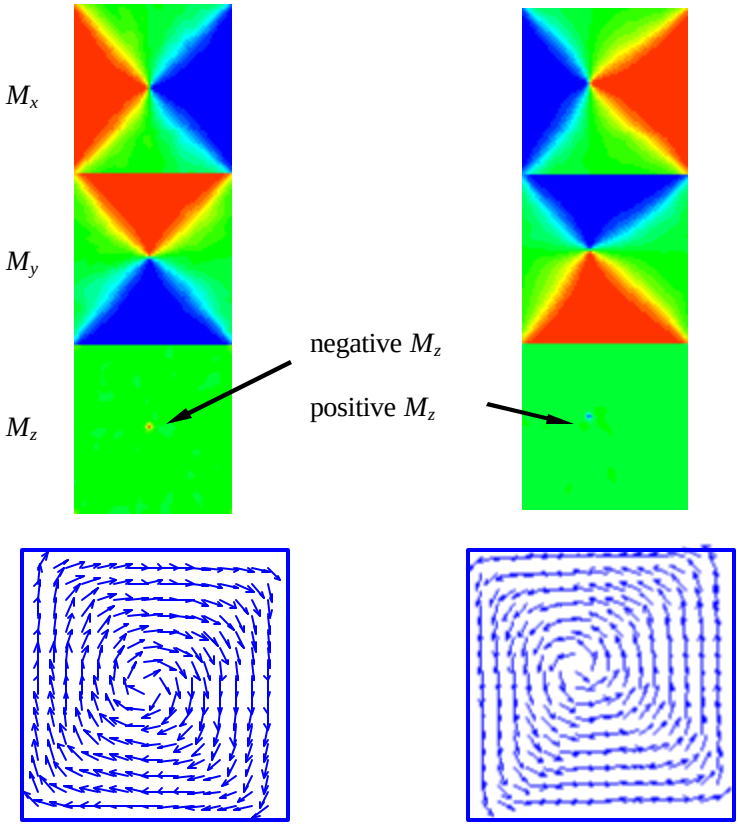


Fig.(3.5) (*color*) Appearance of the "C" state in a Permalloy rectangular particle with dimension of 500x250x10(nm^3). The initial state is a random distribution and the damping constant 0.008 is used in the simulations. (a), easy axis bias field 60(Oe); note the asymmetry against the easy axis. (b), add a transverse bias field 40(Oe), and there is not much difference in comparison to (a); (c), finally the "C" state is stressed into the symmetric "S" state when a much larger bias field is applied. It is the hard axis field that causes the anti-aligned "C-domain" to flip.

When bias fields are small (case (a) and (b)), the effects of the sample's edges and corners become significant and a "C" state appears instead of "flower" state or "S" state. When the fields are strong enough (case (c)), the "S" state is restored by the dominance of external fields (mainly along the hard axis) against the shape effects.

3.2.5 Out-of-plane field

When a weak out-of-plane bias field H_z is applied, it would not be able to change the film's equilibrium state, since the material's crystalline structure and the external field applied during the film's growth are believed to generate strong anisotropy and bound all magnetization spins in-plane [10]. This is also verified by experiments and simulations. Fig.(3.6) shows an interesting but reasonable phenomenon: opposite direction of H_z will result in opposite orientation of the vortex core. Certainly, this is not the only reason to explain why people observe differently spiraling vortices, but one could at least know its significance.



(a) H_z= +260(Oe) (b) H_z= –260(Oe)

Fig.(3.6) (*color*) "Landau" states when out-of-plane bias field H_z=260(Oe) is applied. The size of the particle is 400x400x10(nm^3); no in-plane field is applied. A random distribution is set to be the initial state and the damping constant 0.008 is used in the simulations. H_z won't break the vortex structure, but different field directions will make opposite spiraling directions of the vortex – see the vortex cores in different colors in spatial image representing for M_z .

If H_z is very strong, the magnetization in thin films will be forcibly pulled away from in-plane position. Simulations show that it will form some flux-closure states, see Fig.(3.7). In these simulations, an "S" initial state in no field space is

chosen. Out-of-plane fields with different magnitudes are then applied suddenly to pull the sample to a new equilibrium. There's no major difference even when H_z is as strong as 1200(Oe). Keep increasing the out-of-plane field and remarkable changes eventually occur. (c) and (d) of Fig.(3.7) show a "diamond" state and a "Landau" state, respectively. Not like those in section 2.2 of this chapter, the macrospins no longer stay in-plane now (note the light-blue color in M_z profiles). The strong external field gives a big level-up to the sample's energy hierarchy, leading to transitions between different equilibrium states. The significance of energy will be discussed more intensively in the next section.

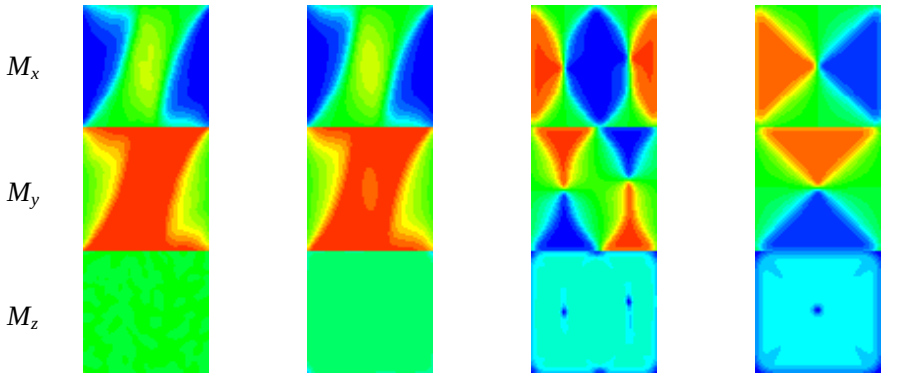


Fig.(3.7) (*color*) Appearance of flux-closure equilibrium states when the out-of-plane bias field is getting stronger. The size of the particle is 400x400x15(nm^3), and no in-plane field is applied. A " S" state is set to be the initial state – just what is shown in (a). The damping constant is 0.008. The M_z profiles shown in (c) and (d) indicate the significant effect of strong H_z.

3.2.6 Energy hierarchies of magnetic states

The total energy of a magnetic state is a valuable parameter to identify the selectivity and stability among different states. There are an infinite number of magnetic states (see discussion in section 3.2.1) with all possible energies, and they make some sort of energy "landscape". Those states in local energy minimums are more stable, and they turn out to be the equilibrium states. Between them are energy "barriers", which can be overcome by pumping extra energy into the system, so that transitions between different equilibrium states may occur.

To calculate the energy of equilibrium states depends on so many aspects that a general conclusion is hard to make. Different element sizes, magnetic parameters or numeric algorithms may lead to different relative energy between two states. Below I quote some results from two groups to show this complexity.

J. Miltat *et al*. [9] did some numerical calculations on a Permalloy rectangular platelet with dimension of 500x250x10(nm^3). In zero external fields, the energy hierarchy reads 0.01695 for the "Landau" state and 0.02086 for the "diamond" state (in units of $M_s^2V/2$ or $\mu_0 M_s^2 V/2$ in SI units).[4] Thus, the "diamond" state lies on a higher energy level (but still in a "local" energy minimum). Calculations also show that the "S" state has an energy hierarchy of 0.01910, significantly lower than the "flower" state's 0.02181. This explains the transition from the "flower" state to the "S" state shown in Fig.(3.4c). It is also possible to see transitions between them and flux-closure states, which has been shown in the last section.

[4] In this simulation, the lattice dimension is 64x32, so V=500x250x10/64/32=610.35(nm^3)=6.1x10^{-27}(m^3); for Permalloy M_s=860(emu/cm^3)=860(kA/m), the energy unit is 0.5x $4p$ /10^7(N/A^2)x860^2($10^6$$A^2$/$m^2$)x6.1x$10^{-27}$($m^3$)=2.83x$10^{-21}$(J).

W. Rave and A. Hubert [38] studied the "standard problem 1 (SP1)" which deals with a 2000x1000x20(nm^3) thin-film element. From this relatively large sample, they got a total energy density of 0.00484 for the "Landau" state and 0.00453 for the "diamond" state (in unit of $\mu_0 M_s^2/2$), which means the "diamond" state has even lower energy than the "Landau" state. They also showed that the "S" and "C" states have almost the same energy density of 0.00865, higher than the "flower" state's 0.00980.

3.3 Selection of the damping constant

Using the LLG approach to calculate the equilibrium states, an artificially assigned (usually much larger than the truth) damping constant is often selected to bring the system into equilibrium more quickly, and shorten the simulation's runtime [8, 23, 30]. Basically this trick is successful, but in some cases, different damping constants do make difference, and results in fakes on judgment of the sample's equilibrium states. I present a test simulation here to show this.

General information about the sample is described in Fig.(3.8). A uniform distribution is set as the initial state[5], and a strong out-of-plane field 3600(Oe) is applied suddenly when t=0. In the case $a = 1$, the simulation converges quickly and forms a diamond state in both particles. Then in a period of about 4(ns), the "diamond" domain in the smaller particle collapses gradually, probably due to the

[5] Usually a random distribution is set as the initial state, giving a sufficient non-equilibrium for the sample to relax and find its correct quasi-static state. In this simulation, however, a strong out-of-plane field will drive the sample away from equilibrium steeply (as shown before), so it doesn't make much difference to initialize the simulation with a uniform distribution.

long range interaction with the larger particle. Eventually it sits in the lowest energy state – "Landau" state, while the diamond state in the larger particle remains very stable. In another case $\alpha = 0.008$ – the experimental value for Permalloy samples we currently use – the simulation converges much slower, but it leads to a "Landau" state in both particles. Because of the weak damping and the magnetic coupling between two particles, the vortex cores do not quickly come to rest in the center of particles, but keep orbiting around the center slowly until t>13(ns).

In conclusion, if the damping constant is set too large in simulations, the ordering of non-equilibrium magnetization may occur locally due to rapid convergence, making more complicated domain structures. Although complicated, these structures have sufficient low energy hierarchy (*e.g.* the diamond state in Fig.(3.8a)) to be very stable under simulation conditions. If used to predict or compare with experiments, errors may occur.

In the following part of this thesis, the damping constant will be set 0.008 for all simulations on Permalloy samples, unless otherwise specified.

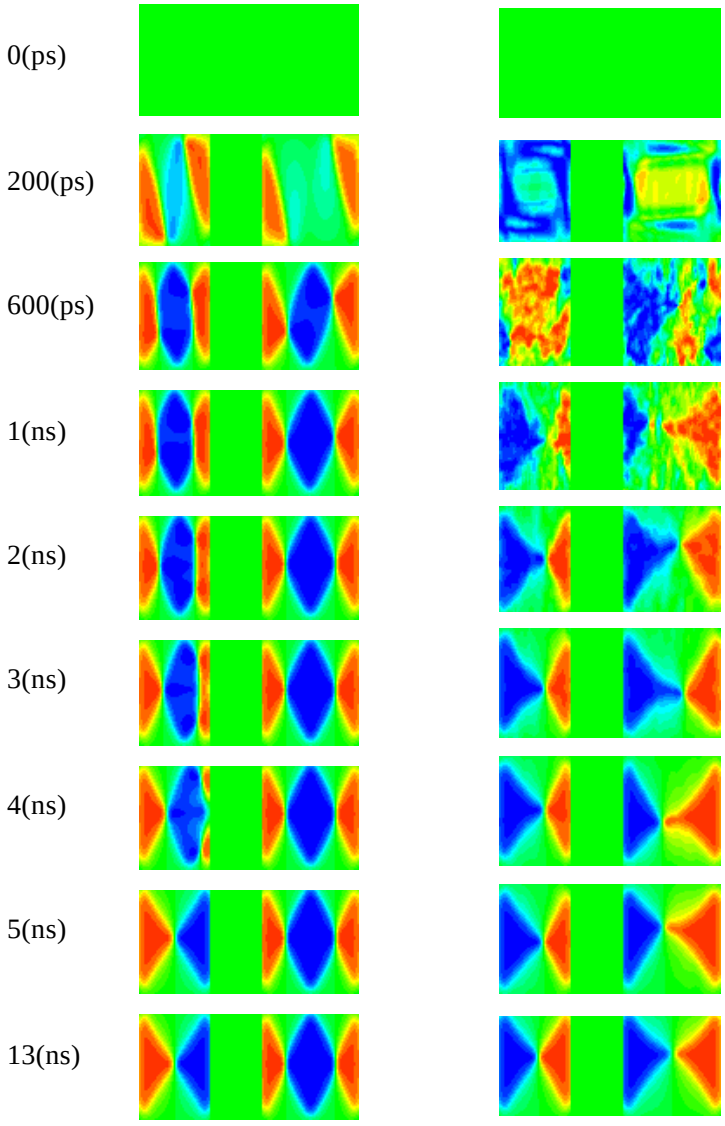


Fig.(3.8) (*color*) Simulated time frames of M_x profile showing the formation of equilibrium states. The sample's thickness is 15(nm), consisting of two rectangular Permalloy platelets in dimensions of 275x220(nm^2) and 275x300(nm^2), respectively, and the separation between them is 170(nm). A 3600(Oe) out-of-plane bias field is applied. The damping constant is set to be 1 for the left column and 0.008 for the right.

3.4 Comparison with experimental results

In the previous section, I compared some simulation results with published images from direct observations. Our group has also made some patterned Permalloy films on Si_3N_4 membrances (Belov *et. al.*), for analysis by electron holography at Brookhaven National Lab (Belleggia, Schofield, Zhu *et. al.*). In this section I present a comparison of simulation with these new measurements.

3.4.1 Individual particles

Patterned Permalloy particles are deposited on a large substrate. Separations between them may vary from hundreds of nanometers to a few micrometers. When a magnetic platelet is more than about the effective dipole size away from every near neighbor on the substrate (measured edge to edge), we treat it independently. Fig.(3.9a) is a TEM image of a square Permalloy platelet with dimension of ~860x840x37(nm^3). One should notice that the four corners have finite radii of curvature. I modified the definition of the mask array in the simulation codes, so that the modified shape due to these round corners is taken into account. Fig.(3.9b-c) shows the simulation results; similar to Fig.(3.3a), it is also a "Landau" state.

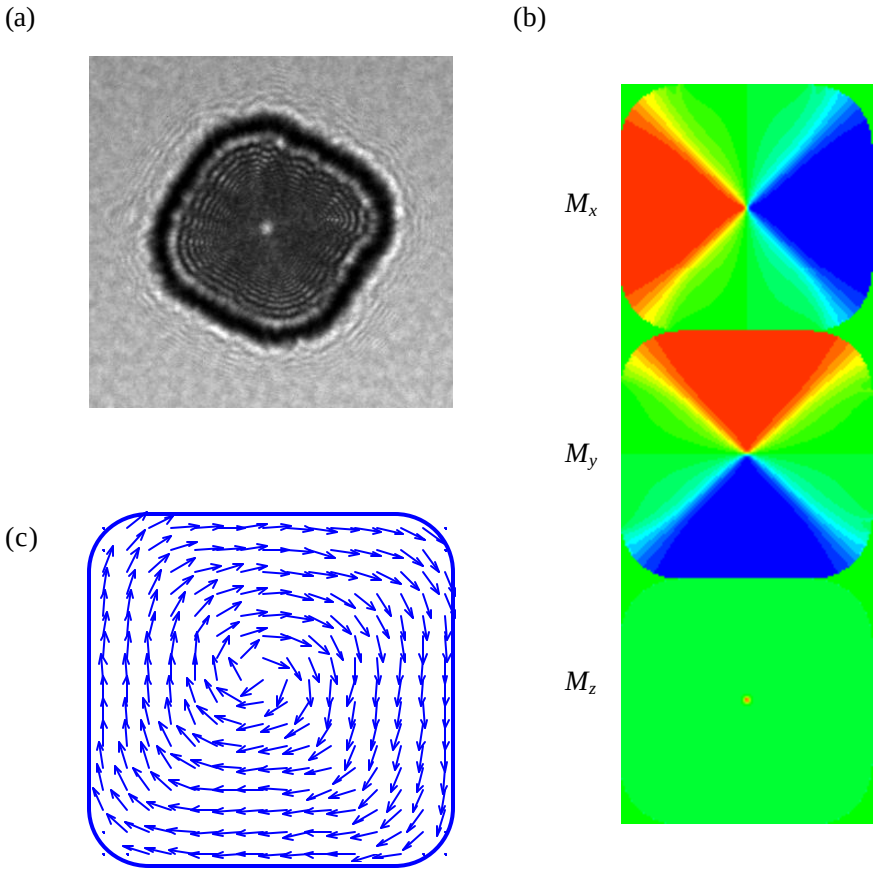


Fig.(3.9) (*color*) (a), TEM image of the Permalloy particle, 860x840x37(nm^3). There is ~260(G) out-of-plane remanent field in the Lorentz lens chamber. (b), Simulation image of the sample. Just to clarify: the corner areas are also displayed in green color because I set mask(0,*i*,*j*)=0 there. (c), the vector map of the structure.

3.4.2 Particle array – long range coupling

We also have samples with Permalloy particles aligning up and forming an array. Fig.(3.10) gives some TEM images from one of these arrays. There are five

square particles, with their sizes ranging from ~200x200x10(nm^3) to ~1.4x1.4(μm^2)x10(nm).

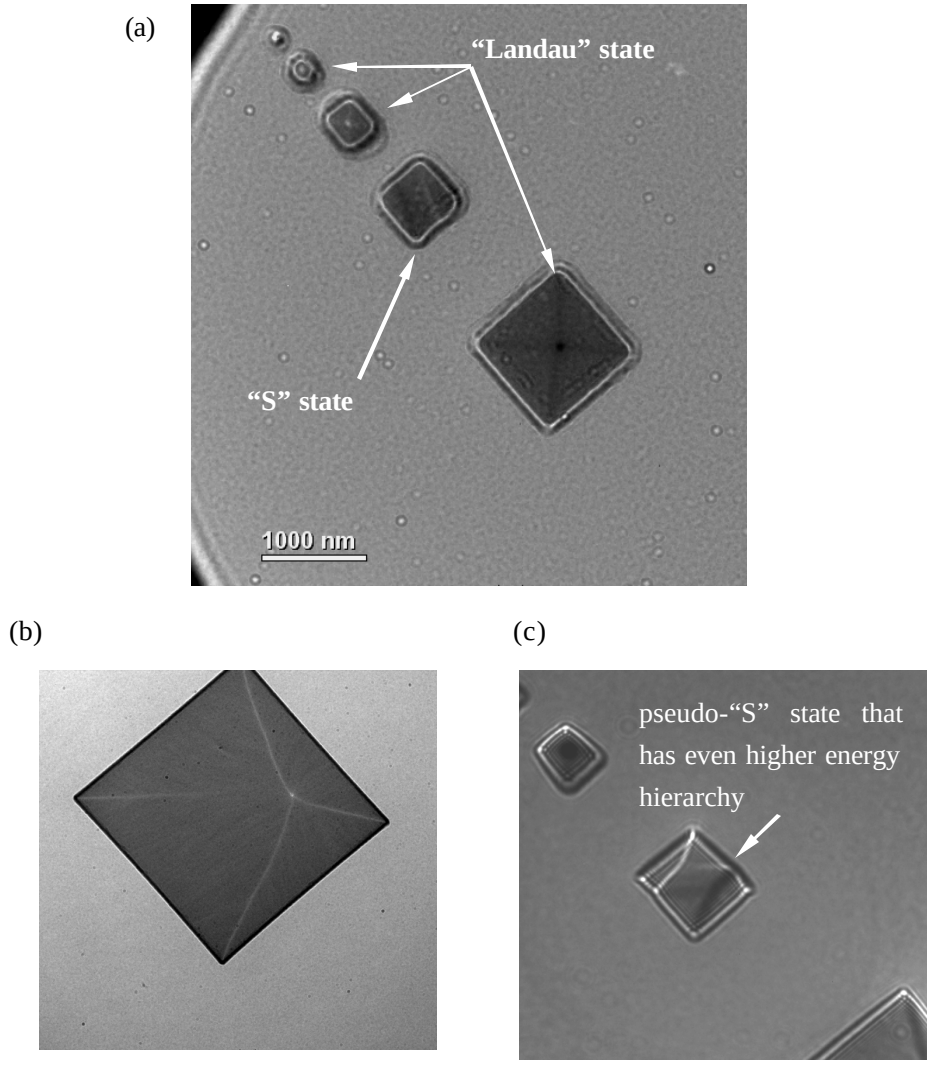


Fig.(3.10) Magnetic coupling of Permalloy particles in an array. (a), topview of the whole structure, one of them in "S" state. (b), zoom-in image of one of the particles in a "Landau" state, but the vortex core shifts off the square center, possibly due to the "attraction" by neighboring particles which produce a small in-plane field on it. (c), obtained from another imaging; one particle has turned into a state that is more complicated than "S".

The remanent magnetic field in the objective lens of the TEM is ~260(Oe) and is perpendicular to the film surface. Since there is almost no in-plane field, most of these particles are in the ground energy state with different vortex orientations, while a little strangely, one particle is in the "S" state (Fig.(3.10a)) or some distortion of that (Fig.(3.10c)).

Simulation of an array can also reproduce this diversity. It is important to keep in mind that this phenomenon is not deterministic, *i.e.* among all the particles you can't predict which one will end up in a non-ground state.

Since the diversity of the magnetic particles' equilibrium states is in nature a probabilistic problem, I designed a test simulation to investigate it in another way. I put a lot of identical particles in a big frame, set a random distribution initial state on all of them, and let them relax under identical magnetic circumstances. Fig.(3.11) shows a simulated image containing 36 particles. One remarkable thing is the "S" state can be seen in one of the particles, which means even these small magnetic particles do have a finite probability to relax into a metastable "S" state.

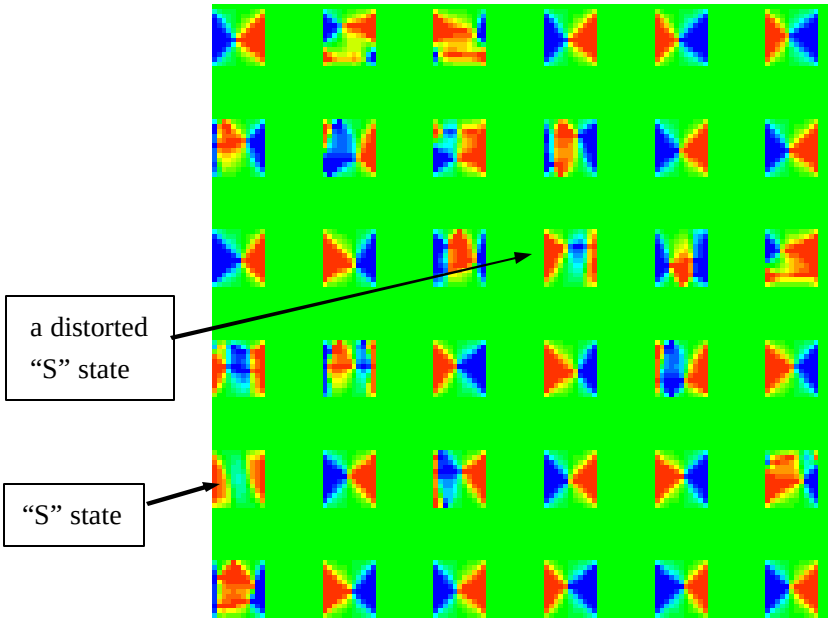


Fig.(3.11) (*color*) Various equilibrium states in a 6x6 array. Each Permalloy platelet has a dimension of 400x400x10(nm^3), and the separation between them is 400(nm). Only a 260(Oe) out-of-plane external field is applied. Many of them have been pumped up from their ground state.

Chapter Four

Magnetization reversal

Stroboscopic microscopy based on magneto-optical Kerr effect (MOKE) is introduced. It is a powerful tool to investigate ultrafast magnetization reversal dynamics. Simulations results are presented and compared with the experiments. They agree with each other pretty well, and some unconformities are discussed through test simulations, showing further that the simulations on reversal problems are successful.

4.1 Introduction

4.1.1 In-plane dynamics of ultrathin magnetic film

As discussed in Chapter One, the exchange interaction is a short-range term. Its characteristic range, known as the exchange length, is

$$L = \sqrt{\frac{A}{2p\, M_s^2}} \tag{4.1}$$

where A is the exchange stiffness. For soft materials such as Permalloy, people usually take $A = 1\times10^{-6}(\text{erg/cm})$, and $L = 5(\text{nm})$. The thickness of most thin films I discuss in this thesis is not far beyond this length, and can be treated in a

quasi-two-dimensional way [9]. In such films, the exchange interactions tend to make a uniform magnetization, and the stray field energy is minimized if the magnetization is in-plane [8, 12]. So generally, the large-angle movement of the magnetization is mostly in-plane.

