fidimag Documentation

Release 0.1

Weiwei Wang

February 26, 2016

Contents

Contents:

CHAPTER 1

How to install

1.1 Install prerequisites

On Ubuntu systems, we need to run the following commands:

```
# required to compile fidimag
apt-get install cython python-numpy
# required for tests and running it
apt-get install python-pytest python-pyvtk ipython python-matplotlib mayavi2
```

These are available in the script *bin/install-ubuntu-packages.sh* for convenience.

1.2 Install external libraries (FFTW and Sundials)

Run the install.sh by using

```
bash install.sh
```

in the fidimag/bin folder.

1.3 Install fidimag

In the fidimag folder type

```
make
```

to build fidimag.

1.4 Add the fidimag to python path

Add the following to your .bashrc file:

```
export PYTHONPATH=/path/to/fidimag:$PYTHONPATH
```

for example, suppose fidimag is in the directory of ~/work, then:

```
export PYTHONPATH=~/work/fidimag:$PYTHONPATH
```

1.5 Add the library path to LD_LIBRARY_PATH

By default, the libraries are installed in fidimag/local, so in order to run fidimag we need to include the libs path in LD_LIBRARY_PATH, so please add the following to your .bashrc file:

```
export LD_LIBRARY_PATH=/path/to/fidimag/local/lib:$LD_LIBRARY_PATH
```

for instance:

```
export LD_LIBRARY_PATH=~/work/fidimag/local/lib:$LD_LIBRARY_PATH
```

1.6 Adding OOMMF path to the system

For the tests that call OOMMF (in micro/tests), we need to tell the system where to find it. The way it is currently set up (in util/oommf.py), we need to find our `oommf.tcl` file, and add the path to it to `.bashrc` in this way:

```
export OOMMF_PATH=/opt/oommf/oommf-1.2a5bis
```

CHAPTER 2

Installing on Iridis

Few additional notes for installing on iridis.

2.1 Loading Modules

Need to load the hg and numpy modules. This can be done by

```
module load hg numpy
```

These modules will only be loaded for your current session on iridis. To have the modules automatically loaded at login, you can add them to the module initlist by

```
module initadd hg numpy
```

2.2 Installing PyTest and PyVTK

py.test and PyVTK need to be installed locally (at /home/$USER/.local/bin/) by

```
pip install --user pytest pyvtk
```

The following path needs to be added to the .bashrc file

```
export PATH=~/.local/bin:$PATH
```

2.3 Cloning Repository

Can only be done with ssh keys.

CHAPTER 3

Quick test

Go to the fidimag/tests folder and type "py.test", a similar result is expected

```
============================= test session starts ==============================
platform linux2 -- Python 2.7.5 -- pytest-2.4.2
collected 20 items

test_anis.py .
test_demag.py ..
test_dmi.py ..
test_energy.py .
test_exch.py ....
test_mesh.py ..
test_sim.py .......
test_stt.py .

========================== 20 passed in 2.31 seconds ===========================
```

3.1 How to set up a virtual machine via vagrant

- install vagrant on your host machine
- run:

```
vagrant init ubuntu/trusty64
```

  to set up a basic linux machine.

- run:

```
vagrant up
```

  to start the machine.

- ssh into the machine with X-forwarding:

```
vagrant ssh -- -X
```

Then within the virtual machine:

```
aptitude install git
git clone https://github.com/fangohr/fidimag.git
cd fidimag/bin
sudo sh install-ubuntu-packages.sh
```

```
sh install.sh
cd ..
make
```

To run the tests:

```
cd /home/vagrant/fidimag/tests
py.test
```

Notes:

- some tests will fail as OOMMF is not installed
- it seems that we need an active X server, on OS X, one may need to install XQuartz before the tests can pass (even 'import fidimag' failed without a working X server).

