



A case study of computational science in Jupyter notebooks: JOOMMF

Demonstrator introduction - Hans Fangohr

University of Southampton

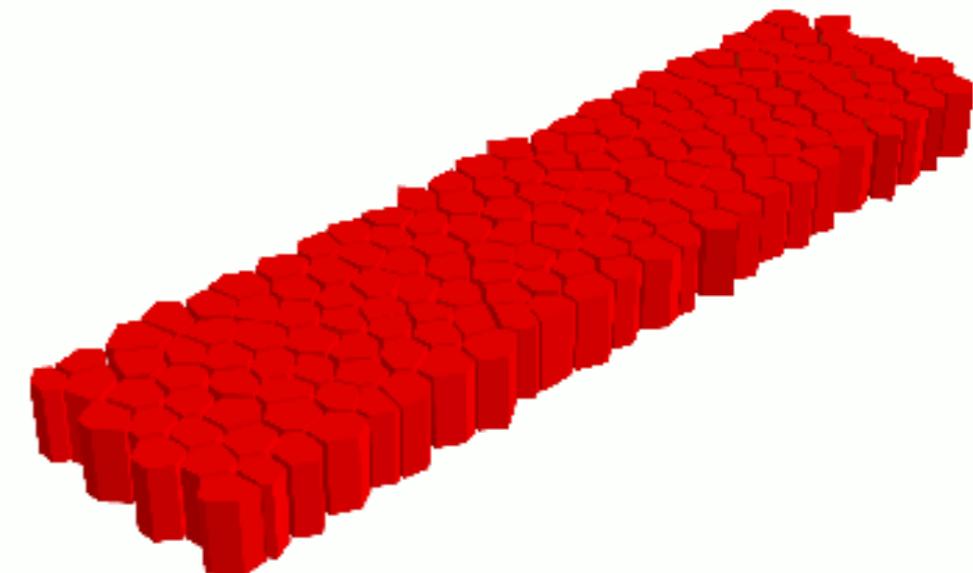
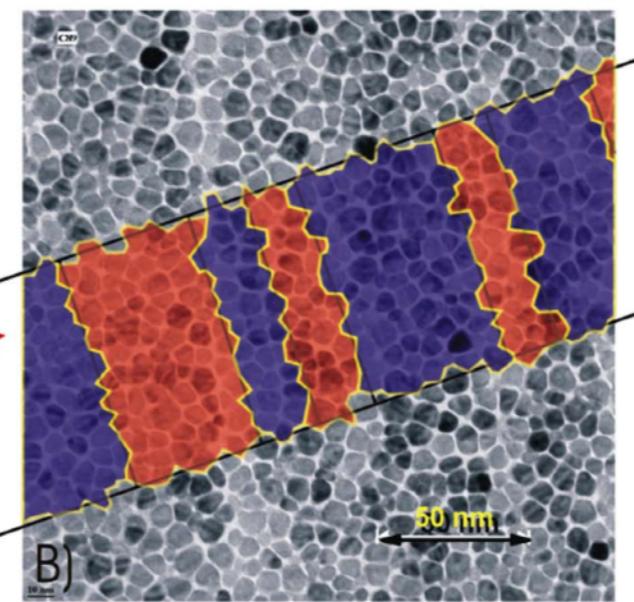
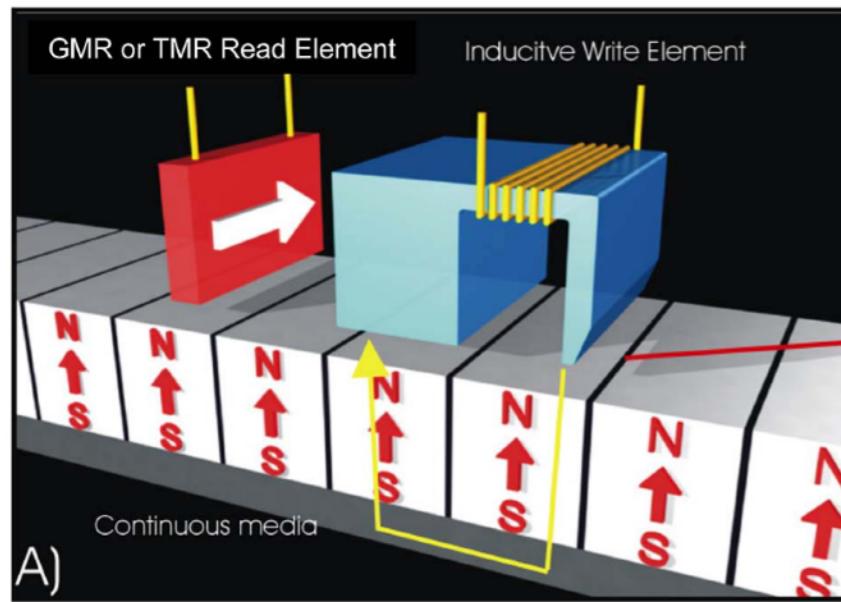
Overview

- Introduction Micromagnetic Research
- State-of-the-art micromagnetics simulation tool
- Beyond state-of-the art: Jupyter OOMMF
- Outlook

Micromagnetics
(magnetism at small length scales,
typically nanometre to micrometre)

Why magnetic nanostructures?

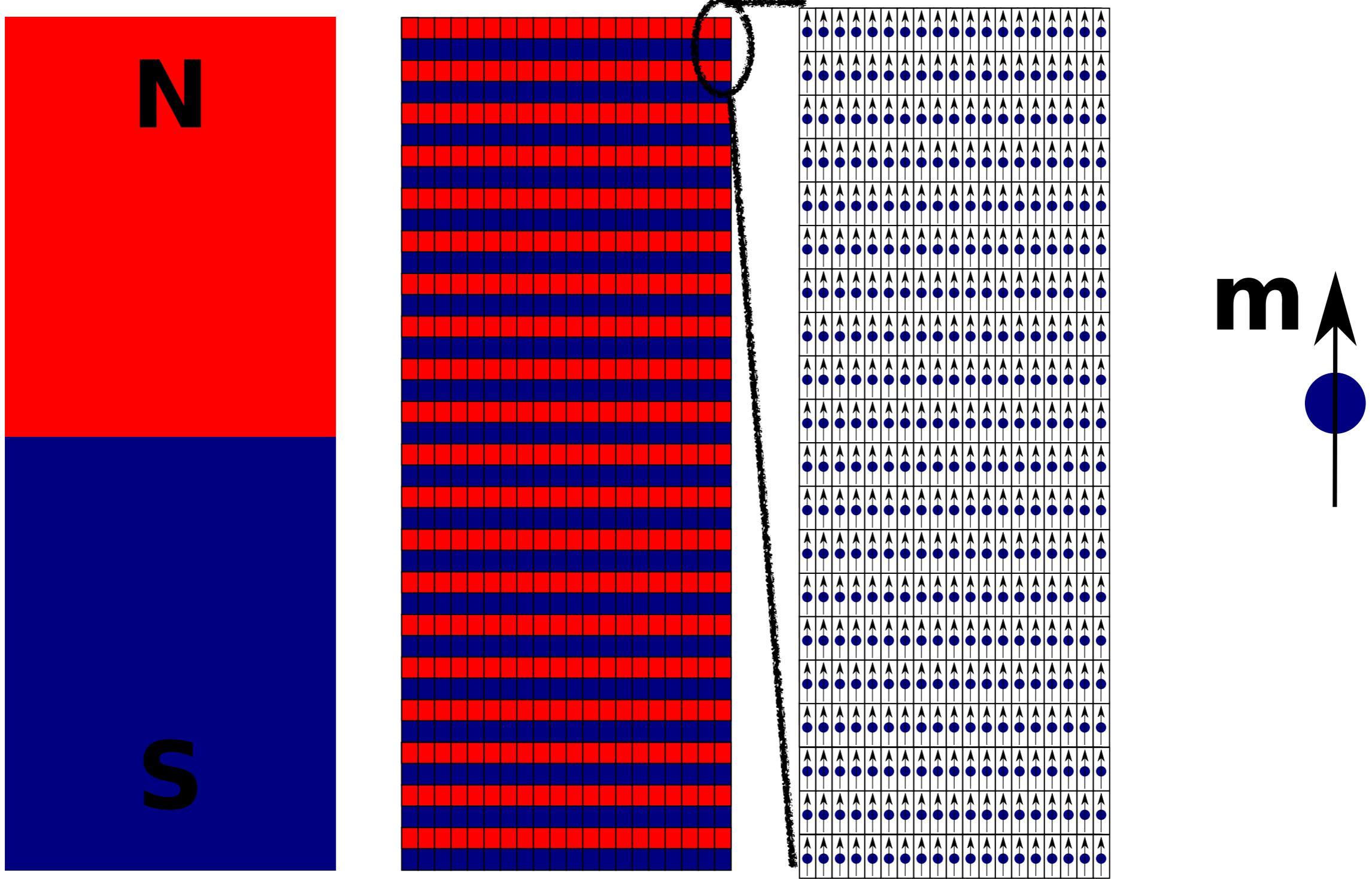
1. Interesting complex system with tuneable parameters and experiments
2. Applications include
 - magnetic data storage (hard disk)
 - cancer diagnostics and therapy
 - low energy magnetic logic (spintronics, skyrmionics)



E. Dobisz et. al., Proceedings of IEEE 96, 1836 (2008)

Curtis & Fangohr (2011)