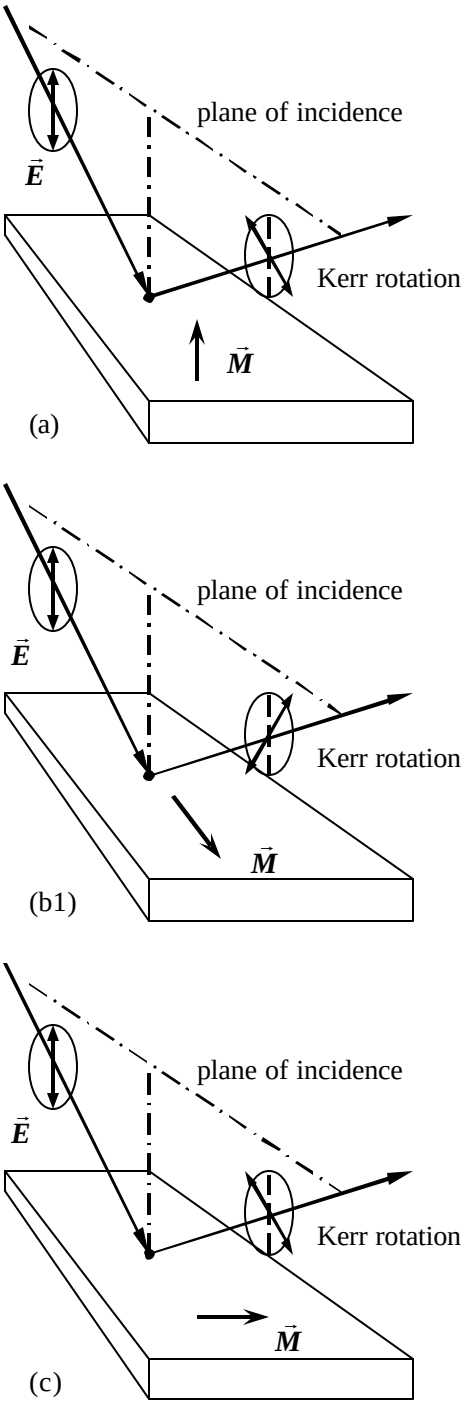
Researchers go forward on sub-nanosecond time scales and sub-micrometer spatial scales, because new tools enable this and industrial interests require advanced magnetic devices for information communication and storage, such as ultrahigh speed read/write heads, and magnetic random access memories (MRAM).

4.1.2 Measurement techniques

Stroboscopic microscopy for investigation of dynamics of magnetic materials requires combining a source of short light pulses such as a mode-locked laser, with a polarizing microscope for magneto-optical imaging, and synchronizing to repetitive magnetic response driven some form of ultrafast pulse generator [31].

Magneto-optical imaging is based primarily on the magneto-optical Kerr and Faraday effect(s). The Kerr effect refers to the changes in the intensity or polarization state of light reflected from a magnetic material. By measuring these changing optical signals, we are able to calculate the corresponding magnetic quantities, thus gain the knowledge of dynamic magnetic states of interested samples.

Depending on the relative geometry of the incoming light and the magnetization orientation in the material, the Kerr effect has three basic modes, illustrated in Fig.(4.1). For out-of-plane magnetization in the sample, it is known as the polar Kerr effect. For in-plane magnetization parallel to the plane of incidence of the reflected light, it is known as the longitudinal Kerr effect; and for in-plane magnetization perpendicular to the plane of incidence, it is known as the transverse Kerr effect [10].



Polar and transverse modes: those beams with polarization perpendicular to the plane of incidence do NOT have Kerr rotations.

Longitudinal modes: The Kerr rotations always occur, no matter what the angle between incident beam's polarization and the plane of incidence.

Fig.(4.1) Three modes of magneto-optic Kerr effect: (a), polar Kerr effect; (b), longitudinal Kerr effect, where (b1) and (b2) are different cases in which incident beam's polarization is parallel or perpendicular to plane of incidence, respectively; (c), transverse Kerr effect.

Fig.(4.2) illustrates a measurement system employing time-resolved scanning Kerr effect microscopy. A pulse generator is used to produce a chain of identical current pulses. When these pulses flow through the transmission line, an in-plane magnetic field will be induced to drive the sample out of equilibrium. The interval between pulses is fixed and long enough to let the sample return to equilibrium after each excitation.

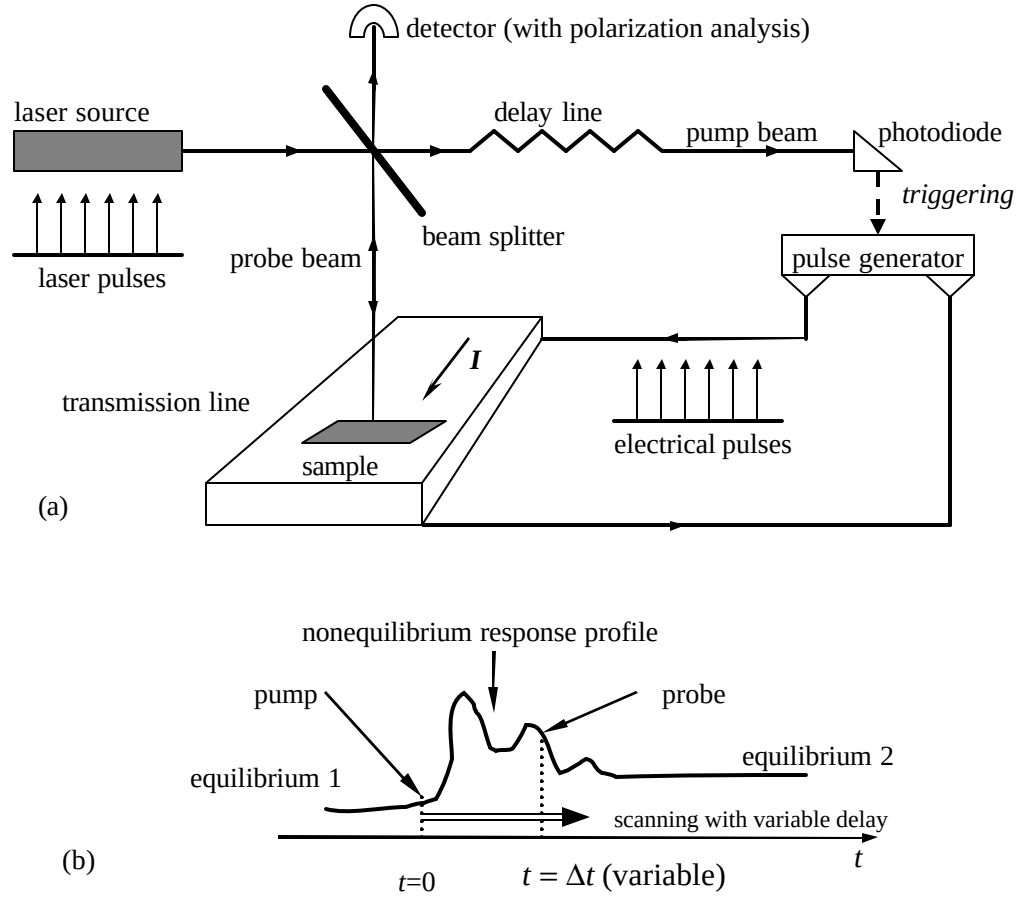


Fig.(4.2) Pump-probe measurement. (a), pump and probe beams are split in phase and guided to their respective destinations. The pump beam is used to generate the magnetic field that changes the sample's magnetic states, and the probe beam is used to measure this change. (b), the timing information is provided by the delay line setting. Repeated probing with varied time delays can measure out the whole nonequilibrium profile.

Ti-sapphire laser pulses are focused to measure the sample's Kerr signals. In the pump-probe scheme, the laser pulse is first split into two pieces, one pump and one probe. At time t=0 the pump beam triggers a photodiode to turn on the pulse generator, and the sample is excited (or "pumped") by induced magnetic field. After a short time interval Δt, the sample is probed by the probe beam. Δt, the time delay, can be varied continuously by changing optical path length, or by electronic delay of the photodiode signal.

When the incoming probe beam reflects from the sample surface, information about the magnetization components is stored in the outgoing modulated beam (the "Kerr signal"). In high numerical aperture microscopic imaging, quadrant detectors are capable of extracting the in-plane and out-of-plane Kerr signals simultaneously [12, 13]. The signals are output to a computer-controlled data acquisition system. It should be pointed out that these data represent the change of magnetization, not the absolute values, and therefore must be supplemented with separate information about the equilibrium magnetic state.

4.2 Simulation results; bias-field dependence

4.2.1 General settings

Our group has performed extensive measurements on ultrafast magnetization reversal. The sample that I'm interested in is a rectangular Permalloy ($Ni_{80}Fe_{20}$) platelet with dimensions of 2x10(μm^2)x15(nm). The sample design and external

field configuration are described in Fig.(4.3). In the 180^0 reversal configuration, the sample is first saturated in the longitudinal bias field H_l, and then a switching pulse H_s is applied antiparallel to H_l in order to flip the magnetization. In some experiments, an in-plane transverse bias field H_t is added to manipulate different reversal modes [3, 33, 34].

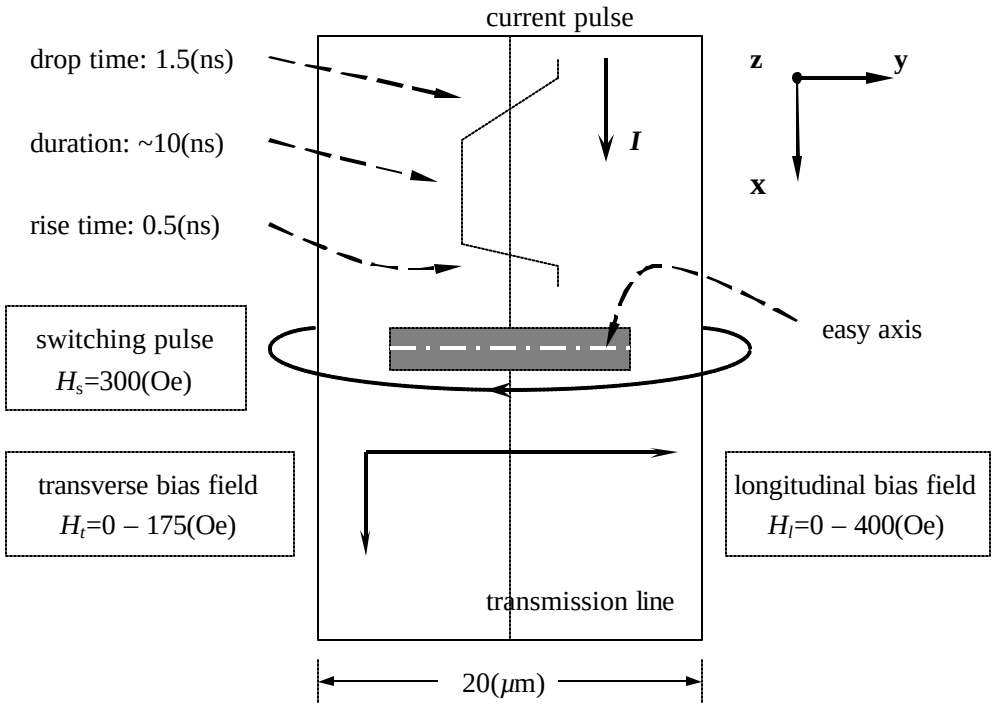


Fig.(4.3) Geometry and field configuration of the magnetization reversal experiments. Note the Cartesian coordinates shown here are different with our group's published results [3, 4], in order to keep consistent with the settings in the simulation code. This will not affect any physics results.

In the simulation, the sample is divided into a lattice with 128x512 cells. I use a uniform distribution as the initial state, *i.e.*, the normalized magnetization M_x=0, M_y=1, M_z=0 for each cell. The samples must have considerable time to reach their quasi-static states. Only the DC bias field exists during this "cool-down" period,

and it can be simulated as a quasi-static problem. The sample will rest in the "S" state, which we treat as the initial state. Then the switching pulse rises up with a linear profile (as sketched in Fig.(4.3)).

4.2.2 No transverse bias field (H_t = 0)

In this case, all external magnetic fields – the DC bias field and the transient switching field – are parallel to the sample's easy axis.

First let's take H_l=60(Oe), H_s=-300(Oe). Fig.(4.4) illustrates the switching process. It starts from the two ends. The edge domains grow and meet in the center of the film, and then expand along the hard axis direction to reach saturation. The sample's magnetization can be plotted in a time-resolved way (shown later). In this problem, a primary concern is about the switching of easy axis component (M_y), and M_y images are shown in common publications [3]. While in some other publications, spatial profiles of the spherical coordinate f are displayed in order to show the in-plane rotation of the magnetization [27]. As mentioned in section 4.1.1 of this chapter, the dynamics is in-plane, then we simply have $M_y = M_s \cos f$. Since M_s is a constant, these two display schemes are equivalent.

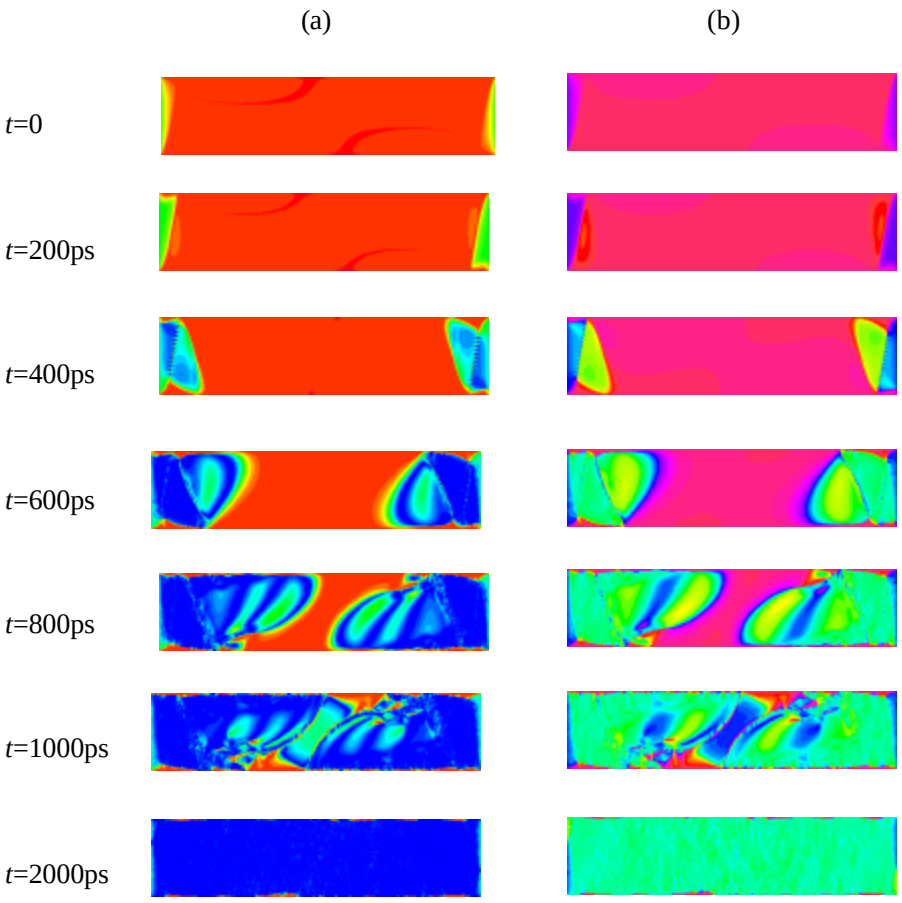


Fig.(4.4) (*color*) Individual time frames showing the reversal process when H_t=0. The left column (a) represents for M_y component and the right column (b) represents for in-plane angle component. Color bars refer to section 2.5.

4.2.3 Transverse bias field; coherent reversal

A remarkable difference in the reversal has been found by applying a transverse bias field H_t [3]. In the simulation, the "S" state is set to be the initial configuration; but in Fig.(4.5a) we can see that because of the transverse bias field, the magnetization has shifted away from the easy axis – the equilibrium position when H_t=0 – comparing to Fig.(4.4a). This means the sample is in a higher energy level. This is very important in understanding the different reversal modes, because the sample is now starting from a different initial configuration, higher on the potential energy landscape (which includes a "barrier" corresponding to high magnetostatic energy with magnetization along the hard axis direction), but also where there is an initial torque on the magnetization throughout the specimen [33, 34].

The sample responds earlier to the switching pulse, and the reversal time decreases dramatically. Edge domains in the initial configuration will expand quickly along the easy axis direction and form a narrow domain parallel to the easy axis. The domain walls then expand transversely in the hard axis direction until saturation is reached. This domain wall motion dominates the whole reversal process, which is in nature different from H_t=0 case. The stripe-like domain nucleation process is avoided, thus the reversal gains a significant speed-up. Fig.(4.6) shows the great improvement. Since the rising time of the switching pulse is about 500(ps), the transverse bias field yields an almost "synchronous" reversal.

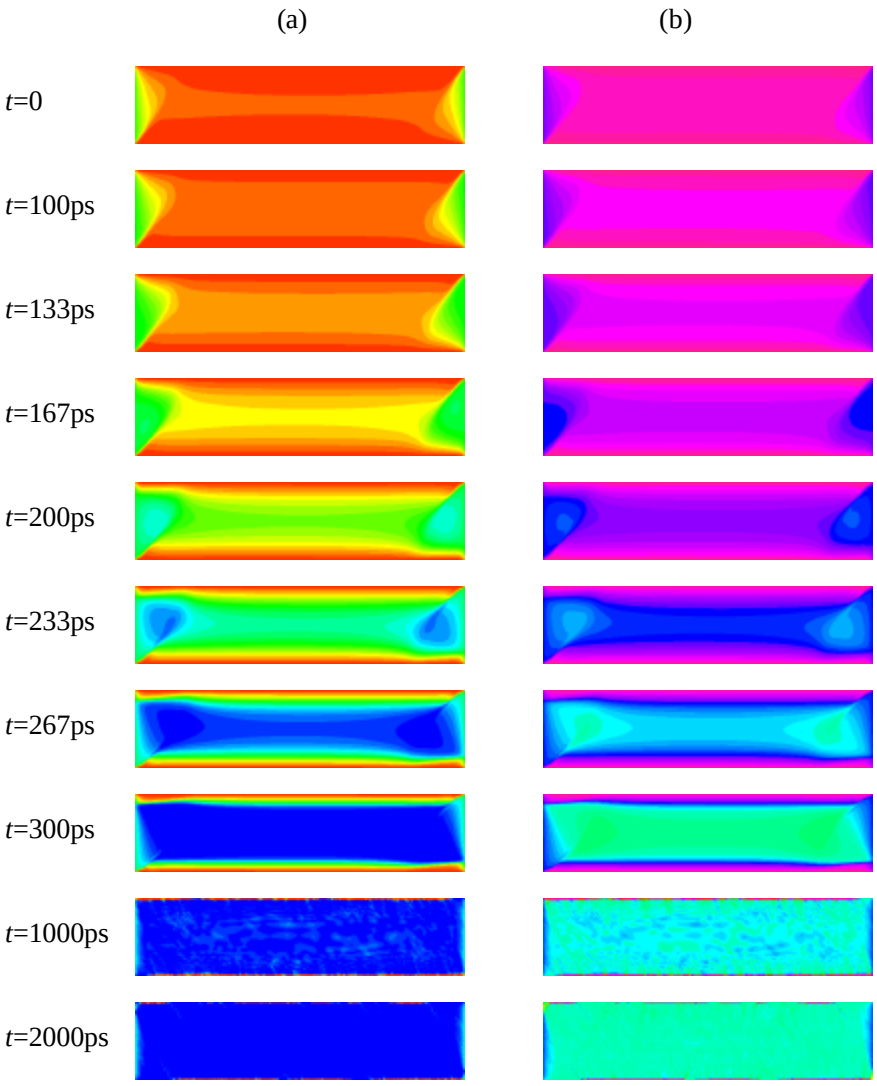


Fig.(4.5) (*color*) Individual time frames showing the reversal process when the transverse bias field is H_t=65.3(Oe). (a), M_y images; (b), in-plane angle images. Color bars refer to section 2.5.

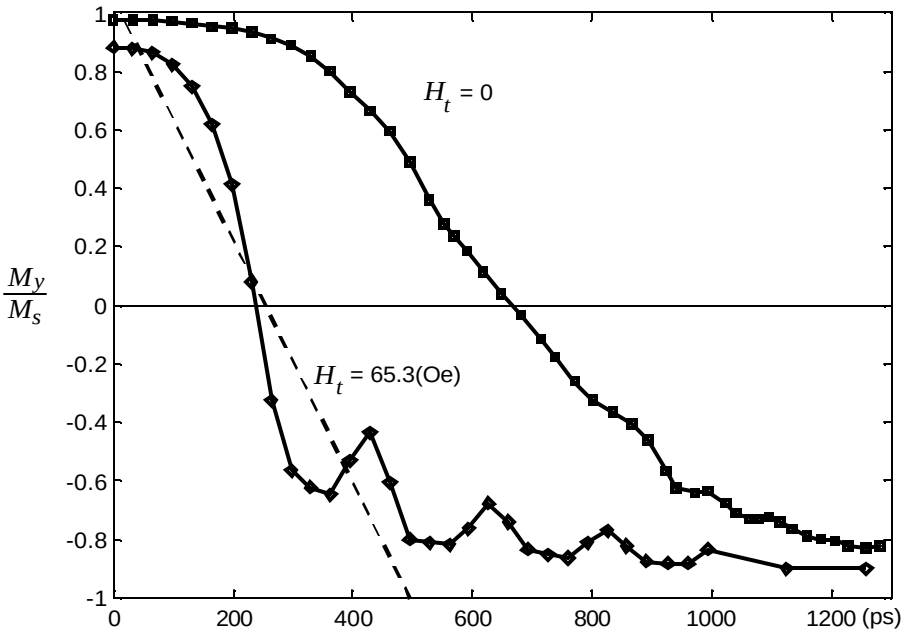


Fig.(4.6) Simulated magnetization reversal as a function of time with or without the transverse bias field. The longitudinal field is fixed at ~60(Oe). The switching pulse is assumed to have a linear rising time ~500(ps), shown by the dashed line.

4.3 Effects of thermal fluctuations

I show here some comparisons between existing experimental data and my own simulation results. Different switching behaviors have been measured and typical results are shown in Fig.(4.7) [3]. This is the 2x10(μm^2)x15(nm) Permalloy sample, and the field configurations and magnetic parameters are the same with those in my simulation. It has been shown that the simulation results match these processes pretty well (refer to Ref.[4] and Fig.(4.4-5)).

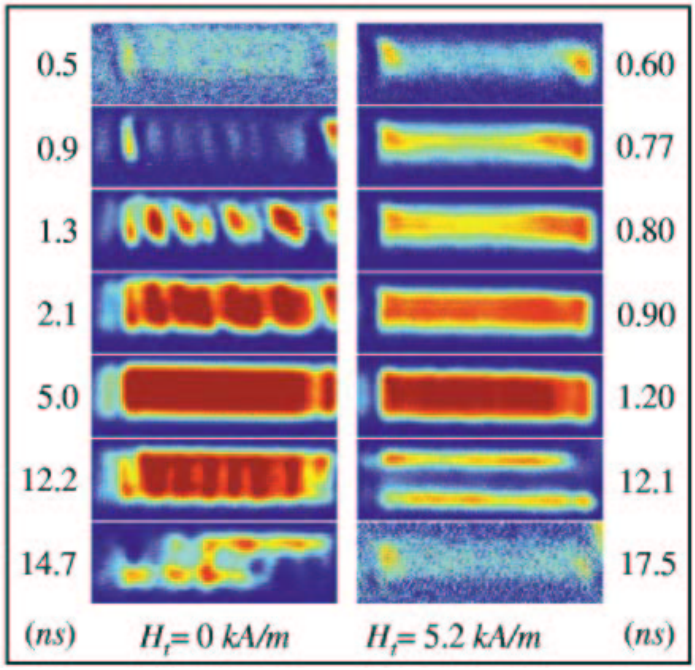
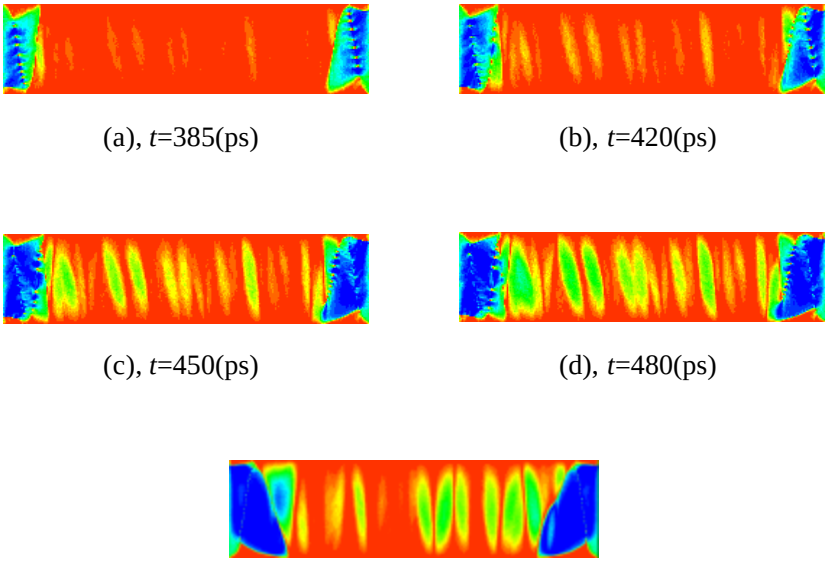


Fig.(4.7) (*color*) Spatial distributions of the in-plane magnetization component in the easy axis direction [3]. In both cases, the longitudinal bias field is 60.3(Oe). The left volumn refers to transverse field H_t=0, and the right volumn H_t=65.3(Oe) (5.2 *k*A/m in SI unit). Note: it uses a different display scheme from what I use, but that will not affect the comparison.

