A case study of computational science in Jupyter notebooks: JOOMMF

Demonstrator introduction - Hans Fangohr

University of Southampton

Overview

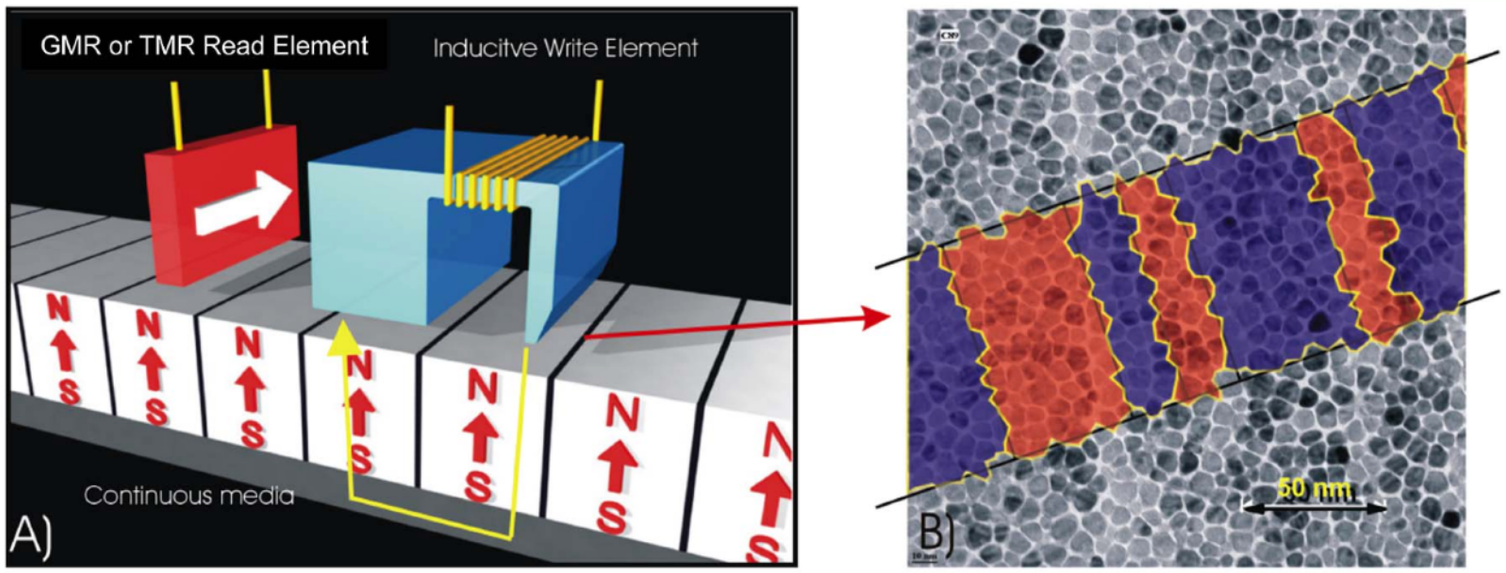
- What is micromagnetics?
- State-of-the-art micromagnetics simulation tool
- Beyond state-of-the art: Jupyter OOMMF
  (Object Oriented MicroMagnetic Framework)
- Outlook

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

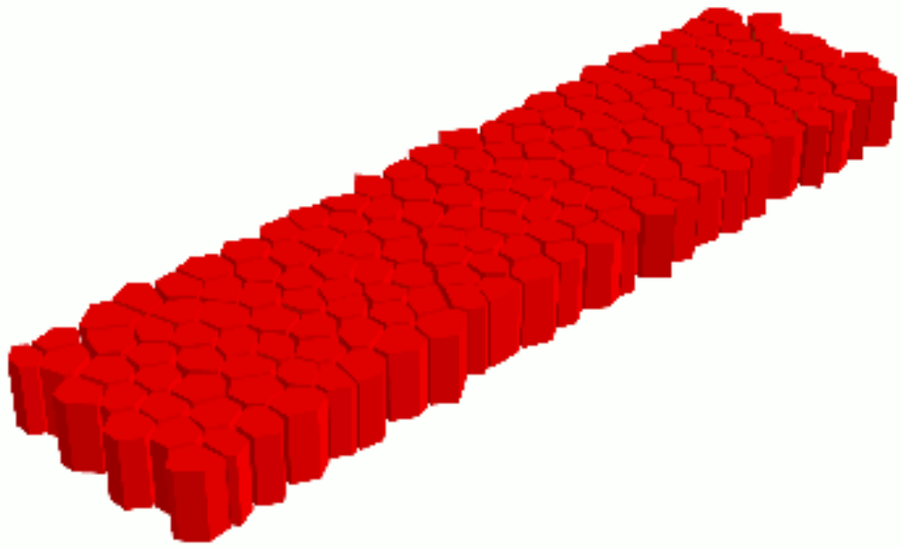
Why magnetic nanostructures?

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)

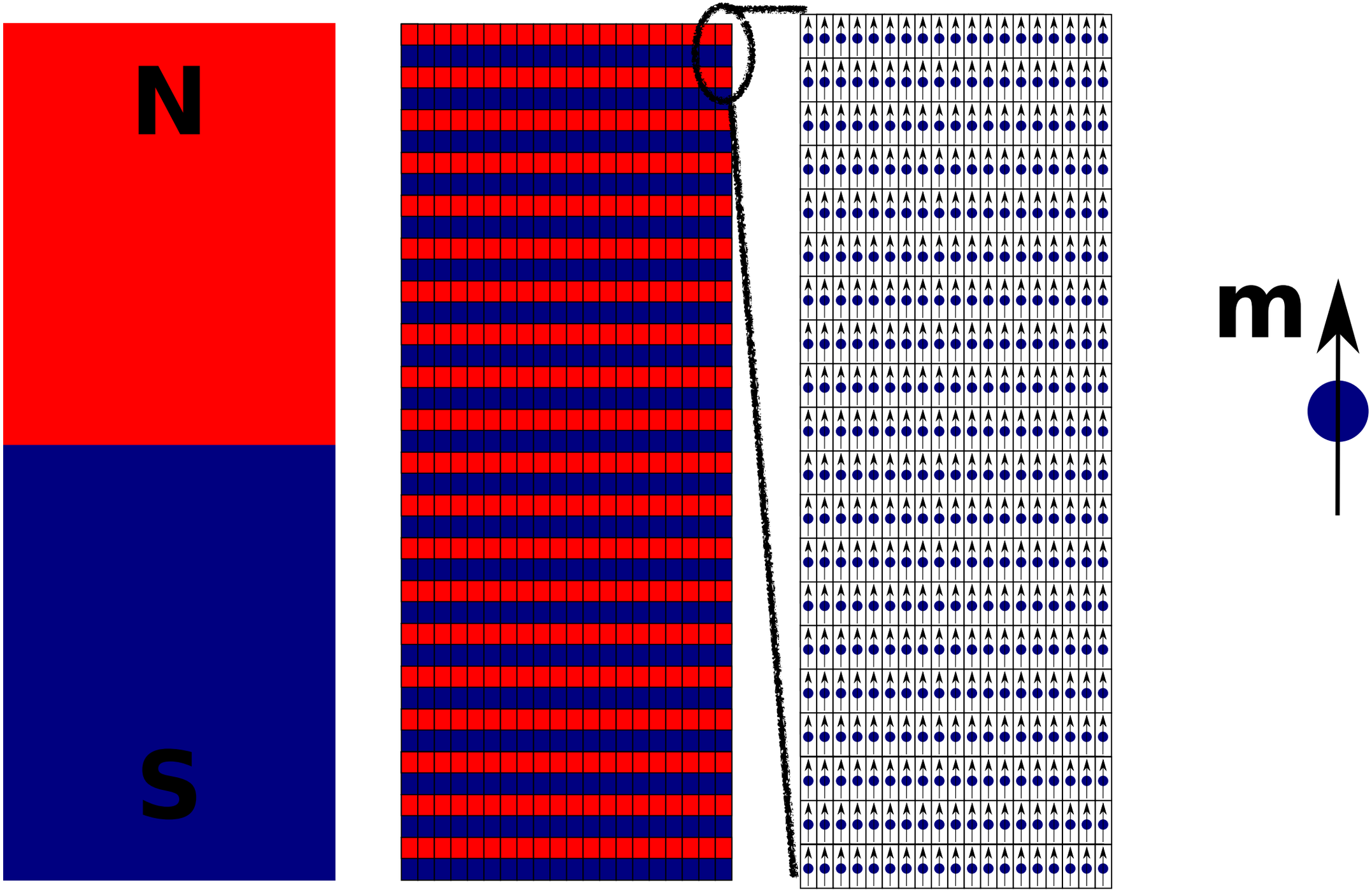


E. Dobisz et. al., Proceedings of IEEE 96, 1836 (2008)



Curtis & Fangohr (2011)