Magnetic moment



Magnetisation dynamics

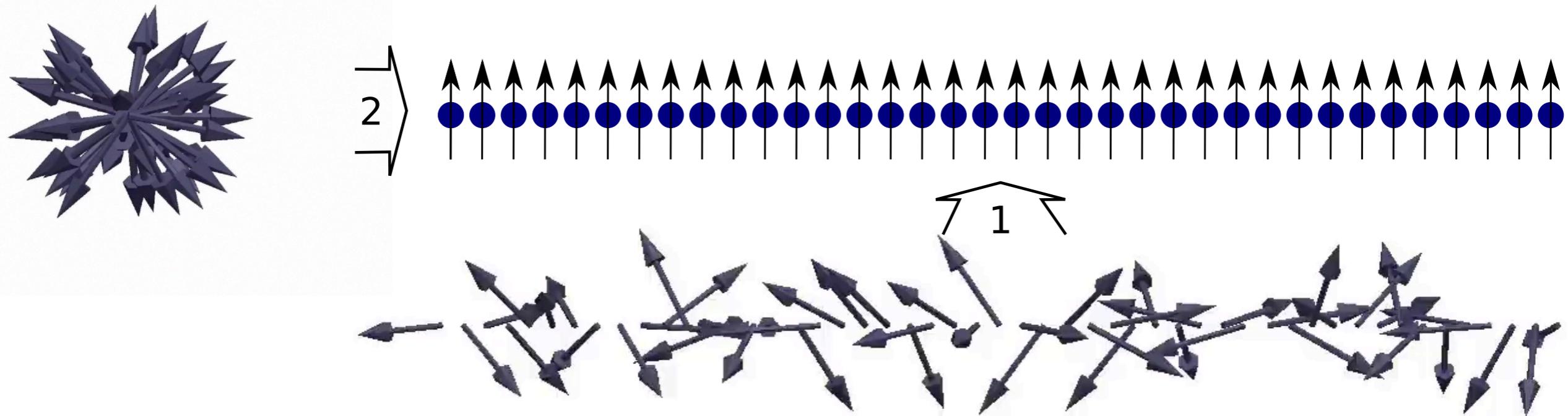
- Landau-Lifshitz-Gilbert (LLG) equation

$$\frac{\partial \mathbf{m}}{\partial t} = \underbrace{\gamma^* \mathbf{m} \times \mathbf{H}_{\text{eff}}}_{\text{precession}} + \underbrace{\alpha \mathbf{m} \times \frac{\partial \mathbf{m}}{\partial t}}_{\text{damping}}$$

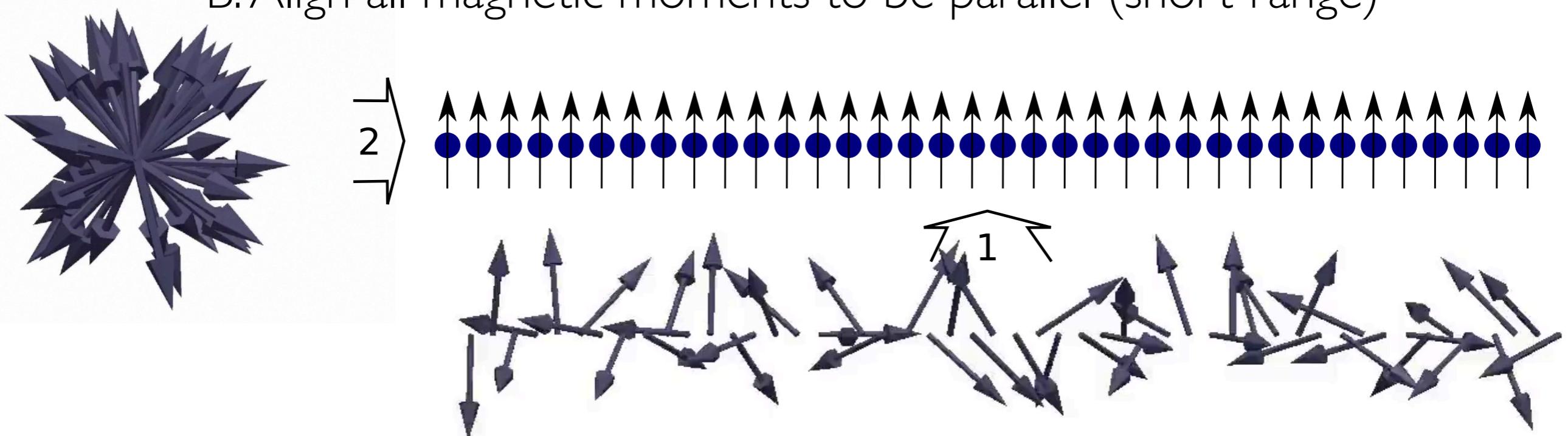
The diagram illustrates the decomposition of magnetization dynamics into precession and damping components. At the top, a magnetization vector \mathbf{m} (blue arrow) is shown precessing around an effective field \mathbf{H}_{eff} (red arrow). A derivative term $\frac{\partial \mathbf{m}}{\partial t}$ is shown as a dark grey arrow pointing downwards. This is labeled "precession" and "damping". Below this, the equation is shown again, separated by a horizontal line. The effective field \mathbf{H}_{eff} is shown as equal to its initial value plus a derivative term, with a circled dot indicating the starting point.

Different types of physics

A: Align the magnetic moment to an external field pointing up (local)

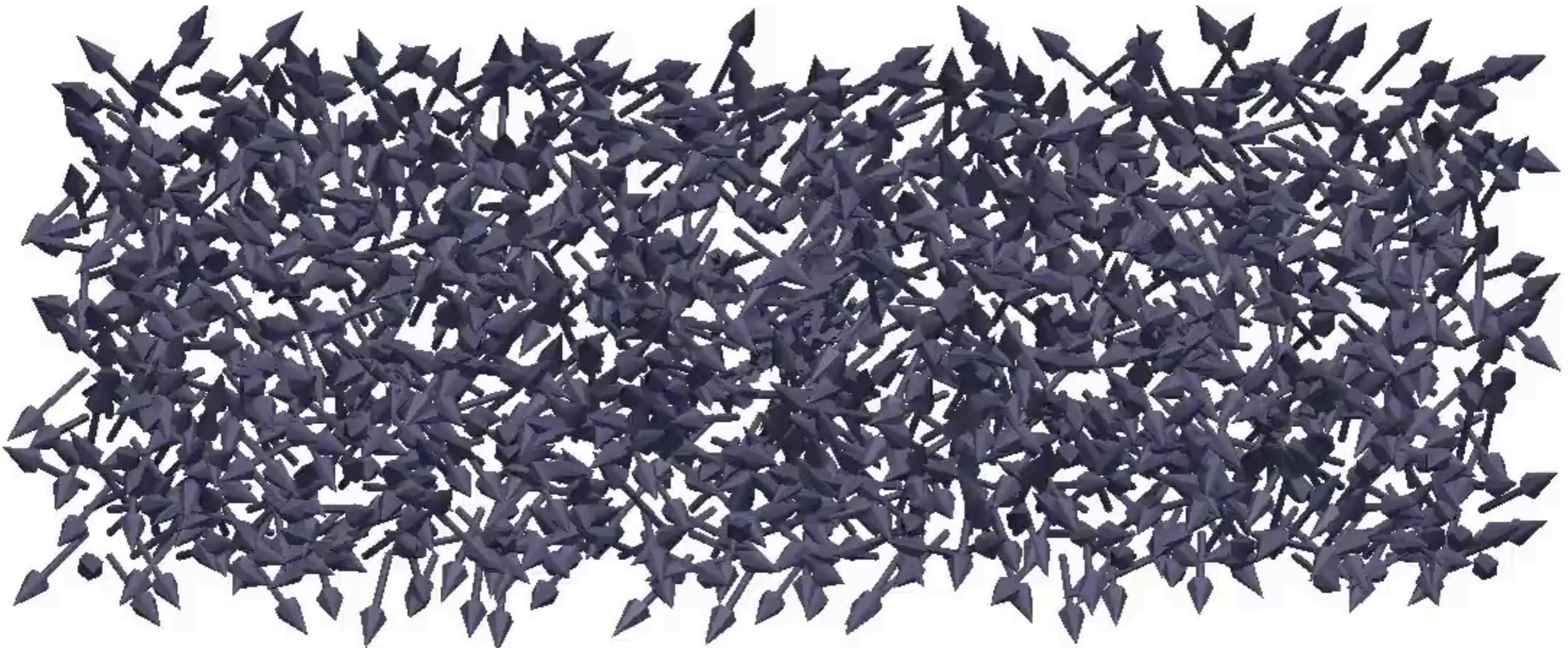


B: Align all magnetic moments to be parallel (short-range)



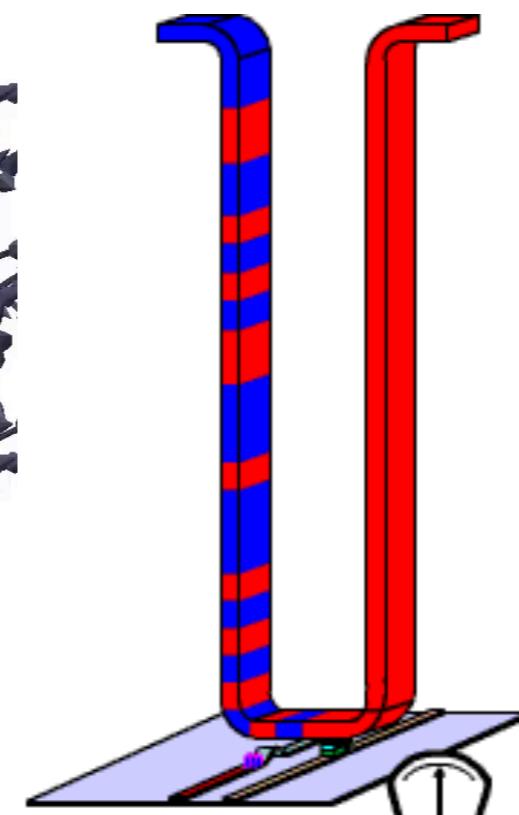
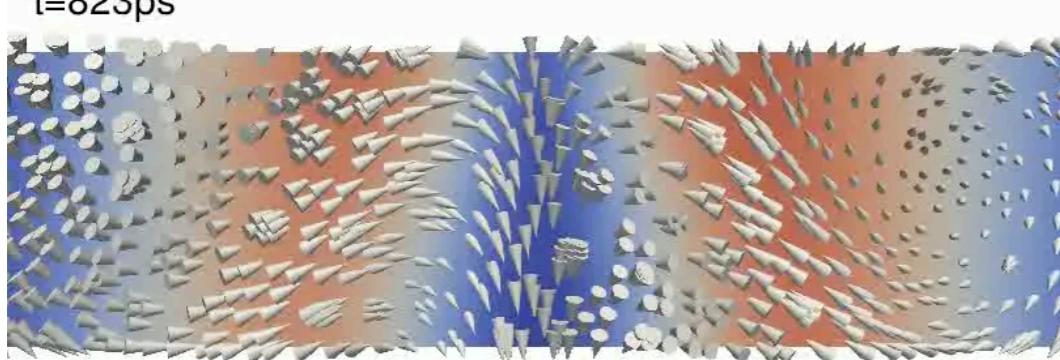
More complicated case