However, the simulated domain-nucleation modes are different from measured results, especially when no transverse bias field is applied. In simulations, small magnetic domains first flip from two ends of the sample, then expand and merge in the center; while the observed process reveals that some stripe-like domains arise throughout the whole sample in the early stage of reversal, showing some random characteristics, which implies that stochastic thermal fluctuations play a role in it.

I added the thermal term in the simulation using Eq.(1.36) and obtained corrected spatial images. Fig.(4.8a-d) shows some of them when T=300(K). The stripe-like domains appear as desired (although a little earlier than experiment, since we assume the DC swithing field holds a linear rising profile in simulations, while the real magnetic pulses rise exponentially in the early stage of excitings, thus have smaller switching field than simulations). Fig.(4.8e) is from another "benchmarking" using Scheinfein's code, indicating my thermal term calculation is successful.



(a), t=385(ps)

(b), t=420(ps)

(c), t=450(ps)

(d), t=480(ps)

(e), by Scheinfein's code, t=490(ps)

Fig.(4.8) (*color*) Simulated time frames of M_y spatial distributions under room temperature T=300(K). Other sample settings are identical with the case described in section 4.2.2. The last frame is obtained from Scheinfein's code that is in agreement with (d) regardless the tiny time gap of 10(ps).

Another useful comparison is about the time evolution of averaged magnetization for the whole sample. Local energy fluctuation brought by thermal effects should make the magnetization spins easier to break the energy barriers and shorten the reversal time. This is shown in Fig.(4.9), which plots the easy-axis average magnetization as a function of time under the different termperature 0(K) (*i.e.*, to neglect the thermal term) and 300(K). It would be valuable for both theory advances and industrial applications (a field called "thermo-assisted switching") to investigate the change of reversal time as a function of temperature, which is one of the future tasks.

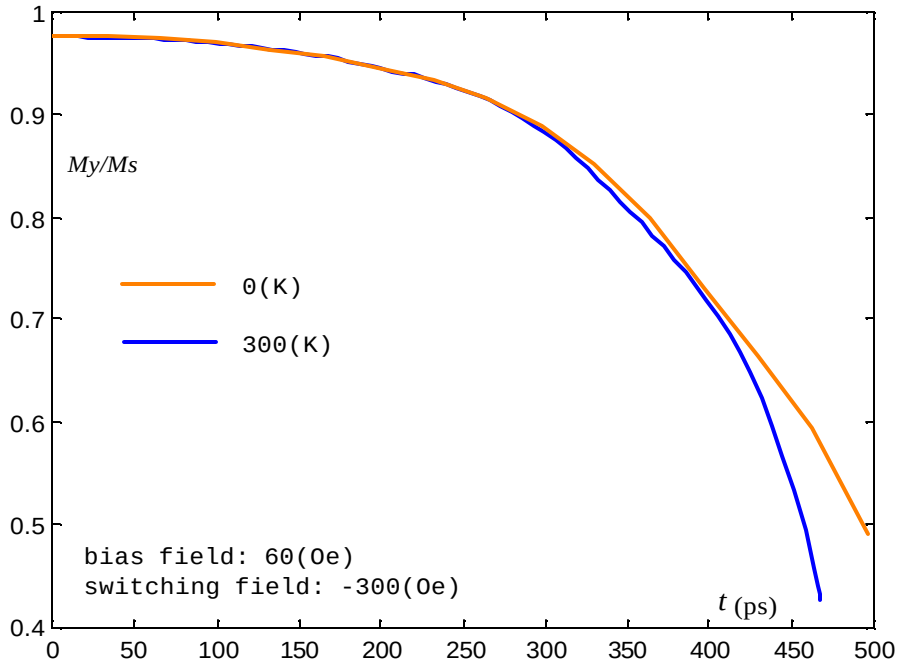


Fig.(4.9) (*color*) Time .vs. averaged magnetization along the easy axis, based on the same simulations discussed in Fig(4.4) and Fig.(4.8).

Chapter Five

Ferromagnetic resonance

When simulating problems of ferromagnetic resonance (FMR), numerical faults will arise, with some checkerboard patterns appearing. After solving this puzzle, FMR simulation results in both time and spatial domains will be presented, and they will be compared with experimental results and empirical formula.

5.1 Introduction

Stroboscopic measurements on the out-of-plane magnetization (M_z) offer a good handle on the study of small angle excitations of a thin film magnetic element. The magnetization's small angle departure from the equilibrium orientation will be followed by precession about the direction of local effective field, which is largely in-plane as discussed in the last chapter. Hence the polar component M_z is one of the oscillating transverse components of magnetization, as in magnetic resonance. Ferromagnetic resonance (FMR), driven by an out-of-plane magnetic pulse, exhibits typical small angle dynamics, and is the main topic of this chapter.

Ferromagnetic resonance data are collected by time domain measurements and spatial scanning. The apparatus for an FMR experiment is similar to what is used in reversal problems. The primary difference is the direction of the transient

magnetic pulse, now out-of-plane.

The sample under experimental investigation here is a square Permalloy element with dimensions of 4x4(μm^2)x15(nm). An adjustable in-plane bias field is applied in the easy axis direction. The excitation pulse has a bell-shaped profile, which I have approximated by a sine function in the simulation. The pulse width is 500 picosecond, and the peak magnitude is 8(Oe).

5.2 Numerical limitation – checkerboard puzzle

A serious challenge came up when studying the simulated M_z spatial images for very small flipping angles. I encountered some checkerboard-like "scars" without physical meaning. Fig.(5.1) shows an example. M_z values in neighboring cells have astonishing divergence. Suspecting all kinds of possibilities, I examined this problem both on the real 4x4(μm^2) Permalloy element and some test samples (with smaller sizes, so that simulations run faster). Eventually I figured out that the numerical precision setting was the main cause.

In the FORTRAN77 simulation code, there are two parameters to identify the precision of the Runge-Kutta integrator: "TOL" (tolerance) and "THRES" (threshold). TOL controls the relative error tolerance; THRES is an array with the length equal to the number of equations to be solved, THRES(i) is the threshold for the ith solution component. Usually they're set to be 10^{-6} or even larger, and this is OK for most problems. The tendency is to keep them large because the smaller they are, the slower the simulation runs.

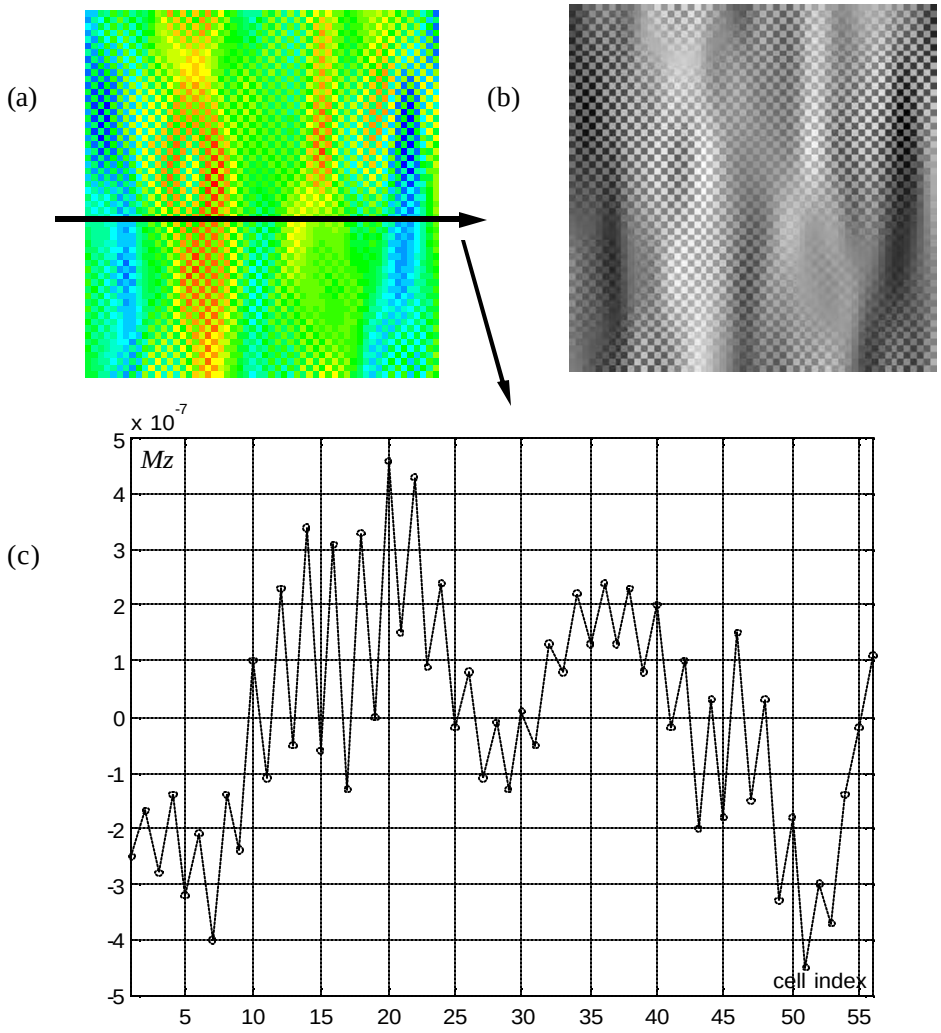


Fig.(5.1) (*color*) (a), a 56x56 cell block extracted from a FMR simulation image; (b), grey map of (a); (c), a scan line showing divergent data in neighboring cells.

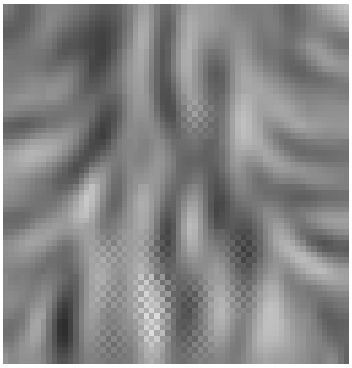
In out-of-plane FMR problems, the situation is changed. Since the magnetization is almost in-plane, the absolute value of M_z/M_s might be even smaller than 10^{-7} (see some of the points in Fig.(5.1c))! When this happens, the two parameters become void, because numerical errors which have been permitted by them (*i.e.*, tolerated errors) are even larger than the magnitude of desired data. This explains

why some data points just look like some kind of non-random noise. "Real" data are submerged in this numerical noise.

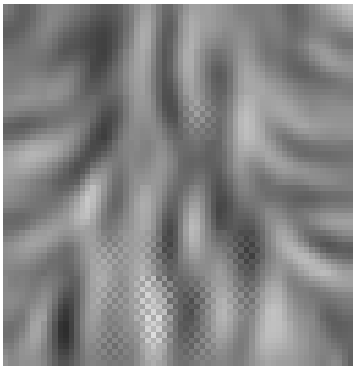
I reset these two parameters to overcome this problem. Fig.(5.2) shows some results from an imaginary simulation (the simulation on the real sample is too slow to test this puzzle conveniently). This is a 640x640x10(nm^3) sample, with a 64x64 grid, so every cell is a 10(nm) cube. I ran a number of simulations in identical conditions, only changed the values of TOL and THRES. Checkerboard patterns keep arising until TOL=10^{-7} and THRES=10^{-6}. When TOL=10^{-6} and THRES=10^{-7}, checkerboard patterns still appear but look much better than the case when TOL=THRES=10^{-6}. When TOL=THRES=10^{-7}, the checkerboard patterns completely vanish. This proves that both the two parameters are related to the numerical limitation; and by setting them properly, the problem would be solved, although that will cost more execution time. Simulation results in the next sections of this chapter are all obtained with safe settings.

In FORTRAN90 version simulation codes, numerical precision is determined by just one parameter, "precision", which limits the minimum angle change of magnetization under a critical value. Control this parameter carefully and checkerboard patterns would disappear.

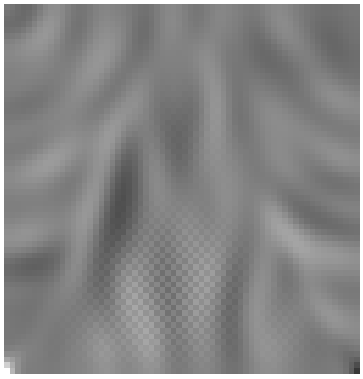
(a). TOL=THRES=10^{-5}



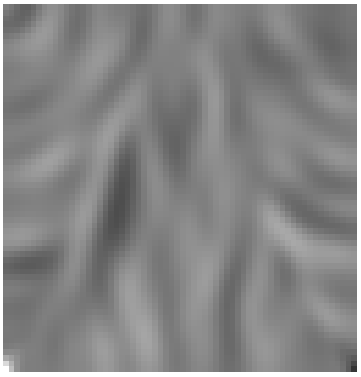
(b). TOL=THRES=10^{-6}



(c). TOL=10^{-6}, THRES=10^{-7}



(d). TOL=10^{-7}, THRES=10^{-6}



(e). TOL=THRES=10^{-7}

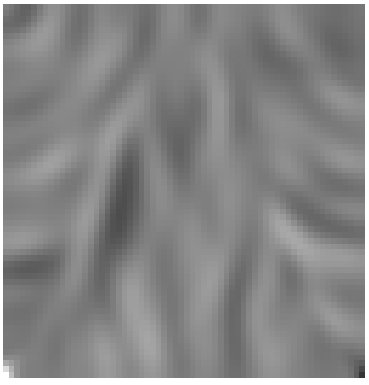


Fig.(5.2) Checkerboard patterns come out if precision settings are not good enough (a, b, c).

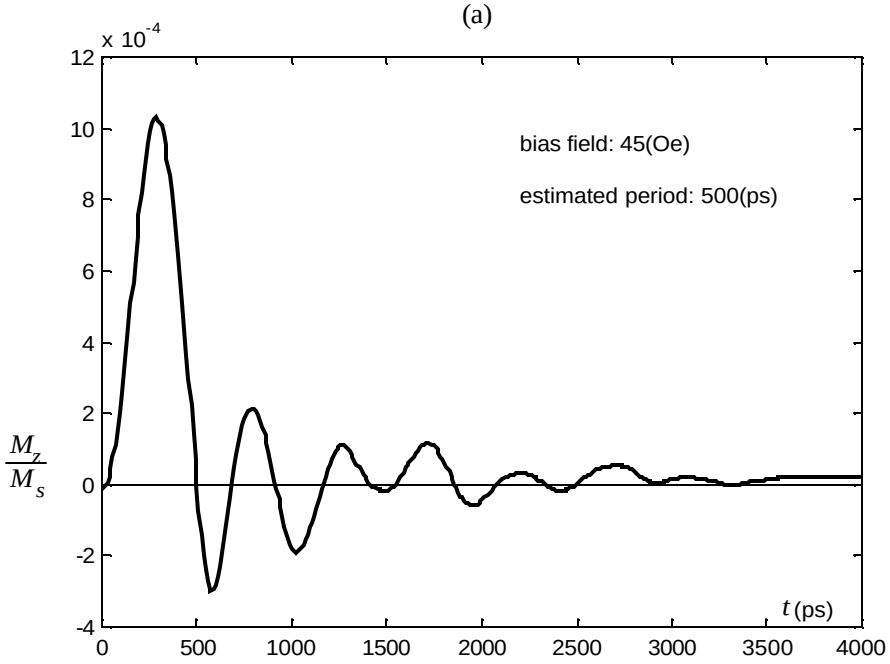
5.3 Results – time domain analysis

The "effective field", which would set the scale for the precession frequency of a "test spin" dropped into a given location, varies quite strongly as a function of position in our inhomogeneously magnetized sample. This field variation defines a profile analogous to potential surfaces or potential wells in quantum mechanics, and forms a landscape for magnetic excitations (eigenmodes) with varied frequencies and wavelengths. Our broad band pulse FMR experiments can drive and detect a variety of these modes [5, 35, 37].

As a zero-order approximation, however, we can compare the calculated FMR response spatially averaged over the whole element to the uniform mode of oscillation in an infinite thin film, predicted by the famous "Kittel formula" [35].

Normalized M_z (averaged over the whole sample) as a function of time, with different easy axis bias field applied, are plotted in Fig.(5.3) separately. Larger bias field has stronger in-plane confinement on the magnetization, thus the out-of-plane oscillation has smaller magnitude and larger frequency as the bias field increases.

(a)


x 10^-4
12
10
8
6
4
2
0
-2
-4
0
500
1000
1500
2000
2500
3000
3500
4000
bias field: 45(Oe)
estimated period: 500(ps)
$\frac{M_z}{M_s}$
t (ps)

(b)

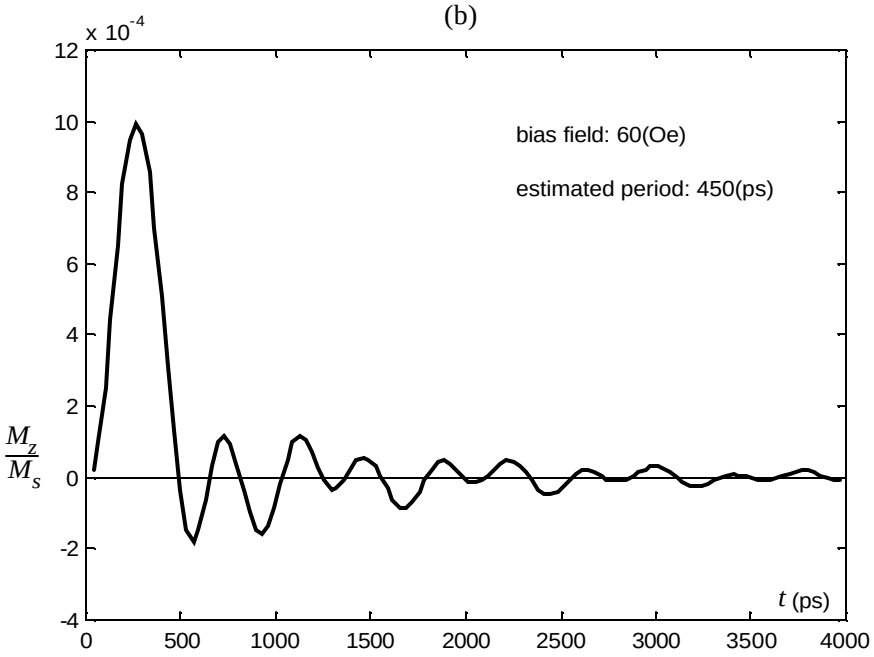
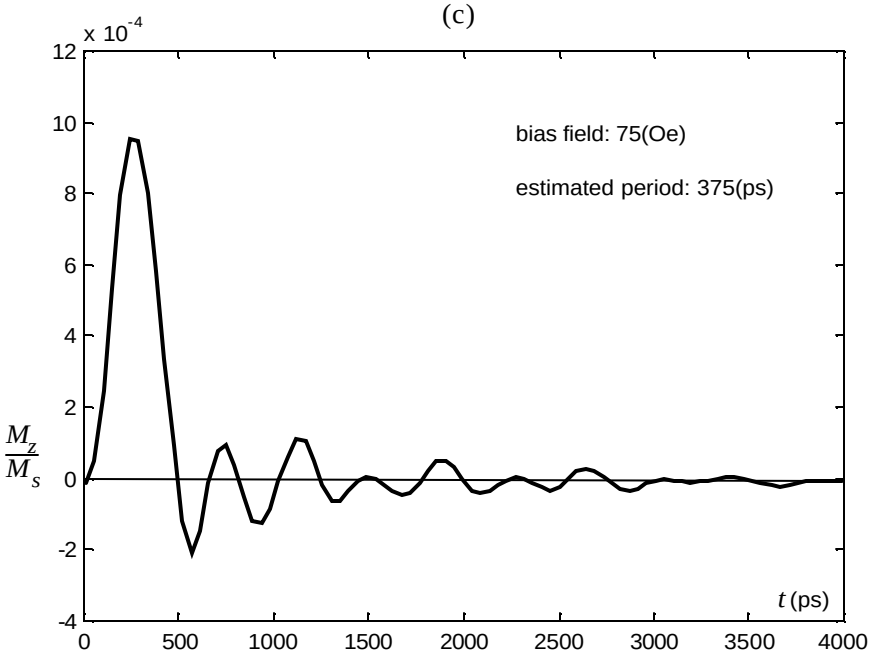

x 10^-4
12
10
8
6
4
2
0
-2
-4
0
500
1000
1500
2000
2500
3000
3500
4000
bias field: 60(Oe)
estimated period: 450(ps)
$\frac{M_z}{M_s}$
t (ps)

Fig.(5.3a-b)

(c)


x 10^-4
12
10
8
6
4
2
0
-2
-4
0
500
1000
1500
2000
2500
3000
3500
4000
bias field: 75(Oe)
estimated period: 375(ps)
$\frac{M_z}{M_s}$
t (ps)

(d)

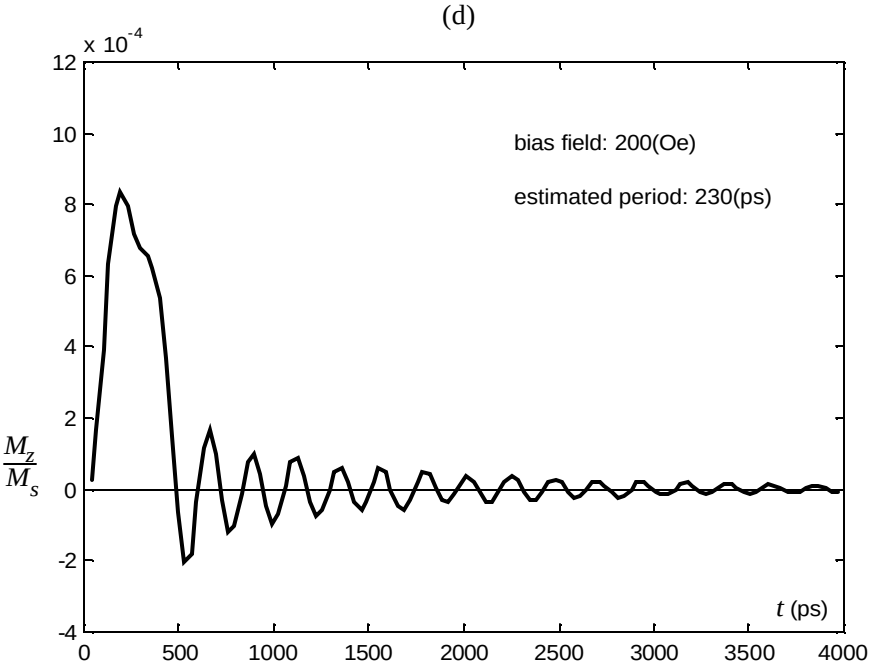

x 10^-4
12
10
8
6
4
2
0
-2
-4
0
500
1000
1500
2000
2500
3000
3500
4000
bias field: 200(Oe)
estimated period: 230(ps)
$\frac{M_z}{M_s}$
t (ps)

Fig.(5.3c-d)

(e)

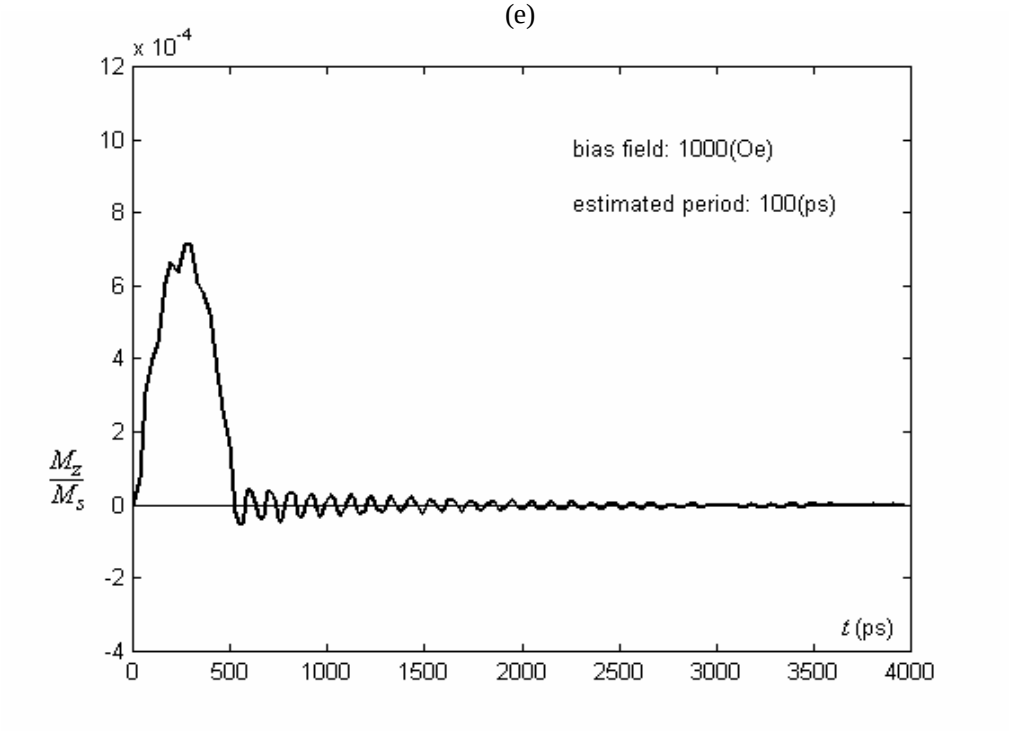


Fig.(5.3) FMR curves for different easy axis bias fields (H_y=45, 60, 75, 200, 1000(Oe)). When the oscillation has sufficient high frequency, the first "nonresonace" peak due to the original excitation has been superposed by subsequent resonance pulses, shown in above picture when bias field is 1000(Oe) (also when bias field is 200(Oe), but not so explicit).