CHAPTER 4

Core equations

4.1 Landau-Lifshitz-Gilbert (LLG) equation

The dynamic of magnetic moment $\vec{\mu}_i$ is governed by LLG equation,

$$\frac{\partial \vec{S}_i}{\partial t} = -\frac{\gamma}{(1+\alpha^2)}\vec{S}_i \times (\vec{H}_i + \alpha \vec{S}_i \times \vec{H}_i)]$$

where $\vec{S}_i$ is the unit vector of magnetic moment,

$$\vec{S}_i = \frac{\vec{\mu}_i}{\mu_s},$$

and $\vec{\mu}_s = |\vec{\mu}_i|$, the effective field $\vec{H}_i$ is defined as

$$\vec{H}_i = -\frac{1}{\mu_s}\frac{\partial \mathcal{H}}{\partial \vec{S}_i}.$$

4.2 Interactions

In the level of atomic moments, $\vec{\mu}_s$, originated from the angular momentum of electrons in atoms, could be employed as the base unit in simulations. There are several typical interactions between magnetic moments. The total Hamiltonian is the summation of them.

$$\mathcal{H} = \mathcal{H}_{ex} + \mathcal{H}_{an} + \mathcal{H}_{d} + \mathcal{H}_{ext}$$

4.2.1 Exchange interaction

The classical Heisenberg Hamiltonian with the nearest-neighbor exchange interaction,

$$\mathcal{H}_{ex} = -J \sum_{<i,j>} \vec{S}_i \cdot \vec{S}_j$$

where the summation is taken only once for each pair, so the effective field is

$$\vec{H}_{i,ex} = \frac{J}{\mu_s} \sum_{<i,j>} \vec{S}_j$$

In the continuum limit the exchange energy could be written,

$$E_{ex} = \int_{\Omega} A(\nabla \vec{m})^2 dx$$

so the corresponding effective field is

$$\vec{H} = \frac{2A}{\mu_0 M_s} \nabla^2 \vec{m}$$

Once we implemented the Heisenberg exchange interaction, the effective field in the continuum case can be computed by the same codes with

$$J_x = 2A \frac{\Delta y \Delta z}{\Delta x}$$

Note that we needs the factor of μ_0 to convert the units from T to A/m.

4.2.2 Anisotropy

The Hamiltonian for uniaxial anisotropy with easy axis in x direction is expressed as,

$$\mathcal{H}_{an} = -D \sum_i (\vec{S}_{x,i})^2$$

and the corresponding field is

$$\vec{H}_{i,an} = \frac{2D}{\mu_s} S_{x,i} \vec{e}_x$$

4.2.3 UniaxialAnisotropy

The UniaxialAnisotropy energy of the magnetic system is defined as

$$E_{anis} = \int_{\Omega} K[1 - (\vec{m} \cdot \vec{u})^2] dx$$

the the effective field is,

$$\vec{H} = \frac{2K}{\mu_0 M_s} (\vec{m} \cdot \vec{u}) \vec{u}$$

4.2.4 Dipolar interaction

The Hamiltonian for dipolar interaction is,

$$\mathcal{H}_d = -\frac{\mu_0}{4\pi} \sum_{i<j} \frac{3(\vec{\mu}_i \cdot \vec{e}_{ij})(\vec{\mu}_j \cdot \vec{e}_{ij}) - \vec{\mu}_i \cdot \vec{\mu}_j}{r_{ij}^3}$$

$$\vec{H}_{i,d} = \frac{\mu_0}{4\pi} \sum_{i \neq j} \frac{3\vec{e}_{ij}(\vec{\mu}_j \cdot \vec{e}_{ij}) - \vec{\mu}_j}{r_{ij}^3} \tag{4.1}$$

4.2.5 Dzyaloshinskii-Moriya interaction (DMI)

DMI is an antisymmetric, anisotropic exchange coupling beteen spins (magnetic moments),

$$\mathcal{H}_{dmi} = \sum_{<i,j>} \vec{D}_{ij} \cdot [\vec{S}_i \times \vec{S}_j]$$

Note that $\vec{a} \cdot (\vec{b} \times \vec{c}) = (\vec{a} \times \vec{b}) \cdot \vec{c}$, the effective field can be computed by

$$\vec{H}_i = -\frac{1}{\mu_s} \frac{\partial \mathcal{H}}{\partial \vec{S}_i} = \frac{1}{\mu_s} \sum_{<i,j>} \vec{D}_{ij} \times \vec{S}_j$$

For bulk materials $\vec{D}_{ij} = D\vec{r}_{ij}$ and for interfacial DMI one has $\vec{D}_{ij} = D\vec{r}_{ij} \times \vec{e}_z$, in both cases the vector $\vec{D}_{ij}$ such that $\vec{D}_{ij} = -\vec{D}_{ji}$.