- Two-dimensional sample.
- Four interactions included (Exchange, Zeeman, Anisotropy, Dzyaloshinskii-Moriya energy (DMI))

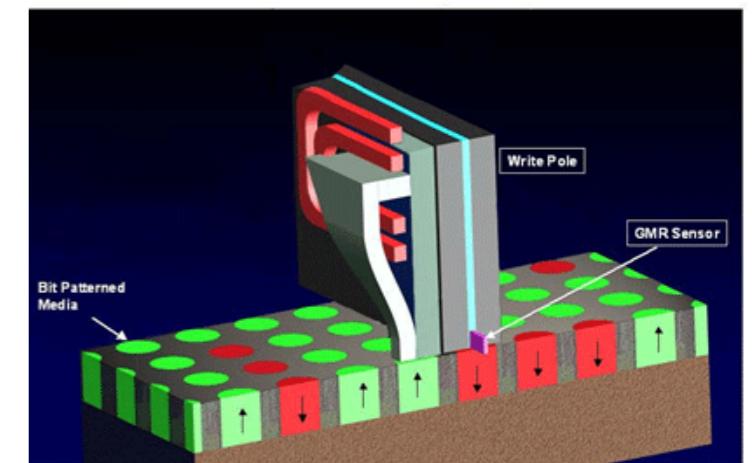


Computational magnetism important

- The number of problems that can be solved analytically is very limited.
- Experimental techniques do not provide enough spatial and temporal resolution.



Parkin, Science, 320, 190 (2008)



Bit-patterned media (Seagate)

Micromagnetic model

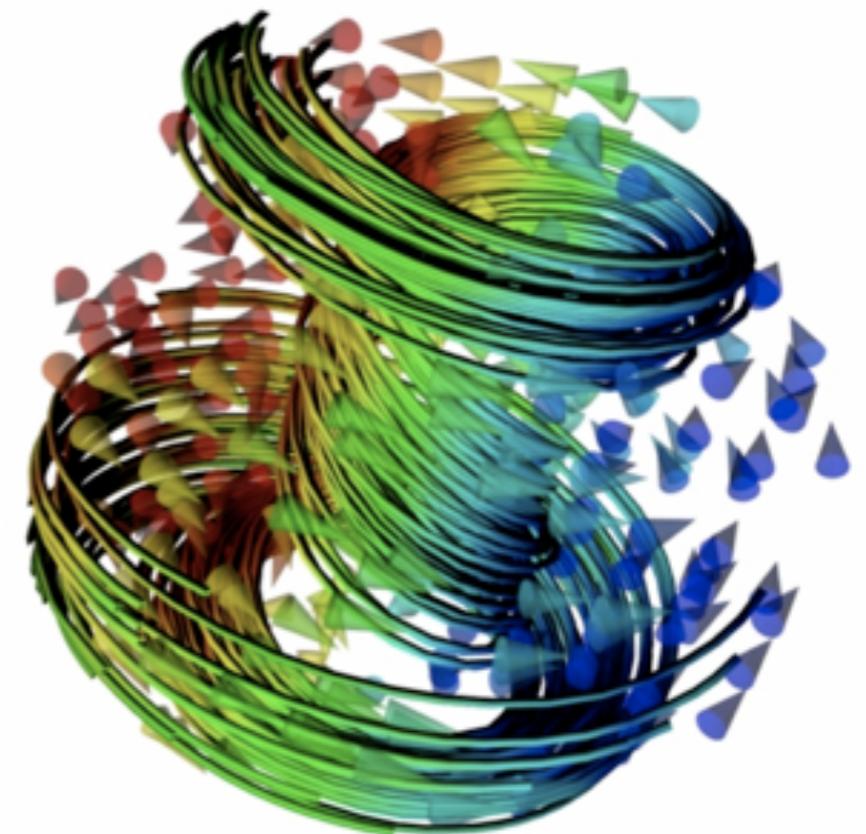
- Coarse graining to go from atoms to continuous magnetisation, known as *micromagnetic model*
- Magnetisation in sample V is described by a continuous vector field $\mathbf{m}(\mathbf{r})$:

$$\mathbf{m} : V \mapsto \mathbb{R}^3 \quad V \subset \mathbb{R}^3$$

- We have an equation of motion

$$\frac{\partial \mathbf{m}}{\partial t} = \mathbf{f}(\mathbf{m})$$

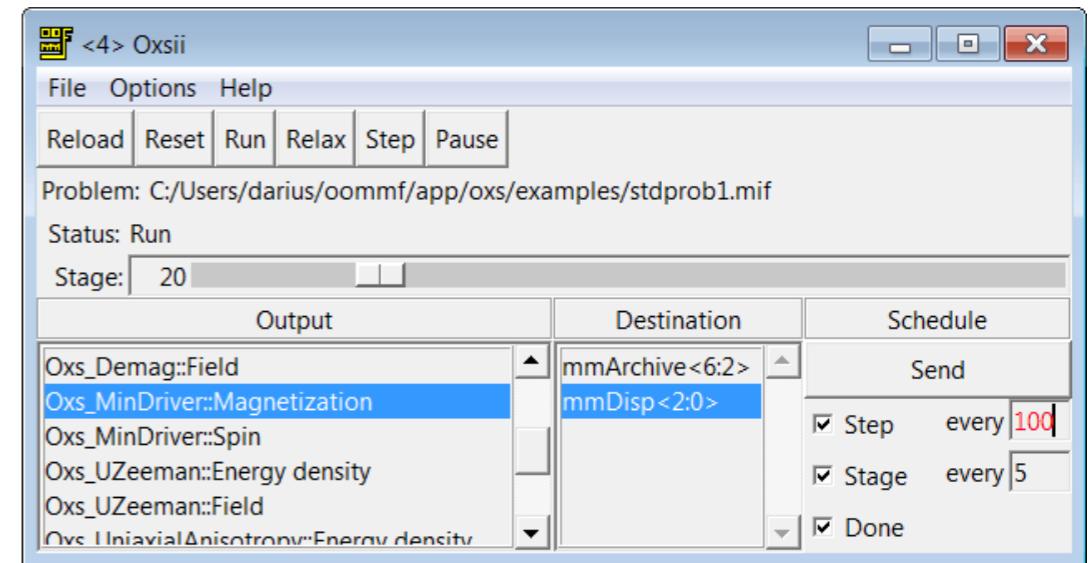
- \mathbf{f} is complicated, involves PDEs



State of the art
micromagnetic simulation
tool

Object Oriented MicroMagnetic Framework (oommf)

- Probably the most widely used simulation tool
- Developed at NIST, USA
- Cited over 2200 times in scientific publications
- Written in C++, some Tcl glue / interface