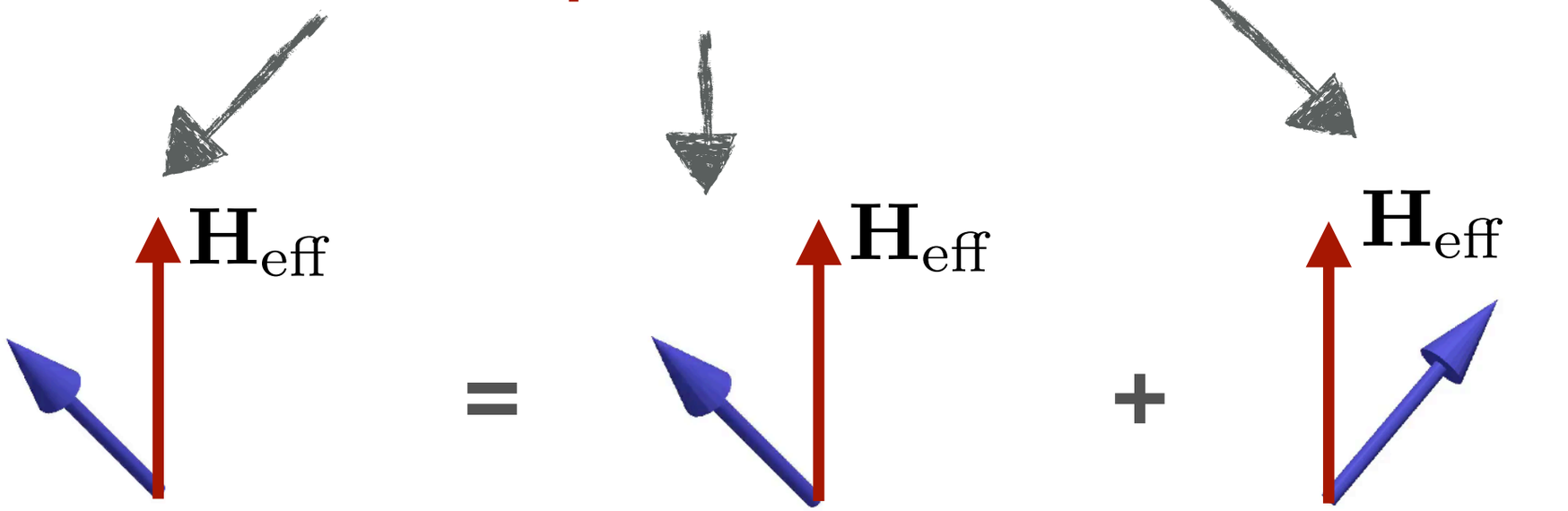
Magnetic moment

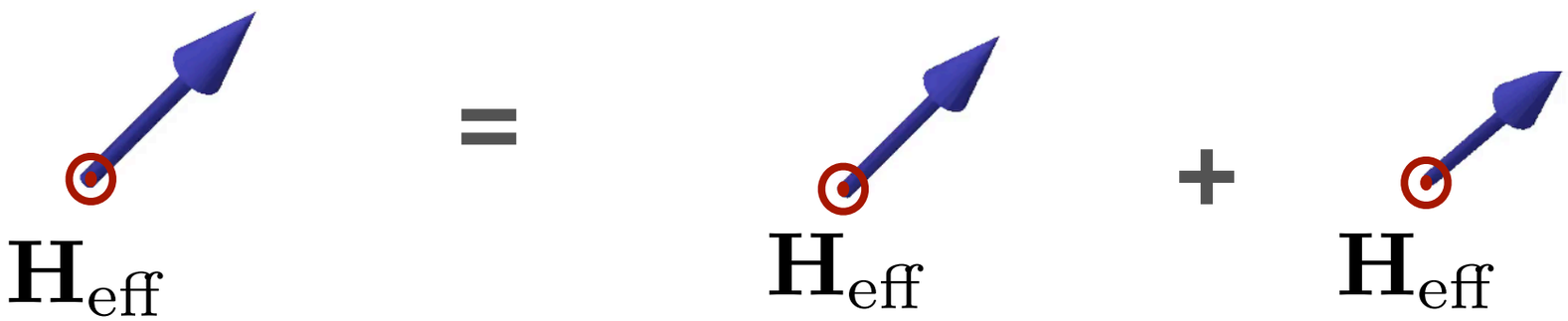


Magnetisation dynamics

- Landau-Lifshitz-Gilbert (LLG) equation

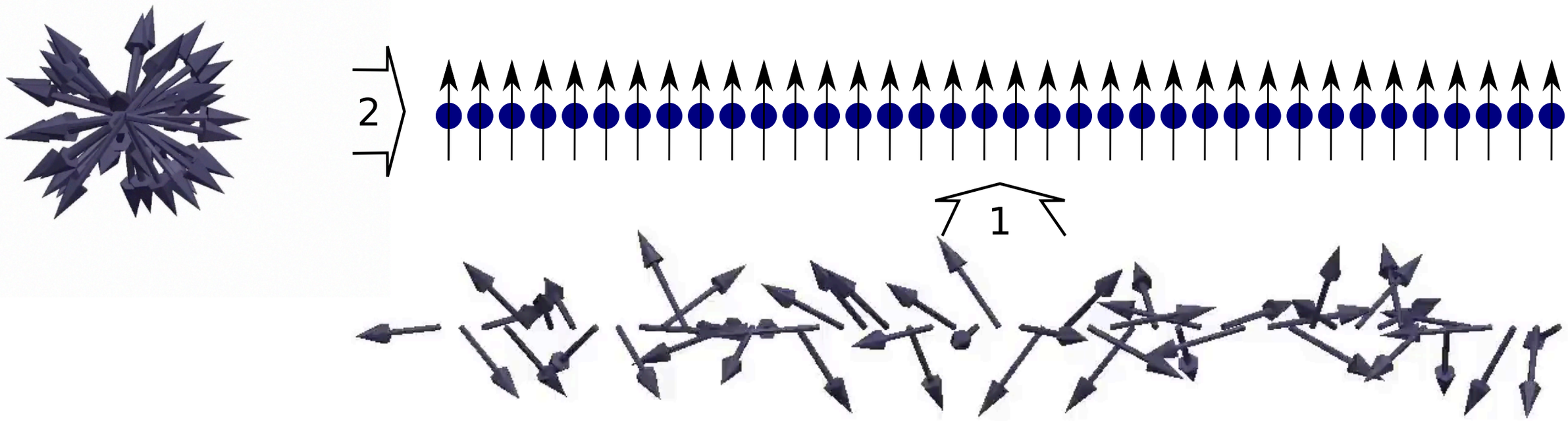
$$\frac{\partial \mathbf{m}}{\partial t} = \underbrace{\gamma^* \mathbf{m} \times \mathbf{H}_{\text{eff}}}_{\textbf{precession}} + \overbrace{\alpha \mathbf{m} \times \frac{\partial \mathbf{m}}{\partial t}}^{\textbf{damping}}$$



