Journal of Magnetism and Magnetic Materials 387 (2015) 13–18

Contents lists available at ScienceDirect

Journal of Magnetism and Magnetic Materials

journal homepage: www.elsevier.com/locate/jmmm

A full-fledged micromagnetic code in fewer than 70 lines of NumPy

Claas Abert [a,*], Florian Bruckner [a], Christoph Vogler [b], Roman Windl [a], Raphael Thanhoffer [a], Dieter Suess [a]

[a] Christian Doppler Laboratory of Advanced Magnetic Sensing and Materials, Institute of Solid State Physics, Vienna University of Technology, Austria
[b] Institute of Solid State Physics, Vienna University of Technology, Austria

ARTICLE INFO

Article history:
Received 21 January 2015
Received in revised form
12 March 2015
Accepted 25 March 2015
Available online 27 March 2015

Keywords:
Micromagnetics
Finite-difference method
Python

ABSTRACT

We present a complete micromagnetic finite-difference code in fewer than 70 lines of Python. The code makes a large use of the NumPy library and computes the exchange field by finite differences and the demagnetization field with a fast convolution algorithm. Since the magnetization in finite-difference micromagnetics is represented by a multi-dimensional array and the NumPy library features a rich interface for this data structure, the code we present is an ideal starting point for the development of novel algorithms.

1. Introduction

Micromagnetic simulations have become an important tool for the investigation of ferromagnetic nanostructures. A lot has been published on algorithms and programming paradigms used for the numerical solution of the micromagnetic equations and a variety of open and closed source micromagnetic codes are available. These codes can be roughly divided into those acting on regular cuboid grids [1–3] and those acting on irregular grids [4–9]. Irregular-grid codes usually employ finite-elements or fast-multipole methods for spatial discretization [10]. A popular class of regular grid methods applies the finite-difference method for the computation of the exchange field and a fast convolution for the computation of the demagnetization field [11]. In this work we present a complete micromagnetic code of the latter kind that is written in only 70 lines of Python and that makes extensive use of the NumPy library [12].

The code we will present is not able to compete with mature finite-difference codes in terms of performance and flexibility. However, it delivers all essential building blocks of a micromagnetic code and is therefore perfectly suited for prototyping new micromagnetic algorithms. In particular, the NumPy library is a good choice for the code we will present, since the magnetization in finite-difference micromagnetics is represented by an n-dimensional array and the NumPy library has a very powerful interface for n-dimensional arrays that supports a large variety of operations.

The paper is structured as follows. In Section 2 we give a brief overview of the micromagnetic model. In Sections 3–5 the implementation of the most important micromagnetic subproblems is described. The code we will present is validated by numerical experiments in Section 6.

2. Micromagnetic model

The central equation of dynamic micromagnetics is the Landau–Lifshitz–Gilbert equation that describes the motion of a continuous magnetization configuration $\boldsymbol{m}$ in an effective field $\boldsymbol{H}_{\text{eff}}$ is

$$\frac{\partial \boldsymbol{m}}{\partial t} = -\frac{\gamma}{1+\alpha^2}\boldsymbol{m}\times\boldsymbol{H}_{\text{eff}} - \frac{\alpha\gamma}{1+\alpha^2}\boldsymbol{m}\times(\boldsymbol{m}\times\boldsymbol{H}_{\text{eff}}) \tag{1}$$

where γ is the reduced gyromagnetic ratio and $\alpha \geq 0$ is a dimensionless damping constant. The effective field $\boldsymbol{H}_{\text{eff}}$ is given by the negative variational derivative of the free energy:

$$\boldsymbol{H}_{\text{eff}} = -\frac{1}{\mu_0 M_s}\frac{\delta U}{\delta \boldsymbol{m}} \tag{2}$$

where μ_0 is the magnetic constant and M_s is the saturation magnetization. Contributions to the effective field usually include the demagnetization field, the exchange field, the external Zeeman field and terms describing anisotropic effects. In this work we focus on the numerical computation of the demagnetization field, see Section 3, and the exchange field, see Section 4, as well as the integration of the Landau–Lifshitz–Gilbert equation, see Section 5.

For the numerical solution of (1) and (2) a regular cuboid grid is used for the spatial discretization. Every simulation cell is of size $\Delta r_1 \times \Delta r_2 \times \Delta r_3$ and can be addressed by a multi-index $\boldsymbol{i} = (i_1, i_2, i_3)$. All spatially varying quantities such as the magnetization $\boldsymbol{m}$ are thus represented by an n-dimensional array:

* Corresponding author.
E-mail address: claas.abert@tuwien.ac.at (C. Abert).

http://dx.doi.org/10.1016/j.jmmm.2015.03.081
0304-8853/

$$\boldsymbol{m}(\boldsymbol{r}) \approx \boldsymbol{m}_{\boldsymbol{i}}. \tag{3}$$

The Python library NumPy provides the class `ndarray` for this purpose which supports a large number of operations. Despite Python being a scripting language, all collective operations of `ndarray` have a good performance due to the native implementation of the NumPy library.