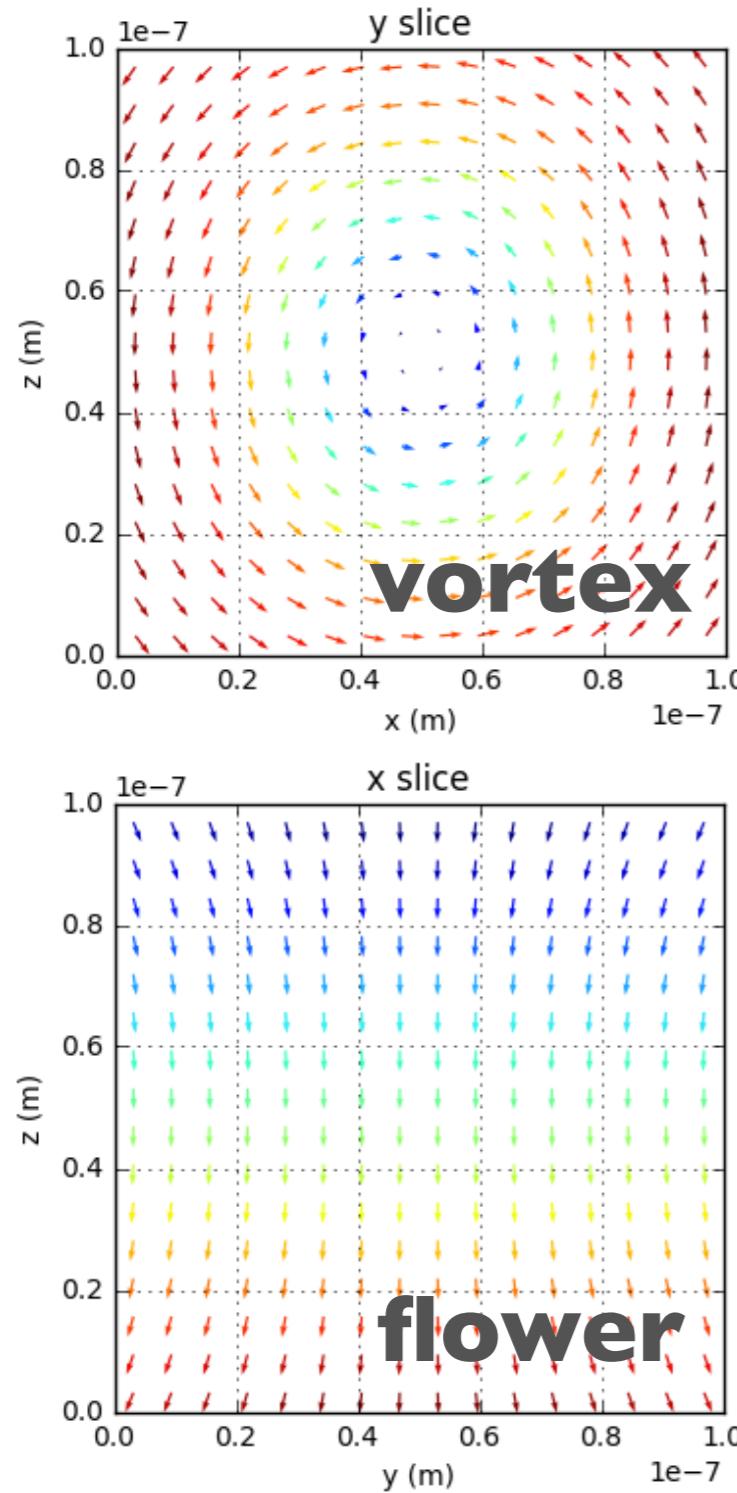
GUI

The screenshot shows a code editor window displaying a Tcl script named stdprob3.mif. The script contains various commands and variables related to micro-magnetic simulations, such as setting auxiliary variables, calculating magnetostatic energy density, defining cube dimensions, and setting exchange coefficients. It also includes a proc definition for CantedVortexInit and coordinate transformations. The code editor has tabs for stdprob3.mif and sublime, and a status bar indicating the current file, Git branch, and line numbers.

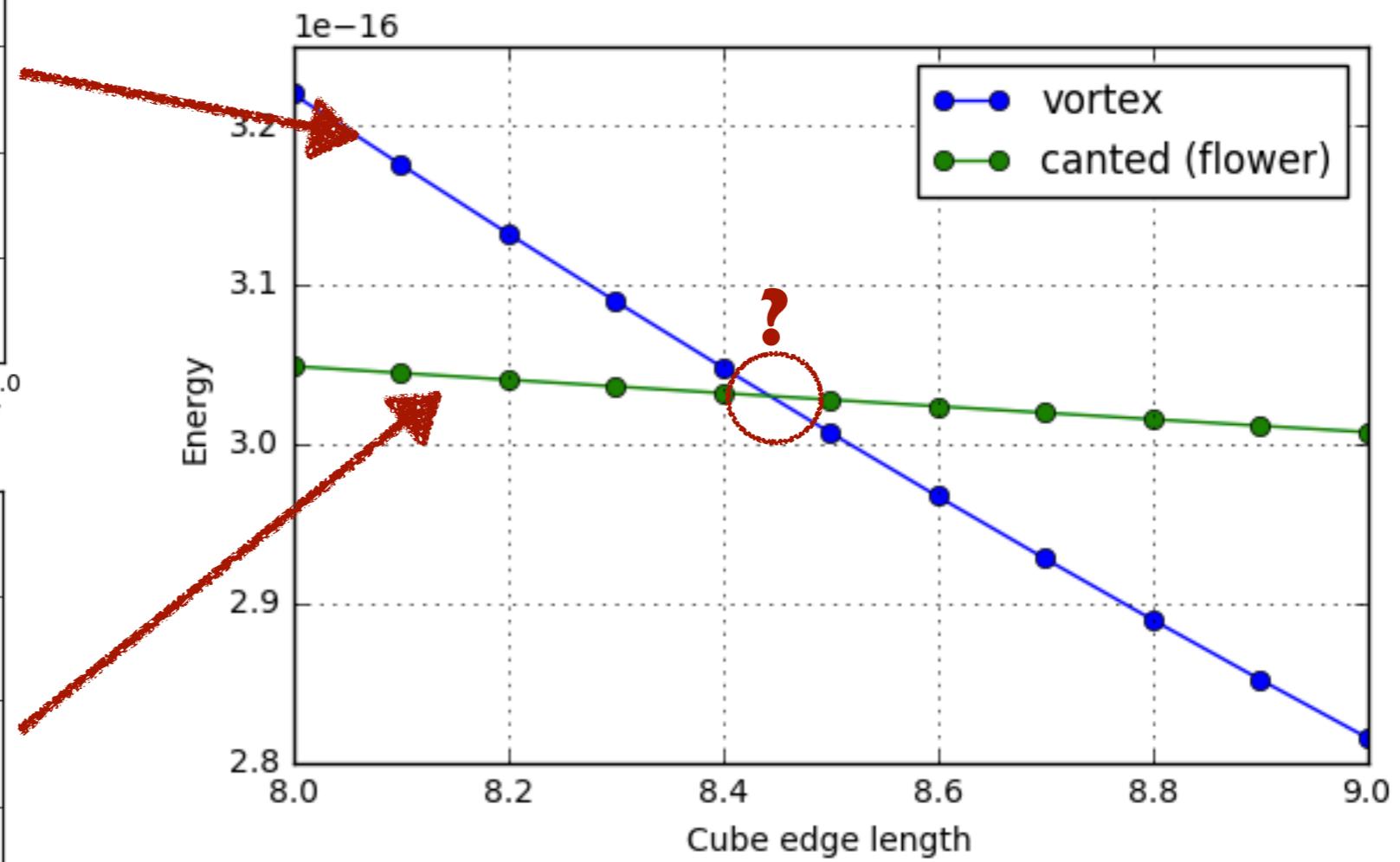
```
26 #####  
27 # Auxiliary variables:  
28  
29 # Work out Ms so magnetostatic energy density, Km=0.5*mu0*Ms^2,  
30 # is 1e6 J/m^3  
31 set Km 1e6  
32 set Ms [expr {sqrt(2*$Km/$mu0)}]  
33  
34 # Arbitrarily set cube dimension to 100 nm, and compute cellsize and  
35 # exchange length based on parameters L and N.  
36 set cubesize 100e-9 ;# Cube dimension in meters  
37 set cellsize [expr {$cubesize/$N}] ;# In meters  
38 set lex [expr {$cubesize/$L}] ;# exchange length  
39  
40 # Set K1 to 0.1*Km  
41 set K1 [expr {$Km/10.}]  
42  
43 # Compute A so that cubesize is requested number of exchange lengths  
44 set A [expr {0.5*$mu0*$Ms*$Ms+$lex*$lex}] ;# Exchange coefficient, J/m  
45  
46 #####  
47 Report "A=$A, K1=$K1, Ms=$Ms, lex=$lex, L=$L, seed=$seed"  
48  
49 #####  
50 # Tcl script for CantedVortex proc  
51 #  
52 #  
53 # Coordinate transform to select initial vortex orientation:  
54 proc CantedVortexInit { vec } {  
55     proc Mag { v } {  
56         set v0 [lindex $v 0]  
57         set v1 [lindex $v 1]  
58         set v2 [lindex $v 2]  
59     }  
60 }
```

Tcl config file

Research work flow example



For what cube edge length
have vortex and flower states the same
energy?