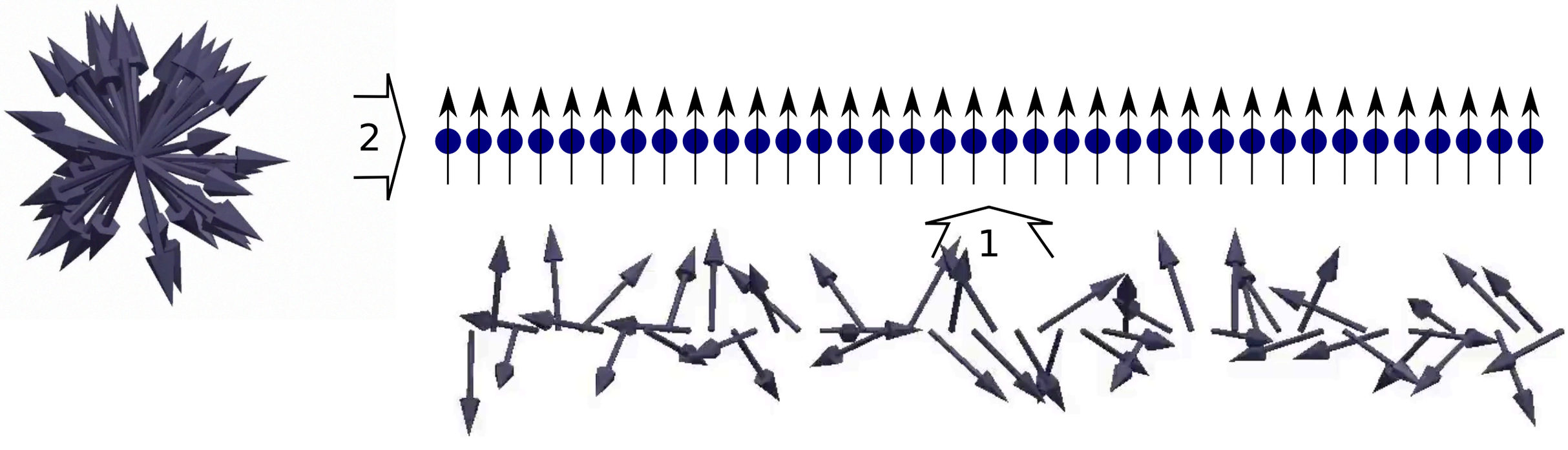


Different types of physics

A: Align the magnetic moment to an external field

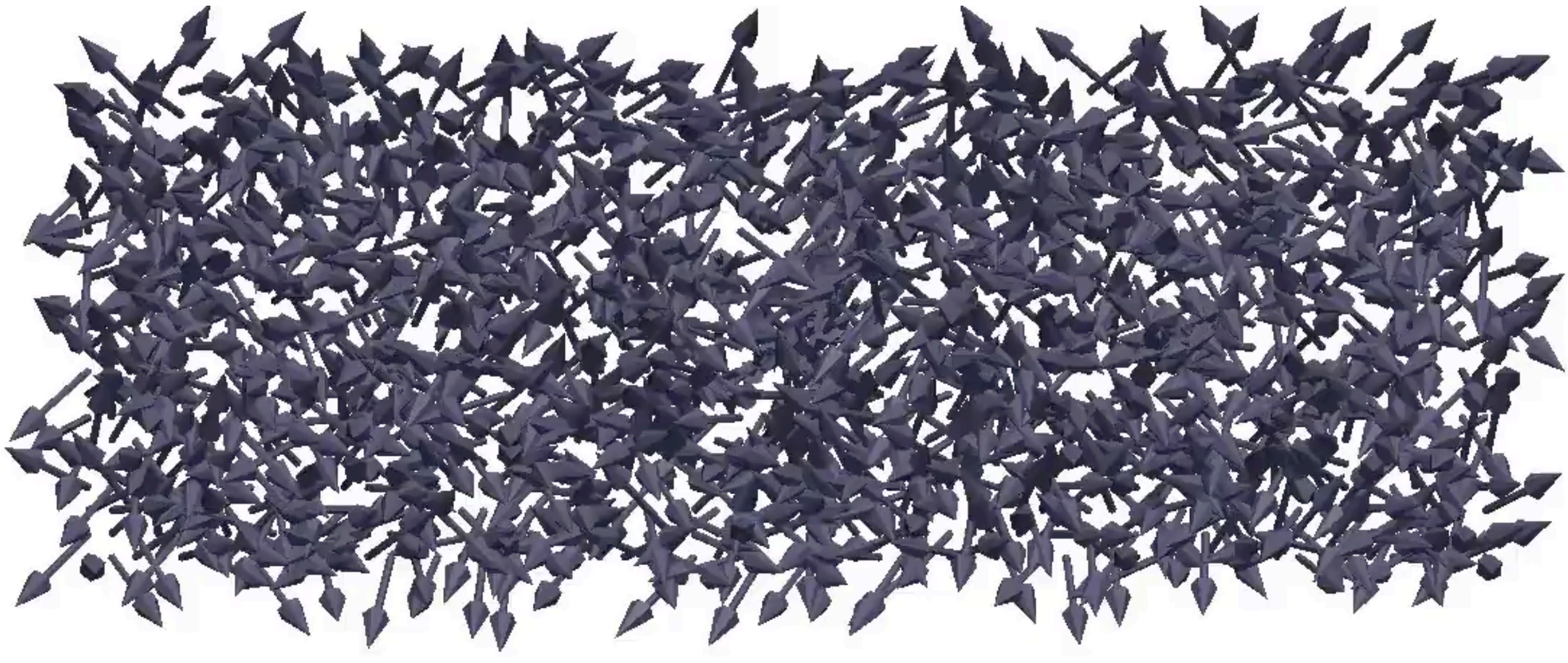


B: Align all magnetic moments to be parallel (short-range)



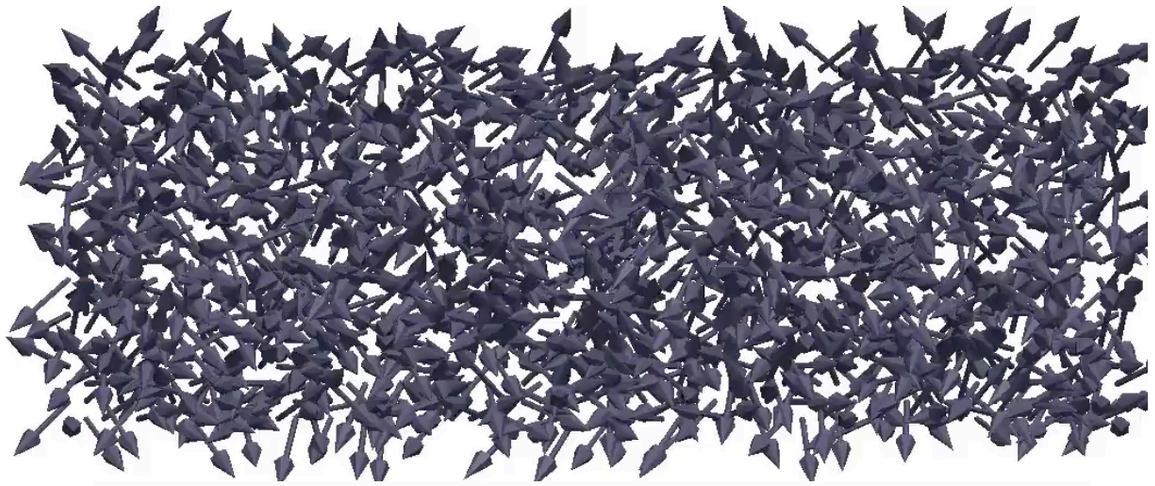
More complicated case

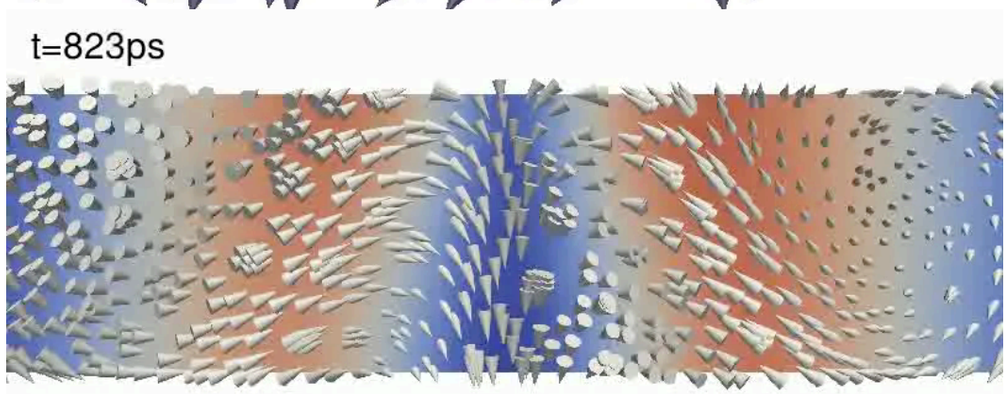
- Two-dimensional sample.
- Four interactions included (Exchange, Zeeman, Anisotropy, Dzyaloshinskii-Moriya energy (DMI))

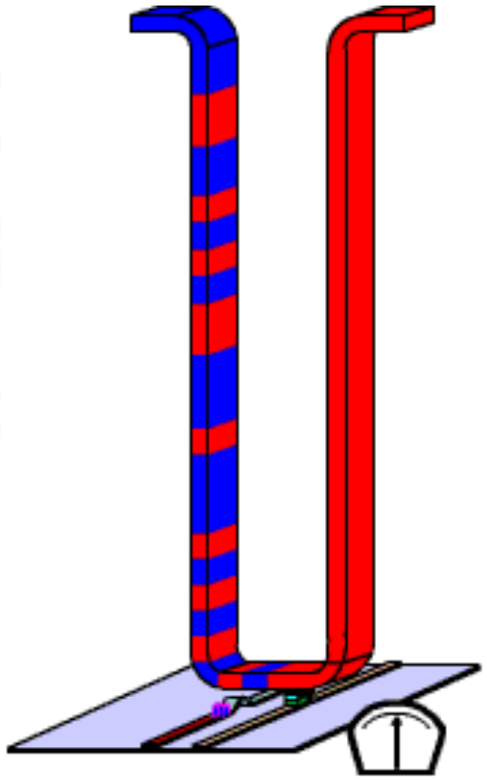


Computational magnetism important

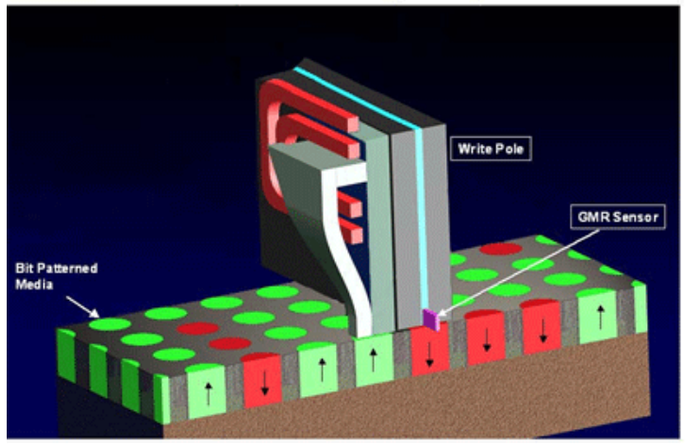
- The number of problems that can be solved analytically is very limited.
- Experimental techniques do not provide enough spatial and temporal resolution.