In Fig.(5.3d-e) we notice superimposed oscillations in the first "nonresonance" peak, indicating that the FMR mode is established immediately after the excitation. Fig.(5.4) shows an experiment curve and a simulation curve under the same parameter configuration. This is not a careful comparison, but simply to show that both of them exhibit this kind of superposition.

Fig.(5.3) also provides the simulated resonance periods for different cases. The

stronger the bias field, the high frequency mode is launched. A quantitative comparison can be performed based on this field-frequency dependence.

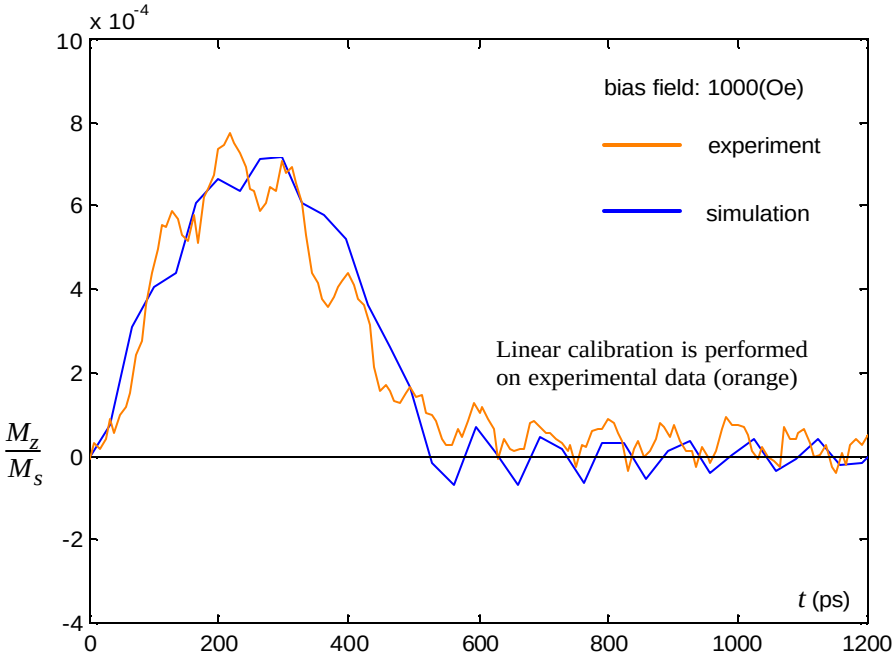


Fig.(5.4) (*color*) Experimental .vs. simulated FMR curves when H_{bias}=1000(Oe).

The Kittel's formula describes the relation between FMR frequency and static external field [35], and it is fit for current simulations. The equation reads:

$$w = g_0 \sqrt{H_{\text{ext}} \left(H_{\text{ext}} + 4p M_s \right)} \tag{5.1}$$

Substitute H_{ext} with the values shown in Fig.(5.3), we will get corresponding theoretical values for FMR periods (reciprocal of the frequencies). A comparison between the simulation results and the Kittel formula is shown in Fig.(5.5). They agree with each other very well.

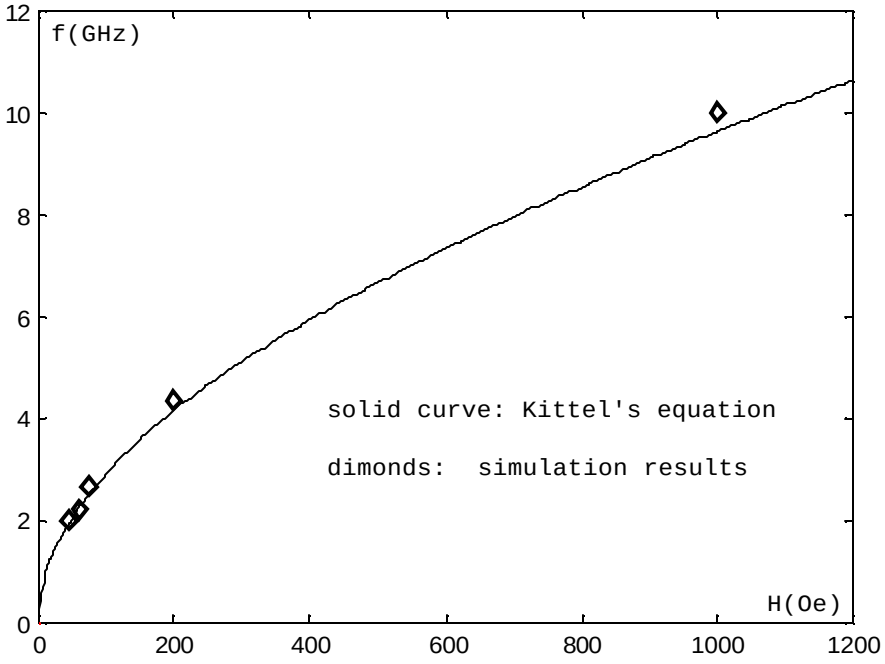
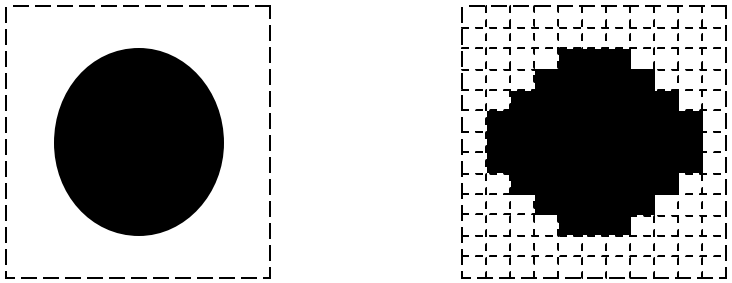


Fig.(5.5) External bias field .vs. FMR frequency.

5.4 Results – spatial images

Spatial distribution of magnetization is measured by repeated pump-probe procedures with the probe spot scanning across the whole sample surface. A new feature to be introduced here is a round pinhole punched at the center of the sample. This is one of the efforts to study higher order FMR modes. The pinhole introduces a quadrapolar topography on the preciously uniform effective field or "potential surface" background, hereby influences the symmetry of the excited

eigenmode. The diameter of the pinhole is ~240(nm). In simulations, the pinhole is implemented by a square grid mask with a staircase boundary, see Fig.(5.4). This may cause high frequency spin waves in the simulations that would not be present in real samples.



(a) (b)

Fig.(5.6) Sketch map: definition of the pinhole.
(a), real sample; (b), simulation.

Fig.(5.7a-b) shows some images taken in experiments. They are spatial distribution of z-component Kerr signals. We could notice the two 90^0 domain walls, indicating that this sample is excited from the quasi-static "C" state. I use the "C" state as the element's initial state in the simulation, too. Fig.(5.7c) shows all three $\boldsymbol{M}$ components after the sample has been allowed a long time to equilibrate.

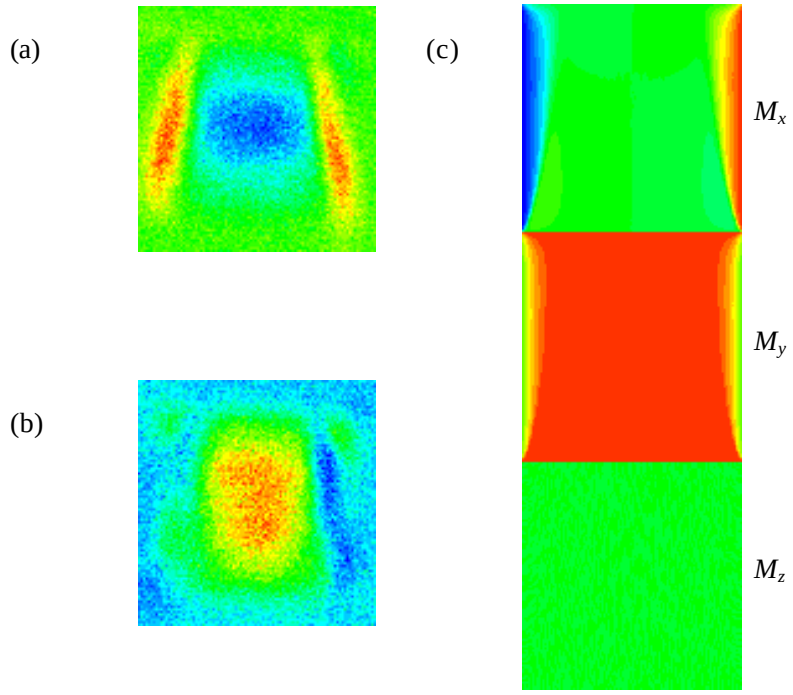


Fig.(5.7) (*color*) (a), (b), experimental images imply that the sample's initial state should be "C" state; (c), the quasi-static "C" state in the simulation.

Preliminary FMR simulation results are shown in Fig.(5.8). The left column refers to the no-defect sample, and the right column refers to the sample with the pinhole. Four snapshots are taken when the resonance reaches the 1st peak, the 1st valley, the 2nd peak and the 2nd valley, respectively. Because the cells within the pinhole area are assigned (0, 0) in spherical coordinates, some offset and linear expansion/retraction are applied on the data to produce good-contrast images. This leads to different visualization effects for the two columns. A detailed comparison between experiments and simulations is in progress.

The results in spatial profile simulation have some limitations. In the areas close to the pinhole, some odd points and tiny domain stripes arise. Although people do

observe similar structures in real samples, those in simulation results are obviously due to numerical limitations. On the other hand, the two long domain walls associated with the "C" state look much sharper than those in experiment images (see Fig.(5.7a-b)). At the same time, these areas are most vulnerable by checkerboard flaws – the numerical limitation as discussed before. These disagreements expose some configuration differences between experiments and simulations. In simulations, rectangular grid cells make odd boundary conditions, as illustrated in Fig.(5.6). The sample's corners and edges (including the pinhole) produce topological effects. Furthermore, not all dissipating terms are involved in the numerical model, such as laser drifts, electrical noise, mechanic vibration, *etc.* Therefore, the simulated images have somewhat abrupt appearances.

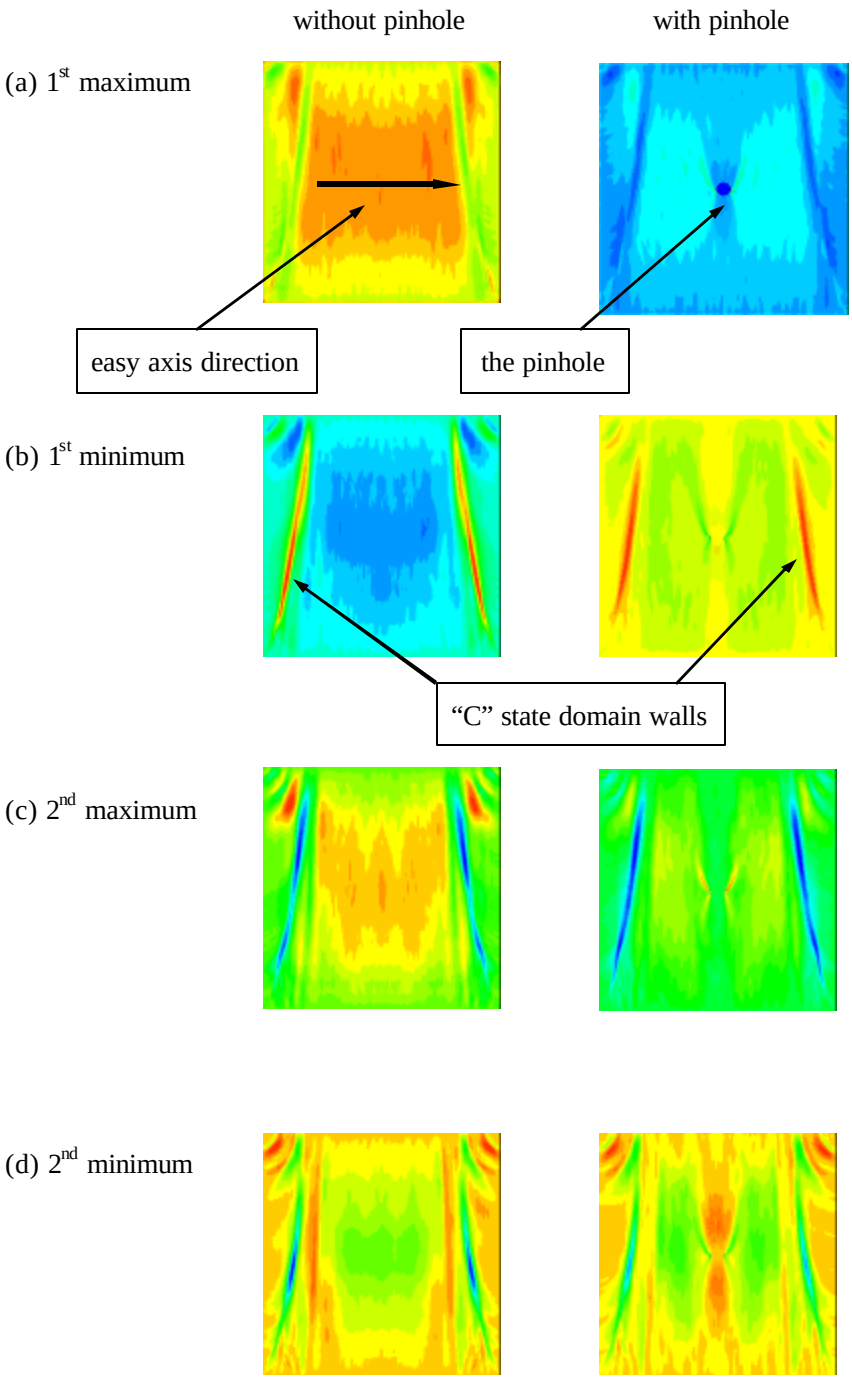


Fig.(5.8) (*color*) Spatial images for FMR study on the 4x4(μm^2) Permalloy sample. Simulation data are rescaled for convenience of color displaying, because absolute value of M_z/M_s is mostly as small as 10^{-4}~10^{-5}.

Epilogue

Conclusion and prospect

So far, micromagnetics in Permalloy thin film microstructures has been investigated – from quasi-static states, to in-plane and out-of-plane dynamics. Micromagnetic simulations show substantive success on reproducing the real processes and predicting valuable phenomena. In principle, the micromagnetic dynamics can be simulated to sufficiently high accuracy based on LLG model, provided external conditions are precisely configured and numerical precisions are good enough. On the other hand, stochastic thermal fluctuations bring random diversities into the motions of magnetization.

The simulation's capability of predicting may play an important role in future applications. Many "test" simulations presented in this thesis show a way how to do studies without real experiments, and this can help people to design future magnetic devices. This creative prospect stimulates my enthusiasm for further simulation works.

More calculations on the thermal term will be very interesting, possibly by doing a large number sampling (say, ~1000 times) and finding relative probabilities for certain processes. This also requires further improvements on the simulation code for less execution time. Another important task is to calculate the energy of magnetic states, which offers a quantitative method to study the stability of the system. Finally, 3D simulation is expected to be implemented based on current 2D version, because multi-layer magnetic samples are being researched in our group.

Reference

1. C. S. Lee, H. Lee, and R. M. Westervelt, **Microelectromagnets for the control of magnetic nanoparticles**, *Appl. Phys. Lett., 79, 20, 2001*

2. J. A. Katine, F. J. Albert, and R. A. Buhrman, **Current-induced realignment of magnetic domains in nanostructured Cu/Co multilayer pillars**, *Appl. Phys. Lett., 76, 3, 2000*

3. B. C. Choi, M. Belov, W. K. Hiebert, G. E. Ballentine, and M. R. Freeman, **Ultrafast magnetization reversal dynamics investigated by time domain imaging**, *Phys. Rev. Lett., 86, 4, 2001*

4. B. C. Choi, G. E. Ballentine, M. Belov, and M. R. Freeman, **Bias-field dependence of the spatiotemporal evolution of magnetization reversal in a mesoscopic $Ni_{80}Fe_{20}$ element**, *Phys. Rev. B, 64, 144418, 2001*

5. W. K. Hiebert, G. E. Balletine, and M. R. Freeman, **Comparison of experimental and numerical micromagnetic dynamics in coherent precessional switching and modal oscillations**, *Phys. Rev. B, 65, 140404, 2002*

6. M. Beleggia, M. A. Schofield, Y. Zhu, M. Malac, Z. Liu, and M. R. Freeman, **Quantitative comparison of magnetic field mapping in TEM with micromagnetic simulations**, *to be published*

7. J. D. Jackson, **Classical electrodynamics, 3rd edition**, *New York Wiley, 1999*

8. M. Mansuripur, **Magnetization reversal dynamics in the media of magneto-optical recording**, *J. Appl. Phys., 63, 12, 1988*

9. Jacques Miltat, Gonçalo Albuquerque, and André Thiaville, **An introduction to micromagnetics in the dynamic regime**, *from Spin dynamics in confined magnetic structures I, Topics Appl. Phys., 83, Springer, 2002*

10. M. Mansuripur, **The physical principles of magneto-optic recording**, *Cambridge University Press, 1995*

11. A. Hubert and R. Schäfer, **Magnetic domains – the analysis of magnetic**

microstructures, *Springer, 1998*

12. W. K. Hiebert, **Experimental Micromagnetic Dynamics: Ultrafast Magnetization Reversal Using Time Resolved Scanning Kerr Effect Microscopy**, *PhD thesis, University of Alberta, Canada, 2001*

13. G. B. Ballentine, **Comparison of Time-Resolved Micromagnetic Dynamics Experiments with Permalloy and Landau-Lifshitz-Gilbert Micromagnetic Simulation**, *PhD thesis, University of Alberta, Canada, 2002*

14. M. E. Schabes and A. Aharoni, **Magnetostatic interaction fields for a three-dimensional array of ferromagnetic cubes**, *IEEE Trans. Magnetics, 23, 6, 1987*

15. Hiroshi Fukushima, Yoshinobu Nakatani, and Nobuo Hayashi, **Volumn average demagnetizing tensor of rectangular prisms**, *IEEE Trans. Magnetics, 34, 1, 1998*

16. J. L. García-Palacios and F. J. Lázaro, **Langevin-dynamics study of the dynamical properties of small magnetic particles**, *Phys. Rev. B, 58, 22, 1998*

17. Leading authors: R. W. Brankin, I. Gladwell, and L. F. Shampine. The original *rksuite* code is available at NETLIB (http://www.netlib.org/ode/rksuite/)

18. H. Gould and J. Tobochnik, **An introduction to computer simulation methods – applications to physical systems, 2nd edition**, *Addison-Wdsley Publishing Company Inc., 1996*

19. FORTRAN90 version of *rksuite* was just released in August of 2002, also available at NETLIB (http://www.netlib.org/ode/rksuite/)

20. R. H. Koch, J. G. Deak, D. W. Abraham, P. L. Trouilloud, R. A. Altman, Y. Lu, W. J. Gallagher, R. E. Scheuerlein, K. P. Roche, and S. S. P. Parkin, **Magnetization reversal in micron-sized magnetic thin films**, *Phys. Rev. Lett. 81, 20, 1998*

21. P. A. Midgley, **An introduction to off-axis electron holography**, *Micron , 32, 2001*

22. R. E. Dunin-Borkowski, M. R. McCartney, and D. J. Smith, **Off-axis electron holography of nanostructured magnetic materials**, *Magnetic nanostructures (edited by H. S. Nalwa), Vol. 1, 2002*

23. R. E. Dunin-Borkowski, M. R. McCartney, B. Kardynal, D. J. Smith, M. R. Scheinfein, **Switching asymmetries in closely coupled magnetic nanostructure arrays**, *Appl.*

Phys. Lett., 75, 17, 1999

24. Y. Zhu, V. V. Volkov, and M. De Graef, **Understanding magnetic structures in permanent magnets via in-situ Lorentz microscopy, interferometric and non-interferometric phase reconstruction**, *J. Electron Microscopy, 50, 2002*

25. Y. Aharonov and D. Bohm, **Significance of electromagnetic potentials in the quantum theory**, *Phys. Rev. 2nd series, 115, 3, 1959*

26. C. Merton, G. D. Skidmore, J. Schmidt, E. D. Dahlberg, H. Wan, and B. Pant, **Magnetic reversal of tapered permalloy bars with holes in the center**, *J. Appl. Phys., 85, 8, 1999*

27. R. H. Koch, J. G. Deak, D. W. Abraham, P. L. Trouilloud, R. A. Altman, Y. Lu, And W. J. Gallagher, **Magnetization reversal in micron-sized magnetic thin films**, *Phys. Rev. Lett., 81, 20, 1998*

28. M. Schneider, H. Hoffmann and J. Zweck, **Lorentz microscopy of circular ferromagnetic permalloy nanodisks**, *Appl. Phys. Lett., Vol. 77, No. 18, 2000*

29. R. P. Cowburn, **Property variation with shape in magnetic nanoelements**, *J. Phys. D, 33, R1, 2000*

30. Y. Nakatani, Y. Uesaka, and N. Hayashi, **Direct solution of the Landau-Lifshitz-Gilbert equation for micromagnetics**, *Jpn. J. Appl. Phys., 28, 12, 1989*

31. M. R. Freeman and W. K. Hiebert, **Stroboscopic Microscopy of Magnetic Dynamics**, *from Spin dynamics in confined magnetic structures I, Topics Appl. Phys., 83, Springer, 2002*

32. M. R. Freeman and B. C. Choi, **Advances in magnetic microscopy**, *Science, 294, 2001*

33. H. W. Schumacher, C. Chappert, R. C. Sousa, P. P. Freitas, and J. Miltat, **Quasi-ballistic magnetization reversal**, *Submitted to Phys. Rev. Lett.*

34. H.W. Schumacher, C. Chappert, P. Crozat, R.C. Sousa, P.P. Freitas, J. Miltat, J. Fassbender, and B. Hillebrands, **Phase coherent precessional magnetization reversal in microscopic spin valve elements,** *Submitted to Phys. Rev. Lett.*

35. W. K. Hiebert, A. Stankiewicz, and M. R. Freeman, **Direct observation of magnetic**

relaxation in a small Permalloy disk by time-resolved scanning Kerr microscopy, *Phys. Rev. Lett., 79, 6, 1997.*

36. C. Kittel, *Phys. Rev., 73, 155, 1948*

37. M. Belov, Z. Liu, R. Sydora and M. R. Freeman, **Modal oscillation symmetry and indirect damping in internally patterned $Ni_{80}Fe_{20}$ thin film microstructures**, *to be published*

38. W. Rave and A. Hubert, **Magnetic ground state of a thin-film element**, *IEEE Trans. Magnetics, 36, 6, 2000*

Appendix I

2D Micromagnetic Simulation code

(FORTRAN77)

Below is the FORTRAN77 version of micromagnetic simulation. The code is conventionally split into 11 files, each containing subroutines or functions that perform certain tasks. The code structure is illustrated in section 2.2.