Step 1: write simulation configuration

```
# MIF 2.1
# MIF Example File: stdprob3.mif
# Description: Sample problem description for muMAG Standard Problem #3

set pi [expr {4*atan(1.0)}]
set mu0 [expr {4*$pi*1e-7}]

Parameter seed 0
RandomSeed $seed ;# Initialize seed to {} to get a seed
## value from the system clock.

#####
# Simulation parameters

Parameter L 8 ;# Cube dimension, in units of exchange length
Parameter N 32 ;# Number of cells along one edge of cube

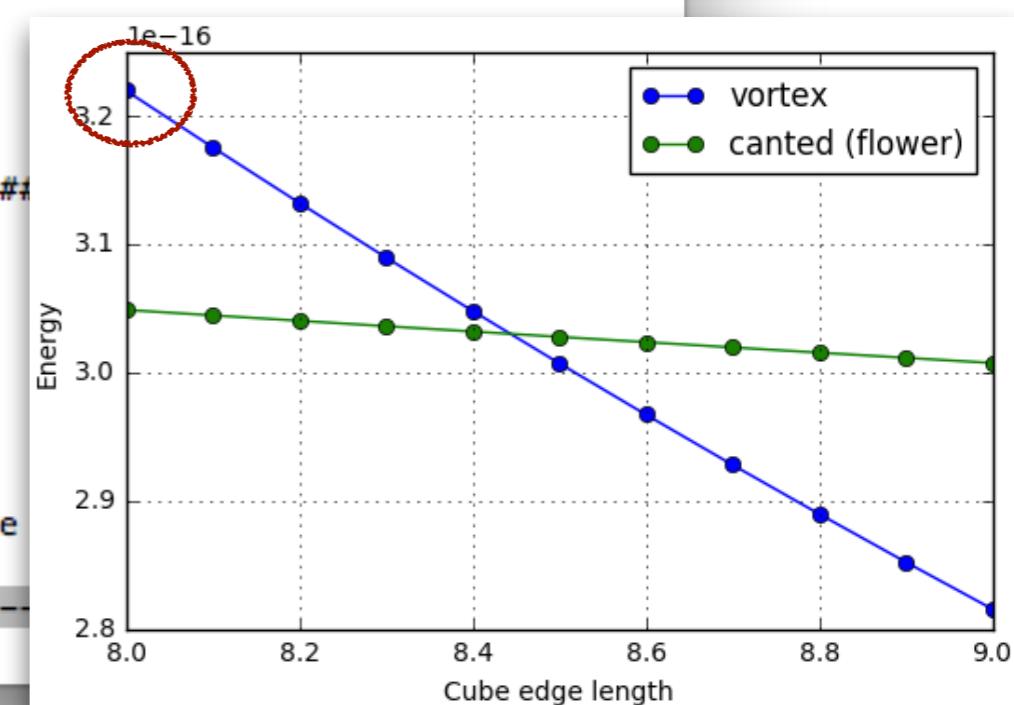
Parameter initial_state "vortex";# Initial state should be
## one of "uniform", "vortex", "cant", "cantvortex", "twisted",
## "random" or "file <filename>"; in the last case <filename> is the
## name of a file to use as the initial configuration.

Parameter stop 1e-3

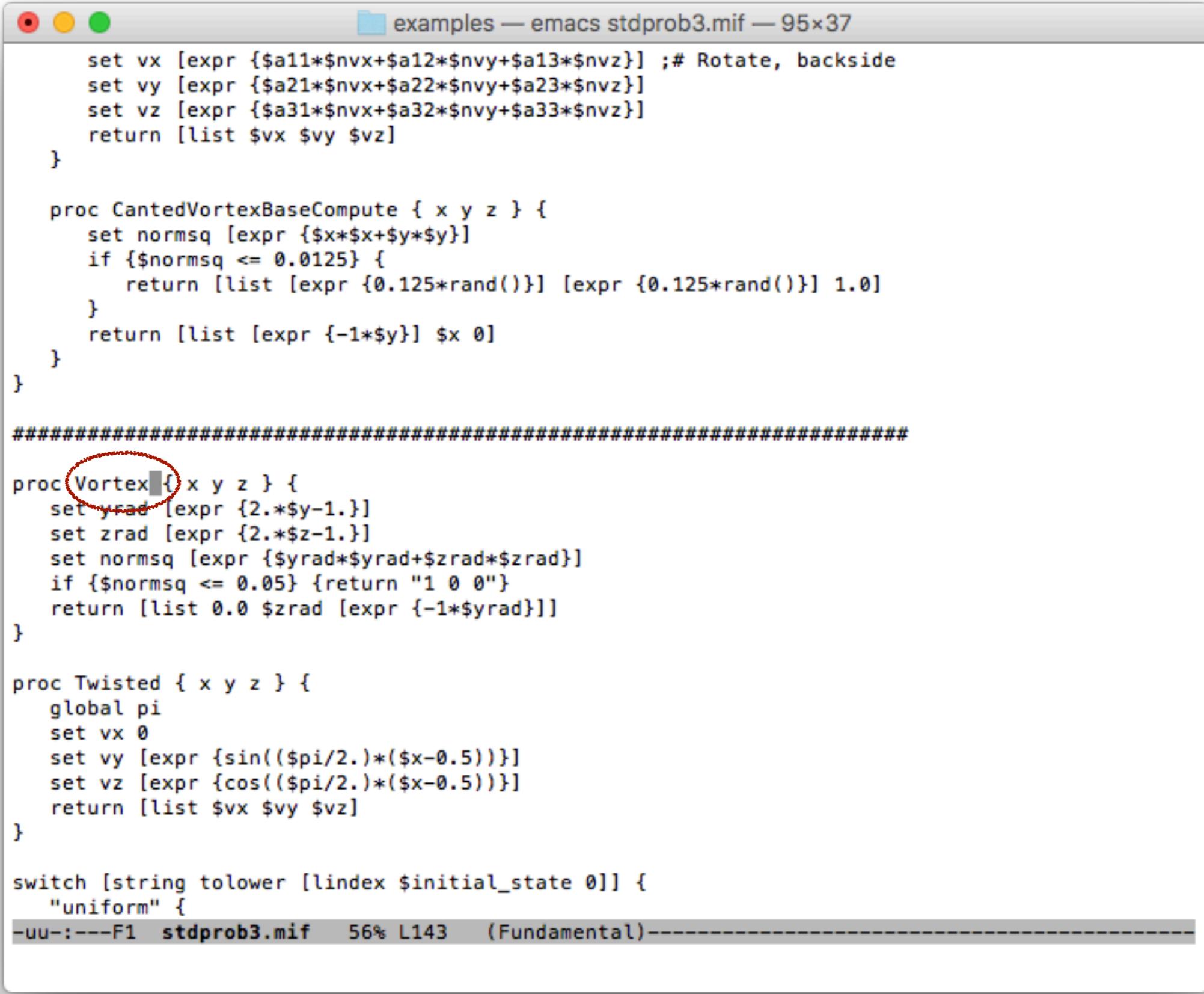
#####
# Auxiliary variables:

# Work out Ms so magnetostatic energy density, Km=0.5*mu0*Ms^2,
# is 1e6 J/m^3
set Km 1e6
set Ms [expr {sqrt(2*$Km/$mu0)}]

# Arbitrarily set cube dimension to 100 nm, and compute cellsize
# exchange length based on parameters L and N.
-uu-:---F1 stdprob3.mif Top L1 (Fundamental)---
```



Step 1: write simulation configuration



The screenshot shows an Emacs window titled "examples — emacs stdprob3.mif — 95x37". The buffer contains a Tcl script with several procedures. A red oval highlights the word "Vortex" in the first procedure definition. The script includes comments and mathematical expressions for vector calculations.

```
set vx [expr {$a11*$nvx+$a12*$nvy+$a13*$nvz}] ;# Rotate, backside
set vy [expr {$a21*$nvx+$a22*$nvy+$a23*$nvz}]
set vz [expr {$a31*$nvx+$a32*$nvy+$a33*$nvz}]
return [list $vx $vy $vz]
}

proc CantedVortexBaseCompute { x y z } {
    set normsq [expr {$x*$x+$y*$y}]
    if {$normsq <= 0.0125} {
        return [list [expr {0.125*rand()}] [expr {0.125*rand()}] 1.0]
    }
    return [list [expr {-1*$y}] $x 0]
}
}

#####
proc Vortex{ x y z } {
    set yrad [expr {2.*$y-1.}]
    set zrad [expr {2.*$z-1.}]
    set normsq [expr {$yrad*$yrad+$zrad*$zrad}]
    if {$normsq <= 0.05} {return "1 0 0"}
    return [list 0.0 $zrad [expr {-1*$yrad}]]
}

proc Twisted { x y z } {
    global pi
    set vx 0
    set vy [expr {\sin((\pi/2.)*($x-0.5))}]
    set vz [expr {\cos((\pi/2.)*($x-0.5))}]
    return [list $vx $vy $vz]
}

switch [string tolower [lindex $initial_state 0]] {
    "uniform" {
-uu-:---F1  stdprob3.mif  56% L143  (Fundamental)-----
```

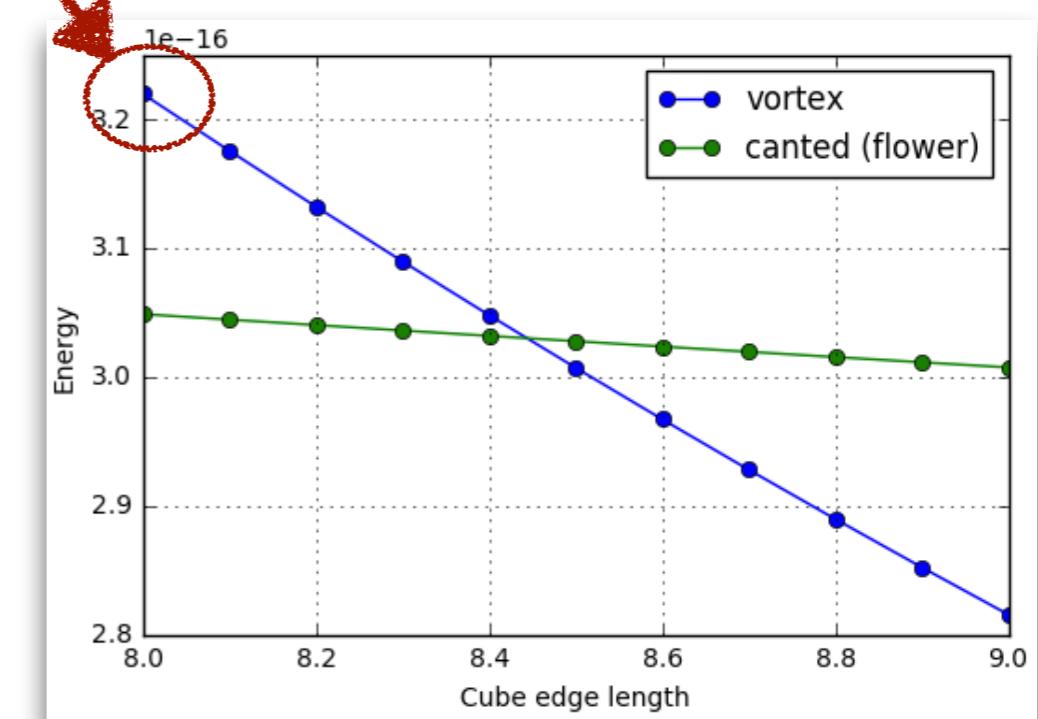
Step 2: run simulation

```
[Marijans-MBP:my_project mb4e10$ ls  
stdprob3.mif  
[Marijans-MBP:my_project mb4e10$ tclsh $00MMFTCL boxsi +fg stdprob3.mif -exitondone 1  
Start: "/Users(mb4e10/my_project/stdprob3.mif"  
Options: -exitondone 1 -threads 2  
Boxsi version 1.2.1.0  
Running on: marijans-macbook-pro.local  
OS/machine: Darwin/x86_64  
User: mb4e10 PID: 72176  
Number of threads: 2  
Mesh geometry: 32 x 32 x 32 = 32 768 cells  
Checkpoint file: /Users(mb4e10/my_project/sp3-vortex-seed0000.restart  
Boxsi run end.  
[Marijans-MBP:my_project mb4e10$ ls  
sp3-vortex-seed0000.odt stdprob3.mif  
Marijans-MBP:my_project mb4e10$ ]
```

