Computational micromagnetics with JOOMMF

Marijan Beg, Ryan A. Pepper, and Hans Fangohr

University of Southampton, UK
European XFEL, Germany

2017-04-03

JOOMMF Project

- Towards embedding OOMMF into the Jupyter Notebook ($\rightarrow$ J-OOMMF)
- Drive OOMMF through Python interface
- Use Python interface in Jupyter Notebook

Terminology: OOMMFC

- OOMMFC Stands for OOMMF Calculator

Example 1

Macrospin

Example 2

Skyrmion

Benefits

Benefits Python Interface

- OOMMF simulation study in single (Python) file
- Multiple simulation runs within the same script
- Exploit existing (Python) libraries and tools

Benefits Jupyter Notebook

- Can embed simulation and data analysis in one document
- interactive exploration
- Reproducibility
- Sharing of study (static output as html, latex, pdf)
- ... ($\rightarrow$ standard problem 3)

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. More details in:
- Marijan Beg, Ryan A. Pepper, Hans Fangohr *User interfaces for computational science: a domain specific language for OOMMF embedded in Python* AIP Advances 7, 056025 (2017) `https://arxiv.org/abs/1609.07432`

How to install?

- See `http://joommf.github.io`
- Instructions for Windows, OSX and Linux

---

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
- Not all OOMMF features supported yet - tell us what is important
- users and questions welcome

Support available

- see `http://joommf.github.io`
- Email, Gitter, Github issues, Workshops, ...

JOOMMF Workshop this Wednesday

Wednesday: main workshop: lectures and exercises

- 10:30 – 12:30 Introduction to Micromagnetics and OOMMF (Mike Donahue)
- 14:00 – 17:00 *Introduction to Jupyter OOMMF* (Hans Fangohr, Marijan Beg, Ryan Pepper, Leoni Breth)

Every day (Mon, Tue, Wed): Installation and help desk

- Monday 17:30-18:30 (Bierstube, red sofas exhibition centre)
- Tuesday 17:00-18:00 (after talks, red sofas)
- Wednesday 9:00 - 10:00 (before workshop, 2nd floor teaching tower block)