In the continuum limit the bulk DMI energy could be written,

$$E_{dmi} = \int_\Omega D_a \vec{m} \cdot (\nabla \times \vec{m}) dx$$

where $D_a = -D/a^2$ and the effective field is

$$\vec{H} = -\frac{2D_a}{\mu_0 M_s} (\nabla \times \vec{m})$$

For the interfacial case, the effective field thus becomes,

$$\vec{H} = \frac{2D}{M_s a^2} (\vec{e}_x \times \frac{\partial \vec{m}}{\partial y} - \vec{e}_y \times \frac{\partial \vec{m}}{\partial x})$$

Compared with the effective field [PRB 88 184422]

$$\vec{H} = \frac{2D_a}{\mu_0 M_s} ((\nabla \cdot \vec{m}) \vec{e}_z - \nabla m_z)$$

we have $D_a = D/a^2$, note that there is no negative sign for the interfacial case.

4.2.6 Zeeman energy

The zeeman energy is,

$$\mathcal{H}_{dmi} = -\sum_i \mu_s \vec{H}_{ext} \cdot \vec{S}_i$$

Basically, we will follow the above equations to write codes.

CHAPTER 5

Extended equations

5.1 Stochastic LLG equation

The implemented equation including the thermal fluctuation effects is

$$\frac{\partial \vec{m}}{\partial t} = -\gamma \vec{m} \times (\vec{H} + \vec{\xi}) + \alpha \vec{m} \times \frac{\partial \vec{m}}{\partial t}$$

where $\vec{\xi}$ is the thermal fluctuation field and is assumed to have the following properties,

$$\left\langle \vec{\xi} \right\rangle = 0, \quad \left\langle \vec{\xi}_i^u, \vec{\xi}_j^v \right\rangle = 2D\delta_{ij}\delta_{uv}$$

and

$$D = \frac{\alpha k_B T}{\gamma \mu_s}$$

5.2 Spin transfer torque (Zhang-Li model)

The extended equation with current is,

$$\frac{\partial \vec{m}}{\partial t} = -\gamma \vec{m} \times \vec{H} + \alpha \vec{m} \times \frac{\partial \vec{m}}{\partial t} + u_0(\vec{j}_s \cdot \nabla)\vec{m} - \beta u_0[\vec{m} \times (\vec{j}_s \cdot \nabla)\vec{m}]$$

Where

$$u_0 = \frac{pg\mu_B}{2|e|M_s} = \frac{pg\mu_B a^3}{2|e|\mu_s}$$

and $\mu_B = |e|\hbar/(2m)$ is the Bohr magneton. Notice that $\partial_x \vec{m} \cdot \vec{m} = 0$ so $u_0(\vec{j}_s \cdot \nabla)\vec{m} = -u_0 \vec{m} \times [\vec{m} \times (\vec{j}_s \cdot \nabla)\vec{m}]$. Besides, we can change the equation to atomistic one by introducing $\vec{s} = -\vec{S}$ where $\vec{S}$ is the local spin such that

$$\vec{M} = -\frac{g\mu_B}{a^3}\vec{S} = \frac{g\mu_B}{a^3}\vec{s}$$

so $u_0 = pa^3/(2|e|s)$, furthermore,

$$\frac{\partial \vec{s}}{\partial t} = -\gamma \vec{s} \times \vec{H} + \frac{\alpha}{s}\vec{s} \times \frac{\partial \vec{s}}{\partial t} + \frac{pa^3}{2|e|s}(\vec{j}_s \cdot \nabla)\vec{s} - \frac{pa^3\beta}{2|e|s^2}[\vec{s} \times (\vec{j}_s \cdot \nabla)\vec{s}]$$