Step 3: extract data from output file

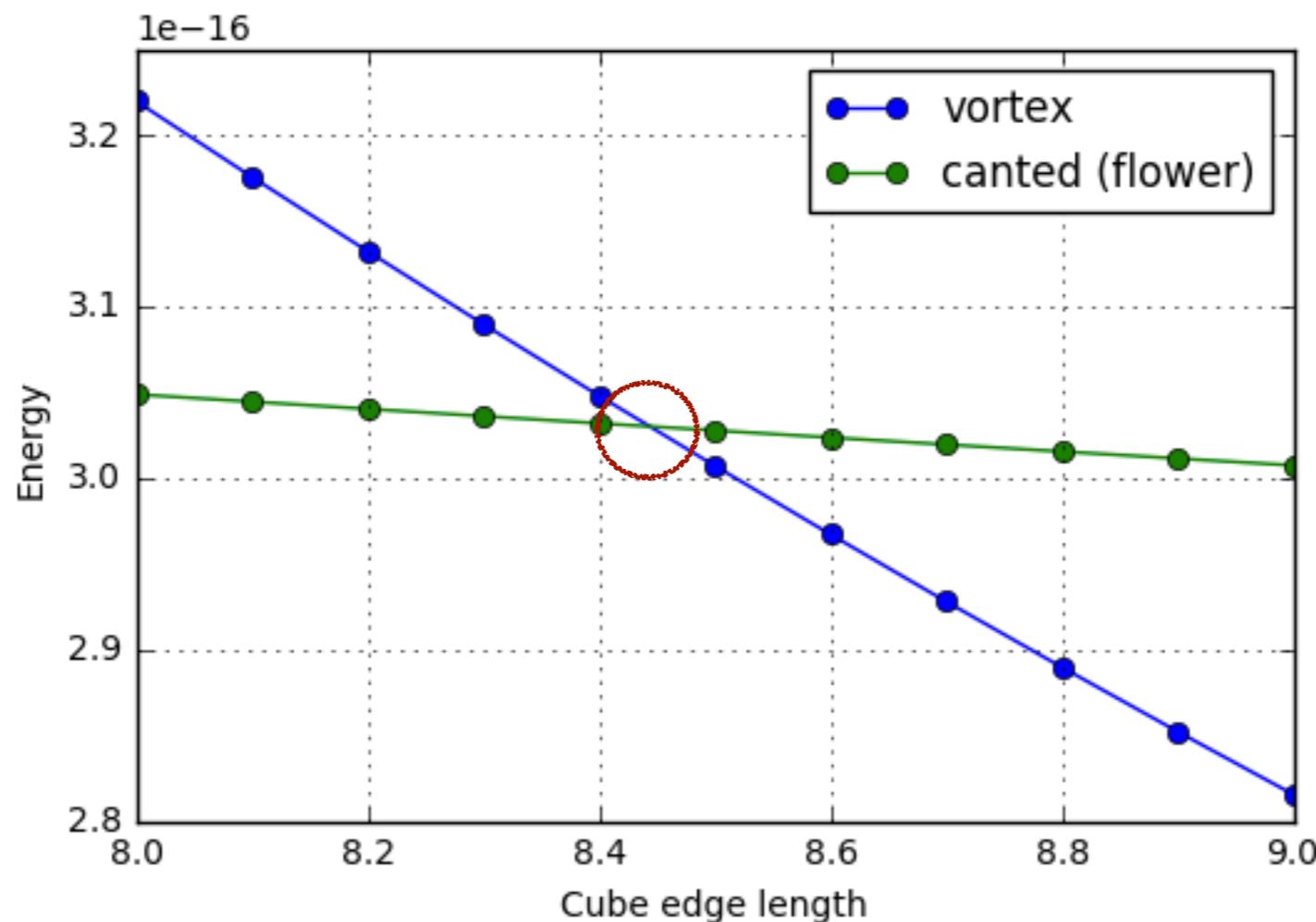
Step 4: gather data, and repeat simulations...

L	flower	vortex
8.0	?	3.23×10^{-16}
8.1	?	?
8.2	?	?
8.3	?	?
8.4	?	?
8.5	?	?
8.6	?	?
8.7	?	?
8.8	?	?
8.9	?	?
9.0	?	?



Postprocessing

- We plot the data we obtained by running separate plotting scripts or by using some Graphical User Interfaces (Python, MATLAB, Excel, Origin...)



- Find crossing (here at ~ 8.45).

Issues with (oommf) workflow

- Time consuming
- Extracting data is repetitive, manual process
(or requires bash scripting)
- Separate post processing and plotting scripts
- Reproducibility?

Jupyter OOMMF

JOOMMF

- Jupyter + OOMMF = JOOMMF
- Micromagnetic Virtual Research Environment
- Enable running OOMMF simulations in Jupyter notebook (through Python interface to OOMMF)

Research example (repeated) with Jupyter OOMMF

[Live demo in Notebook]

Benefits

- The entire workflow is contained in a single document, including computation, post processing and visualisation
- Self documenting & reproducible
- Easy to share, publish, and collaborate
- Later: Executable tutorial for OOMMF Software

Micromagnetic model integration in VRE

[Live demo in Notebook]

Summary

- Virtual Research Environment allows us to have documentation (text and equations), code, code outputs (text, figures, tables) in a single file
- Python interface [I] supports component-based approach: can combine OOMMF with the tools from Python ecosystem
- Integration into Jupyter Notebook allows to use Scientist's language
- Improved effectiveness and reproducibility
- Future work on visualisation, data analysis, and cloud hosting

[I] OOMMF Python interface and embedded Domain Specific Language: Beg *et al.*, AIP Advances Advances **7**, 056025, (2017) arXiv 1609.07432

Acknowledgements

- Financial support from
 - OpenDreamKit Horizon 2020, European Research Infrastructures project (#676541), <http://opendreamkit.org>
- Further contributions from
 - EPSRC's Centre for Doctoral Training in Next Generation Computational Modelling, <http://ngcm.soton.ac.uk> (#EP/L015382/1) and
 - The Gordon and Betty Moore Foundation through Grant GBMF #4856, by the Alfred P. Sloan Foundation and by the Helmsley Trust.

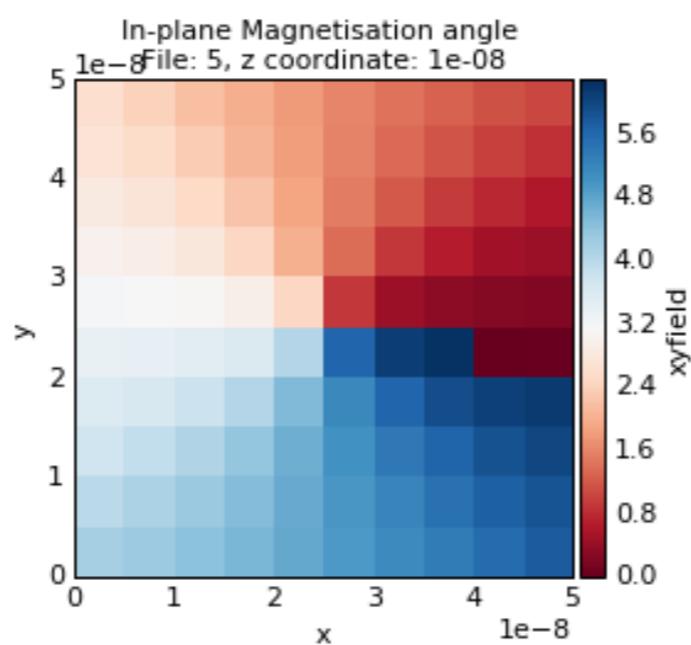
Slides end here

- Remaining slides are for questions only and are likely not to be used

Outlook

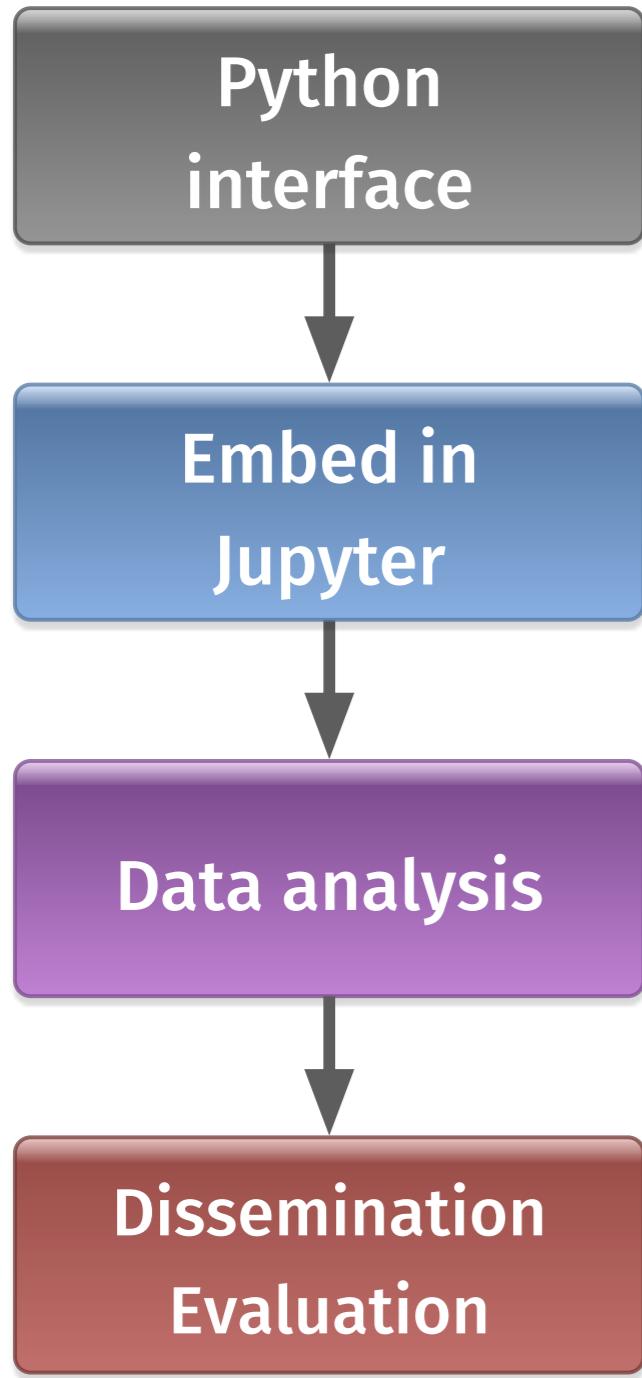
```
In [16]: #PYTEST_VALIDATE_IGNORE_OUTPUT  
inplane = joommftools.create_inplane_holomap(files[:10], slicecoords, 'z')  
inplane
```