1. Contents in the file "*global2d.inc*":

```
c*****************************************************************
c Global definitions of parameters. Globalization is achieved by
c including this file in every subroutine and function. Be aware:
c this method does not allow for any communication between units,
c unless variables are placed in COMMON declaration.
c*****************************************************************
    IMPLICIT NONE
    INTEGER*4 nxmax, nymax, Nmax, nx2, ny2, kmax
    REAL*8 sizex, sizey, thick, un, Pi, dux, duy, duz, d2x, d2y, d2z
! Working array size
    PARAMETER(kmax=1000)
    PARAMETER(nxmax=64, nymax=64)
    PARAMETER(nx2=2*nxmax, ny2=2*nymax)
    PARAMETER(Nmax=2*nxmax*nymax)
! Dimensions of the rectangular grid in nm
    PARAMETER(sizex=800.d0, sizey=800.d0, thick=15.d0)
! un is a length unit in nm (=sqrt(2*A/Ms^2, where A is the exchange
! constant. Ms is the saturated magnetization. A=10-6(erg/cm), and
! 4*Pi*Ms=10.8(kGauss))
    PARAMETER(un=16.455122d0, Pi=3.14159265358979324d0)
! Relative size of the cell
    PARAMETER(dux=sizex/nxmax/un, duy=sizey/nymax/un, duz=thick/un)
    PARAMETER(d2x=1/(dux*dux), d2y=1/(duy*duy), d2z=1/(duz*duz))
```

```
! Mask array: defines shape of the sample.
    INTEGER*1 mask(-2:2,0:nxmax+1,0:nymax+1)
    INTEGER*1 maskp(-2:2,-1:nxmax,-1:nymax)
    COMMON /mask1/ mask
    EQUIVALENCE (mask,maskp)
! Definition of damping constant, evaluated in "sim2d.f".
    REAL*8 alpha
    COMMON /param/ alpha
!
    CHARACTER*32 datini
    CHARACTER*32 datlast
    CHARACTER*3 datint
! Note: datini must be the same as datlast
    PARAMETER(datini="data_last.dat")
    PARAMETER(datint="xyz")
    PARAMETER(datlast="data_last.dat")
```

2. Contents in the file "*sim2d.f*":

```
    PROGRAM sim2d
c*****************************************************************
c Program traces the dynamical development of magnetization process
c in arbitrarily shaped sample with space- and time-dependent external
c field. The main program is a customizable driver, the code below is
c just an example.
c*****************************************************************
    INCLUDE 'global2d.inc'
c Paramaters & procedures for timing
    CHARACTER*24 the_time
    REAL*4 tarr(2), ttot, ETIME
c Parameters for RKSUITE
    INTEGER cstep
    COMMON /cstep/ cstep
    INTEGER*4 method, lenwrk, total, cost, stepok
    PARAMETER(lenwrk=10*Nmax)
    CHARACTER*1 task
    LOGICAL errass, message, ex
    REAL*8 work(lenwrk), thres(Nmax), wast, hnext
    REAL*8 eps, h1
```

```
    EXTERNAL sigterm, sigusr1, sigusr2
! Store angles of magnetization distribution
    REAL*8 tp(2,nxmax,nymax)
! Ellipse size
    REAL*8 rx, ry
! Range of integration
    REAL*8 t1, t2
!
    INTEGER*4 kount, i, j
    REAL*8 xm(0:kmax), ym(0:kmax), zm(0:kmax), t(0:kmax), dxsav
    COMMON /PATH/ kount, dxsav, xm, ym, zm, t
!
    cstep=0
c RKSUITE parameter initialization (see RKSUITE documentation)
    eps=0.0000001
c Relative accuracy
    h1=0.0
c Guess for initial step (0 - auto)
    method=1
c Method of Runge-Kutta integration
    task='u'
c Integration procedure
    errass=.FALSE.
    message=.TRUE.
c Absolute accuracy
    DO j=1, Nmax
        thres(j)=0.0000001
    END DO
! Signal handler to catch soft kill signal. Signal number 15,16 and 17
    CALL signal(15,sigterm,-1)
    CALL signal(16,sigusr1,-1)
    CALL signal(17,sigusr2,-1)
c Evaluate the damping constant - defined here to escape
c BLOCK DATA and still keep flexibility to change it
    alpha=0.008
c Period of savings
    dxsav=2.0
c Range of integration
    t1=0.0
    t2=200.0
c Calls to timing procedures
```

```
      CALL FDATE(the_time)
      WRITE(*,*) the_time
      ttot=ETIME(tarr)
c Initialization of demagnetizing field calculation
      CALL initdem
c Information on shape and initial condition are read from
c a file. If the file does not exist it starts from a prescribed
c initial state. If the file does exits it reads the state that
c it got to and the time and begins from there.
      INQUIRE(file=datini, exist=ex)
      IF (ex .eqv. .TRUE.) THEN
          OPEN(10,FILE=datini,FORM='unformatted')
          READ(10) mask, tp, t1, kount
          CLOSE(10)
! Reset the start point if desired
!         t1=0
!         kount=0
      ELSE
          CALL shape(rx,ry)
          CALL iniuni(tp,1)
      END IF
! Perform integration
      CALL SETUP(Nmax,t1,tp,t2,eps,thres,method,task,
      &       errass,h1,work,lenwrk,message)
      CALL ODEINT(tp,t1,t2,work)
!
      CALL FDATE(the_time)
      WRITE(*,*) the_time
      WRITE(*,*) 'Execution time: ', ETIME(tarr), ' sec'
      CALL STAT1(total,cost,wast,stepok,hnext)
      WRITE(*,*) total, cost, stepok
      WRITE(*,*) wast, hnext
!
      END
```

3. Contents in the file “*interupt .f*”:

```
! Signal handlers; exit and resubmit program to queue
      SUBROUTINE sigterm()
```

```
    STOP 99
    RETURN
    END
!
    SUBROUTINE sigusr1()
    STOP 99
    RETURN
    END
!
    SUBROUTINE sigusr2()
    STOP 99
    RETURN
    END
```

4. Contents in the file "*init2d.f*":

```
! Temporal-spatial profile of the external field
      SUBROUTINE hfun(t,i,j,hx,hy,hz)
      INCLUDE 'global2d.inc'
!
       REAL*8 t, hx, hy, hz
       INTEGER*4 i, j
!
       REAL*8 Ms
       PARAMETER(Ms=859.436693)
!
       hx=0/Ms
       hy=0/Ms
       hz=0/Ms
!
       END
! -------------------------------------------------------------
       SUBROUTINE shape
       INCLUDE 'global2d.inc'
!
       INTEGER*4 i, j
! First, fill up the full rectangle
       DO i=1, nxmax
              DO j=1, nymax
```

```
                    mask(-2,i,j)=-1
                    mask(-1,i,j)=-1
                    mask(0,i,j)=1
                    mask(1,i,j)=1
                    mask(2,i,j)=1
             END DO
      END DO
! Detect edges and assign boundary conditions
      DO i=1, nxmax
             DO j=1, nymax
                    IF (mask(0,i,j) .eq. 1) THEN
                          IF (mask(0,i-1,j) .eq. 0) mask(-1,i,j)=0
                           IF (mask(0,i+1,j) .eq. 0) mask(1,i,j)=0
                          IF (mask(0,i,j-1) .eq. 0) mask(-2,i,j)=0
                           IF (mask(0,i,j+1) .eq. 0) mask(2,i,j)=0
                    END IF
             END DO
      END DO
      END
! ------------------------------------------------------------
! ------------------------------------------------------------
      SUBROUTINE iniuni(tp)
      INCLUDE 'global2d.inc'
!
      REAL*8 tp(2,nxmax,nymax)
!
      INTEGER iseed(4)
      REAL*8 rand(nxmax*nymax)
!
      INTEGER*4 i, j
!
      iseed(1)=371
      iseed(2)=49
      iseed(3)=185
      iseed(4)=229
      CALL dlarnv(2,iseed,nxmax*nymax,rand)
!
      DO i=1, nxmax
             DO j=1, nymax
                    tp(1,i,j)=mask(0,i,j)*Pi/2
    &                      +(1-mask(0,i,j))*rand(i*j)*0.0000001
```

```
                  tp(2,i,j)=mask(0,i,j)*Pi/2
!                  tp(2,i,j)=mask(0,i,j)*rand(i*j)*Pi
              END DO
       END DO
!
       END
! ------------------------------------------------------------
! ------------------------------------------------------------
       SUBROUTINE initdem
       INCLUDE 'global2d.inc'
! ********************************************************************
! Initializes FFTW and calculates Fourier transforms of demagnetizing
! coefficients matrices. It should be done only once per program
! execution.
! ********************************************************************
       INTEGER*4 i, j, Nnorm
       PARAMETER(Nnorm=2*Nmax)
! Demagnetization functions
       REAL*8 dmxx, dmyy, dmzz, dmxy
! Demagnetization matrices
       REAL*8 kxx(0:nx2+1,0:ny2-1)
       REAL*8 kyy(0:nx2+1,0:ny2-1)
       REAL*8 kzz(0:nx2+1,0:ny2-1)
       REAL*8 kxy(0:nx2+1,0:ny2-1)
! Workspace
       REAL*8 table(((15+nx2)+2*(ny2+15)))
       REAL*8 tab2(((15+nx2)+2*(ny2+15)))
       REAL*8 work(nx2+4*ny2)
       COMMON /dcoef/ kxx, kyy, kzz, kxy, table, tab2, work
! Demagnetization matrices are filled with the respective values
! of the demagnetizing coefficients.
       DO j=0, nymax-1
              DO i=0, nxmax-1
                     kxx(i,j)=dmxx(i,j,0,dux,duy,duz)
                     kyy(i,j)=dmyy(i,j,0,dux,duy,duz)
                     kzz(i,j)=dmzz(i,j,0,dux,duy,duz)
                     kxy(i,j)=dmxy(i,j,0,dux,duy,duz)
              END DO
       END DO
!
       DO j=0, nymax-1
```

```
            DO i=nxmax+1, 2*nxmax-1
                kxx(i,j)=dmxx(i-2*nxmax,j,0,dux,duy,duz)
                kyy(i,j)=dmyy(i-2*nxmax,j,0,dux,duy,duz)
                kzz(i,j)=dmzz(i-2*nxmax,j,0,dux,duy,duz)
                kxy(i,j)=dmxy(i-2*nxmax,j,0,dux,duy,duz)
            END DO
        END DO
!
        DO j=nymax+1, 2*nymax-1
            DO i=0, nxmax-1
                kxx(i,j)=dmxx(i,j-2*nymax,0,dux,duy,duz)
                kyy(i,j)=dmyy(i,j-2*nymax,0,dux,duy,duz)
                kzz(i,j)=dmzz(i,j-2*nymax,0,dux,duy,duz)
                kxy(i,j)=dmxy(i,j-2*nymax,0,dux,duy,duz)
            END DO
        END DO
!
        DO j=nymax+1, 2*nymax-1
            DO i=nxmax+1, 2*nxmax-1
                kxx(i,j)=dmxx(i-2*nxmax,j-2*nymax,0,dux,duy,duz)
                kyy(i,j)=dmyy(i-2*nxmax,j-2*nymax,0,dux,duy,duz)
                kzz(i,j)=dmzz(i-2*nxmax,j-2*nymax,0,dux,duy,duz)
                kxy(i,j)=dmxy(i-2*nxmax,j-2*nymax,0,dux,duy,duz)
            END DO
        END DO
! Middle planes (lines) are padded with 0 to keep 2^n size
        DO i=0, 2*nxmax-1
            kxx(i,nymax)=0
            kyy(i,nymax)=0
            kzz(i,nymax)=0
            kxy(i,nymax)=0
        END DO
!
        DO j=0, 2*nymax-1
            kxx(nxmax,j)=0
            kyy(nxmax,j)=0
            kzz(nxmax,j)=0
            kxy(nxmax,j)=0
        END DO
! Transform normization
        DO j=0, ny2-1
```

```
              DO i=0, nx2-1
                    kxx(i,j)=kxx(i,j)/Nnorm
                    kyy(i,j)=kyy(i,j)/Nnorm
                    kzz(i,j)=kzz(i,j)/Nnorm
                    kxy(i,j)=kxy(i,j)/Nnorm
              END DO
       END DO
! FFTW initialization
       CALL DZFFT2D(0,nx2,ny2,1.d0,kxx,nx2+2,kxx,nxmax+1,table,work,0)
       CALL ZDFFT2D(0,nx2,ny2,1.d0,kxx,nx2+2,kxx,nxmax+1,tab2,work,0)
! Conversion to Fourier space
       CALL DZFFT2D(1,nx2,ny2,1.d0,kxx,nx2+2,kxx,nxmax+1,table,work,0)
       CALL DZFFT2D(1,nx2,ny2,1.d0,kyy,nx2+2,kyy,nxmax+1,table,work,0)
       CALL DZFFT2D(1,nx2,ny2,1.d0,kzz,nx2+2,kzz,nxmax+1,table,work,0)
       CALL DZFFT2D(1,nx2,ny2,1.d0,kxy,nx2+2,kxy,nxmax+1,table,work,0)
!
       END
```

5. Contents in the file "*rk2d.f*":

```
    SUBROUTINE ODEINT(y,x1,x2,work)
!*********************************************************************
! Intermediate driver for RKSUITE solver. Integrates system of ODEs
! from x1 to x2 with initial condition y, in steps of dxsav.
! The stepping procedure do not take into account any physical
! organization of cells.
! See RKSUITE documentation for details of UT (or CT) usage.
!*********************************************************************
    INCLUDE 'global2d.inc'
!
    REAL*8 y(Nmax), x1, x2, work(*)
    INTEGER*4 i, j, flag, lenwrk
    REAL*8 x, t_eff
    PARAMETER(t_eff=66.160297d0)
    REAL*8 xmag, ymag, zmag ! functions calculating magnetic moment
! Arrays transfering condition of the system between UT calls
    REAL*8 dy(Nmax), ymax(Nmax)
! Saving parameters
    INTEGER*4 kount
```

```
    REAL*8 dxsav, xm(0:kmax), ym(0:kmax), zm(0:kmax), t(0:kmax)
    COMMON /PATH/ kount, dxsav, xm, ym, zm, t
!
    EXTERNAL derivs
!
    x=x1+dxsav
! Initial condition statistics (kount=0)
    t(kount)=x1*t_eff
    xm(kount)=ymag(y)/(nxmax*nymax)
    ym(kount)=xmag(y)/(nxmax*nymax)
    zm(kount)=zmag(y)/(nxmax*nymax)
!
    OPEN(7,FILE='average.dat',FORM='formatted',ACCESS='append')
    WRITE(7,300) kount, t(kount), xm(kount), ym(kount), zm(kount)
    CLOSE(7)
!
    CALL sav(kount,y)
! Stepping loop
    DO WHILE (x .lt. x2)
    CALL UT(derivs,x,x1,y,dy,ymax,work,flag)
! Force theta phi to be in appropriate quadrants
    DO j=1, Nmax
        IF (y(j) .gt. 2*Pi) THEN
            y(j)=y(j)-2*Pi
        ELSE IF (y(j) .lt. -2*Pi) THEN
            y(j)=y(j)+2*Pi
        END IF
    END DO
! Recording of intermediate states
    IF (kount .le. kmax-1) THEN
        kount=kount+1
        t(kount)=x1*t_eff
        xm(kount)=ymag(y)/(nxmax*nymax)
        ym(kount)=xmag(y)/(nxmax*nymax)
        zm(kount)=zmag(y)/(nxmax*nymax)
!
        OPEN(7,FILE='average.dat',FORM='formatted',ACCESS='append')
        WRITE(7,300) kount, t(kount), xm(kount), ym(kount), zm(kount)
        CLOSE(7)
!
        CALL sav(kount,y)
```

```
! Recording of the full state of the system. Allows to
! continue calculations if the program execution is broken
       OPEN(1,FILE=datlast,FORM='unformatted')
       WRITE(1) mask, y, x1, kount
       CLOSE(1)
!
    END IF
!
    x=x1+dxsav
!
    END DO
!
    x=x2
! Final call to UT
    CALL UT(derivs,x,x1,y,dy,ymax,work,flag)
! Recording of final state (see details above)
    kount=kount+1
    t(kount)=x1*t_eff
    xm(kount)=ymag(y)/(nxmax*nymax)
    ym(kount)=xmag(y)/(nxmax*nymax)
    zm(kount)=zmag(y)/(nxmax*nymax)
!
    OPEN(7,FILE='average.dat',FORM='formatted',ACCESS='append')
    WRITE(7,300) kount, t(kount), xm(kount), ym(kount), zm(kount)
    CLOSE(7)
!
    CALL sav(kount,y)
!
    OPEN(1,FILE=datlast,FORM='unformatted')
    WRITE(1) mask, y, x1, kount
    CLOSE(1)
 300FORMAT(I8,' ',F12.4,3(' ',F12.8))
!
    END
```

6. Contents in the file "*deriv2d.f*":

```
    SUBROUTINE derivs(x,tp,dtp)
!*******************************************************
! Procedure calculates theta- and phi- components
! of the effective magnetic field and finds
! respective time derivatives dtp(1,i,j,k) and dtp(2,i,j,k)
```

```
! Attention: for time-dependent field its components
! will have to be given as functions of t.
!*********************************************************
    INCLUDE 'global2d.inc'
!
    REAL*8 x ! independent variable (time)
    REAL*8 tp(2,nxmax,nymax), dtp(2,nxmax,nymax)
    REAL*8 hd(2,nxmax,nymax) ! demagnetizing & external field
    INTEGER*4 i, j
    REAL*8 hth, hph ! exchange field; overall effective field
!
    CALL hdem(x,tp,hd) ! demagnetizing and external field
! Calculation of the effective field
    DO j=1, nymax
        DO i=1, nxmax
!
            hth=-((SIN(tp(1,i,j))*COS(tp(1,i+mask(-1,i,j),j))
    &               -COS(tp(1,i,j))*SIN(tp(1,i+mask(-1,i,j),j))
    &               *COS(tp(2,i,j)-tp(2,i+mask(-1,i,j),j)))
    &               +(SIN(tp(1,i,j))*COS(tp(1,i+mask(1,i,j),j))
    &               -COS(tp(1,i,j))*SIN(tp(1,i+mask(1,i,j),j))
    &               *COS(tp(2,i,j)-tp(2,i+mask(1,i,j),j))))*d2x
    &               -((SIN(tp(1,i,j))*COS(tp(1,i,j+mask(-2,i,j)))
    &               -COS(tp(1,i,j))*SIN(tp(1,i,j+mask(-2,i,j)))
    &               *COS(tp(2,i,j)-tp(2,i,j+mask(-2,i,j))))
    &               +(SIN(tp(1,i,j))*COS(tp(1,i,j+mask(2,i,j)))
    &               -COS(tp(1,i,j))*SIN(tp(1,i,j+mask(2,i,j)))
    &               *COS(tp(2,i,j)-tp(2,i,j+mask(2,i,j)))))*d2y
    &               +hd(1,i,j)
!
            hph=-(SIN(tp(1,i+mask(-1,i,j),j))
    &               *SIN(tp(2,i,j)-tp(2,i+mask(-1,i,j),j))
    &               +SIN(tp(1,i+mask(1,i,j),j))
    &               *SIN(tp(2,i,j)-tp(2,i+mask(1,i,j),j)))*d2x
    &               -(SIN(tp(1,i,j+mask(-2,i,j)))
    &               *SIN(tp(2,i,j)-tp(2,i,j+mask(-2,i,j)))
    &               +SIN(tp(1,i,j+mask(2,i,j)))
    &               *SIN(tp(2,i,j)-tp(2,i,j+mask(2,i,j))))*d2y
    &               +hd(2,i,j)
! Calculation of theta- and phi- derivatives
            dtp(1,i,j)=(alpha*hth+hph)*mask(0,i,j)
```

```
            IF (SIN(tp(1,i,j)) .eq. 0) THEN
                WRITE(*,*) 'bad coordinate system'
            END IF
            dtp(2,i,j)=(alpha*hph-hth)/SIN(tp(1,i,j))*mask(0,i,j)
        END DO
    END DO
!
    END
```

7. Contents in the file "*demag2d.f*":

```
    SUBROUTINE hdem(x,tp,hd)
!*********************************************************
! Calculates the demagnetizing field.
!*********************************************************
    INCLUDE 'global2d.inc'
!
    REAL*8 x, hanis1, hanis2
    REAL*8 tp(2,0:nxmax-1,0:nymax-1) !Spherical components of M
    REAL*8 hd(2,0:nxmax-1,0:nymax-1)
!
    INTEGER*4 i, j, nmaxc
    PARAMETER(nmaxc=2*(nxmax+1)*nymax) ! Transform size
! Fourier images of demagnetizing matrices
    COMPLEX*16 kxx(nmaxc)
    COMPLEX*16 kyy(nmaxc)
    COMPLEX*16 kzz(nmaxc)
    COMPLEX*16 kxy(nmaxc)
    REAL*8 table((nx2+15)+2*(ny2+15))
    REAL*8 tab2((nx2+15)+2*(ny2+15))
    REAL*8 work(nx2+4*ny2)
    COMMON /dcoef/ kxx, kyy, kzz, kxy, table, tab2, work
! Cartesian components of M
    REAL*8 xm(0:nx2+1,0:ny2-1)
    REAL*8 ym(0:nx2+1,0:ny2-1)
    REAL*8 zm(0:nx2+1,0:ny2-1)
! Fourier images of Mx, My, Mz, as calculated by real -> complex transform
    COMPLEX*16 xmc(nmaxc)
    COMPLEX*16 ymc(nmaxc)
```

```
    COMPLEX*16 zmc(nmaxc)
! These matrices are equivalenced to save space
    EQUIVALENCE (xm,xmc), (ym,ymc), (zm,zmc)
! The external field components
    REAL*8 hx, hy, hz
! Cartesian components of Hd
    REAL*8 xm1(0:nx2+1,0:ny2-1)
    REAL*8 ym1(0:nx2+1,0:ny2-1)
    REAL*8 zm1(0:nx2+1,0:ny2-1)
! Fourier images of Hx, Hy, Hz, as calculated by real -> complex transform
    COMPLEX*16 xm1c(nmaxc)
    COMPLEX*16 ym1c(nmaxc)
    COMPLEX*16 zm1c(nmaxc)
! These matrices are equivalenced to save space
    EQUIVALENCE (xm1,xm1c),(ym1,ym1c),(zm1,zm1c)
! Thermal term
    INTEGER nrand, num, iseed(4), flag
    REAL*8 dt, dxsav, Ms, kbolts, temp, gamma, eps, t_eff, volume
    PARAMETER(nrand=3*nxmax*nymax) ! 1 random number for each M-component
    REAL*8 rand(nrand), idum(2)
    REAL gasdev
    PARAMETER(Ms=859.436693d0) ! in unit of emu/cm^3
    PARAMETER(kbolts=1.380650d-16) ! in unit of erg/K
    PARAMETER(temp=300.0)      ! in unit of K
    PARAMETER(gamma=1.760860d+7)   ! in unit of 1/(sec*Gauss)
    PARAMETER(t_eff=66.082897d-12)

       COMMON /PATH/ dxsav
! The first octant (quadrant) of Cartesian M components is
! filled with the respective values. Rest are zero-padded.
    DO j=0, nymax-1
        DO i=0, nxmax-1
!
            xm(i+nxmax,j+nymax)=0
            ym(i+nxmax,j+nymax)=0
            zm(i+nxmax,j+nymax)=0
!
            xm(i+nxmax,j)=0
            ym(i+nxmax,j)=0
            zm(i+nxmax,j)=0
!
```

```
            xm(i,j+nymax)=0
            ym(i,j+nymax)=0
            zm(i,j+nymax)=0
!
            xm(i,j)=SIN(tp(1,i,j))*COS(tp(2,i,j))*maskp(0,i,j)
            ym(i,j)=SIN(tp(1,i,j))*SIN(tp(2,i,j))*maskp(0,i,j)
            zm(i,j)=COS(tp(1,i,j))*maskp(0,i,j)
!
        END DO
    END DO
! Conversion of M components to Fourier space
    CALL DZFFT2D(1,nx2,ny2,1.d0,xm,nx2+2,xmc,nxmax+1,table,work,0)
    CALL DZFFT2D(1,nx2,ny2,1.d0,ym,nx2+2,ymc,nxmax+1,table,work,0)
    CALL DZFFT2D(1,nx2,ny2,1.d0,zm,nx2+2,zmc,nxmax+1,table,work,0)
! Fourier images of M are multipled by demagnetizing
! coefficients transform - this is a convolution in
! the Fourier space, that produces image of Hd field.
    DO i=1, nmaxc
        xm1c(i)=kxx(i)*xmc(i)+kxy(i)*ymc(i)
        ym1c(i)=kxy(i)*xmc(i)+kyy(i)*ymc(i)
        zm1c(i)=kzz(i)*zmc(i)
    END DO
! Conversion of Hd to real space
    CALL ZDFFT2D(-1,nx2,ny2,1.d0,xm1c,nxmax+1,xm1,nx2+2,tab2,work,0)
    CALL ZDFFT2D(-1,nx2,ny2,1.d0,ym1c,nxmax+1,ym1,nx2+2,tab2,work,0)
    CALL ZDFFT2D(-1,nx2,ny2,1.d0,zm1c,nxmax+1,zm1,nx2+2,tab2,work,0)
! Generate Gaussian random numbers
    iseed(1)=2*(INT(10*tp(1,13,65)/tp(2,16,47)))**2+1
    iseed(2)=2*(INT(10*tp(1,68,11)/tp(2,26,34)))**2+1
    iseed(3)=2*(INT(10*tp(2,18,10)/tp(1,54,40)))**2+1
    iseed(4)=2*(INT(10*tp(2,21,52)/tp(1,31,76)))**2+1
!
    DO num=1, nrand
         rand(num)=gasdev(idum,iseed)
    END DO
!
    dt=0.1*t_eff
    volume=(sizex/nxmax)*(sizey/nymax)*thick*1.d-21    ! in unit of cm^3
    eps=2.0*alpha*kbolts*temp/(gamma*volume*Ms)
! Spherical components of demagnetizing and external fields.
    DO j=0, nymax-1
```

```
        DO i=0, nxmax-1
! The external field
            CALL hfun(x,i+1,j+1,hx,hy,hz)
! Combination of demagnetizing, external, and anisotropy fields
            xm1(i,j)=xm1(i,j)+hx
            ym1(i,j)=ym1(i,j)+hy
            zm1(i,j)=zm1(i,j)+hz
! Include thermal term if necessary
            xm1(i,j)=xm1(i,j)+sqrt(eps/dt)*rand(nxmax*j+3*i+1)/Ms
            ym1(i,j)=ym1(i,j)+sqrt(eps/dt)*rand(nxmax*j+3*i+2)/Ms
            zm1(i,j)=zm1(i,j)+sqrt(eps/dt)*rand(nxmax*j+3*i+3)/Ms
! Checkpoint for thermal term
    IF (x .le. -38.0 .and. i .eq. 64 .and. j .eq. 256) THEN
        OPEN(22,FILE='thermal.dat',FORM='formatted',ACCESS='append')
        WRITE(22,220) eps, dt, sqrt(eps/dt)*rand(nxmax*j+3*i+1)/Ms
        CLOSE(22)
 220    FORMAT(3(E12.6,' '))
    END IF
! Conversion to spheric coordinates
            hd(1,i,j)=xm1(i,j)*COS(tp(1,i,j))*COS(tp(2,i,j))
    &               +ym1(i,j)*COS(tp(1,i,j))*SIN(tp(2,i,j))
    &               -zm1(i,j)*SIN(tp(1,i,j))
    &               +hanis1(i+1,j+1,tp)
            hd(2,i,j)=-xm1(i,j)*SIN(tp(2,i,j))
    &               +ym1(i,j)*COS(tp(2,i,j))
    &               +hanis2(i+1,j+1,tp)
!
        END DO
    END DO
!
    END
!
! Gaussian random number generator
!
    FUNCTION gasdev(idum,iseed)
    REAL gasdev
    INTEGER iset
    INTEGER iseed(4)
!
    DOUBLE PRECISION idum(2)
    REAL v1, v2, rsq, fac, gset
```

```
      SAVE iset, gset
      DATA iset/0/
!
      IF (iset .eq. 0) THEN
 1        CALL dlarnv(1,iseed,2,idum)
          v1=2*idum(1)-1
          v2=2*idum(2)-1
          rsq=v1**2+v2**2
          IF (rsq .ge. 1 .or. rsq .eq. 0) GOTO 1
          fac=sqrt(-2.*log(rsq)/rsq)
          gset=v1*fac
          gasdev=v2*fac
          iset=1
      ELSE
          gasdev=gset
          iset=0
      END IF
!
      RETURN
      END
```