Parkin, Science, 320, 190 (2008)

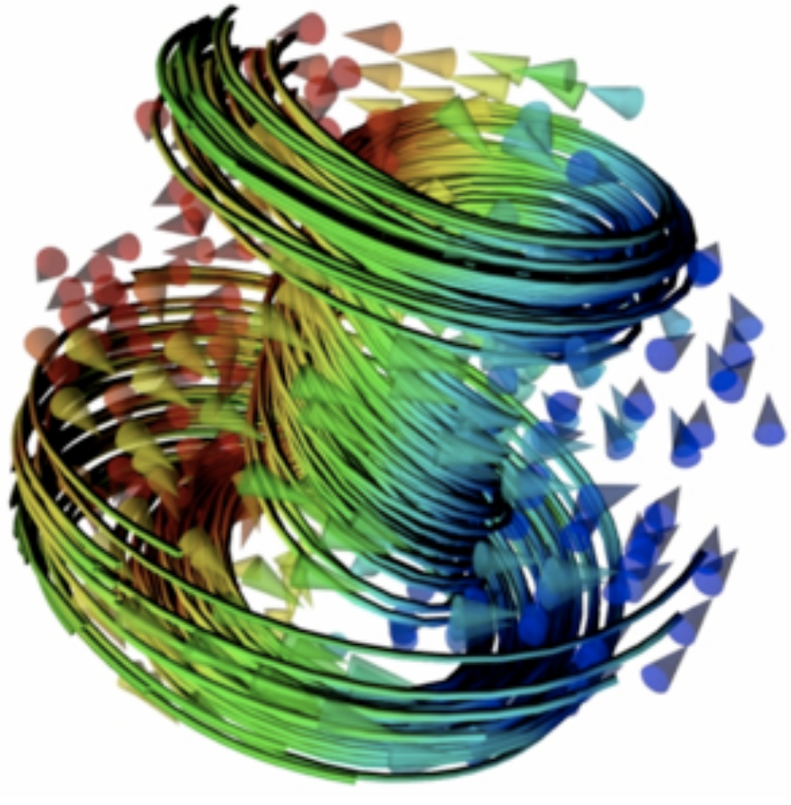


Bit-patterned media (Seagate)

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* **m**(**r**):

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



- We have an equation of motion

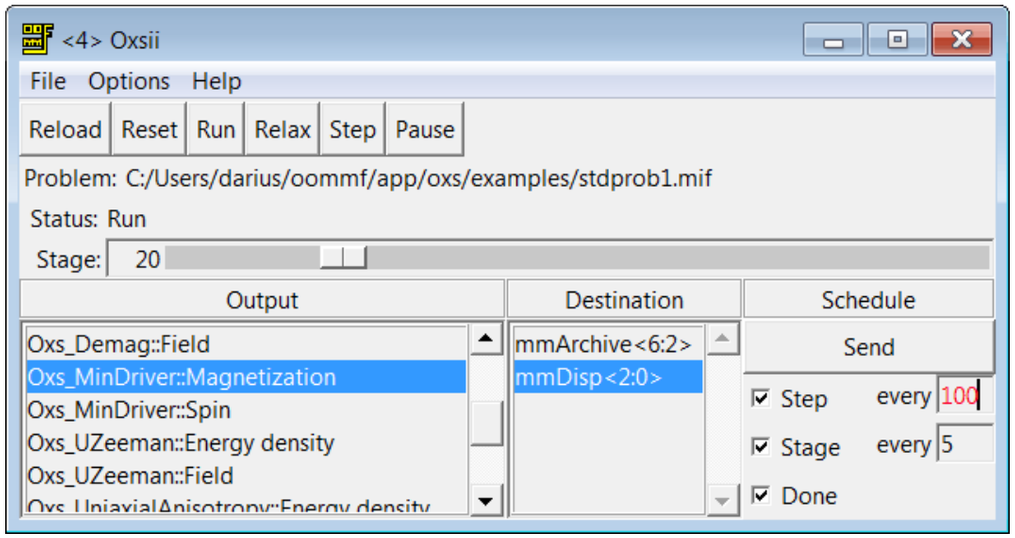
$$\frac{\partial \mathbf{m}}{\partial t} = \mathbf{f}(\mathbf{m})$$

- **f** is complicated, involves PDEs

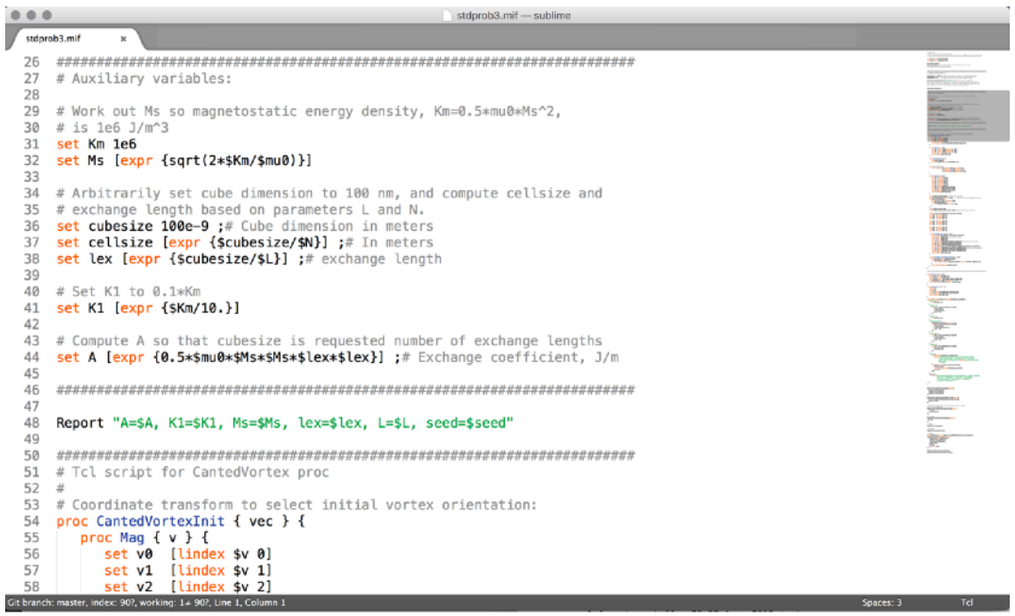
State of the art micromagnetic simulation tool