Out[16]:



File:	<input type="text" value="5.0"/> 
z coordinate:	<input type="text" value="1e-08"/> 

Road map JOOMMF



OOMMF	T3.8 - Python
Latex representation	
Matplotlib	T4.11 - Jupyter
Pandas	
Holoviews	T4.8 - 3d vis
Widgets	T4.14 - cloud
interactive 3d vis	T4.13 - interactive doc
	T2.8 - workshops
	T7.4 - evaluation

Micromagnetic computational problem

- semi-discretised: PDE in every timestep, coupled set of ODEs to integrate time
- Dynamics of magnetisation is defined by a stiff set of equations (different timescales)
- Different length scales (exchange vs demagnetisation)
- “More difficult than CFD”

PDEs for micromagnetics

- Magnetic scalar potential and demagnetisation field

$$\phi(\mathbf{r}') = \frac{M_s}{4\pi} \left(\underbrace{- \int_{\Omega} \frac{\nabla \cdot \mathbf{m}(\mathbf{r}')}{\|\mathbf{r} - \mathbf{r}'\|} dV}_{\text{volume term}} + \int_{\partial\Omega} \frac{\mathbf{m}(\mathbf{r}') \cdot \mathbf{n}}{\|\mathbf{r} - \mathbf{r}'\|} dS \right)$$

surface term

$$\mathbf{H}_d(\mathbf{r}) = -\nabla\phi(\mathbf{r})$$

- Effective field

$$H_{\text{eff}}(\mathbf{m}) = \underbrace{\frac{2A}{\mu_0 M_s} \nabla^2 \mathbf{m}}_{\text{exchange}} - \underbrace{\frac{2D}{\mu_0 M_s} (\nabla \times \mathbf{m})}_{\text{DMI}} + \underbrace{\mathbf{H}}_{\text{Zeeman}} + \underbrace{\mathbf{H}_d}_{\text{demagnetisation}}$$

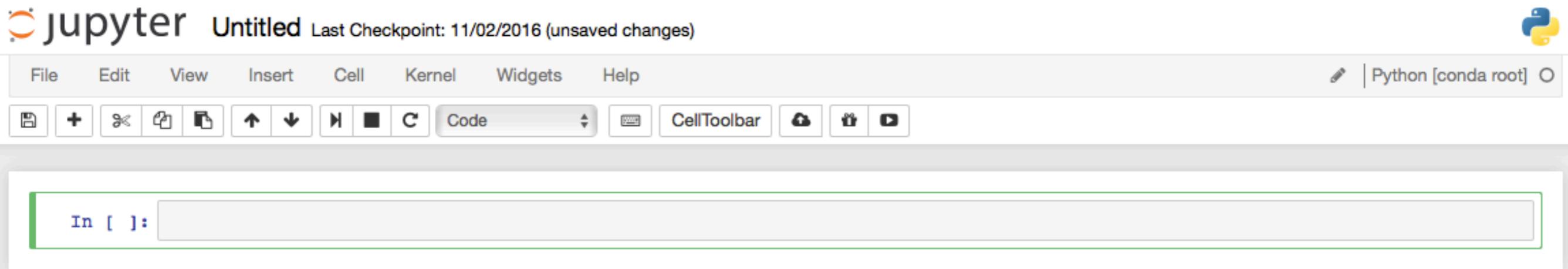
- LLG equation

$$\frac{\partial \mathbf{m}}{\partial t} = \underbrace{\gamma^* \mathbf{m} \times \mathbf{H}_{\text{eff}}}_{\text{precession}} + \underbrace{\alpha \mathbf{m} \times \frac{\partial \mathbf{m}}{\partial t}}_{\text{damping}} + u(|\mathbf{m}| - 1) \underbrace{\frac{\mathbf{m}}{|\mathbf{m}|}}_{\text{norm correction}}$$

OOMMF-Python communication

- The core OOMMF computational routines are implemented in C++, higher level functionality implemented in Tcl
- Possible solutions for interfacing Tcl and C++ with Python include swig, boost, cython
- We decided communicating with OOMMF via MIF (configuration) files:
 - This provides us robustness since we do not need to adapt the interface depending on the platform.
 - Re-use of all functionality in Tc
 - Easier to debug, even for end users

jupyter notebook

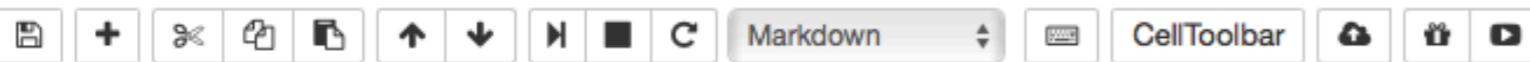


text and equations

jupyter vre_workflow Last Checkpoint: 11/02/2016 (autosaved)



File Edit View Insert Cell Kernel Widgets Help | Python [conda root] O



JOOMMF workflow in Virtual Research Environment

Problem specification

This problem is to calculate a single domain limit of a cubic magnetic particle. This is the size L of equal energy for the so-called flower state (which one may also call a splayed state or a modified single-domain state) on the one hand, and the vortex or curling state on the other hand.

Geometry:

A cube with edge length, L , expressed in units of the intrinsic length scale, $l_{\text{ex}} = \sqrt{A/K_m}$, where K_m is a magnetostatic energy density, $K_m = \frac{1}{2} \mu_0 M_s^2$.

Material parameters:

- uniaxial anisotropy K_u with $K_u = 0.1 K_m$, and with the easy axis directed parallel to a principal axis of the cube $(0, 0, 1)$,
- exchange energy constant is $A = \frac{1}{2} \mu_0 M_s^2 l_{\text{ex}}^2$.

More details about the standard problem 3 can be found in Ref. 1.

text and equations

jupyter vre_workflow Last Checkpoint: 11/02/2016 (unsaved changes) Python [conda root]

File Edit View Insert Cell Kernel Widgets Help

CellToolbar

JOOMMF workflow in Virtual Research Environment

Problem specification

This problem is to calculate a single domain limit of a cubic magnetic particle. This is the size L of equal energy for the so-called flower state (which one may also call a splayed state or a modified single-domain state) on the one hand, and the vortex or curling state on the other hand.

Geometry:

A cube with edge length, L , expressed in units of the intrinsic length scale, $l_{\text{ex}} = \sqrt{A/K_m}$, where K_m is a magnetostatic energy density, $K_m = \frac{1}{2}\mu_0 M_s^2$.

Material parameters:

- uniaxial anisotropy K_u with $K_u = 0.1K_m$, and with the easy axis directed parallel to a principal axis of the cube (0, 0, 1),
- exchange energy constant is $A = \frac{1}{2}\mu_0 M_s^2 l_{\text{ex}}^2$.

More details about the standard problem 3 can be found in Ref. 1.

code

jupyter standard_problem3 Last Checkpoint: 11/02/2016 (unsaved changes) Python [default] O

File Edit View Insert Cell Kernel Widgets Help

Code CellToolbar

• exchange energy constant is $A = \frac{1}{2}\mu_0 M_s^2 l_{ex}^2$.

More details about the standard problem 3 can be found in Ref. 1.

Simulation

Firstly, we import all necessary modules.

```
In [1]: import discretisedfield as df  
import oommfc as oc
```

The following two functions are used for initialising the system's magnetisation [1].

```
In [2]: import numpy as np  
  
# Function for initiaising the flower state.  
def m_init_flower(pos):  
    x, y, z = pos[0]/1e-9, pos[1]/1e-9, pos[2]/1e-9  
    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:
```

visualisation

jupyter standard_problem3 Last Checkpoint: 11/02/2016 (unsaved changes) 

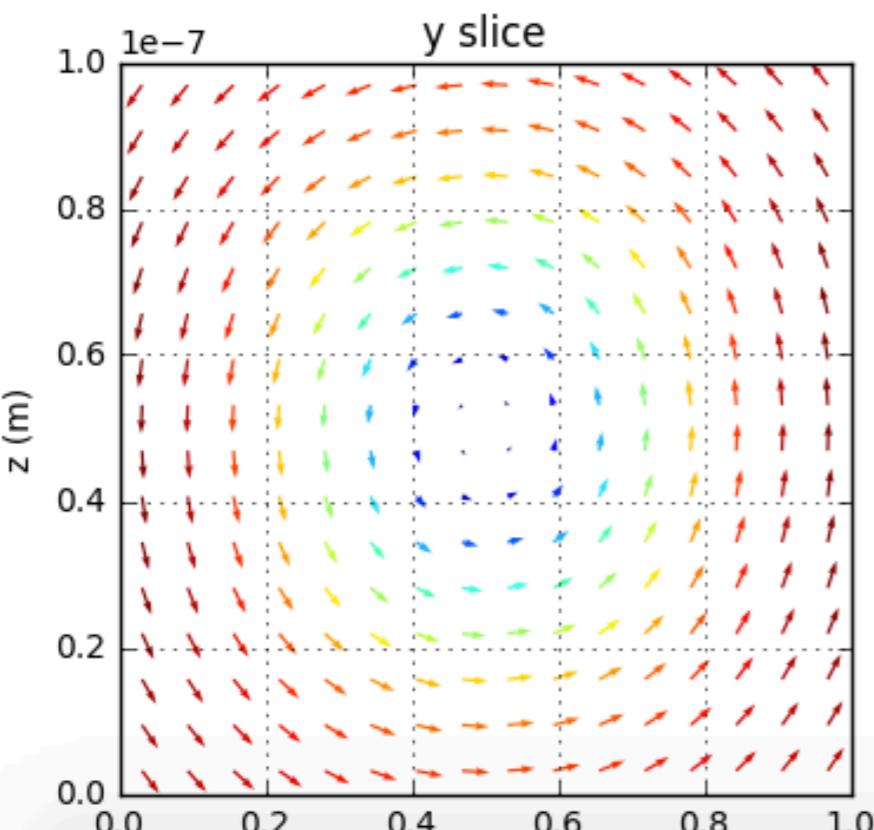
File Edit View Insert Cell Kernel Widgets Help | Python [default] O

Relaxed magnetisation states

Now, we show the magnetisation configurations of two relaxed states.