8. Contents in the file "*anisot2d.f*":

```
c************************************************************
c This is the first version to calculate anisotropy field.
c We assume a uniaxial anisotropy with the easy axis along
c the y (actually x - we change coordinates later) axis.
c From Wayne Hiebert's data we have Hk=8-10 Oe
c a value which may change with new deposition conditions.
c In this routine Hk is normalized by Ms (not 4*Pi*Ms).
c this gives Hk=8.594366927(Oe).
c************************************************************
! Uniaxial anisotropy term
! The theta-component
      REAL*8 FUNCTION hanis1(i,j,tp)
      INCLUDE 'global2d.inc'
!
      INTEGER*4 i, j
      REAL*8 tp(1:2,1:nxmax,1:nymax), Hk
```

```
!
    PARAMETER(Hk=0.01)
!
    hanis1=Hk*SIN(2*tp(1,i,j))*(SIN(tp(2,i,j)))**2*mask(0,i,j)
!
    END

! The phi-component
    REAL*8 FUNCTION hanis2(i,j,tp)
    INCLUDE 'global2d.inc'
!
    INTEGER*4 i, j
    REAL*8 tp(1:2,1:nxmax,1:nymax), Hk
!
    PARAMETER(Hk=0.01)
!
    hanis2=Hk*SIN(tp(1,i,j))*SIN(2*tp(2,i,j))*mask(0,i,j)
!
    END
```

9. Contents in the file "*demfor2d.f*":

```
!*****************************************************************
! Functions describing demagnetizing tensor components. i,j,k
! define relative position of two cells, while dx,dy,dz define
! their sixe
! See: Y.Nakatani, Y.Uesaka, N.Hayashi, "Direct Solution of the
! Landau-Lifshitz-Gilbert equation for micromagnetics",
! Jap.J.Appl.Phys., 28 (1989) 2485-507.
!*****************************************************************
    REAL*8 FUNCTION dmxx(i,j,k,dx,dy,dz)
!
    INTEGER*4 i, j, k
    REAL*8 dx, dy, dz
!
    dmxx=ATAN((dy*dz*(-0.5+j)*(-0.5+k))/
    &       (dx*(-0.5+i)*SQRT(dx**2*(-0.5+i)**2+dy**2*(-0.5+j)**2+
    &       dz**2*(-0.5+k)**2)))-ATAN((dy*dz*(-0.5+j)*(-0.5+k))/
    &       (dx*(0.5+i)*SQRT(dx**2*(0.5+i)**2+dy**2*(-0.5+j)**2+
```

```
     &       dz**2*(-0.5+k)**2)))-ATAN((dy*dz*(0.5+j)*(-0.5+k))/
     &       (dx*(-0.5+i)*SQRT(dx**2*(-0.5+i)**2+dy**2*(0.5+j)**2+
     &       dz**2*(-0.5+k)**2)))+ATAN((dy*dz*(0.5+j)*(-0.5+k))/
     &       (dx*(0.5+i)*SQRT(dx**2*(0.5+i)**2+dy**2*(0.5+j)**2+
     &       dz**2*(-0.5+k)**2)))-ATAN((dy*dz*(-0.5+j)*(0.5+k))/
     &       (dx*(-0.5+i)*SQRT(dx**2*(-0.5+i)**2+dy**2*(-0.5+j)**2+
     &       dz**2*(0.5+k)**2)))+ATAN((dy*dz*(-0.5+j)*(0.5+k))/
     &       (dx*(0.5+i)*SQRT(dx**2*(0.5+i)**2+dy**2*(-0.5+j)**2+
     &       dz**2*(0.5+k)**2)))+ATAN((dy*dz*(0.5+j)*(0.5+k))/
     &       (dx*(-0.5+i)*SQRT(dx**2*(-0.5+i)**2+dy**2*(0.5+j)**2+
     &       dz**2*(0.5+k)**2)))-ATAN((dy*dz*(0.5+j)*(0.5+k))/
     &       (dx*(0.5+i)*SQRT(dx**2*(0.5+i)**2+dy**2*(0.5+j)**2+
     &       dz**2*(0.5+k)**2)))
!
     END
! -------------------------------------------------------
     REAL*8 FUNCTION dmyy(i,j,k,dx,dy,dz)
!
     INTEGER*4 i, j, k
     REAL*8 dx, dy, dz
!
     dmyy=ATAN((dx*dz*(-0.5+i)*(-0.5+k))/
     &       (dy*(-0.5+j)*SQRT(dx**2*(-0.5+i)**2+dy**2*(-0.5+j)**2+
     &       dz**2*(-0.5+k)**2)))-ATAN((dx*dz*(0.5+i)*(-0.5+k))/
     &       (dy*(-0.5+j)*SQRT(dx**2*(0.5+i)**2+dy**2*(-0.5+j)**2+
     &       dz**2*(-0.5+k)**2)))-ATAN((dx*dz*(-0.5+i)*(-0.5+k))/
     &       (dy*(0.5+j)*SQRT(dx**2*(-0.5+i)**2+dy**2*(0.5+j)**2+
     &       dz**2*(-0.5+k)**2)))+ATAN((dx*dz*(0.5+i)*(-0.5+k))/
     &       (dy*(0.5+j)*SQRT(dx**2*(0.5+i)**2+dy**2*(0.5+j)**2+
     &       dz**2*(-0.5+k)**2)))-ATAN((dx*dz*(-0.5+i)*(0.5+k))/
     &       (dy*(-0.5+j)*SQRT(dx**2*(-0.5+i)**2+dy**2*(-0.5+j)**2+
     &       dz**2*(0.5+k)**2)))+ATAN((dx*dz*(0.5+i)*(0.5+k))/
     &       (dy*(-0.5+j)*SQRT(dx**2*(0.5+i)**2+dy**2*(-0.5+j)**2+
     &       dz**2*(0.5+k)**2)))+ATAN((dx*dz*(-0.5+i)*(0.5+k))/
     &       (dy*(0.5+j)*SQRT(dx**2*(-0.5+i)**2+dy**2*(0.5+j)**2+
     &       dz**2*(0.5+k)**2)))-ATAN((dx*dz*(0.5+i)*(0.5+k))/
     &       (dy*(0.5+j)*SQRT(dx**2*(0.5+i)**2+dy**2*(0.5+j)**2+
     &       dz**2*(0.5+k)**2)))
!
     END
! -------------------------------------------------------
```

```
      REAL*8 FUNCTION dmzz(i,j,k,dx,dy,dz)
!
      INTEGER*4 i, j, k
      REAL*8 dx, dy, dz
!
      dmzz=ATAN((dx*dy*(-0.5+i)*(-0.5+j))/
     &      (dz*SQRT(dx**2*(-0.5+i)**2+dy**2*(-0.5+j)**2+
     &      dz**2*(-0.5+k)**2)*(-0.5+k)))-
     &      ATAN((dx*dy*(0.5+i)*(-0.5+j))/
     &      (dz*SQRT(dx**2*(0.5+i)**2+dy**2*(-0.5+j)**2+
     &      dz**2*(-0.5+k)**2)*(-0.5+k)))-
     &      ATAN((dx*dy*(-0.5+i)*(0.5+j))/
     &      (dz*SQRT(dx**2*(-0.5+i)**2+dy**2*(0.5+j)**2+
     &      dz**2*(-0.5+k)**2)*(-0.5+k)))+
     &      ATAN((dx*dy*(0.5+i)*(0.5+j))/
     &      (dz*SQRT(dx**2*(0.5+i)**2+dy**2*(0.5+j)**2+
     &      dz**2*(-0.5+k)**2)*(-0.5+k)))-
     &      ATAN((dx*dy*(-0.5+i)*(-0.5+j))/
     &      (dz*(0.5+k)*SQRT(dx**2*(-0.5+i)**2+dy**2*(-0.5+j)**2+
     &      dz**2*(0.5+k)**2)))+ATAN((dx*dy*(0.5+i)*(-0.5+j))/
     &      (dz*(0.5+k)*SQRT(dx**2*(0.5+i)**2+dy**2*(-0.5+j)**2+
     &      dz**2*(0.5+k)**2)))+ATAN((dx*dy*(-0.5+i)*(0.5+j))/
     &      (dz*(0.5+k)*SQRT(dx**2*(-0.5+i)**2+dy**2*(0.5+j)**2+
     &      dz**2*(0.5+k)**2)))-ATAN((dx*dy*(0.5+i)*(0.5+j))/
     &      (dz*(0.5+k)*SQRT(dx**2*(0.5+i)**2+dy**2*(0.5+j)**2+
     &      dz**2*(0.5+k)**2)))
!
      END
! -------------------------------------------------------------
      REAL*8 FUNCTION dmxy(i,j,k,dx,dy,dz)
!
      INTEGER*4 i, j, k
      REAL*8 dx, dy, dz
!
      dmxy=-LOG(ABS(SQRT(dx**2*(-0.5+i)**2+dy**2*(-0.5+j)**2+
     &      dz**2*(-0.5+k)**2)+dz*(-0.5+k)))+
     &      LOG(ABS(SQRT(dx**2*(0.5+i)**2+dy**2*(-0.5+j)**2+
     &      dz**2*(-0.5+k)**2)+dz*(-0.5+k)))+
     &      LOG(ABS(SQRT(dx**2*(-0.5+i)**2+dy**2*(0.5+j)**2+
     &      dz**2*(-0.5+k)**2)+dz*(-0.5+k)))-
     &      LOG(ABS(SQRT(dx**2*(0.5+i)**2+dy**2*(0.5+j)**2+
```

```
    &       dz**2*(-0.5+k)**2)+dz*(-0.5+k)))+LOG(ABS(dz*(0.5+k)+
    &       SQRT(dx**2*(-0.5+i)**2+dy**2*(-0.5+j)**2+
    &       dz**2*(0.5+k)**2)))-
    &       LOG(ABS(dz*(0.5+k)+SQRT(dx**2*(0.5+i)**2+
    &       dy**2*(-0.5+j)**2+
    &       dz**2*(0.5+k)**2)))-LOG(ABS(dz*(0.5+k)+
    &       SQRT(dx**2*(-0.5+i)**2+dy**2*(0.5+j)**2+
    &       dz**2*(0.5+k)**2)))+
    &       LOG(ABS(dz*(0.5+k)+SQRT(dx**2*(0.5+i)**2+
    &       dy**2*(0.5+j)**2+
    &       dz**2*(0.5+k)**2)))
!
    END
```

10. Contents in the file "*output2d.f*":

```
    REAL*8 FUNCTION xmag(tp)
!*****************************************************
! Function calculates x-component of the sample
! magnetic moment in arbitrary units
!*****************************************************
    INCLUDE 'global2d.inc'
!
    REAL*8 tp(2,nxmax,nymax)
    INTEGER*4 i, j
!
    xmag=0
    DO j=1, nymax
        DO i=1, nxmax
            xmag=xmag+SIN(tp(1,i,j))*COS(tp(2,i,j))*mask(0,i,j)
        END DO
    END DO
!
    END
! -----------------------------------------------------
    REAL*8 FUNCTION ymag(tp)
!*****************************************************
! Function calculates y-component of the sample
! magnetic moment in arbitrary units
```

```
!*****************************************************
    INCLUDE 'global2d.inc'
!
    REAL*8 tp(2,nxmax,nymax)
    INTEGER*4 i, j
!
    ymag=0
    DO j=1, nymax
        DO i=1, nxmax
            ymag=ymag+SIN(tp(1,i,j))*SIN(tp(2,i,j))*mask(0,i,j)
        END DO
    END DO
!
    END
! -----------------------------------------------------
    REAL*8 FUNCTION zmag(tp)
!*****************************************************
! Function calculates z-component of the sample
! magnetic moment in arbitrary units
!*****************************************************
    INCLUDE 'global2d.inc'
!
    REAL*8 tp(2,nxmax,nymax)
    INTEGER*4 i, j
!
    zmag=0
    DO i=1, nxmax
        DO j=1, nymax
            zmag=zmag+COS(tp(1,i,j))*mask(0,i,j)
        END DO
    END DO
!
    END
! -----------------------------------------------------
    SUBROUTINE sav(kount,y)
    INCLUDE 'global2d.inc'
!
    INTEGER*4 kount
    REAL*8 y(2,nxmax,nymax)
    INTEGER*4 i, j
    REAL*8 xm(nxmax,nymax), ym(nxmax,nymax), zm(nxmax,nymax)
```

```
!
    CHARACTER*14 itoa
!
    DO i=1, nxmax
        DO j=1, nymax
            xm(i,j)=SIN(y(1,i,j))*COS(y(2,i,j))*mask(0,i,j)
            ym(i,j)=SIN(y(1,i,j))*SIN(y(2,i,j))*mask(0,i,j)
            zm(i,j)=COS(y(1,i,j))*mask(0,i,j)
        END DO
    END DO
!
    OPEN(1,FILE="xyz."//itoa(kount),form='formatted',
     &      status='unknown')
!
    DO i=nxmax, 1, -1
        WRITE(1,100) (xm(i,j), j=1, nymax)
    END DO
    DO i=nxmax, 1, -1
        WRITE(1,100) (ym(i,j), j=1, nymax)
    END DO
    DO i=nxmax, 1, -1
        WRITE(1,100) (zm(i,j), j=1, nymax)
    END DO
!
    CLOSE(1)
100 FORMAT(1024(f12.8))
!
    END
! ------------------------------------------
! Convert integer to character array (string)
    CHARACTER*14 FUNCTION itoa(value)
    IMPLICIT NONE
    INTEGER value
    INTEGER number
    CHARACTER*14 string
    LOGICAL minus
!
    number=value
    minus=.FALSE.
!
    IF (number .lt. 0) THEN
```

```
        minus=.TRUE.
        number=-number
    END IF
!
    string=CHAR(ICHAR('0')+MOD(number, 10))
    number=number/10
!
    DO WHILE (number .ne. 0)
        string=CHAR(ICHAR('0')+MOD(number,10))//string
        number=number/10
    END DO
!
    IF (minus .eqv. .TRUE.) THEN
        string=CHAR(ICHAR('-'))//string
    END IF
!
    itoa=string
    RETURN
!
    END
```

11. Contents in the file "*rksuite.f*":

See Ref.[17] for the original code.

Appendix II

2D Micromagnetic Simulation code

(FORTRAN90)

Below is the FORTRAN90 version of micromagnetic simulation. The code is conventionally split into 10 files, each containing subroutines or functions that perform certain tasks. The code structure is illustrated in section 2.2. Note: the file " globals.f' must be compiled first because it has a module that is used widely in other parts of the code.

1. Contents in the file "*globals.f*":

```
! Global settings.
    MODULE globals
    IMPLICIT NONE
! Name of the file for storing intermediate results
    CHARACTER(LEN=12), PARAMETER :: datapool='datapool.dat'
! Physics constants
    DOUBLE PRECISION, PARAMETER :: Pi=3.1415926535898
    DOUBLE PRECISION, PARAMETER :: Kbolts=1.3806503D-16
! Integration time range
    DOUBLE PRECISION, PARAMETER :: tmin=0.0
    DOUBLE PRECISION, PARAMETER :: tmax=200.0
    DOUBLE PRECISION, PARAMETER :: dtsav=1
! Cells numbers and actual sizes of the sample
    INTEGER, PARAMETER :: nxmax=64
    INTEGER, PARAMETER :: nymax=128
    INTEGER, PARAMETER :: isample=32
    INTEGER, PARAMETER :: jsample=64
    INTEGER, SAVE :: mask(-2:2,0:nxmax+1,0:nymax+1)
    INTEGER, PARAMETER :: Nmax=2*nxmax*nymax
    DOUBLE PRECISION, PARAMETER :: sizex=275.0
```

```
    DOUBLE PRECISION, PARAMETER :: sizey=960.0
    DOUBLE PRECISION, PARAMETER :: thick=30.0
! Parameters for the sample
    DOUBLE PRECISION, PARAMETER :: temp=300.0
    DOUBLE PRECISION, PARAMETER :: Ms=859.436693
    DOUBLE PRECISION, PARAMETER :: treduced=66.160297D-12
    DOUBLE PRECISION, PARAMETER :: un=16.455122
    DOUBLE PRECISION, PARAMETER :: alpha=0.008
    DOUBLE PRECISION, PARAMETER :: gamma=1.7588D+7
! Things used in effective field calculation
    DOUBLE PRECISION, PARAMETER :: dux=sizex/nxmax/un
    DOUBLE PRECISION, PARAMETER :: duy=sizey/nymax/un
    DOUBLE PRECISION, PARAMETER :: duz=thick/un
    DOUBLE PRECISION, PARAMETER :: d2x=1/(dux*dux)
    DOUBLE PRECISION, PARAMETER :: d2y=1/(duy*duy)
    DOUBLE PRECISION, PARAMETER :: d2z=1/(duz*duz)
!
! Below are what the FFT process needs for demagnetizing calculation
!
! Transform size
    INTEGER, PARAMETER :: nmaxc=2*(nxmax+1)*nymax
    INTEGER, PARAMETER :: nx2=2*nxmax
    INTEGER, PARAMETER :: ny2=2*nymax
    INTEGER, PARAMETER :: Nrand=3*nxmax*nymax
    INTEGER, PARAMETER :: Nnorm=4*nxmax*nymax
! Fourier images of demagnetizing matrices
    DOUBLE COMPLEX, SAVE :: kxx(nmaxc)
    DOUBLE COMPLEX, SAVE :: kyy(nmaxc)
    DOUBLE COMPLEX, SAVE :: kzz(nmaxc)
    DOUBLE COMPLEX, SAVE :: kxy(nmaxc)
! Workspace
    DOUBLE PRECISION, SAVE :: table1((nx2+15)+2*(ny2+15))
    DOUBLE PRECISION, SAVE :: table2((nx2+15)+2*(ny2+15))
    DOUBLE PRECISION, SAVE :: work(nx2+4*ny2)
! Flags for FFT procedure
    INTEGER, PARAMETER :: FFTW_FORWARD=-1, FFTW_BACKWARD=1
    INTEGER, PARAMETER :: FFTW_REAL_TO_COMPLEX=-1
    INTEGER, PARAMETER :: FFTW_COMPLEX_TO_REAL=1
    INTEGER, PARAMETER :: FFTW_ESTIMATE=0, FFTW_MEASURE=1
    INTEGER, PARAMETER :: FFTW_IN_PLACE=8, FFTW_USE_WISDOM=16
!
```

```
    END MODULE globals
```

2. Contents in the file "*sim2d.f*":

```
    PROGRAM sim2d
    USE globals
    IMPLICIT NONE
!*************************************************************
! This program traces the dynamical movement of magnetization
! in arbitrarily shaped sample with space- and time-dependent
! external field. The main procedure is a customizable driver.
! The code below is a typical example.
!*************************************************************
! External subroutines/functions.
    EXTERNAL sigterm, sigusr1, sigusr2
    EXTERNAL initdem, shape, initmag, saveimage, ODE2D
!
! Spherical coordinates angles of magnetization distribution.
    DOUBLE PRECISION :: tp(1:2,1:nxmax,1:nymax)
!
! Current temporal point, integration step and number of output.
    DOUBLE PRECISION :: tnow
    INTEGER :: nstep, noutput
!
! Auxilliary variables.
    INTEGER :: i, j
    LOGICAL :: filestatus
!
! Signal handler for (software) kill signals, signal number 15,16,17.
    CALL signal(15, sigterm, -1)
    CALL signal(16, sigusr1, -1)
    CALL signal(17, sigusr2, -1)
!
! Monitoring checkpoint.
    OPEN(999,FILE='eye.txt',ACCESS='sequential',POSITION='append')
    WRITE(999,*) '# Simulation starts:'
    WRITE(999,*)
    CLOSE(999)
!
```

```
! Initialize FFT space and demagnetizing tensor coefficients.
    CALL initdem
!
! See if the job is to be resumed.
    INQUIRE(FILE=datapool,EXIST=filestatus)
!
    IF (filestatus .EQV. .FALSE.) THEN
! Set the start point.
        tnow=tmin
        nstep=0
        noutput=0
! Topological features of the sample; Initialize mask()
        CALL shape(mask)
! Initial uniform distribution of magnetizations
        CALL initmag(tp,mask)
! Save the initial distribution (1st image)
        CALL saveimage(noutput,tp)
        noutput=noutput+1
!
    ELSE
! Checkpoint

    OPEN(999,FILE='eye.txt',ACCESS='sequential',POSITION='append')
        WRITE(999,*) '# Reading data from datapool.dat:'
        CLOSE(999)
!
        OPEN(10,FILE=datapool,FORM='unformatted')
        READ(10) tnow, tp, mask, nstep, noutput
        CLOSE(10)
! Checkpoint

    OPEN(999,FILE='eye.txt',ACCESS='sequential',POSITION='append')
        WRITE(999,*) '> nstep,noutput,tnow=', nstep, noutput, tnow
        WRITE(999,*) '# Workspace refreshed.'
        WRITE(999,*)
        CLOSE(999)
!
    END IF
! Checkpoint
    OPEN(999,FILE='eye.txt',ACCESS='sequential',POSITION='append')
    WRITE(999,*) '# Integration starts:'
```

```
    WRITE(999,*)
    CLOSE(999)
!
    CALL ODE2D(tp,tnow,tmax,nstep,noutput)
!
! Checkpoint
    OPEN(999,FILE='eye.txt',ACCESS='sequential',POSITION='append')
    WRITE(999,*) '# Simulation finished!'
    CLOSE(999)
!
    END PROGRAM sim2d
```

3. Contents in the file "*interupt.f*":

```
! Signal handler
    SUBROUTINE sigterm()
!   IMPLICIT NONE
! Exit and resubmit program to queue
        STOP 99
    RETURN
!   END SUBROUTINE sigterm
    END
!
! ><><><><><><><><><><><><><><><><><><><><><><><><
!
    SUBROUTINE sigusr1()
!   IMPLICIT NONE
! Exit and resubmit program to queue
        STOP 99
    RETURN
!   END SUBROUTINE sigusr1
    END
!
! ><><><><><><><><><><><><><><><><><><><><><><><><
!
    SUBROUTINE sigusr2()
!   IMPLICIT NONE
! Exit and resubmit program to queue
        STOP 99
```

```
    RETURN
!   END SUBROUTINE sigusr2
    END
!
```