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



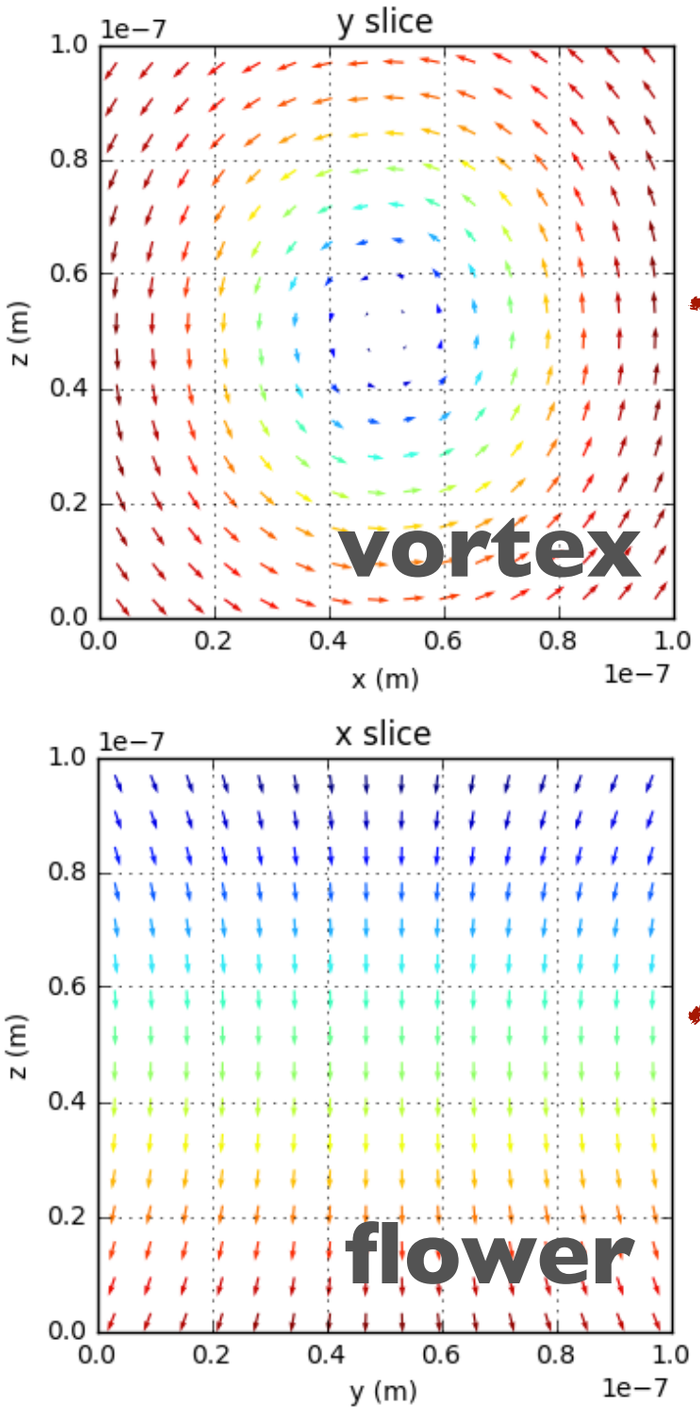
GUI

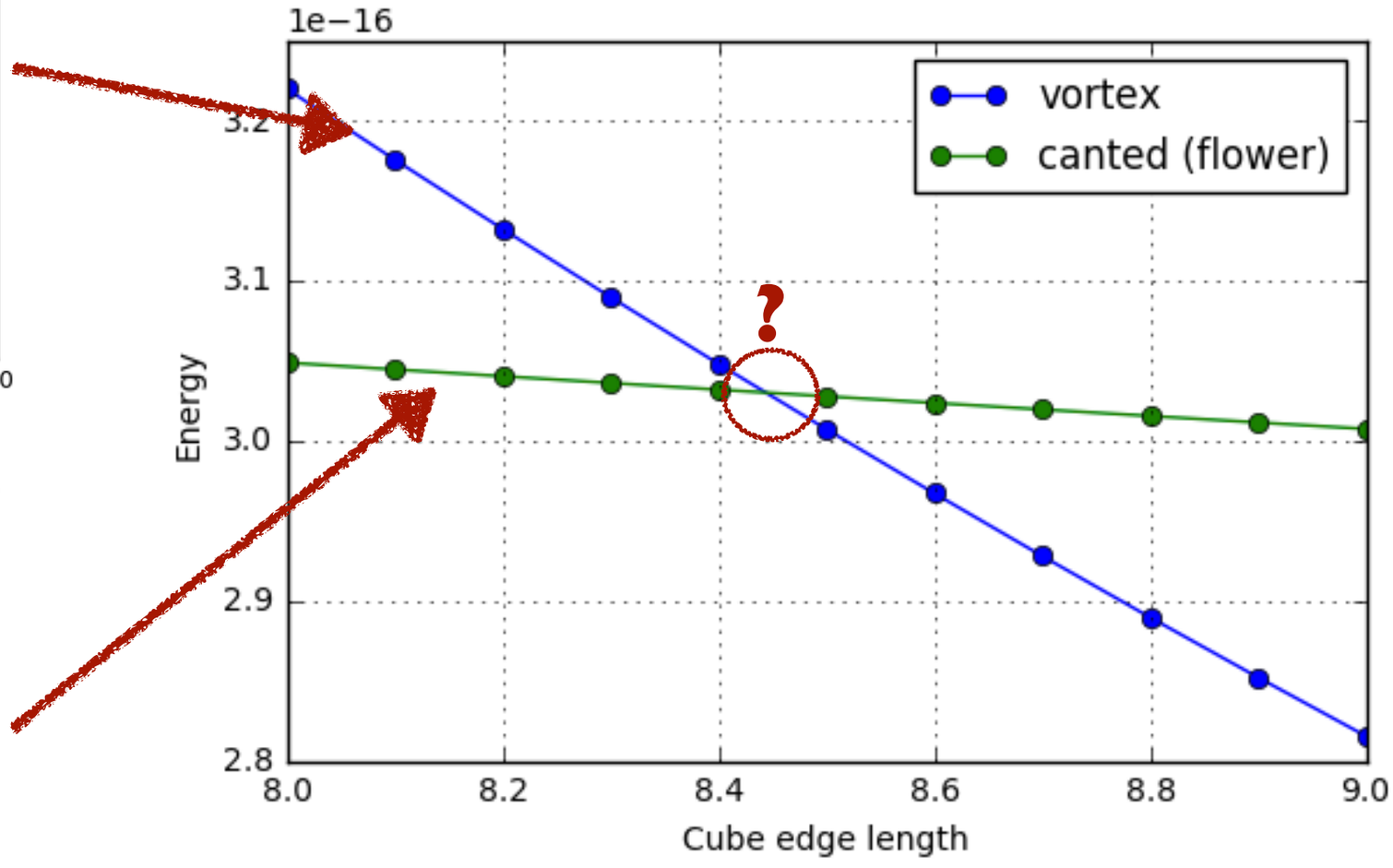


Tcl config file

Research workflow example

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

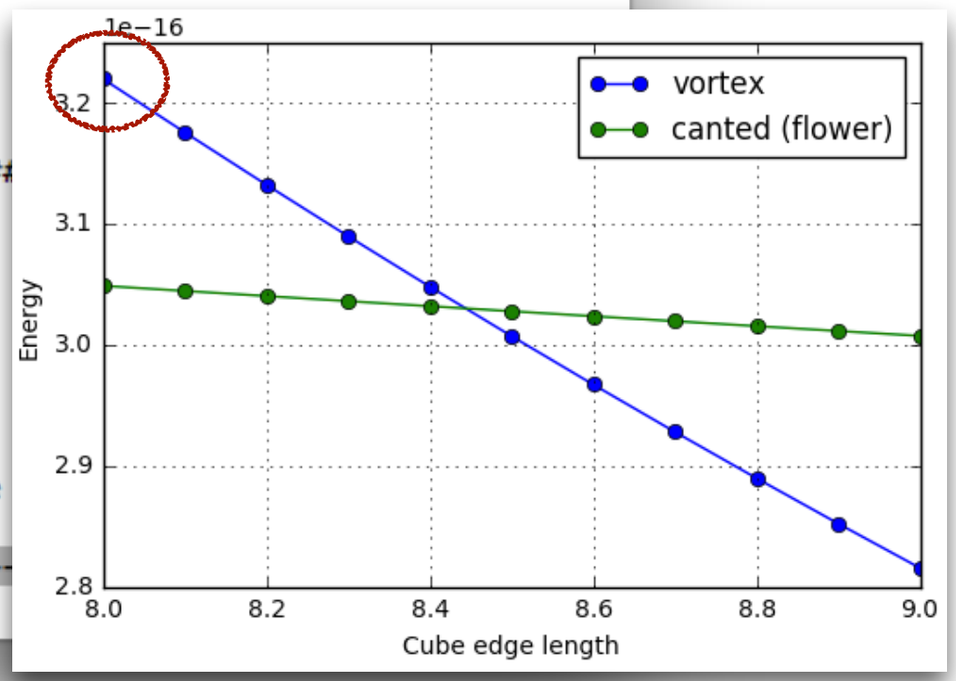




Step 1: write simulation configuration

my_project — IPython: Users/mb4e10 — emacs -nw stdprob3.mif — 95×37

```
# 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", "canted", "cantedvortex", "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)--------------
```