However, we perfer the normalised equaiton here, after changing it to LL form, we obtain,

$$(1+\alpha^2)\frac{\partial \vec{m}}{\partial t} = -\gamma \vec{m} \times \vec{H} - \alpha\gamma \vec{m} \times (\vec{m} \times \vec{H}) + (1+\alpha\beta)u_0(\vec{j}_s \cdot \nabla)\vec{m} - (\beta - \alpha)u_0[\vec{m} \times (\vec{j}_s \cdot \nabla)\vec{m}]$$

although in principle $\partial_x \vec{m}$ is always perpendicular to $\vec{m}$, it's better to take an extra step to remove its longitudinal component, therefore, the real equation written in codes is,

$$(1+\alpha^2)\frac{\partial \vec{m}}{\partial t} = -\gamma \vec{m} \times \vec{H}_\perp + \alpha\gamma \vec{H}_\perp + (1+\alpha\beta)u_0 \vec{\tau}_\perp - (\beta-\alpha)u_0(\vec{m} \times \vec{\tau}_\perp)$$

where $\vec{\tau} = (\vec{j}_s \cdot \nabla)\vec{m}$ is the effective torque generated by current.

5.3 Nonlocal spin transfer torque (full version of Zhang-li model)

The LLG equation with STT is given by,

$$\frac{\partial \vec{m}}{\partial t} = -\gamma \vec{m} \times \vec{H} + \alpha \vec{m} \times \frac{\partial \vec{m}}{\partial t} + \vec{T}$$

where $\vec{T}$ is the spin transfer torque. In the local form the STT is given by,

$$\vec{T}_{loc} = u_0(\vec{j}_s \cdot \nabla)\vec{m} - \beta u_0[\vec{m} \times (\vec{j}_s \cdot \nabla)\vec{m}]$$

And in general case, the spin transfer torque could be computed by,

$$\vec{T} = -\frac{\vec{m} \times \delta\vec{m}}{\tau_{sd}}$$

where τ_{sd} is the s-d exchange time and $\delta\vec{m}$ is the nonequilibrium spin density governed by

$$\frac{\partial \delta\vec{m}}{\partial t} = D\nabla^2 \delta\vec{m} + \frac{\vec{m} \times \delta\vec{m}}{\tau_{sd}} - \frac{\delta\vec{m}}{\tau_{sf}} + u_0(\vec{j}_s \cdot \nabla)\vec{m}$$

By changing the LLG equation to LL form, we obtain,

$$(1+\alpha^2)\frac{\partial \vec{m}}{\partial t} = -\gamma \vec{m} \times \vec{H} - \alpha\gamma \vec{m} \times (\vec{m} \times \vec{H}) + \vec{T} + \alpha \vec{m} \times \vec{T}$$

i.e.,

$$(1+\alpha^2)\frac{\partial \vec{m}}{\partial t} = -\gamma \vec{m} \times \vec{H} - \alpha\gamma \vec{H}_\perp - \frac{\vec{m} \times \delta\vec{m}}{\tau_{sd}} + \alpha\frac{\delta\vec{m}}{\tau_{sd}}$$

CHAPTER 6

Spin transfer torque (Slonczewski type)

We consider the LLG equation with a Slonczewski-type extension [PRB 91 064423 (2015)],

$$\frac{\partial \vec{m}}{\partial t} = -\gamma \vec{m} \times \vec{H} + \alpha \vec{m} \times \frac{\partial \vec{m}}{\partial t} + u \vec{m} \times (\vec{p} \times \vec{m})$$

where $\vec{p}$ is the unit vector in the direction of the spin polarization. Similar to the Zhang-Li case, the implemented equation in the code is,

$$(1+\alpha^2)\frac{\partial \vec{m}}{\partial t} = -\gamma \vec{m} \times \vec{H}_\perp + \alpha\gamma \vec{H}_\perp + u\vec{p}_\perp + \alpha u \vec{m} \times \vec{p}_\perp$$

where $\vec{p}_\perp = \vec{p} - (\vec{m} \cdot \vec{p})\vec{m}$.

CHAPTER 7

Developing notes

7.1 Data structure

The first thing we may want to know is that how we label each spin in space, which can be peered from the index method in Mesh class

```
def index(self, i, j, k):
    idx = i*self.nyz + j*self.nz + k
    return idx
```

So we label the cells along z axis first then along y axis and the last is x, therefore we using the following code to loop all cells

```
for (i = 0; i < nx; i++) {
        for (j = 0; j < ny; j++) {
                for (k = 0; k < nz; k++) {
                        index = nyz * i + nz * j + k;
                        //do something ...
                }
        }
}
```