4. Contents in the file "*init2d.f*":

```
    SUBROUTINE hext(t,i,j,hx,hy,hz)
    USE globals
    IMPLICIT NONE
!*********************************************************************
! Space/time-dependences of the external magnetic field.
!*********************************************************************
    DOUBLE PRECISION :: t, hx, hy, hz
    INTEGER :: i, j
! Biasing x-field
    DOUBLE PRECISION, PARAMETER :: hbiasx=0.0/Ms
! Biasing y-field
    DOUBLE PRECISION, PARAMETER :: hbiasy=0.0/Ms
! Biasing z-field
    DOUBLE PRECISION, PARAMETER :: hbiasz=-3600.0/Ms
!
    hx=hbiasx
    hy=hbiasy
    hz=hbiasz
!
    END SUBROUTINE hext
! ------------------------------------------------------------------
! ><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
! ------------------------------------------------------------------
    SUBROUTINE shape
    USE globals
    IMPLICIT NONE
!************************************************
! Procedure prepares a mask defining the shape of
! the sample
! - interior:  mask(0,i,j)=1
! - exterior:  mask(0,i,j)=0
!
```

```
! Other components are used for data branching in
! exchange energy calculation
! - internal sites:
!    mask(-1,i,j)=-1  mask(1,i,j)=1 (for x-direction)
!    mask(-2,i,j)=-1  mask(2,i,j)=1 (for y-direction)
! - boundary sites: sign changes to maintain
!   proper boundary conditions
!
! Parameter rx*ry defines the shape:
! <1 - full rectangle nxmax x nymax
! >=1 - ellipse with axes 2*rx and 2*ry,
! overlapped on the original rectangle
!************************************************
    INTEGER :: i, j
! Calculation for full rectangle
    mask=0
!
    DO i=1, nxmax
        DO j=1, nymax
            mask(-2,i,j)=-1
            mask(-1,i,j)=-1
            mask(0,i,j)=1
            mask(1,i,j)=1
            mask(2,i,j)=1
        END DO
    END DO
! Sample's outer edge
    DO i=1, nxmax
        mask(-2,i,1)=0
        mask(2,i,nymax)=0
    END DO
!
    DO j=1, nymax
        mask(-1,1,j)=0
        mask(1,nxmax,j)=0
    END DO
! The defect
    DO i=imin, imax
        DO j=jmin+1, jmax-1
            mask(-2,i,j)=0
            mask(-1,i,j)=0
```

```
            mask(0,i,j)=0
            mask(1,i,j)=0
            mask(2,i,j)=0
        END DO
    END DO
!
    DO i=imin, imax
        mask(-2,i,jmax)=0
        mask(2,i,jmin)=0
    END DO
!
    END SUBROUTINE shape
! ------------------------------------------------------------
! ><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
! ------------------------------------------------------------
    SUBROUTINE initmag(tp)
    USE globals
    IMPLICIT NONE
    DOUBLE PRECISION :: tp(1:2,1:nxmax,1:nymax)
!************************************************
! Procedure fills tp matrix with in-plane uniform
! distribution (idir=0 - x-direction, idir=1 - y,
! idir=2 - z).
! Grid sites outside the sample borders are filled
! formally with z orientation.
!************************************************
!
    INTEGER :: i, j
    DOUBLE PRECISION :: rand(1:nxmax,1:nymax)
!
    CALL RANDOM_NUMBER(rand)
!
    DO j=1, nymax
        DO i=1, nxmax
            tp(1,i,j)=mask(0,i,j)*Pi/2
    &              +(1-mask(0,i,j))*(1-rand(i,j))*0.0001
            tp(2,i,j)=mask(0,i,j)*Pi/2
        END DO
    END DO
!
    END SUBROUTINE initmag
```

```
! ------------------------------------------------------------
! ><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
! ------------------------------------------------------------
! Initialize demagnetizing field computation. Do it only once!
    SUBROUTINE initdem
    USE globals
    IMPLICIT NONE
!
    DOUBLE PRECISION, EXTERNAL :: dmxx, dmyy, dmzz, dmxy
!
    DOUBLE PRECISION :: kxxr(0:nx2+1,0:ny2-1)
    DOUBLE PRECISION :: kyyr(0:nx2+1,0:ny2-1)
    DOUBLE PRECISION :: kzzr(0:nx2+1,0:ny2-1)
    DOUBLE PRECISION :: kxyr(0:nx2+1,0:ny2-1)
! duxn(), duyn(), duzn() are noise-modified values for dux, duy, duz
    DOUBLE PRECISION :: duxn(0:2*nxmax-1,0:2*nymax-1)
    DOUBLE PRECISION :: duyn(0:2*nxmax-1,0:2*nymax-1)
    DOUBLE PRECISION :: duzn(0:2*nxmax-1,0:2*nymax-1)
    DOUBLE PRECISION :: noise(0:2*nxmax-1,0:2*nymax-1)
!
    INTEGER :: i, j
!
    CALL RANDOM_NUMBER(noise)
    DO i=0, 2*nxmax-1
        DO j=0, 2*nymax-1
            duxn(i,j)=dux*(1+0.01*(noise(i,j)-0.5))
            duyn(i,j)=duy*(1+0.01*(noise(i,j)-0.5))
            duzn(i,j)=duz*(1+0.01*(noise(i,j)-0.5))
        END DO
    END DO
! Demagnetization matrices are filled with the respective values
! of the demagnetizing coefficients.
    DO j=0, nymax-1
        DO i=0, nxmax-1
            kxxr(i,j)=dmxx(i,j,0,duxn(i,j),duyn(i,j),duzn(i,j))
            kyyr(i,j)=dmyy(i,j,0,duxn(i,j),duyn(i,j),duzn(i,j))
            kzzr(i,j)=dmzz(i,j,0,duxn(i,j),duyn(i,j),duzn(i,j))
            kxyr(i,j)=dmxy(i,j,0,duxn(i,j),duyn(i,j),duzn(i,j))
        END DO
    END DO
!
```

```
    DO j=0, nymax-1
        DO i=nxmax+1, 2*nxmax-1
            kxxr(i,j)=dmxx(i-2*nxmax,j,0,
    &               duxn(i,j),duyn(i,j),duzn(i,j))
            kyyr(i,j)=dmyy(i-2*nxmax,j,0,
    &               duxn(i,j),duyn(i,j),duzn(i,j))
            kzzr(i,j)=dmzz(i-2*nxmax,j,0,
    &               duxn(i,j),duyn(i,j),duzn(i,j))
            kxyr(i,j)=dmxy(i-2*nxmax,j,0,
    &               duxn(i,j),duyn(i,j),duzn(i,j))
        END DO
    END DO
!
    DO j=nymax+1, 2*nymax-1
        DO i=0,nxmax-1
            kxxr(i,j)=dmxx(i,j-2*nymax,0,
    &               duxn(i,j),duyn(i,j),duzn(i,j))
            kyyr(i,j)=dmyy(i,j-2*nymax,0,
    &               duxn(i,j),duyn(i,j),duzn(i,j))
            kzzr(i,j)=dmzz(i,j-2*nymax,0,
    &               duxn(i,j),duyn(i,j),duzn(i,j))
            kxyr(i,j)=dmxy(i,j-2*nymax,0,
    &               duxn(i,j),duyn(i,j),duzn(i,j))
        END DO
    END DO
!
    DO j=nymax+1, 2*nymax-1
        DO i=nxmax+1, 2*nxmax-1
            kxxr(i,j)=dmxx(i-2*nxmax,j-2*nymax,0,
    &               duxn(i,j),duyn(i,j),duzn(i,j))
            kyyr(i,j)=dmyy(i-2*nxmax,j-2*nymax,0,
    &               duxn(i,j),duyn(i,j),duzn(i,j))
            kzzr(i,j)=dmzz(i-2*nxmax,j-2*nymax,0,
    &               duxn(i,j),duyn(i,j),duzn(i,j))
            kxyr(i,j)=dmxy(i-2*nxmax,j-2*nymax,0,
    &               duxn(i,j),duyn(i,j),duzn(i,j))
        END DO
    END DO
! Middle planes (lines) are padded with 0 to keep 2^n size
    DO i=0, 2*nxmax-1
        kxxr(i,nymax)=0
```

```
        kyyr(i,nymax)=0
        kzzr(i,nymax)=0
        kxyr(i,nymax)=0
    END DO
!
    DO j=0, 2*nymax-1
        kxxr(nxmax,j)=0
        kyyr(nxmax,j)=0
        kzzr(nxmax,j)=0
        kxyr(nxmax,j)=0
    END DO
! Transform normization
    DO j=0, ny2-1
        DO i=0, nx2-1
            kxxr(i,j)=kxxr(i,j)/Nnorm
            kyyr(i,j)=kyyr(i,j)/Nnorm
            kzzr(i,j)=kzzr(i,j)/Nnorm
            kxyr(i,j)=kxyr(i,j)/Nnorm
        END DO
    END DO
! FFTW initialization
    CALL DZFFT2D(0,nx2,ny2,1.d0,kxxr,nx2+2,kxx,nxmax+1,table1,work,0)
    CALL ZDFFT2D(0,nx2,ny2,1.d0,kxx,nx2+2,kxxr,nxmax+1,table2,work,0)
! Conversion to Fourier space
    CALL DZFFT2D(1,nx2,ny2,1.d0,kxxr,nx2+2,kxx,nxmax+1,table1,work,0)
    CALL DZFFT2D(1,nx2,ny2,1.d0,kyyr,nx2+2,kyy,nxmax+1,table1,work,0)
    CALL DZFFT2D(1,nx2,ny2,1.d0,kzzr,nx2+2,kzz,nxmax+1,table1,work,0)
    CALL DZFFT2D(1,nx2,ny2,1.d0,kxyr,nx2+2,kxy,nxmax+1,table1,work,0)
!
    END SUBROUTINE initdem
```

5. Contents in the file “*ode2d.f*”:

```
    SUBROUTINE ODE2D(y,xnow,xmax,nstep,noutput)
    USE globals
    IMPLICIT NONE
!
    DOUBLE PRECISION :: y(1:2,1:nxmax,1:nymax)
    DOUBLE PRECISION :: xnow, xmax
```

```
    INTEGER :: nstep, noutput
!
    DOUBLE PRECISION, SAVE :: x, dx
    DOUBLE PRECISION, SAVE :: t, Mx, My, Mz
    LOGICAL, SAVE :: save_image
! Timing variables
    INTEGER :: clock_start, clock_end, clock_rate
    DOUBLE PRECISION :: elapsed_time
! External routines
    DOUBLE PRECISION, EXTERNAL :: xmag, ymag, zmag
    EXTERNAL :: RKLIU, derivs, saveimage
!
    clock_start=0
    clock_end=0
    CALL SYSTEM_CLOCK(COUNT_RATE=clock_rate) ! Find the timing rate
    x=xnow
    dx=0.5*dtsav    ! Initial step size; factor 0.5 - just for safety
!
    DO WHILE (x<=xmax)
!
        t=x*treduced*1.0D+12    ! t in picosecond
        Mx=xmag(y)/(nxmax*nymax)
        My=ymag(y)/(nxmax*nymax)
        Mz=zmag(y)/(nxmax*nymax)
!
        elapsed_time=(clock_end-clock_start)/clock_rate

    OPEN(8,FILE='average.dat',ACCESS='sequential',POSITION='append')
        WRITE(*,100) nstep, save_image, t, Mx, My, Mz
        WRITE(8,100) nstep, save_image, t, Mx, My, Mz
 100FORMAT(I8,X,L,X,F12.4,3(X,F12.8))
        CLOSE(8)
        nstep=nstep+1
!
        save_image=.FALSE.
!
        CALL SYSTEM_CLOCK(COUNT=clock_start)   ! Start timing
        CALL RK2D(derivs,x,dx,y,noutput,save_image)
        CALL SYSTEM_CLOCK(COUNT=clock_end) ! Stop timing
!
        IF (save_image .EQV. .TRUE.) THEN
```

```
            CALL saveimage(noutput,y)
            noutput=noutput+1
        END IF
!
        xnow=x
!
        OPEN(11,FILE=datapool,FORM='unformatted')
        WRITE(11) xnow, y, mask, nstep, noutput
        CLOSE(11)
!
    END DO
!
    END SUBROUTINE ODE2D
```

6. Contents in the file "*rk2d.f*":

```
!**********************************************************************
**
! Cash Karp's embedded Runge-Kutta ODE solver with adaptive stepsize
! control, designedly for 'sim2d' program, replacing previous 'UT'.
!**********************************************************************
**
    SUBROUTINE RK2D(derivs,x,dx,y,noutput,save_image)
    USE globals
    IMPLICIT NONE
!
    EXTERNAL :: derivs
    DOUBLE PRECISION :: x, dx, y(1:Nmax)
    INTEGER :: noutput
    LOGICAL :: save_image
!
    DOUBLE PRECISION :: dy(1:Nmax)
    INTEGER :: i, ntry
    LOGICAL :: retry
    DOUBLE PRECISION :: err0, errmax
    DOUBLE PRECISION, SAVE :: ytemp(1:Nmax), yembd(1:Nmax)
    DOUBLE PRECISION, SAVE :: k1(1:Nmax), k2(1:Nmax), k3(1:Nmax)
    DOUBLE PRECISION, SAVE :: k4(1:Nmax), k5(1:Nmax), k6(1:Nmax)
!
```

```
    DOUBLE PRECISION, PARAMETER :: precision=0.00001, safety=0.9
    DOUBLE PRECISION, PARAMETER :: exp1=0.2, exp2=0.25
    DOUBLE PRECISION, PARAMETER :: a2=0.2, a3=0.3, a4=0.6, a5=1.0,
a6=0.875
    DOUBLE PRECISION, PARAMETER :: b21=0.2, b31=0.075, b32=0.225
    DOUBLE PRECISION, PARAMETER :: b41=0.3, b42=-0.9, b43=1.2
    DOUBLE PRECISION, PARAMETER :: b51=-11./54., b52=2.5
    DOUBLE PRECISION, PARAMETER :: b53=-70./27., b54=35./27.
    DOUBLE PRECISION, PARAMETER :: b61=1631./55296., b62=175./512.
    DOUBLE PRECISION, PARAMETER :: b63=575./13824.
    DOUBLE PRECISION, PARAMETER :: b64=44275./110592., b65=253./4096.
    DOUBLE PRECISION, PARAMETER :: c1=37./378., c2=0, c3=250./621.
    DOUBLE PRECISION, PARAMETER :: c4=125./594., c5=0, c6=512./1771.
    DOUBLE PRECISION, PARAMETER :: d1=2825./27648., d2=0
    DOUBLE PRECISION, PARAMETER :: d3=18575./48384., d4=13525./55296.
    DOUBLE PRECISION, PARAMETER :: d5=277./14336., d6=.25
! Initial values for self-adapting procedures
    ntry=0
    retry=.TRUE.
    errmax=0
! The precision for theta and phi
    err0=precision*Pi
!
    DO WHILE ((retry .EQV. .TRUE.) .AND. (ntry<=5))
!
        IF (x+dx>tmin+(noutput+1)*dtsav .AND. x>0) THEN
            save_image=.TRUE.
        END IF
!
        CALL derivs(x,dx,y,dy,mask)
        k1=dx*dy
        ytemp=y+b21*k1
!
        CALL derivs(x+a2*dx,dx,ytemp,dy,mask)
        k2=dx*dy
        ytemp=y+b31*k1+b32*k2
!
        CALL derivs(x+a3*dx,dx,ytemp,dy,mask)
        k3=dx*dy
        ytemp=y+b41*k1+b42*k2+b43*k3
!
```

```
        CALL derivs(x+a4*dx,dx,ytemp,dy,mask)
        k4=dx*dy
        ytemp=y+b51*k1+b52*k2+b53*k3+b54*k4
!
        CALL derivs(x+a5*dx,dx,ytemp,dy,mask)
        k5=dx*dy
        ytemp=y+b61*k1+b62*k2+b63*k3+b64*k4+b65*k5
!
        CALL derivs(x+a6*dx,dx,ytemp,dy,mask)
        DO i=1, Nmax
            k6(i)=dx*dy(i)
            ytemp(i)=y(i)+c1*k1(i)+c2*k2(i)+c3*k3(i)
    &               +c4*k4(i)+c5*k5(i)+c6*k6(i)
            yembd(i)=y(i)+d1*k1(i)+d2*k2(i)+d3*k3(i)
    &               +d4*k4(i)+d5*k5(i)+d6*k6(i)
! Compute the largest discrepancy of ytemp and yembd
            errmax=MAX(errmax,ABS(ytemp(i)-yembd(i)))
        END DO
!
        IF (errmax<=err0) THEN
            dx=dx*safety*(err0/errmax)**exp1
            retry=.FALSE.
        ELSE
            dx=dx*safety*(err0/errmax)**exp2
            retry=.TRUE.
            ntry=ntry+1
        END IF
!
    END DO
!
    x=x+dx
    y=ytemp
!
    END SUBROUTINE rk2d
```

7. Contents in the file “*deriv2d.f*”:

```
    SUBROUTINE derivs(t,dt,tp0,dtp)
    USE globals
```

```
    IMPLICIT NONE
!*********************************************************
! Procedure calculates theta- and phi- components
! of the effective magnetic field and finds
! respective time derivatives dtp(1,i,j,k) and dtp(2,i,j,k)
! Attention: for time-dependent field its components
! will have to be given as functions of t.
!*********************************************************
!
    DOUBLE PRECISION :: t, dt
    DOUBLE PRECISION :: tp0(1:2,1:nxmax,1:nymax)
    DOUBLE PRECISION :: dtp(1:2,1:nxmax,1:nymax)
!
    DOUBLE PRECISION :: noise(1:nxmax,1:nymax)
    DOUBLE PRECISION :: d2xn(1:nxmax,1:nymax)
    DOUBLE PRECISION :: d2yn(1:nxmax,1:nymax)
    DOUBLE PRECISION :: d2zn(1:nxmax,1:nymax)
!
    DOUBLE PRECISION :: hdem(1:2,0:nxmax-1,0:nymax-1)
    DOUBLE PRECISION :: tp(1:2,0:nxmax+1,0:nymax+1)    ! dummy array.
    INTEGER :: i, j
    DOUBLE PRECISION :: hth, hph   ! Overall effective field
    DOUBLE PRECISION :: xhgth, xhgph
!
    EXTERNAL :: hdemag
!
    CALL hdemag(t,dt,tp0,hdem) ! demagnetizing and external field
!
    tp=0
    tp(1:2,1:nxmax,1:nymax)=tp0(1:2,1:nxmax,1:nymax)
!
    CALL RANDOM_NUMBER(noise)
    DO i=1, nxmax
        DO j=1, nymax
            d2xn(i,j)=d2x*(1+0.01*(noise(i,j)-0.5))
            d2yn(i,j)=d2y*(1+0.01*(noise(i,j)-0.5))
            d2zn(i,j)=d2z*(1+0.01*(noise(i,j)-0.5))
        END DO
    END DO
! Calculation of the effective field
    DO j=1, nymax
```

```
        DO i=1, nxmax
! Exchange field + sum of external and demagnetizing fields
            xhgth=-((SIN(tp(1,i,j))*COS(tp(1,i+mask(-1,i,j),j))
    &               -COS(tp(1,i,j))*SIN(tp(1,i+mask(-1,i,j),j))
    &               *COS(tp(2,i,j)-tp(2,i+mask(-1,i,j),j)))
    &               +(SIN(tp(1,i,j))*COS(tp(1,i+mask(1,i,j),j))
    &               -COS(tp(1,i,j))*SIN(tp(1,i+mask(1,i,j),j))
    &               *COS(tp(2,i,j)-tp(2,i+mask(1,i,j),j))))*d2xn(i,j)
    &               -((SIN(tp(1,i,j))*COS(tp(1,i,j+mask(-2,i,j)))
    &               -COS(tp(1,i,j))*SIN(tp(1,i,j+mask(-2,i,j)))
    &               *COS(tp(2,i,j)-tp(2,i,j+mask(-2,i,j))))
    &               +(SIN(tp(1,i,j))*COS(tp(1,i,j+mask(2,i,j)))
    &               -COS(tp(1,i,j))*SIN(tp(1,i,j+mask(2,i,j)))
    &               *COS(tp(2,i,j)-tp(2,i,j+mask(2,i,j))))*d2yn(i,j)
!
            xhgph=-(SIN(tp(1,i+mask(-1,i,j),j))
    &               *SIN(tp(2,i,j)-tp(2,i+mask(-1,i,j),j))
    &               +SIN(tp(1,i+mask(1,i,j),j))
    &               *SIN(tp(2,i,j)-tp(2,i+mask(1,i,j),j)))*d2xn(i,j)
    &               -(SIN(tp(1,i,j+mask(-2,i,j)))
    &               *SIN(tp(2,i,j)-tp(2,i,j+mask(-2,i,j)))
    &               +SIN(tp(1,i,j+mask(2,i,j)))
    &               *SIN(tp(2,i,j)-tp(2,i,j+mask(2,i,j))))*d2yn(i,j)
!
            hth=xhgth+hdem(1,i-1,j-1)
            hph=xhgph+hdem(2,i-1,j-1)
!
            dtp(1,i,j)=(alpha*hth+hph)*mask(0,i,j)
            dtp(2,i,j)=(alpha*hph-hth)/SIN(tp(1,i,j))*mask(0,i,j)
!
            IF (i==isample .AND. j==jsample) THEN
                OPEN(9,FILE='xhgtot.dat',ACCESS='sequential',
    &                   POSITION='append')
                WRITE(9,900) t, hdem(1,i-1,j-1), hdem(2,i-1,j-1),
    &                   xhgth,xhgph, hth, hph, dtp(1,i,j), dtp(2,i,j)
 900FORMAT(9(X,F12.6))
                CLOSE(9)
            END IF
!
        END DO
    END DO
```

```
!
    END SUBROUTINE derivs
```