Vortex state:

```
In [11]: %matplotlib inline
system = minimise_system_energy(8, m_init_vortex)
fig = system.m.plot_slice('y', 50e-9, xscale=4)
```



multiple simulation runs

jupyter standard_problem3 Last Checkpoint: 11/02/2016 (unsaved changes) 

File Edit View Insert Cell Kernel Widgets Help  Python [default] 

Energy crossing

Now, we can plot the energies of both vortex and flower states as a function of cube edge length. This will give us an idea where the state transition occurs.

```
In [13]: L_array = np.linspace(8, 9, 11) # values of L for which the system is relaxed.

vortex_energies = []
flower_energies = []

for L in L_array:
    vortex = minimise_system_energy(L, m_init_vortex)
    flower = minimise_system_energy(L, m_init_flower)

    vortex_energies.append(vortex.total_energy())
    flower_energies.append(flower.total_energy())

# Plot the energy dependences.
import matplotlib.pyplot as plt
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([8.0, 9.0])
plt.grid()
plt.legend()
```

plotting

jupyter standard_problem3 Last Checkpoint: 11/02/2016 (autosaved)

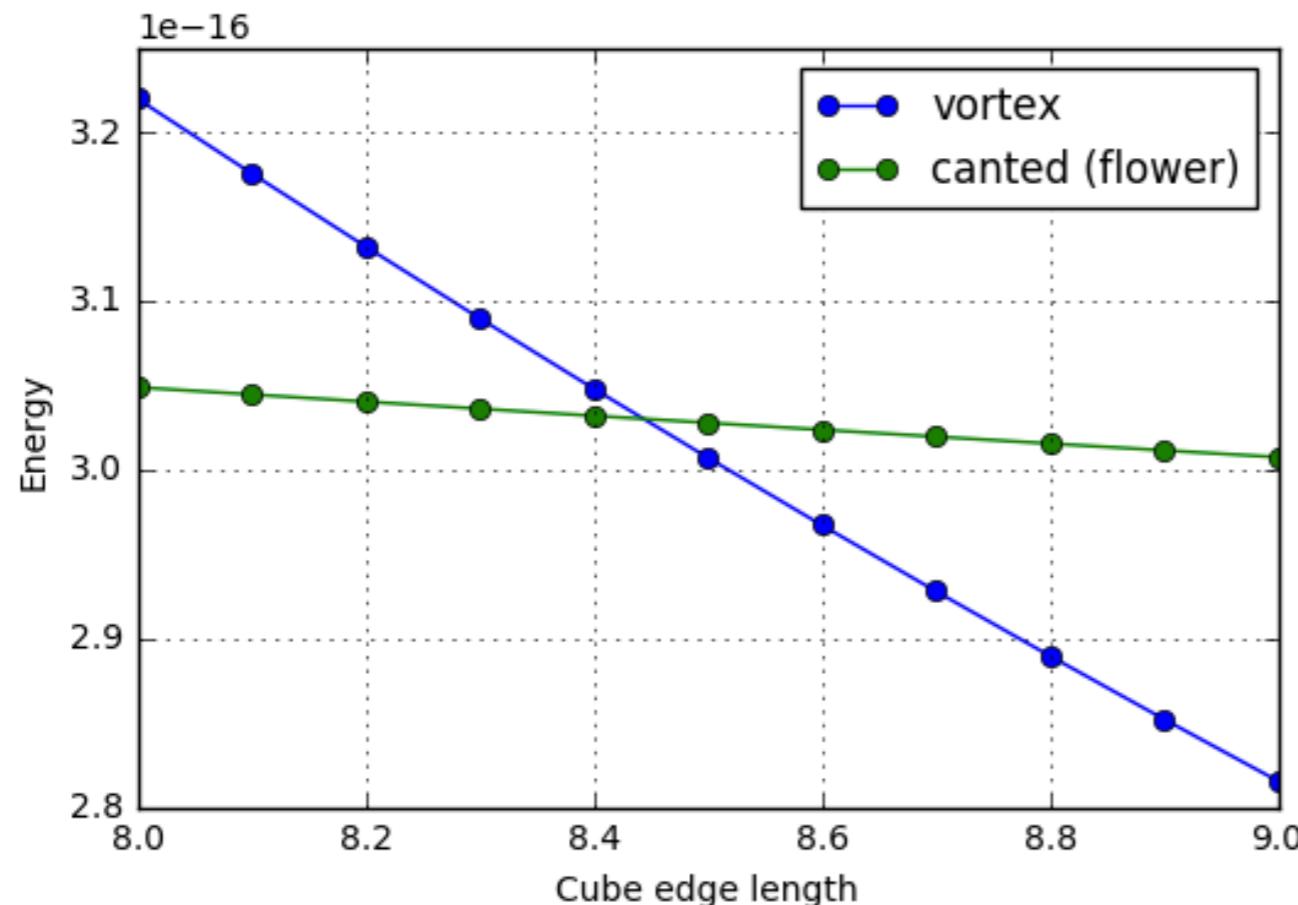
File Edit View Insert Cell Kernel Widgets Help

Python [default] C



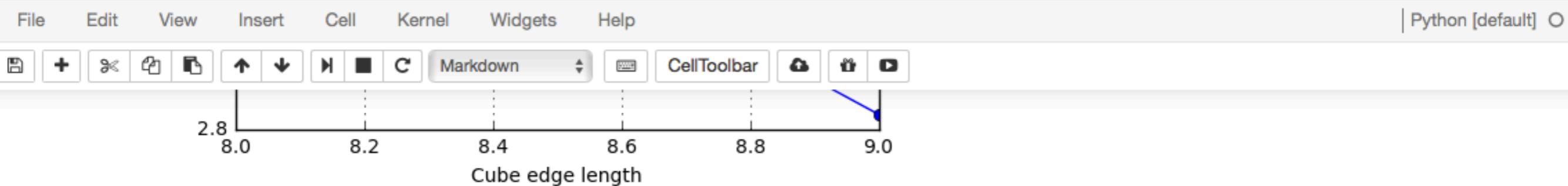
```
plt.ylim()
plt.legend()
```

Out[8]: <matplotlib.legend.Legend at 0x112caab70>



energy crossing

jupyter standard_problem3 Last Checkpoint: 11/02/2016 (autosaved)



We now know that the energy crossing occurs between $8l_{\text{ex}}$ and $9l_{\text{ex}}$, so a bisection algorithm can be used to find the exact crossing.

```
In [9]: from scipy.optimize import bisect

def energy_difference(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, 9, xtol=0.1)

print("The transition between vortex and flower states occurs at {}*lex".format(cross_section))
```

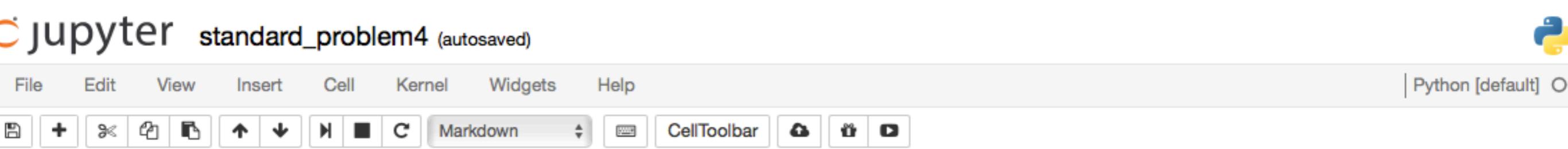
The transition between vortex and flower states occurs at 8.4375*lex

References

[1] μMAG Site Directory <http://www.ctcms.nist.gov/~rdm/mumag.org.html>

Tables

jupyter standard_problem4 (autosaved)



Postprocessing

When we drove the system using the `TimeDriver`, we specified that we want to save the magnetisation configuration $n = 200$ times. A detailed table of all computed parameters from the last simulation can be shown from the datatable (`system.dt`), which is a pandas dataframe [2].

For instance, if we want to show the last 10 rows in the table, we run:

In [16]: `system.dt.tail(5)`

Out[16]:

	E	Ecount	max_dm/dt	dE/dt	deltaE	Eex	max_spin_angle	stage_max_spin_angle	run_max_spin_angle	Ed
195	-2.676805e-18	3802.0	1168.558002	-1.701626e-09	-2.292409e-21	9.509917e-20	3.988227	4.335480	29.984064	8.53911 19
196	-2.685941e-18	3821.0	1264.690715	-1.936455e-09	-2.633442e-21	8.603004e-20	4.234452	4.234452	29.984064	8.85139 19
197	-2.695973e-18	3840.0	1385.550409	-2.055475e-09	-2.820054e-21	8.010709e-20	4.782664	4.782664	29.984064	9.07919 19
198	-2.706283e-18	3859.0	1350.603218	-2.048108e-09	-2.831441e-21	7.558538e-20	4.688752	4.815510	29.984064	9.22294 19
199	-2.716260e-18	3878.0	1189.283501	-1.925038e-09	-2.680093e-21	7.113525e-20	4.377352	4.688752	29.984064	9.29227 19