7.2 Units

We try to follow the SI units in the whole development, for example the gyromagnetic ratio γ is 1.76e11 rad/(s T). Because we use the sampe code to compute the exchange field and DMI field for both the atomic moments and the continuum case, so we need to take care of the relation between the two. For example, the saturated magnetization M_s can be computed by

$$M_s = \frac{\hbar\gamma S}{a^3}$$

7.3 Exchange and DMI energy

Although we can compute the total energy according to the direct definitions, however, we also can compute them using the corresponding effective fields, the energy density is

$$\mathcal{H}_i = -\vec{S}_i \cdot \vec{H}_i$$

where

$$\vec{H}_i = -\frac{\partial \mathcal{H}}{\partial \vec{S}_i}$$

and thus the total energy is

$$\mathcal{H} = \frac{1}{2} \sum_i \mathcal{H}_i$$

7.4 Anisotropy and external field energy

We compute them directly from the direct definitions.

CHAPTER 8

Fidimag: A Basic Simulation

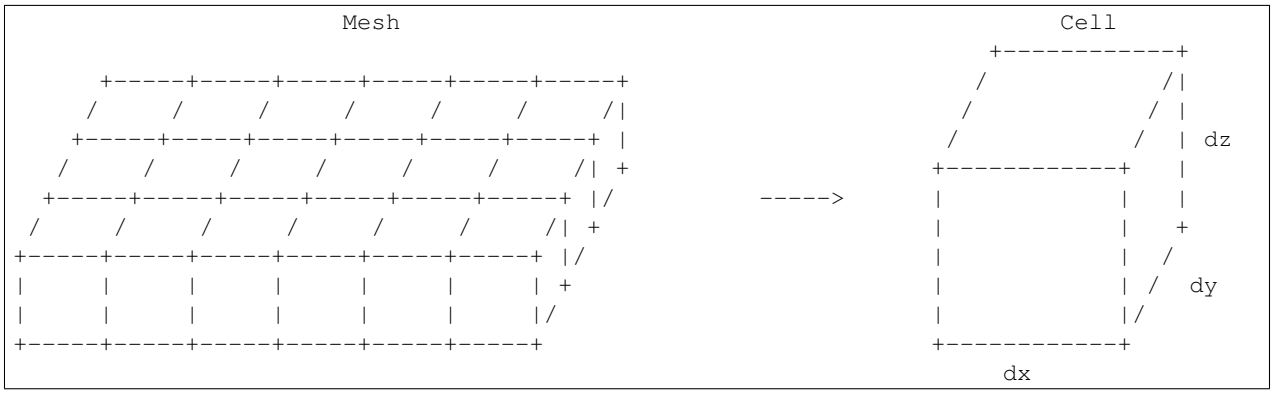
TODO: - [x] imports - [x] Meshes (+ parameters) - [] Material parameters - [] simulation object - [] adding energies - [] setting magnetisation - [] saving data - [] relax simulation - [] viewing data

This notebook is a guide to the essential commands required to write and run basic Fidimag simulations.

The first step is to import Fidimag. Numpy and Matplotlib are also imported for later use, to demonstrate visualising simulations results.

```
In [1]: import fidimag
        from fidimag.common import CuboidMesh
        import numpy as np
        import matplotlib.pyplot as plt
```

8.1 Meshes



We need to create a mesh. Meshes are created by specifying the dimensions of the finite difference cells, (dx, dy, dz) and the number of cells in each direction, (nx, ny, nz).

The cell dimensions are defined by dimensionless units. The dimensions of the mesh/cells are integrated by the parameter, unit_length.

In the the following example, the (cuboid) mesh consists of 6x3x1 cells (nx=6, ny=3 and nz=1), with each cell comprising of the dimensions, dx=3, dy=3 and dz=4. The unit_length = 1e-9 (nm).

Thus, the total size of the mesh is 18nm x 9nm x 4nm.

```
In [2]: nx, ny, nz = 6, 3, 1
        dx, dy, dz = 3, 3, 4 #nm
```

```
        unit_length = 1e-9 # define the unit length of the dx units to nm.

        mesh = CuboidMesh(nx=nx, ny=ny, nz=nz, dx=dx, dy=dy, dz=dz, unit_length=unit_length
```

In [3]:
```
nx
```

Out[3]:
```
6
```

In [ ]: