A case study of computational science in Jupyter notebooks: JOOMMF

Hans Fangohr* and Marijan Beg

Faculty of Engineering and the Environment, University of Southampton, UK

* email: fangohr@soton.ac.uk, @ProfCompMod

Edinburgh, United Kingdom (20 January 2017)

Python – AN IntroduCtion

Hans Fangohr, fangohr@soton.ac.uk, BioSimGrid Meeting 25/11/2004

Overview

- Why Python
- How to get started
- Interactive Python (IPython)
- Installing extra modules
- Lists
- For-loops
- if-then
- modules and name spaces
- while
- string handling
- file-input, output
- functions
- Numerical computation
- some other features
  - ▷ long numbers
  - ▷ exceptions
  - ▷ dictionaries

Why Python?

→ Chapter 1, p3 in *LP*

- All sorts of reasons ;-)
  - ▷ Object-oriented scripting language
  - ▷ power of high-level language
  - ▷ portable, powerful, free
  - ▷ mixable (glue together with C/C++, Fortran, . . .)
  - ▷ easy to use (save time developing code)
  - ▷ easy to learn
  - ▷ (in-built complex numbers)
- Today:
  - ▷ easy to learn
  - ▷ some interesting features of the language
  - ▷ use as tool for small sysadmin/data processing/collecting tasks