8. Contents in the file "*demag2d.f*":

```
    SUBROUTINE hdemag(t,dt,tp,hdem)
    USE globals
    IMPLICIT NONE
!
    DOUBLE PRECISION :: t, dt
    DOUBLE PRECISION :: tp(1:2,1:nxmax,1:nymax)
    DOUBLE PRECISION :: hdem(1:2,0:nxmax-1,0:nymax-1)
!************************************************************
! Calculates the demagnetizing field.
!************************************************************
!
    EXTERNAL :: hext
    DOUBLE PRECISION, EXTERNAL :: hanis, gasdev
!
    INTEGER :: i, j
    DOUBLE PRECISION :: eps
    DOUBLE PRECISION :: hextx, hexty, hextz    ! External field
    DOUBLE PRECISION :: rand_Gauss(1:Nrand)
    DOUBLE PRECISION :: rseed(4)
    INTEGER :: iseed(4)
    DOUBLE PRECISION :: idum(2)
! Cartesian components of Mx, My, Mz
    DOUBLE PRECISION :: xm(0:nx2+1,0:ny2-1)
    DOUBLE PRECISION :: ym(0:nx2+1,0:ny2-1)
    DOUBLE PRECISION :: zm(0:nx2+1,0:ny2-1)
! Fourier images of Mx,My,Mz, as calculated by real->complex transform
    DOUBLE COMPLEX :: xmc(nmaxc)
    DOUBLE COMPLEX :: ymc(nmaxc)
    DOUBLE COMPLEX :: zmc(nmaxc)
! Cartesian components of Hd
    DOUBLE PRECISION :: xm1(0:nx2+1,0:ny2-1)
    DOUBLE PRECISION :: ym1(0:nx2+1,0:ny2-1)
    DOUBLE PRECISION :: zm1(0:nx2+1,0:ny2-1)
! Fourier images of Hx,Hy,Hz, as calculated by real->complex transform
```

```
    DOUBLE COMPLEX :: xm1c(nmaxc)
    DOUBLE COMPLEX :: ym1c(nmaxc)
    DOUBLE COMPLEX :: zm1c(nmaxc)
! Timing variables
    INTEGER :: clock_start, clock_end, clock_rate
    DOUBLE PRECISION :: elapsed_time
!
    eps=2.0*alpha*Kbolts*temp/((un*1.d-7)**3*Ms*gamma*dux*duy*duz)
! The first octant (quadrant) of Cartesian M components is
! filled with the respective values. The rest are zero-padded.
    xm=0
    ym=0
    zm=0
    DO j=1, nymax
        DO i=1, nxmax
            xm(i-1,j-1)=SIN(tp(1,i,j))*COS(tp(2,i,j))*mask(0,i,j)
            ym(i-1,j-1)=SIN(tp(1,i,j))*SIN(tp(2,i,j))*mask(0,i,j)
            zm(i-1,j-1)=COS(tp(1,i,j))*mask(0,i,j)
        END DO
    END DO
! Timing routine
    CALL SYSTEM_CLOCK(COUNT_RATE=clock_rate) ! Find the timing rate
    CALL SYSTEM_CLOCK(COUNT=clock_start)   ! Start timing
! Conversion of M components to Fourier space
    CALL DZFFT2D(1,nx2,ny2,1.d0,xm,nx2+2,xmc,nxmax+1,table1,work,0)
    CALL DZFFT2D(1,nx2,ny2,1.d0,ym,nx2+2,ymc,nxmax+1,table1,work,0)
    CALL DZFFT2D(1,nx2,ny2,1.d0,zm,nx2+2,zmc,nxmax+1,table1,work,0)
! Fourier images of M are multipled by demagnetizing
! coefficients transform - this is a convolution in
! the Fourier space, that produces image of Hd field.
    DO i=1, nmaxc
        xm1c(i)=kxx(i)*xmc(i)+kxy(i)*ymc(i)
        ym1c(i)=kxy(i)*xmc(i)+kyy(i)*ymc(i)
        zm1c(i)=kzz(i)*zmc(i)
    END DO
! Conversion of Hd to real space
    CALL ZDFFT2D(-1,nx2,ny2,1.d0,xm1c,nxmax+1,xm1,nx2+2,table2,work,0)
    CALL ZDFFT2D(-1,nx2,ny2,1.d0,ym1c,nxmax+1,ym1,nx2+2,table2,work,0)
    CALL ZDFFT2D(-1,nx2,ny2,1.d0,zm1c,nxmax+1,zm1,nx2+2,table2,work,0)
!
    CALL SYSTEM_CLOCK(COUNT=clock_end) ! Stop timing
```

```
    elapsed_time=(clock_end-clock_start)/clock_rate
    OPEN(4,FILE='FFTtime.dat',ACCESS='sequential',POSITION='append')
    WRITE(4,400) t, elapsed_time
 400FORMAT(2(X,F12.6))
    CLOSE(4)
! Prepare Guassian-type random numbers for thermo-term
    CALL RANDOM_NUMBER(rseed)
    iseed=INT(rseed*100+100)
    iseed(4)=iseed(4)*2-1   ! the last random seed must be ODD.
    DO i=1, Nrand
         rand_Gauss(i)=gasdev(idum,iseed)
    END DO
! Calculation of spherical components of demagnetizing and external
fields.
    DO j=0, nymax-1
        DO i=0, nxmax-1
! External field
        CALL hext(t,i+1,j+1,hextx,hexty,hextz)
!
        IF (i==isample-1 .AND. j==jsample-1) THEN

    OPEN(7,FILE='demext.dat',ACCESS='sequential',POSITION='append')
        WRITE(7,700) t, xm1(i,j), ym1(i,j), zm1(i,j),
    &          hextx, hexty, hextz, hanis(i+1,j+1,tp),
    &          SQRT(eps*dt*treduced)*rand_Gauss(nxmax*j+3*i+1)/Ms
 700FORMAT(9(X,F12.6))
        CLOSE(7)
        END IF
!
        xm1(i,j)=xm1(i,j)+hextx+SQRT(eps*dt*treduced)
    &          *rand_Gauss(nxmax*j+3*i+1)/Ms
        ym1(i,j)=ym1(i,j)+hexty
    &          +SQRT(eps*dt*treduced)*rand_Gauss(nxmax*j+3*i+2)/Ms
        zm1(i,j)=zm1(i,j)+hextz+SQRT(eps*dt*treduced)
    &          *rand_Gauss(nxmax*j+3*i+3)/Ms
! Conversion to spheric coordinates
        hdem(1,i,j)=xm1(i,j)*COS(tp(1,i+1,j+1))*COS(tp(2,i+1,j+1))
    &          +ym1(i,j)*COS(tp(1,i+1,j+1))*SIN(tp(2,i+1,j+1))
    &          -zm1(i,j)*SIN(tp(1,i+1,j+1))
        hdem(2,i,j)=-xm1(i,j)*SIN(tp(2,i+1,j+1))
    &          +ym1(i,j)*COS(tp(2,i+1,j+1))
```

```
!
        END DO
    END DO
!
    END SUBROUTINE hdemag
! ------------------------------------------------------------------
! ><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
! ------------------------------------------------------------------
! Anisotropy component of effective field
    DOUBLE PRECISION FUNCTION hanis(i,j,tp)
    USE globals
    IMPLICIT NONE
    INTEGER :: i, j
    DOUBLE PRECISION :: tp(1:2,1:nxmax,1:nymax)
!
! Anisotropy constant, y-direction only
    DOUBLE PRECISION, PARAMETER :: Hky=0.01
!
    hanis = Hky*(SIN(tp(1,i,j))*SIN(tp(2,i,j))*mask(0,i,j))**2
    &       *SIN(tp(1,i,j))*SIN(tp(2,i,j))
    &       /ABS(SIN(tp(1,i,j))*SIN(tp(2,i,j)))
!
    END FUNCTION hanis
! ------------------------------------------------------------------
! ><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
! ------------------------------------------------------------------
    FUNCTION gasdev(idum,iseed)
    IMPLICIT NONE
    DOUBLE PRECISION :: gasdev
    DOUBLE PRECISION :: idum(2)
    INTEGER :: iseed(4)
!
    INTEGER, SAVE :: iset
    DOUBLE PRECISION, SAVE :: gset
    DOUBLE PRECISION ::  v1, v2, rsq, fac
    DATA iset/0/
!
    IF (iset==0) THEN
 1      CALL dlarnv(1,iseed,2,idum)
        v1=2*idum(1)-1
        v2=2*idum(2)-1
```

```
        rsq=v1**2+v2**2
        IF (rsq>=1 .or. rsq==0) GOTO 1
        fac=SQRT(-2.*LOG(rsq)/rsq)
        gset=v1*fac
        gasdev=v2*fac
        iset=1
    ELSE
        gasdev=gset
        iset=0
    END IF
!
    END FUNCTION gasdev
```

9. Contents in the file "*functions.f*":

```
!*************************************************************
! Demagnetizing tensor components. i,j,k define relative position
! of two cells, while dx,dy,dz define their size.
! See: Y.Nakatani, Y.Uesaka, N.Hayashi, "Direct Solution of the
! Landau-Lifshitz-Gilbert equation for micromagnetics",
! Jap.J.Appl.Phys., 28 (1989) 2485-507.
!*************************************************************
    FUNCTION dmxx(i,j,k,dx,dy,dz)
    IMPLICIT NONE
!
    DOUBLE PRECISION :: dmxx
    INTEGER :: i, j, k
    DOUBLE PRECISION :: dx, dy, dz
!
    dmxx=ATAN((dy*dz*(-0.5+j)*(-0.5+k))
    &   /(dx*(-0.5+i)*SQRT(dx**2*(-0.5+i)**2+dy**2*(-0.5+j)**2
    &   +dz**2*(-0.5+k)**2)))-ATAN((dy*dz*(-0.5+j)*(-0.5+k))
    &   /(dx*(0.5+i)*SQRT(dx**2*(0.5+i)**2+dy**2*(-0.5+j)**2
    &   +dz**2*(-0.5+k)**2)))-ATAN((dy*dz*(0.5+j)*(-0.5+k))
    &   /(dx*(-0.5+i)*SQRT(dx**2*(-0.5+i)**2+dy**2*(0.5+j)**2
    &   +dz**2*(-0.5+k)**2)))+ATAN((dy*dz*(0.5+j)*(-0.5+k))
    &   /(dx*(0.5+i)*SQRT(dx**2*(0.5+i)**2+dy**2*(0.5+j)**2
    &   +dz**2*(-0.5+k)**2)))-ATAN((dy*dz*(-0.5+j)*(0.5+k))
    &   /(dx*(-0.5+i)*SQRT(dx**2*(-0.5+i)**2+dy**2*(-0.5+j)**2
```

```
    &   +dz**2*(0.5+k)**2)))+ATAN((dy*dz*(-0.5+j)*(0.5+k))
    &   /(dx*(0.5+i)*SQRT(dx**2*(0.5+i)**2+dy**2*(-0.5+j)**2
    &   +dz**2*(0.5+k)**2)))+ATAN((dy*dz*(0.5+j)*(0.5+k))
    &   /(dx*(-0.5+i)*SQRT(dx**2*(-0.5+i)**2+dy**2*(0.5+j)**2
    &   +dz**2*(0.5+k)**2)))-ATAN((dy*dz*(0.5+j)*(0.5+k))
    &   /(dx*(0.5+i)*SQRT(dx**2*(0.5+i)**2+dy**2*(0.5+j)**2
    &   +dz**2*(0.5+k)**2)))
!
    END FUNCTION dmxx
! -------------------------------------------------------
! ><><><><><><><><><><><><><><><><><><><><><><><><><><><><
! -------------------------------------------------------
    FUNCTION dmyy(i,j,k,dx,dy,dz)
    IMPLICIT NONE
!
    DOUBLE PRECISION :: dmyy
    INTEGER :: i, j, k
    DOUBLE PRECISION :: dx, dy, dz
!
    dmyy=ATAN((dx*dz*(-0.5+i)*(-0.5+k))
    &   /(dy*(-0.5+j)*SQRT(dx**2*(-0.5+i)**2+dy**2*(-0.5+j)**2
    &   +dz**2*(-0.5+k)**2)))-ATAN((dx*dz*(0.5+i)*(-0.5+k))
    &   /(dy*(-0.5+j)*SQRT(dx**2*(0.5+i)**2+dy**2*(-0.5+j)**2
    &   +dz**2*(-0.5+k)**2)))-ATAN((dx*dz*(-0.5+i)*(-0.5+k))
    &   /(dy*(0.5+j)*SQRT(dx**2*(-0.5+i)**2+dy**2*(0.5+j)**2
    &   +dz**2*(-0.5+k)**2)))+ATAN((dx*dz*(0.5+i)*(-0.5+k))
    &   /(dy*(0.5+j)*SQRT(dx**2*(0.5+i)**2+dy**2*(0.5+j)**2
    &   +dz**2*(-0.5+k)**2)))-ATAN((dx*dz*(-0.5+i)*(0.5+k))
    &   /(dy*(-0.5+j)*SQRT(dx**2*(-0.5+i)**2+dy**2*(-0.5+j)**2
    &   +dz**2*(0.5+k)**2)))+ATAN((dx*dz*(0.5+i)*(0.5+k))
    &   /(dy*(-0.5+j)*SQRT(dx**2*(0.5+i)**2+dy**2*(-0.5+j)**2
    &   +dz**2*(0.5+k)**2)))+ATAN((dx*dz*(-0.5+i)*(0.5+k))
    &   /(dy*(0.5+j)*SQRT(dx**2*(-0.5+i)**2+dy**2*(0.5+j)**2
    &   +dz**2*(0.5+k)**2)))-ATAN((dx*dz*(0.5+i)*(0.5+k))
    &   /(dy*(0.5+j)*SQRT(dx**2*(0.5+i)**2+dy**2*(0.5+j)**2
    &   +dz**2*(0.5+k)**2)))
!
    END FUNCTION dmyy
! -------------------------------------------------------
! ><><><><><><><><><><><><><><><><><><><><><><><><><><><><
! -------------------------------------------------------
```

```
    FUNCTION dmzz(i,j,k,dx,dy,dz)
    IMPLICIT NONE
!
    DOUBLE PRECISION :: dmzz
    INTEGER :: i, j, k
    DOUBLE PRECISION :: dx, dy, dz
!
    dmzz=ATAN((dx*dy*(-0.5+i)*(-0.5+j))
    &   /(dz*SQRT(dx**2*(-0.5+i)**2+dy**2*(-0.5+j)**2
    &   +dz**2*(-0.5+k)**2)*(-0.5+k)))
    &   -ATAN((dx*dy*(0.5+i)*(-0.5+j))
    &   /(dz*SQRT(dx**2*(0.5+i)**2+dy**2*(-0.5+j)**2
    &   +dz**2*(-0.5+k)**2)*(-0.5+k)))
    &   -ATAN((dx*dy*(-0.5+i)*(0.5+j))
    &   /(dz*SQRT(dx**2*(-0.5+i)**2+dy**2*(0.5+j)**2
    &   +dz**2*(-0.5+k)**2)*(-0.5+k)))
    &   +ATAN((dx*dy*(0.5+i)*(0.5+j))
    &   /(dz*SQRT(dx**2*(0.5+i)**2+dy**2*(0.5+j)**2
    &   +dz**2*(-0.5+k)**2)*(-0.5+k)))
    &   -ATAN((dx*dy*(-0.5+i)*(-0.5+j))
    &   /(dz*(0.5+k)*SQRT(dx**2*(-0.5+i)**2+dy**2*(-0.5+j)**2
    &   +dz**2*(0.5+k)**2)))+ATAN((dx*dy*(0.5+i)*(-0.5+j))
    &   /(dz*(0.5+k)*SQRT(dx**2*(0.5+i)**2+dy**2*(-0.5+j)**2
    &   +dz**2*(0.5+k)**2)))+ATAN((dx*dy*(-0.5+i)*(0.5+j))
    &   /(dz*(0.5+k)*SQRT(dx**2*(-0.5+i)**2+dy**2*(0.5+j)**2
    &   +dz**2*(0.5+k)**2)))-ATAN((dx*dy*(0.5+i)*(0.5+j))
    &   /(dz*(0.5+k)*SQRT(dx**2*(0.5+i)**2+dy**2*(0.5+j)**2
    &   +dz**2*(0.5+k)**2)))
!
    END FUNCTION dmzz
! -------------------------------------------------------------
! ><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
! -------------------------------------------------------------
    FUNCTION dmxy(i,j,k,dx,dy,dz)
    IMPLICIT NONE
!
    DOUBLE PRECISION :: dmxy
    INTEGER :: i, j, k
    DOUBLE PRECISION :: dx, dy, dz
!
    dmxy=-LOG(ABS(SQRT(dx**2*(-0.5+i)**2+dy**2*(-0.5+j)**2
```

```
    &   +dz**2*(-0.5+k)**2)+dz*(-0.5+k)))
    &   +LOG(ABS(SQRT(dx**2*(0.5+i)**2+dy**2*(-0.5+j)**2
    &   +dz**2*(-0.5+k)**2)+dz*(-0.5+k)))
    &   +LOG(ABS(SQRT(dx**2*(-0.5+i)**2+dy**2*(0.5+j)**2
    &   +dz**2*(-0.5+k)**2)+dz*(-0.5+k)))
    &   -LOG(ABS(SQRT(dx**2*(0.5+i)**2+dy**2*(0.5+j)**2
    &   +dz**2*(-0.5+k)**2)+dz*(-0.5+k)))+LOG(ABS(dz*(0.5+k)
    &   +SQRT(dx**2*(-0.5+i)**2+dy**2*(-0.5+j)**2
    &   +dz**2*(0.5+k)**2)))
    &   -LOG(ABS(dz*(0.5+k)+SQRT(dx**2*(0.5+i)**2
    &   +dy**2*(-0.5+j)**2
    &   +dz**2*(0.5+k)**2)))-LOG(ABS(dz*(0.5+k)
    &   +SQRT(dx**2*(-0.5+i)**2+dy**2*(0.5+j)**2
    &   +dz**2*(0.5+k)**2)))
    &   +LOG(ABS(dz*(0.5+k)+SQRT(dx**2*(0.5+i)**2
    &   +dy**2*(0.5+j)**2
    &   +dz**2*(0.5 + k)**2)))
!
    END FUNCTION dmxy
!
!*****************************************************
! Functions that calculate x/y/z-component of the
! sample's magnetic moment in arbitrary units
!*****************************************************
!
    DOUBLE PRECISION FUNCTION xmag(tp)
    USE globals
    IMPLICIT NONE
    DOUBLE PRECISION :: tp(1:2,1:nxmax,1:nymax)
!
    INTEGER i, j
!
    xmag=0
    DO j=1, nymax
        DO i=1, nxmax
            xmag=xmag+SIN(tp(1,i,j))*COS(tp(2,i,j))*mask(0,i,j)
        END DO
    END DO
!
    END FUNCTION xmag
! -----------------------------------------------------
```

```
! ><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
! ------------------------------------------------------
    DOUBLE PRECISION FUNCTION ymag(tp)
    USE globals
    IMPLICIT NONE
    DOUBLE PRECISION :: tp(1:2,1:nxmax,1:nymax)
!
    INTEGER i, j
!
    ymag=0
    DO j=1, nymax
        DO i=1, nxmax
            ymag=ymag+SIN(tp(1,i,j))*SIN(tp(2,i,j))*mask(0,i,j)
        END DO
    END DO
!
    END FUNCTION ymag
! ------------------------------------------------------
! ><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><
! ------------------------------------------------------
    DOUBLE PRECISION FUNCTION zmag(tp)
    USE globals
    IMPLICIT NONE
    DOUBLE PRECISION :: tp(1:2,1:nxmax,1:nymax)
!
    INTEGER i, j
!
    zmag=0
    DO i=1, nxmax
        DO j=1, nymax
            zmag=zmag+COS(tp(1,i,j))*mask(0,i,j)
        END DO
    END DO
!
    END FUNCTION zmag
```

10. Contents in the file “*saveimage.f*”:

```
    SUBROUTINE saveimage(noutput,y)
    USE globals
    IMPLICIT NONE
!
    INTEGER :: noutput
    DOUBLE PRECISION :: y(1:2,1:nxmax,1:nymax)
!*********************************************************
! Procedure saves a current distribution of Mx, My and Mz
! components so that Matlab's 'imshow' command can apply.
!*********************************************************
!
    INTEGER :: i, j
    INTEGER, PARAMETER :: Ndigit=3
    DOUBLE PRECISION, SAVE :: mx(1:nxmax,1:nymax)
    DOUBLE PRECISION, SAVE :: my(1:nxmax,1:nymax)
    DOUBLE PRECISION, SAVE :: mz(1:nxmax,1:nymax)
! Note: number of image files must be less than 999!
    CHARACTER :: digit(1:Ndigit)
! Digit pool
    CHARACTER :: a(0:10)
! int is equal to noutput at the beginning
    INTEGER :: int, index
!
    DO i=1, nxmax
        DO j=1, nymax
            mx(i,j)=SIN(y(1,i,j))*COS(y(2,i,j))*mask(0,i,j)
            my(i,j)=SIN(y(1,i,j))*SIN(y(2,i,j))*mask(0,i,j)
            mz(i,j)=COS(y(1,i,j))*mask(0,i,j)
        END DO
    END DO
!
    a(0)='0'; a(1)='1'; a(2)='2'; a(3)='3'; a(4)='4';
    a(5)='5'; a(6)='6'; a(7)='7'; a(8)='8'; a(9)='9';
!
    digit='0'
    int=noutput
!
    DO i=1, Ndigit
        index=MOD(int,10)
        digit(4-i)=a(index)
        int=FLOOR(int/10.0)
```

```
    END DO
! Making image file's name
    IF (noutput<10) THEN
        OPEN(1,FILE='xyz.'//digit(3),
    &           FORM='formatted',RECL=6144,STATUS='unknown')
    ELSE IF (noutput<100) THEN
        OPEN(1,FILE='xyz.'//digit(2)//digit(3),
    &           FORM='formatted',RECL=6144,STATUS='unknown')
    ELSE IF (noutput<1000) THEN
        OPEN(1,FILE='xyz.'//digit(1)//digit(2)//digit(3),
    &           FORM='formatted',RECL=6144,STATUS='unknown')
    END IF
! Saving data
    DO i=nxmax, 1, -1
        WRITE(1,100)(mx(i,j), j=1, nymax)
    END DO
!
    DO i=nxmax, 1, -1
        WRITE(1,100)(my(i,j), j=1, nymax)
    END DO
!
    DO i=nxmax, 1, -1
        WRITE(1,100)(mz(i,j), j=1, nymax)
    END DO
!
    CLOSE(1)
!
100 FORMAT(512(F12.8))
!
    END SUBROUTINE saveimage
```

Appendix III

Visualization of simulation results

(Matlab programs)

Below are some Matlab programs I wrote to perform visualization tasks. Brief introduction and direction are included in their comment scripts.

11. Contents in the file "*RGBshow.m*":

```
% This function reads a 2D array from the file "file",
% and display it by RGB color scheme. The purpose is to
% display spatial profiles of normalized magnetization,
% thus the elements of the array must be within [-1,+1].
        function output=RGBshow(file)
% Define colors
        colormap= ...
            [0,0,1;0,0.2,1;0,0.4,1;0,0.6,1;0,0.8,1;0,1,1;0,1,0.8; ...
            0,1,0.6;0,1,0.4;0,1,0.2;0,1,0;0.2,1,0;0.4,1,0;0.6,1,0; ...
            0.8,1,0;1,1,0;1,0.8,0;1,0.6,0;1,0.4,0;1,0.2,0;1,0,0];
% Load the image file
        imagedata=load(file);
        imagesize=size(imagedata);
% Customize the area to be displayed
        imin=1;
        imax=imagesize(1);
        jmin=1;
        jmax=imagesize(2);
% Define the color array to be displayed
        imbin=imagedata(imin:imax,jmin:jmax);
        imbmp=zeros(imax-imin+1,jmax-jmin+1,3);
% Fill the color array
        for i=1:imagesize(1)
```

```
            for j=1:imagesize(2)
%
                index=floor(10*imbin(i,j)+10)+1;
                if index<=1
                    index=1;
                end
                if index>=21
                    index=21;
                end
%
                imbmp(i,j,1)=colormap(index,1);
                imbmp(i,j,2)=colormap(index,2);
                imbmp(i,j,3)=colormap(index,3);
%
            end
        end
% Display the color array
        imshow(imbmp);
% Output
        output=imbmp;
        fclose('all');
%
```

12. Contents in the file "*xy2phi.m*":

```
% This function read a 2D array from the file "file",
% and convert the 3D magnetization components into an
% array whose elements represent for the in-plane "phi"
% angles. It is used by the function "RGBshow_angles".
    function output=xy2phi(file)
%
    xyz=load(file);
    dim=size(xyz);
    sizex=dim(1)/3;
    sizey=dim(2);
    phi=zeros(sizex,sizey);
%
    x=xyz(1:sizex,1:sizey);
    y=xyz((sizex+1):(2*sizex),1:sizey);
```

```
%
    for i=1:sizex
        for j=1:sizey
%
            mx=x(i,j);
            my=y(i,j);
            if mx==0
                mx=0.00000001;
            end
            angle=atan(abs(my/mx));
%
            if mx<0 & my>0
                angle=pi-angle;
            elseif mx<0 & my<0
                angle=pi+angle;
            elseif mx>0 & my<0
                angle=2*pi-angle;
            else
                angle=angle;
            end
%
            phi(i,j)=angle*180/pi;
%
        end
    end
%
    output=phi;
    fclose('all');%
%
```

13. Contents in the file “*RGBshow_angles.m*”:

```
% This function does the similar work with “RGBshow”,
% except that the in-plane angles “phi” are displayed
% and the elements in the array “imagedata” must be
% within [0,2*Pi].
    function output=RGBshow_angles(imagedata)
% The color map
    colormap=[255,255,  0; 255,226,  0; 255,197,  0; 255,159,  0; ...
        255,119,  0; 255, 76,  0; 255,  0,  0; 255, 57, 62; ...
```

```
        255, 45,103; 255, 35,136; 255, 26,164; 255, 18,193; ...
        255,  0,255; 230,  0,255; 201,  0,255; 169,  0,255; ...
        145,  0,255; 104,  0,255;   0,  0,255;   0, 36,255; ...
        0, 89,255;   0,130,255;   0,174,255;   0,218,255; ...
        0,255,255;   0,255,195;   0,255,176;   0,255,143; ...
        0,255,115;   0,255, 86;   0,255,  0;  98,255,  0; ...
        149,255,  0; 170,255,  0; 191,255,  0; 214,255,  0];
    colormap=colormap/256;
%
    imagesize=size(imagedata);
% Define the color image array
    image=zeros(imagesize(1),imagesize(2),3);
% Fill the color array
    for i=1:imagesize(1)
        for j=1:imagesize(2)
%
            index=floor(imagedata(i,j)/10)+1;
            if index<=1
                index=1;
            end
            if index>=36
                index=36;
            end
%
            image(i,j,1)=colormap(index,1);
            image(i,j,2)=colormap(index,2);
            image(i,j,3)=colormap(index,3);
%
        end
    end
% Display the color image and output
    imshow(image);
    output=image;
%
    fclose('all');
%
```

14. Contents in the file “*vectormap.m*”:

```
% This function displays the spatial profile of in-plane
% magnetization as arrows pointing to proper directions.
% ‘x’/’y’ refer to Mx/My components arrays;
% ‘nx’/’ny’ are the cell numbers along x/y-axis;
% ‘narrowx’/’narrowy’ are the arrow numbers along x/y-axis.
% ‘nx’ must be an integer times of ‘narrowx’, so is ‘ny’.
    function output=vectormap(x,y,nx,ny,narrowx,narrowy)
%
    vx=zeros(narrowx,narrowy);
    vy=zeros(narrowx,narrowy);
    xfactor=floor(nx/narrowx);
    yfactor=floor(ny/narrowy);
% Fill the vector array
    for i=1:narrowx
        for j=1:narrowy
            vx(i,j)=x((i-0.5)*xfactor,(j-0.5)*yfactor);
            vy(i,j)=y((i-0.5)*xfactor,(j-0.5)*yfactor);
        end
    end
% Display the vector map
    quiver(vx,vy);
    axis equal;
    axis off;
%
    output=xfactor;
%
```

Appendix IV

Standard RGB Color Wheel

This color wheel (available online) is used to calibrate different colors representing for orientations of the magnetization. For example, if the magnetization of a cell points upward, the cell is painted by yellow (color code 1); if it points downward, the cell is painted by blue (color code 19), and so on.