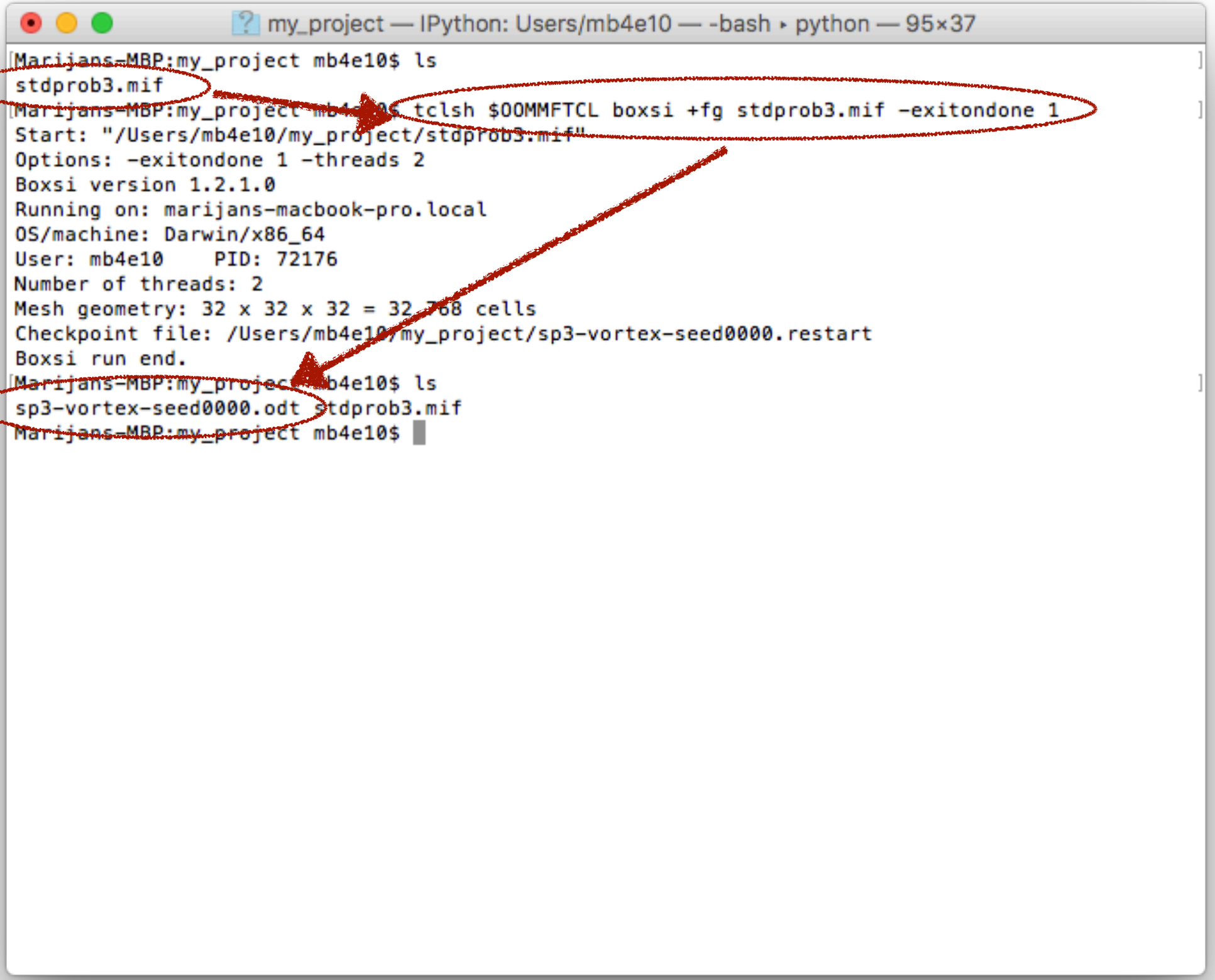


Step 1: write simulation configuration

examples — emacs stdprob3.mif — 95×37

```
      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



Step 3: extract data from output file

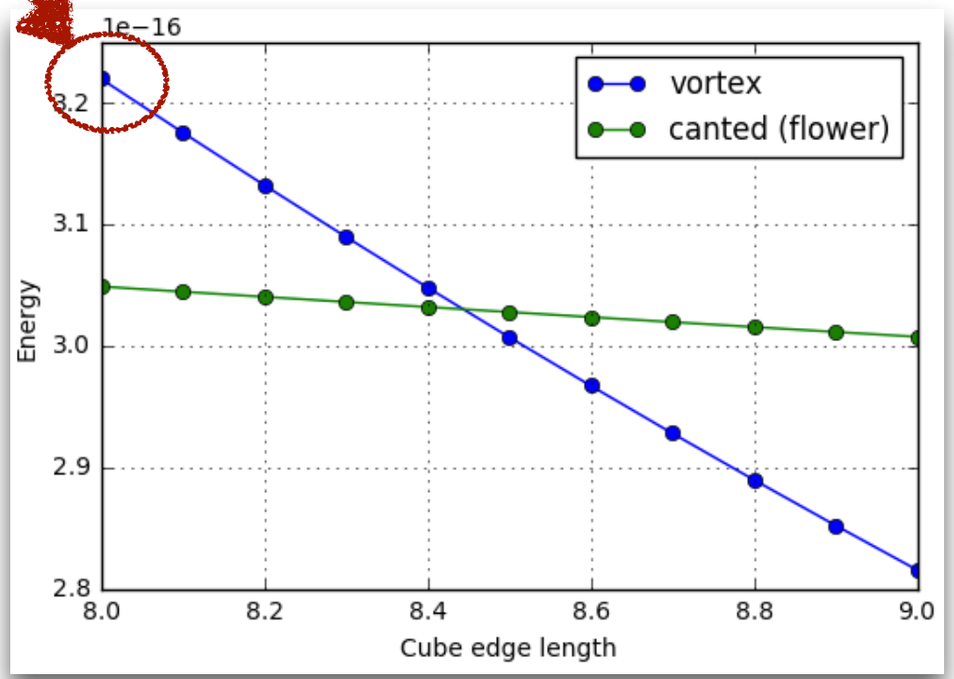
my_project — IPython: Users/mb4e10 — emacs -nw sp3-vortex-seed0000.odt — 95×37

```
# ODT 1.0
# Table Start
# Title: mmArchive Data Table, Wed Nov 16 20:54:28 GMT 2016
# Columns: {Oxs_CGEvolve::Max mxHxm} {Oxs_CGEvolve::Total energy} {Oxs_CGEvolve::Delta E} {Oxs\
_CGEvolve::Bracket count} {Oxs_CGEvolve::Line min count} {Oxs_CGEvolve::Conjugate cycle count}\
 {Oxs_CGEvolve::Cycle count} {Oxs_CGEvolve::Cycle sub count} {Oxs_CGEvolve::Energy calc count}\
 Oxs_UniaxialAnisotropy::Energy Oxs_UniformExchange::Energy {Oxs_UniformExchange::Max Spin Ang\
} {Oxs_UniformExchange::Stage Max Spin Ang} {Oxs_UniformExchange::Run Max Spin Ang} Oxs_Demag:\
:Energy    Oxs_MinDriver::Iteration {Oxs_MinDriver::Stage iteration} Oxs_MinDriver::Stage Oxs_\
MinDriver::mx    Oxs_MinDriver::my    Oxs_MinDriver::mz
# Units:              A/m                                J                                J                \
         {}                                {}                                {}                \
             {}                                {}                                {}                \
               J                                 J                                deg               \
                     deg                                                 deg                              J \
                      {}                                {}                                {}                \
   {}                   {}                   {}
              0.00097778028256529097       3.2257415663518404e-16      0                    \
       353                              326                                 7                    \
           340                              333                                 680                  \
             5.4172367330709765e-17           1.780007679106069e-16               11.011344278380\
658                     90.000000000000014                              90.000000000000014           \
  9.0401021393867362e-17    670                               670                          0          \
           0.40912126717720015  4.6139202285652408e-17 -1.9801501518039851e-16
# Table End
```