Overview

1. Micromagnetics

   Magnetic moment

   Energies

   Dynamics

   OOMMF

2. OOMMF workflow

   Standard problem 3

   Example

3. JOOMMF

   micromagneticmodel

   standard problem 3

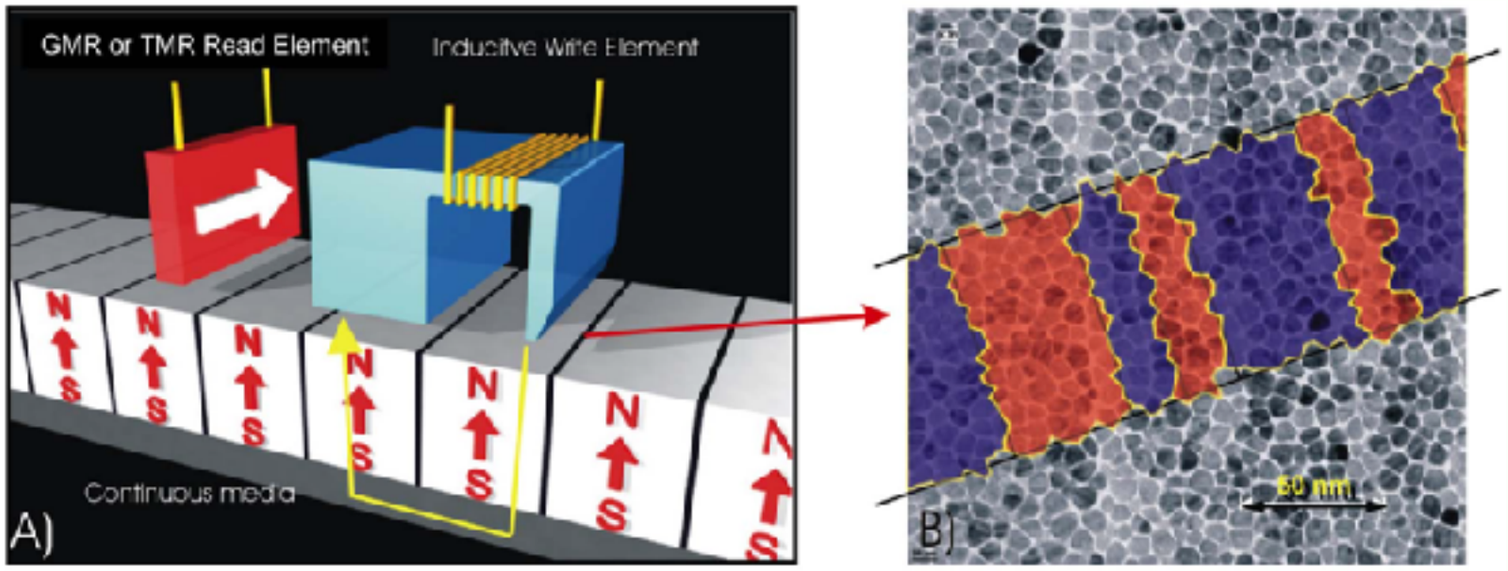
4. Some details & outlook

Micromagnetics

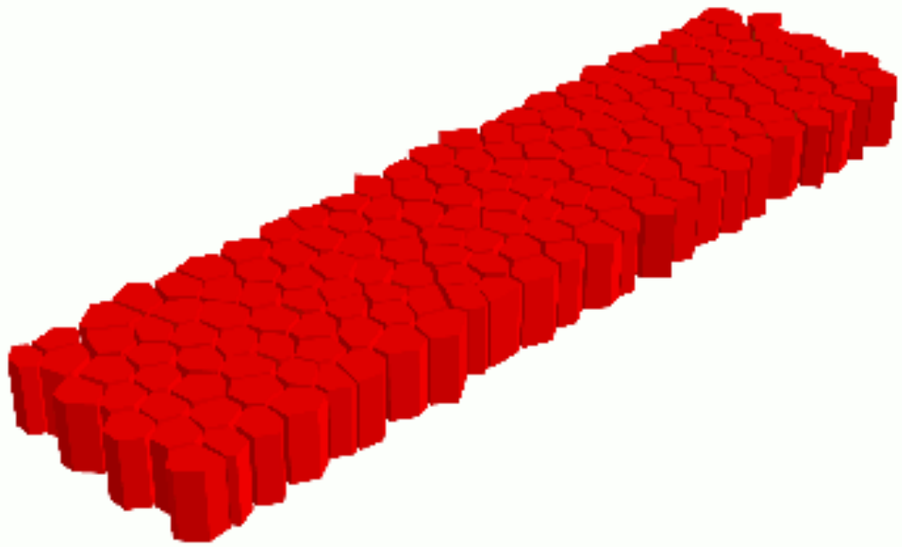
Why magnetic nanostructures?

1. Interesting complex system with tuneable parameters and experiments

2. Real applications, including
   - magnetic data storage (hard disk)
   - cancer therapy
   - low energy magnetic logic (spintronics)

