CF-03 Python interface for OOMMF

Marijan Beg, Ryan A. Pepper, and Hans Fangohr

2016-11-02
Overview – JOOMMF Project

- Towards embedding OOMMF into the Jupyter Notebook (→ J-OOMMF)
- Step 1: Drive OOMMF through Python interface
- Step 2: Develop data analysis tools
- Step 3: Interactive documentation, micromagnetic tutorial, reproducibility

Status

Step 1 Prototype completed

- Python interface for OOMMF: “OOMMF Calculator” ≡ OOMMFC
Standard Problem 3


CF-03: Hans Fangohr: Python interface for OOMMF
Live demonstration 1

```python
In [1]: import oommfc as oc  # access to OOMMF Calculator
    import discretisedfield as df  # other setup to keep the next slides brief
    import numpy as np
    from math import sin, cos, pi, sqrt
    import matplotlib.pyplot as plt
    %matplotlib inline

Micromagnetic standard problem 3

In [2]: def m_init_flower(pos):
    """Given a pos vector pos = (x, y, z), return the magnetisation vector (mx, my, mz) for that position.""
    x, y, z = pos[0]/1e-9, pos[1]/1e-9, pos[2]/1e-9
    # flower pattern:
    mx = 0
    my = 2 * z - 1
    mz = -2 * y + 1
    norm_squared = mx**2 + my**2 + mz**2
    if norm_squared <= 0.05:
        return (1, 0, 0)
    else:
        return (mx, my, mz)

def m_init_vortex(pos):
    """Given a pos vector pos = (x, y, z), return the magnetisation vector (mx, my, mz) for that position.""
    x, y, z = pos[0]/1e-9, pos[1]/1e-9, pos[2]/1e-9
    # vortex pattern
    mx = 0
    my = sin(pi/2 * (x-0.5))
    mz = cos(pi/2 * (x-0.5))
    return (mx, my, mz)
```
Relaxed magnetisation states: vortex state

In [4]: system = minimise_system_energy(8, m_init_vortex)  # calling OOMMF

In [5]: fig = system.m.plot_slice('y', 50e-9, xsize=4)
Live demonstration 3

```python
Flower state:

In [6]: system = minimise_system_energy(8, m_init_flower)

In [7]: fig = system.m.plot_slice('y', 50e-9, xsize=4)
```
Create the energy crossing plot

In [8]:
L_array = np.linspace(8, 9, 3)  # values of L, from 8 to 9 in 4 steps
vortex_energies = []
flower_energies = []

for L in L_array:
    print("Computing vortex L={} using OOMMF".format(L))
    vortex = minimise_system_energy(L, m_init_vortex)
    print("Computing flower L={} using OOMMF".format(L))
    flower = minimise_system_energy(L, m_init_flower)
    vortex_energies.append(vortex.total_energy())  # remember energies for later
    flower_energies.append(flower.total_energy())  # plotting

Computing vortex L=8.0 using OOMMF
Computing flower L=8.0 using OOMMF
Computing vortex L=8.5 using OOMMF
Computing flower L=8.5 using OOMMF
Computing vortex L=9.0 using OOMMF
Computing flower L=9.0 using OOMMF
Live demonstration 5

```python
In [9]: plt.plot(L_array, vortex_energies, 'o-', label='vortex')
plt.plot(L_array, flower_energies, 'o-', label='flower')
plt.xlabel('L (lex)')
plt.ylabel('E')
plt.xlim([7.9, 9.1])
plt.grid()
plt.legend();
```
Use bisection method to find energy crossing automatically

In [10]: from scipy.optimize import bisect

def energy_difference(L):
    print("Computing energy difference at L = {}\).format(L))
    vortex = minimise_system_energy(L, m_init_vortex)
    flower = minimise_system_energy(L, m_init_flower)
    return vortex.total_energy() - flower.total_energy()

cross_section = bisect(energy_difference, 8.3, 8.5, xtol=0.01)

print("The transition between vortex and flower states occurs approximately at {}\).format(cross_section))

Computing energy difference at L = 8.3
Computing energy difference at L = 8.5
Computing energy difference at L = 8.4
Computing energy difference at L = 8.45
Computing energy difference at L = 8.425
Computing energy difference at L = 8.4125
Computing energy difference at L = 8.41875
The transition between vortex and flower states occurs approximately at 8.41875*lex
<table>
<thead>
<tr>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Present</strong></td>
</tr>
<tr>
<td>• OOMMF simulation study in single (Python) file</td>
</tr>
<tr>
<td>• Multiple simulation runs within the same script</td>
</tr>
<tr>
<td>• Exploit existing libraries and tools (root finding)</td>
</tr>
<tr>
<td><strong>Future</strong></td>
</tr>
<tr>
<td>• Embedding interactive simulation data analysis, and visualisation</td>
</tr>
<tr>
<td>• Reproducibility</td>
</tr>
<tr>
<td>• Interactive documentation</td>
</tr>
<tr>
<td>• ...</td>
</tr>
</tbody>
</table>
### How does the interface to OOMMF work?

#### Via MIF files

1. write MIF file
2. execute OOMMF
3. read output files

#### Why ?

- most robust approach
- see [https://arxiv.org/abs/1609.07432](https://arxiv.org/abs/1609.07432) for details
How to install?

1. Need OOMMF natively installed
   (and set variable `OOMMFTCL` to point to `oommf.tcl` file)
   or
   Docker (http://docker.com)

2. Need Python (Suggest Anaconda distribution)

3. Install oommfc via
   
   `$> pip install oommfc`
Is it ready to use?

### Software ready to use?

- Yes(-ish)
- Interface may change, although we try to avoid it
- Beta users and questions welcome

### Installation and support workshop for OOMMFC

- Wednesday 17:00 - 19:00 (today)
- Thursday 17:00 - 19:00 (tomorrow)

Outside ”Galerie 4” on Level 2, drop-in anytime
Summary

Python interface for OOMMF (OOMMF Calculator)

- part of JOOMMF Project
- invite the community to engage
  - with ideas, questions and bug reports
  - subscribe to joommf-news mailing list
  - come to workshop tonight/tomorrow
- http://joommf.github.io

Acknowledgements: Financial support from

- OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541), http://opendreamkit.org
- EPSRC’s Centre for Doctoral Training in Next Generation Computational Modelling, http://ngcm.soton.ac.uk (#EP/L015382/1)
- EPSRC’s Programme grant on Skyrmionics (#EP/N032128/1)