```
-:---F1  sp3-vortex-seed0000.odt   All L1     (Fundamental)----------------------------------
File mode specification error: (error "Buffer format not recognized")
```

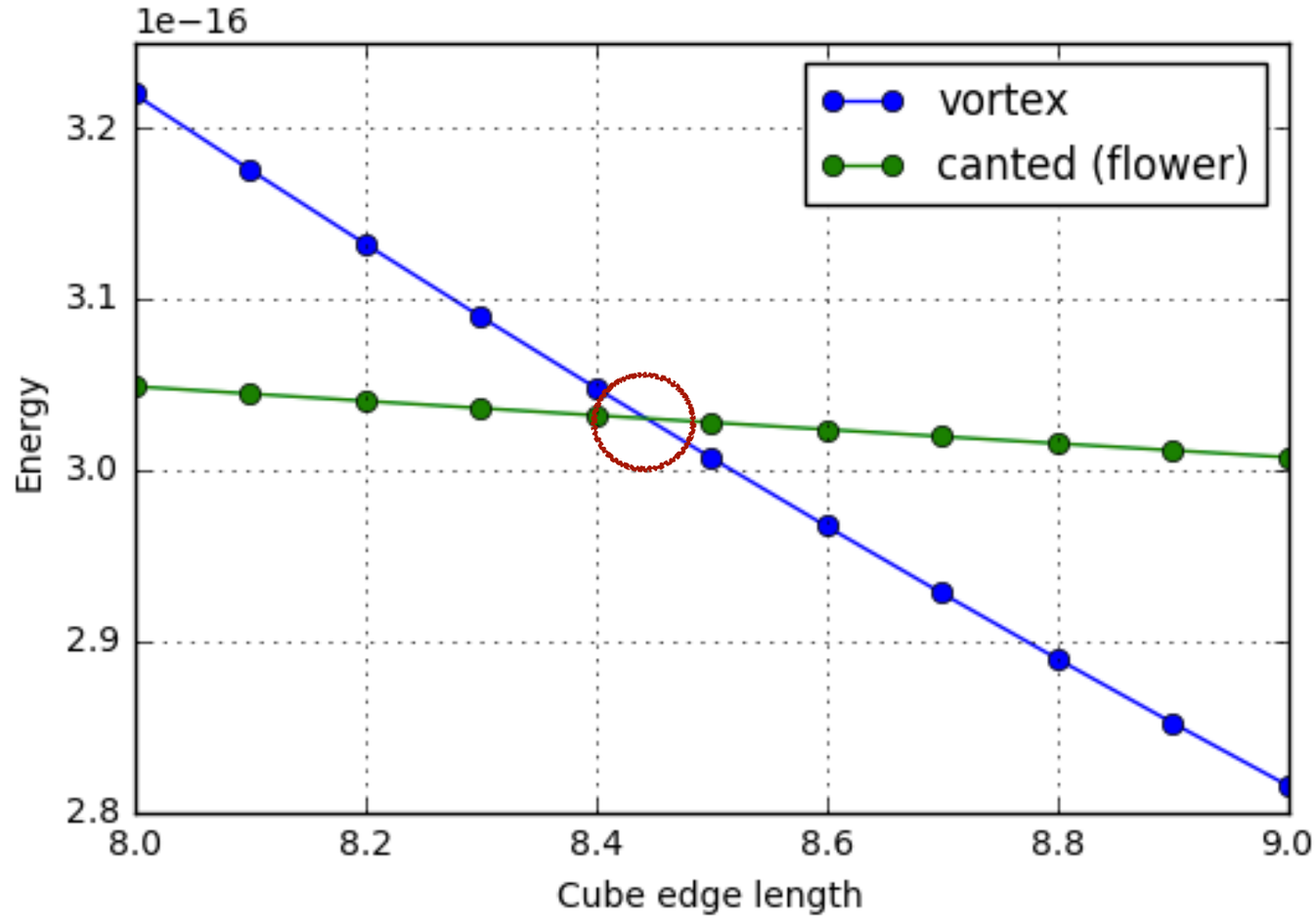
Step 4: gather data, and repeat simulations...

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 [1] 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

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

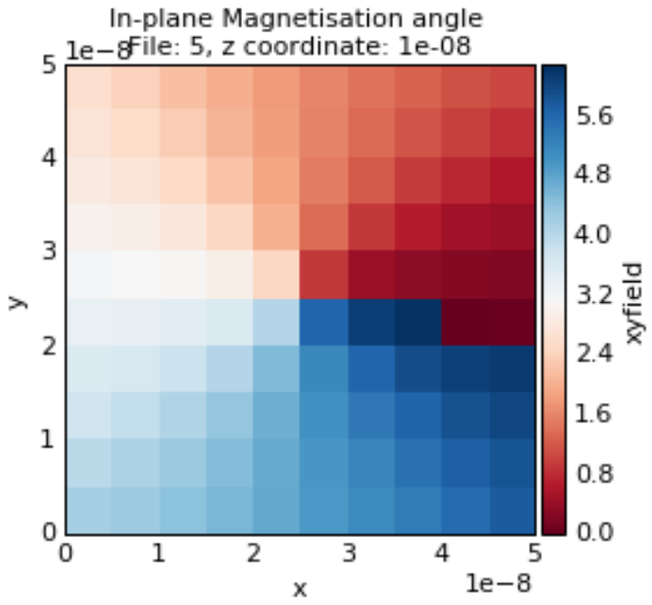
Slides end here

- 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]:



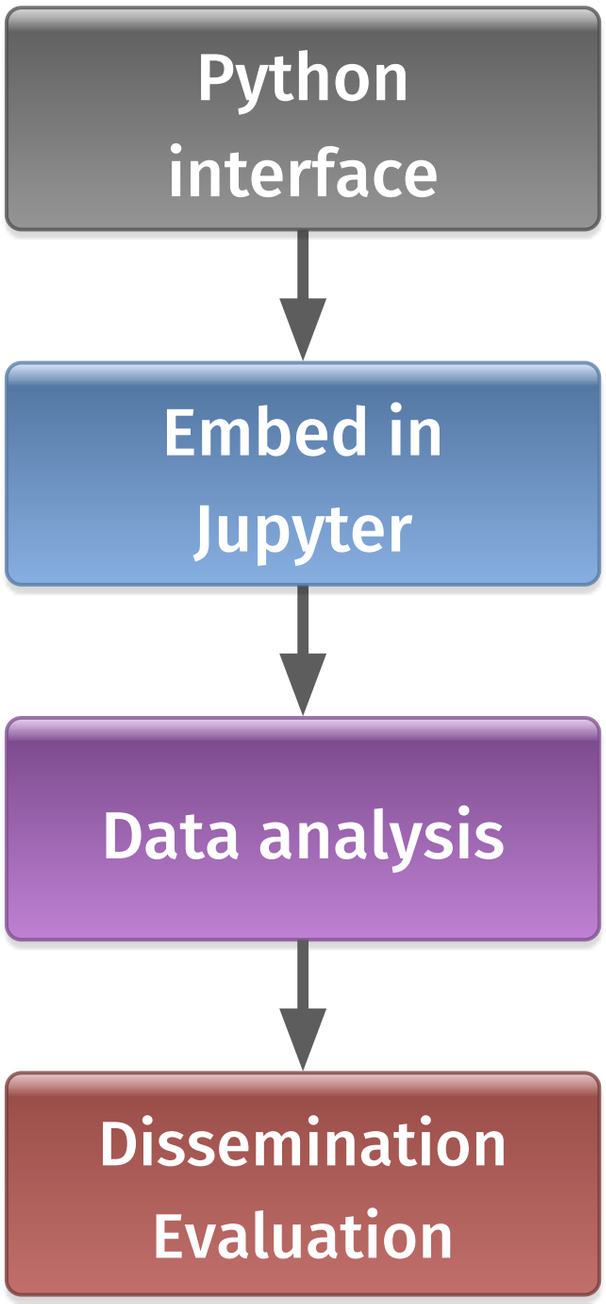
File:

5.0

z coordinate:

1e-08

Road map JOOMMF



OOMMFC

Latex representation

Matplotlib

Pandas

Holoviews

Widgets

interactive 3d vis

T3.8 - Python

T4.11- Jupyter

T4.8 - 3d vis

T4.14 - cloud

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

surface term

$$\phi(\mathbf{r}') = \frac{M_\mathrm{s}}{4\pi}\left(\underbrace{-\int_\Omega \frac{\nabla\cdot\mathbf{m}(\mathbf{r}')}{||\mathbf{r}-\mathbf{r}'||}\,\mathrm{d}V}_{\text{volume term}} + \overbrace{\int_{\partial\Omega} \frac{\mathbf{m}(\mathbf{r}')\cdot\mathbf{n}}{||\mathbf{r}-\mathbf{r}'||}\,\mathrm{d}S}\right)$$

$$\mathbf{H}_\mathrm{d}(\mathbf{r}) = -\nabla\phi(\mathbf{r})$$

- Effective field

DMI

demagnetisation

$$H_\mathrm{eff}(\mathbf{m}) = \underbrace{\frac{2A}{\mu_0 M_\mathrm{s}}\nabla^2\mathbf{m}}_{\text{exchange}} - \overbrace{\frac{2D}{\mu_0 M_\mathrm{s}}(\nabla\times\mathbf{m})} + \underbrace{\mathbf{H}}_{\text{Zeeman}} + \overbrace{\mathbf{H}_\mathrm{d}}$$

- LLG equation

damping

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

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

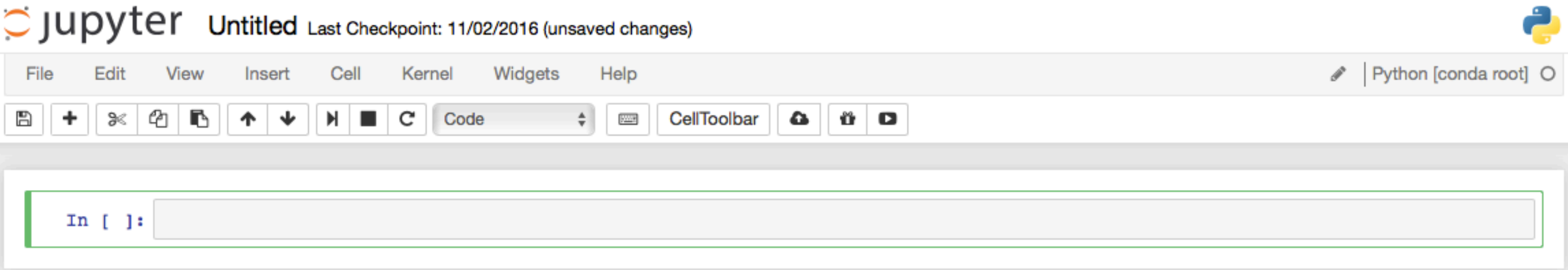
- We decided communicating with OOMMF via MIF (configuration) files:

  - 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]

Markdown 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, $l_\text{ex} = \sqrt{A/K_\text{m}}$,
where $K_\text{m}$ is a magnetostatic energy density, $K_\text{m} = \frac{1}{2}\mu_{0}M_\text{s}^{2}$.

Material parameters:

- uniaxial anisotropy $K_\text{u}$ with $K_\text{u} = 0.1 K_\text{m}$, and with the easy axis directed parallel to a
principal axis of the cube (0, 0, 1),
- exchange energy constant is $A = \frac{1}{2}\mu_{0}M_\text{s}^{2}l_\text{ex}^{2}$.

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)

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

Markdown 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, $l_{\mathrm{ex}} = \sqrt{A/K_{\mathrm{m}}}$, where K_{m} is a magnetostatic energy density, $K_{\mathrm{m}} = \frac{1}{2}\mu_0 M_{\mathrm{s}}^2$.

Material parameters:

- uniaxial anisotropy K_{u} with $K_{\mathrm{u}} = 0.1K_{\mathrm{m}}$, and with the easy axis directed parallel to a principal axis of the cube (0, 0, 1),
- exchange energy constant is $A = \frac{1}{2}\mu_0 M_{\mathrm{s}}^2 l_{\mathrm{ex}}^2$.

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

code

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

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

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]

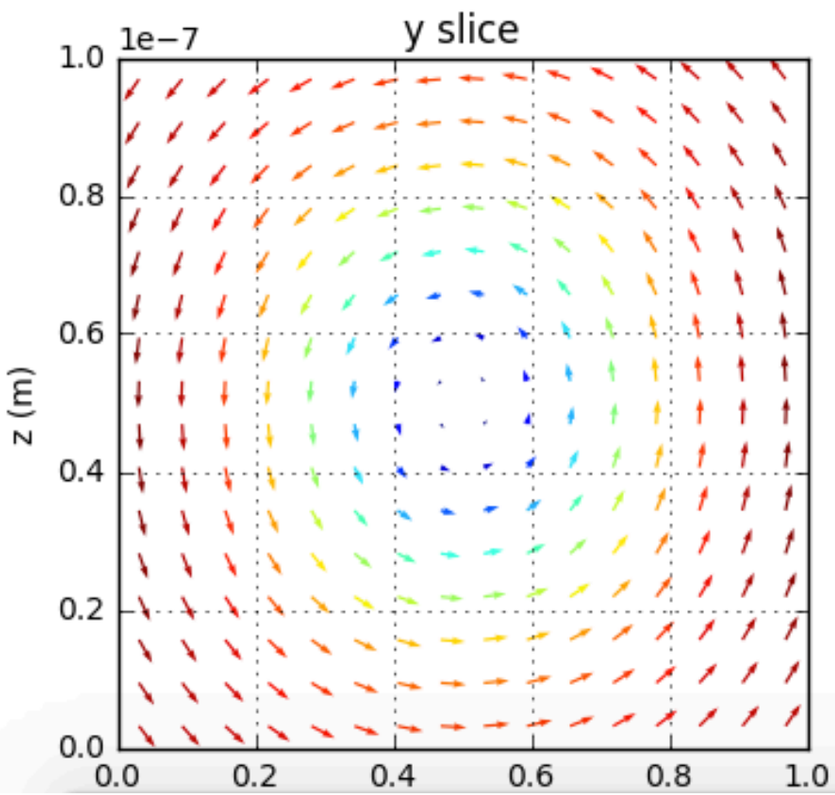
Code CellToolbar

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, xsize=4)
```