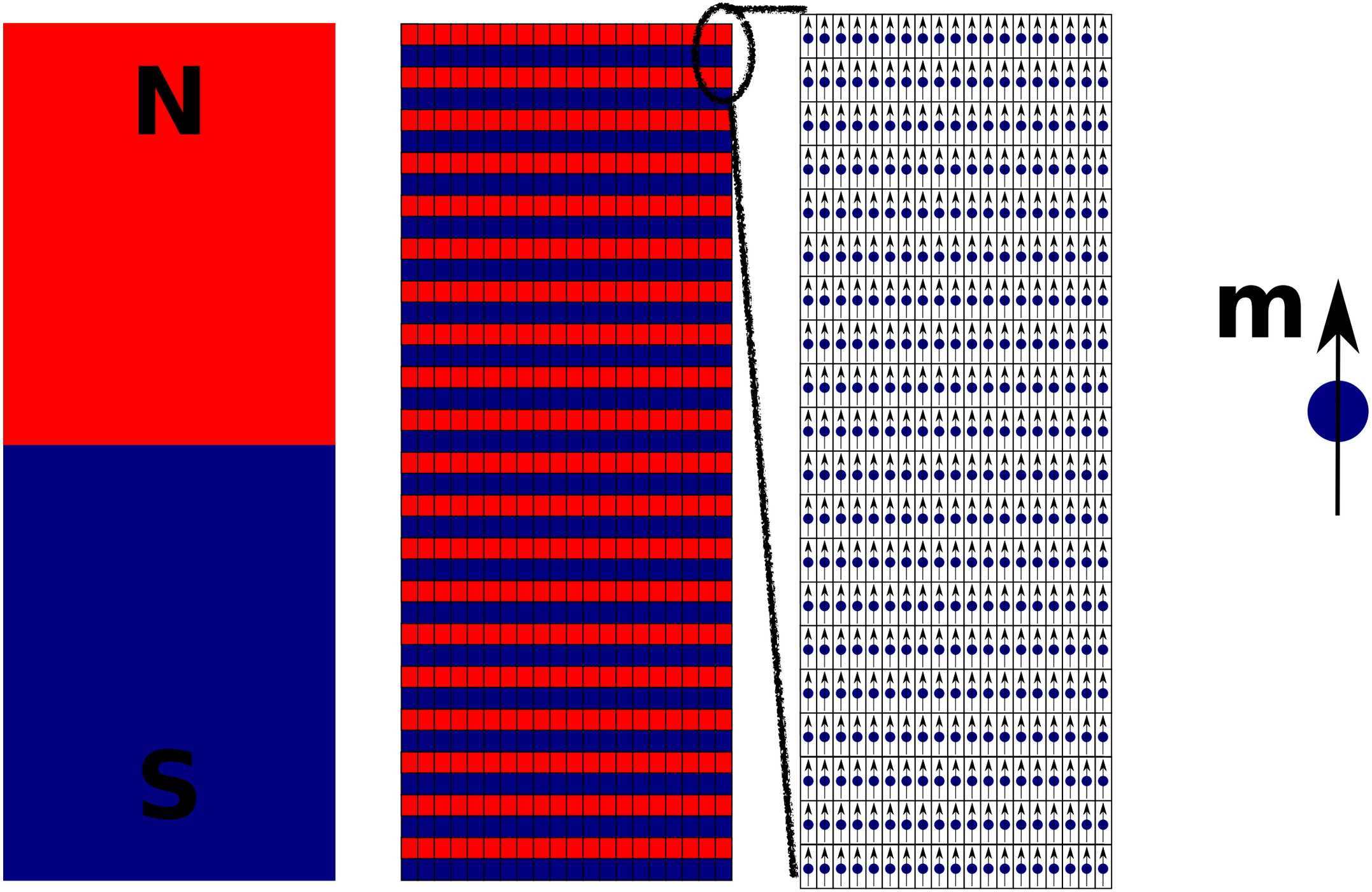


E. Dobisz et. al., Proceedings of IEEE 96, 1836 (2008)



Curtis & Fangohr (2011)