Summary

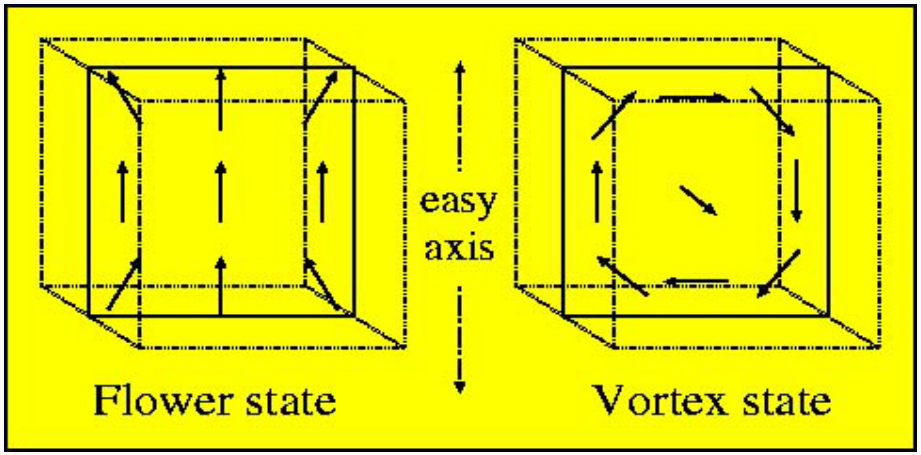
Jupyter-OOMMF

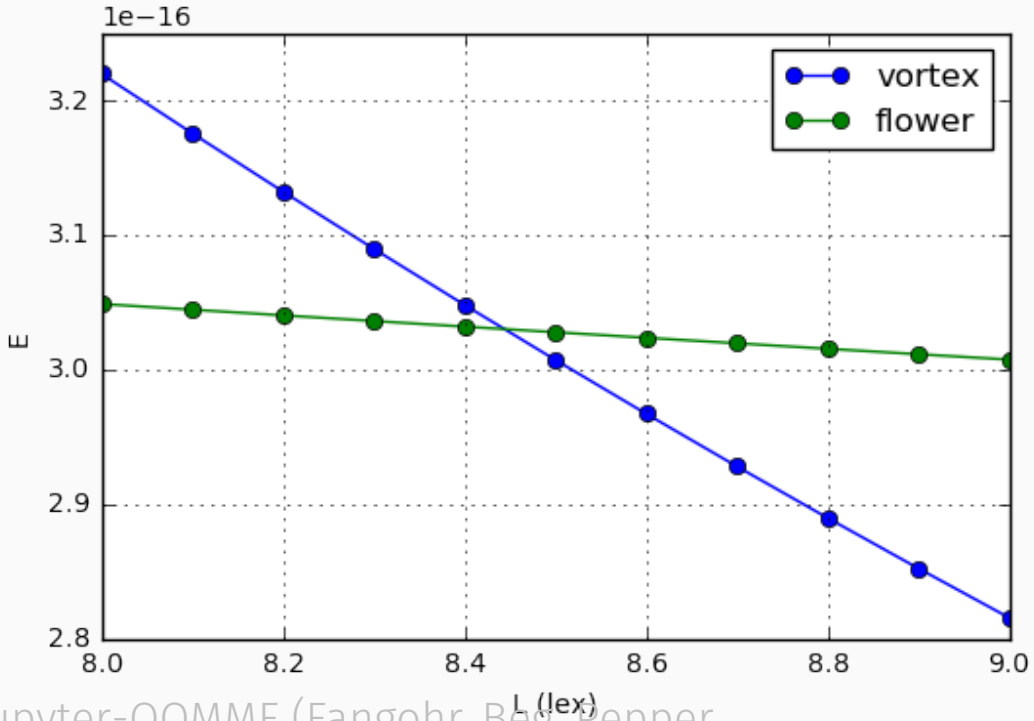
- drive OOMMF from Notebook using Python
- nearly ready to use
- feedback welcome
- http://joommf.github.io

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)

Standard Problem 3





Full problem specification:

`http://www.ctcms.nist.gov/~rdm/spec3.html`

Live demonstration 1

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

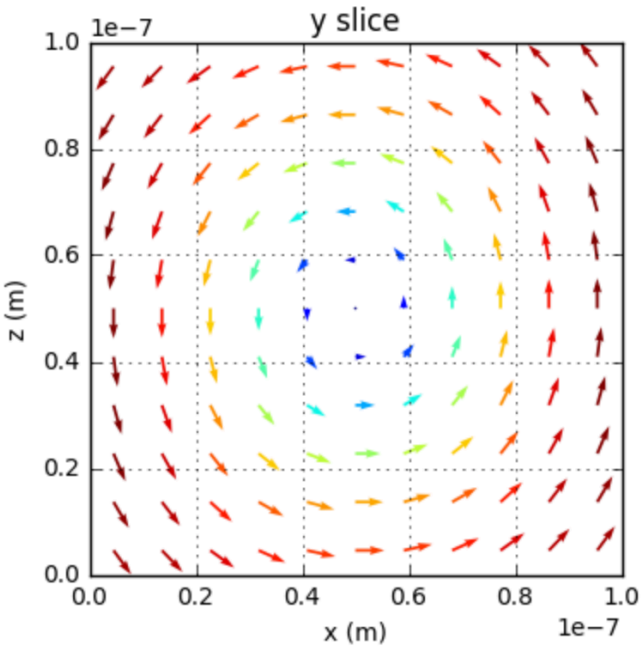
In [2]:
```
def m_init_flower(pos):
    """Given a pos vector pos = (x, y, z), return the magnetisation
    vector (mx, my, mz) for that position."""
    x, y, z = pos[0]/1e-9, pos[1]/1e-9, pos[2]/1e-9
    # flower pattern:
    mx = 0
    my = 2 * z - 1
    mz = -2 * y + 1
    norm_squared = mx**2 +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

Live demonstration 2

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)
```