multiple simulation runs

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

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

Code CellToolbar

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 occurrs.

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

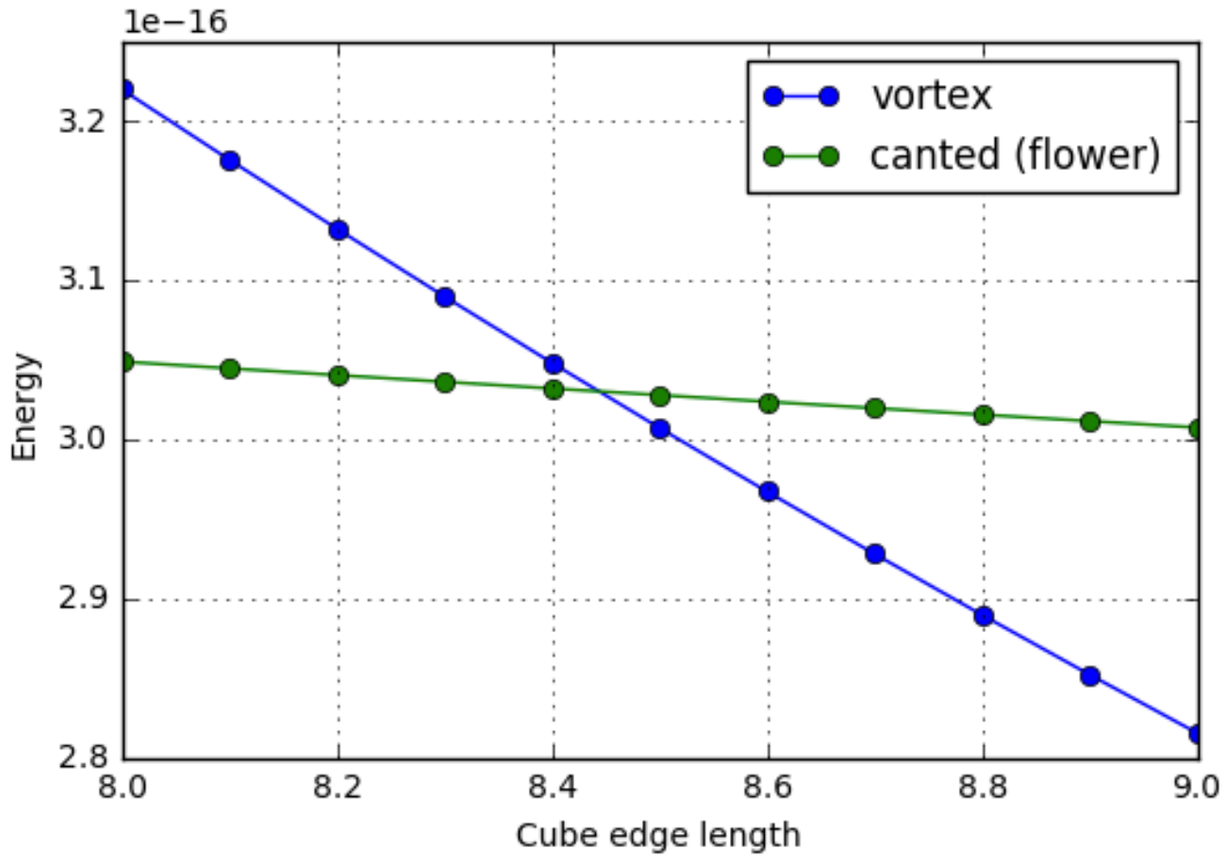
jupyter standard_problem3 Last Checkpoint: 11/02/2016 (autosaved)

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

Markdown CellToolbar

```
plt.legend()
```

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

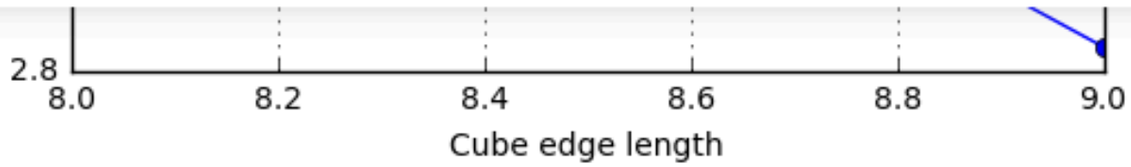


energy crossing

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

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

Markdown CellToolbar



We now know that the energy crossing occurrs between $8l_{ex}$ and $9l_{ex}$, 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)

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

Markdown CellToolbar

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	8.5391119
196	-2.685941e-18	3821.0	1264.690715	-1.936455e-09	-2.633442e-21	8.603004e-20	4.234452	4.234452	29.984064	8.8513919
197	-2.695973e-18	3840.0	1385.550409	-2.055475e-09	-2.820054e-21	8.010709e-20	4.782664	4.782664	29.984064	9.0791919
198	-2.706283e-18	3859.0	1350.603218	-2.048108e-09	-2.831441e-21	7.558538e-20	4.688752	4.815510	29.984064	9.2229419
199	-2.716260e-18	3878.0	1189.283501	-1.925038e-09	-2.680093e-21	7.113525e-20	4.377352	4.688752	29.984064	9.2922719

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.