3. Demagnetization field

The demagnetization field accounts for the dipole–dipole interaction of the elementary magnets. For a continuous magnetization configuration the demagnetization field is given by

$$\boldsymbol{H}_{\mathrm{demag}}(\boldsymbol{r}) = M_{\mathrm{s}} \int_{\Omega} \tilde{\boldsymbol{N}}(\boldsymbol{r}-\boldsymbol{r}')\boldsymbol{m}(\boldsymbol{r}')\,\mathrm{d}\boldsymbol{r}' \tag{4}$$

$$\tilde{\boldsymbol{N}}(\boldsymbol{r}-\boldsymbol{r}') = -\frac{1}{4\pi}\nabla\nabla'\frac{1}{|\boldsymbol{r}-\boldsymbol{r}'|} \tag{5}$$

where $\tilde{\boldsymbol{N}}$ is called demagnetization tensor. This expression has the form of a convolution of the magnetization $\boldsymbol{m}$ with the matrix valued kernel $\tilde{\boldsymbol{N}}$. By choice of a regular grid this convolution structure can also be exploited on the discrete level:

$$\boldsymbol{H}_{\boldsymbol{i}} = M_{\mathrm{s}} \sum_{\boldsymbol{j}} \tilde{\boldsymbol{N}}_{\boldsymbol{i}-\boldsymbol{j}}\boldsymbol{m}_{\boldsymbol{j}} \tag{6}$$

$$\tilde{\boldsymbol{N}}_{\boldsymbol{i}-\boldsymbol{j}} = \frac{1}{\Delta r_1 \Delta r_2 \Delta r_3} \iint_{\Omega_{\mathrm{cell}}} \tilde{\boldsymbol{N}}\left(\sum_k (i_k - j_k)\Delta r_k \boldsymbol{e}_k + \boldsymbol{r} - \boldsymbol{r}'\right)\mathrm{d}\boldsymbol{r}\,\mathrm{d}\boldsymbol{r}' \tag{7}$$

where Ω_{cell} describes a cuboid reference cell and $\boldsymbol{e}_k$ is a unit vector in direction of the kth coordinate axis. Here the magnetization is assumed to be constant within each simulation cell and the field generated by each source cell is averaged over each target cell. This results in a sixfold integral for the computation of the discrete demagnetization tensor $\tilde{\boldsymbol{N}}_{\boldsymbol{i}-\boldsymbol{j}}$. An analytical solution of (7) was derived by Newell et al. [13]. The diagonal element $N^{1,1}$ computes as

$$N^{1,1}_{\boldsymbol{i}-\boldsymbol{j}} = -\frac{1}{4\pi\Delta r_1\Delta r_2\Delta r_3} \sum_{\boldsymbol{k},\boldsymbol{l}\in\{0,1\}} (-1)^{\sum_x k_x + l_x} f[(i_1 - j_1 + k_1 - l_1)\Delta r_1, (i_2 - j_2 + k_2 - l_2)\Delta r_2, (i_3 - j_3 + k_3 - l_3)\Delta r_3] \tag{8}$$

where the auxiliary function f is defined by

$$\begin{aligned} f(r_1, r_2, r_3) = {} & \frac{|r_2|}{2}(r_3^2 - r_1^2)\sinh^{-1}\left(\frac{|r_2|}{\sqrt{r_1^2 + r_3^2}}\right) \\ & + \frac{|r_3|}{2}(r_2^2 - r_1^2)\sinh^{-1}\left(\frac{|r_3|}{\sqrt{r_1^2 + r_2^2}}\right) \\ & - |r_1 r_2 r_3|\tan^{-1}\left(\frac{|r_2 r_3|}{r_1\sqrt{r_1^2 + r_2^2 + r_3^2}}\right) \\ & + \frac{1}{6}(2r_1^2 - r_2^2 - r_3^2)\sqrt{r_1^2 + r_2^2 + r_3^2}. \end{aligned} \tag{9}$$

The elements $N^{2,2}$ and $N^{3,3}$ are obtained by circular permutation of the coordinates:

$$N^{2,2}_{\boldsymbol{i}-\boldsymbol{j}} = N^{1,1}_{(i_2,i_3,i_1)-(j_2,j_3,j_1)} \tag{10}$$

$$N^{3,3}_{\boldsymbol{i}-\boldsymbol{j}} = N^{1,1}_{(i_3,i_1,i_2)-(j_3,j_1,j_2)} \tag{11}$$