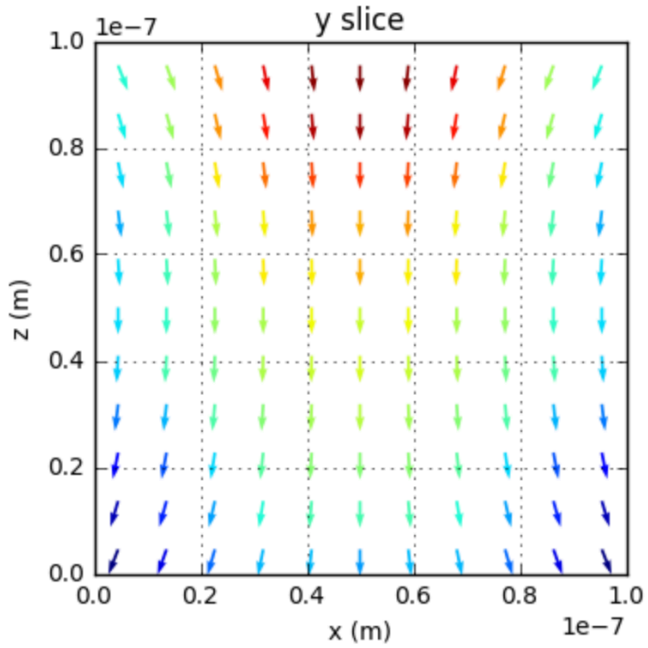


Live demonstration 3

Flower state:

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

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



Live demonstration 4

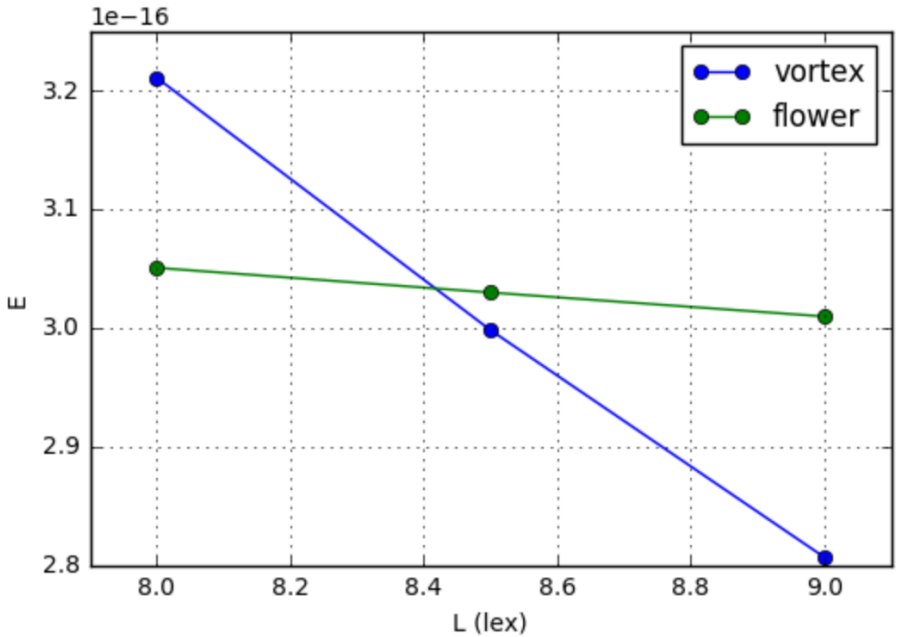
Create the energy crossing plot

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

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();
```



Live demonstration 6

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 {}*lex".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
```