According to Newell the off-diagonal element $N^{1,2}$ is given by

$$N^{1,2}_{\boldsymbol{i}-\boldsymbol{j}} = -\frac{1}{4\pi\Delta r_1\Delta r_2\Delta r_3} \sum_{\boldsymbol{k},\boldsymbol{l}\in\{0,1\}} (-1)^{\sum_x k_x + l_x} g[(i_1 - j_2 + k_1 - l_1)\Delta r_1, (i_2 - j_2 + k_2 - l_2)\Delta r_2, (i_3 - j_3 + k_3 - l_3)\Delta r_3] \tag{12}$$

where the function g is defined by

$$\begin{aligned} g(r_1, r_2, r_3) = {} & (r_1 r_2 r_3)\sinh^{-1}\left(\frac{r_3}{\sqrt{r_1^2 + r_2^2}}\right) \\ & + \frac{r_2}{6}(3r_3^2 - r_2^2)\sinh^{-1}\left(\frac{r_1}{\sqrt{r_2^2 + r_3^2}}\right) \\ & + \frac{r_1}{6}(3r_3^2 - r_1^2)\sinh^{-1}\left(\frac{r_2}{\sqrt{r_1^2 + r_3^2}}\right) \\ & - \frac{r_3^3}{6}\tan^{-1}\left(\frac{r_1 r_2}{r_3\sqrt{r_1^2 + r_2^2 + r_3^2}}\right) \\ & - \frac{r_3 r_2^2}{2}\tan^{-1}\left(\frac{r_1 r_3}{r_2\sqrt{r_1^2 + r_2^2 + r_3^2}}\right) \\ & - \frac{r_3 r_1^2}{2}\tan^{-1}\left(\frac{r_2 r_3}{r_1\sqrt{r_1^2 + r_2^2 + r_3^2}}\right) \\ & - \frac{r_1 r_2\sqrt{r_1^2 + r_2^2 + r_3^2}}{3}. \end{aligned} \tag{13}$$

Again, other off-diagonal elements are obtained by permutation of coordinates:

$$N^{1,3}_{\boldsymbol{i}-\boldsymbol{j}} = N^{1,2}_{(i_1,i_3,i_2)-(j_1,j_3,j_2)} \tag{14}$$

$$N^{2,3}_{\boldsymbol{i}-\boldsymbol{j}} = N^{1,2}_{(i_2,i_3,i_1)-(j_2,j_3,j_1)}. \tag{15}$$

The remaining components of the tensor are obtained by exploiting the symmetry of $\tilde{\boldsymbol{N}}$, i.e. $N^{i,j} = N^{j,i}$.

Listing 1. Definition of auxiliary functions f and g. The very small number `eps` is added to denominators in order to avoid division-by-zero errors.

```
eps = 1e-18

def f(p):
  x, y, z = abs(p[0]), abs(p[1]), abs(p[2])
  return + y/2.0*(z**2-x**2)*asinh(y/(sqrt(x**2+z**2)+eps)) \
         + z/2.0*(y**2-x**2)*asinh(z/(sqrt(x**2+y**2)+eps)) \
         - x*y*z*atan(y*z/(x * sqrt(x**2+y**2+z**2)+eps))   \
         + 1.0/6.0*(2*x**2-y**2-z**2)*sqrt(x**2+y**2+z**2)

def g(p):
  x, y, z = p[0], p[1], abs(p[2])
  return + x*y*z*asinh(z/(sqrt(x**2+y**2)+eps))                \
         + y/6.0*(3.0*z**2-y**2)*asinh(x/(sqrt(y**2+z**2)+eps)) \
         + x/6.0*(3.0*z**2-x**2)*asinh(y/(sqrt(x**2+z**2)+eps)) \
         - z**3/6.0  *atan(x*y/(z*sqrt(x**2+y**2+z**2)+eps))    \
         - z*y**2/2.0*atan(x*z/(y*sqrt(x**2+y**2+z**2)+eps))    \
         - z*x**2/2.0*atan(y*z/(x*sqrt(x**2+y**2+z**2)+eps))    \
         - x*y*sqrt(x**2+y**2+z**2)/3.0
```

The calculation of the discrete demagnetization tensor in Python is straightforward. Listing 1 shows the function definitions for the auxiliary functions f and g. Note that fractions occurring in f and g might feature zero denominators. However, limit considerations show that all fractions tend to zero in this case. In order to avoid division-by-zero errors in the implementation, a very small floating point number `eps` is added to all denominators.

Listing 2. Assembly of the demagnetization tensor $\tilde{\boldsymbol{N}}$. Only the six distinct components of the symmetric tensor are computed.

```
def set_n_demag(c, permute, func):
  it = np.nditer(n_demag[:,:,:,c], flags=['multi_index'], op_flags=['writeonly'])
  while not it.finished:
    value = 0.0
    for i in np.rollaxis(np.indices((2,)*6), 0, 7).reshape(64, 6):
      idx=map(lambda k: (it.multi_index[k]+n[k]-1)%(2*n[k]-1)-n[k]+1,range(3))
      value+=(-1)**sum(i)*func(map(lambda j: (idx[j]+i[j]-i[j+3])*dx[j], permute))
    it[0]=-value/(4*pi*np.prod(dx))
    it.iternext()

n_demag = np.zeros([2*i-1 for i in n] + [6])
for i, t in enumerate(((f,0,1,2),(g,0,1,2),(g,0,2,1),
                       (f,1,2,0),(g,1,2,0),(f,2,0,1))):
  set_n_demag(i, t[1:], t[0])
```