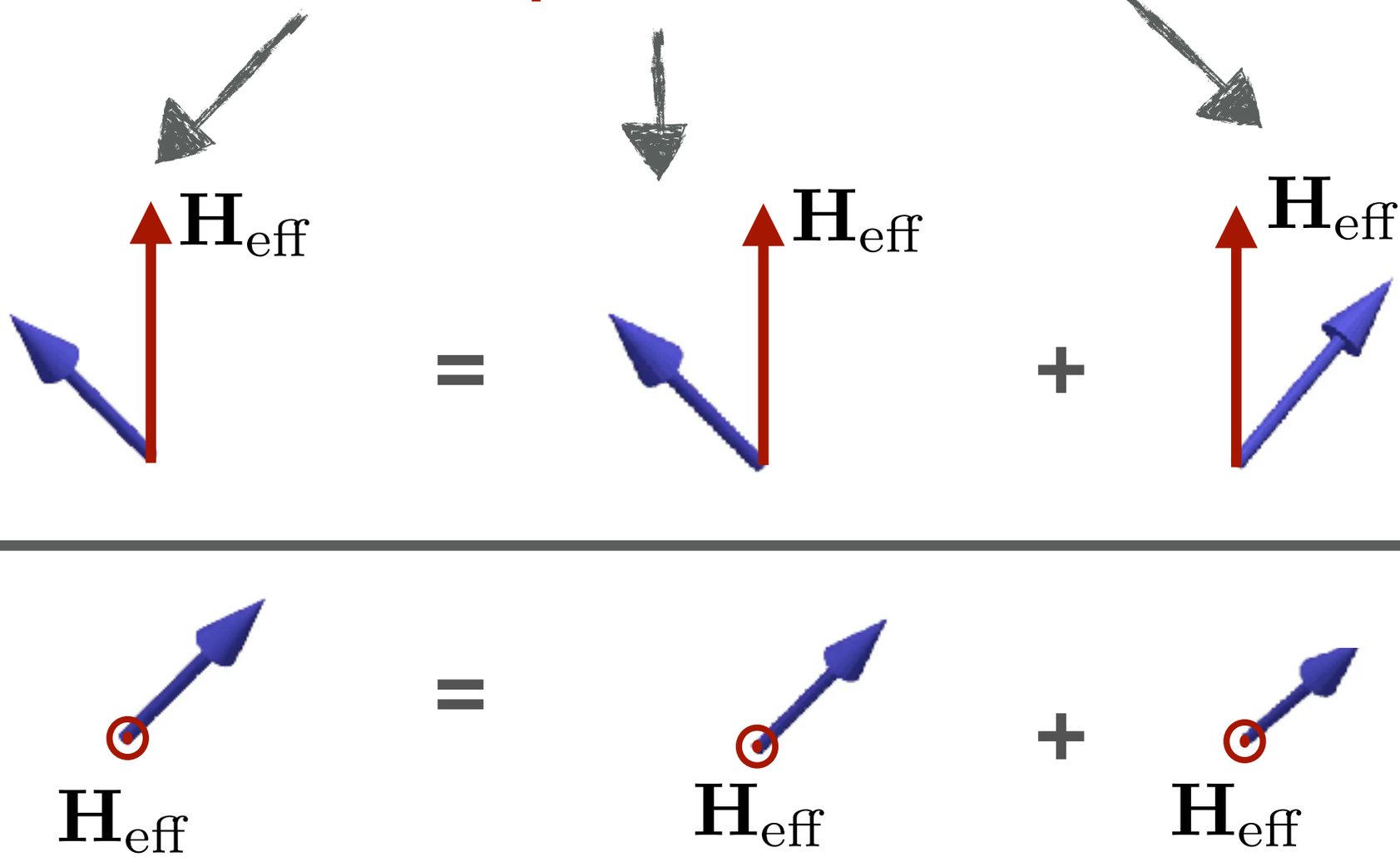
Magnetic moment



Magnetisation dynamics

- Landau-Lifshitz-Gilbert (LLG) equation

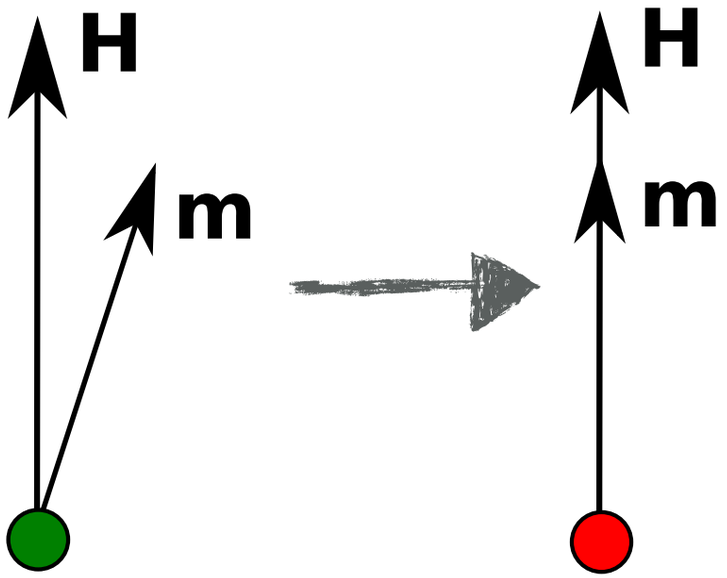
$$\frac{\partial \mathbf{m}}{\partial t} = \underbrace{\gamma^* \mathbf{m} \times \mathbf{H}_{\text{eff}}}_{\textbf{precession}} + \overbrace{\alpha \mathbf{m} \times \frac{\partial \mathbf{m}}{\partial t}}^{\textbf{damping}}$$



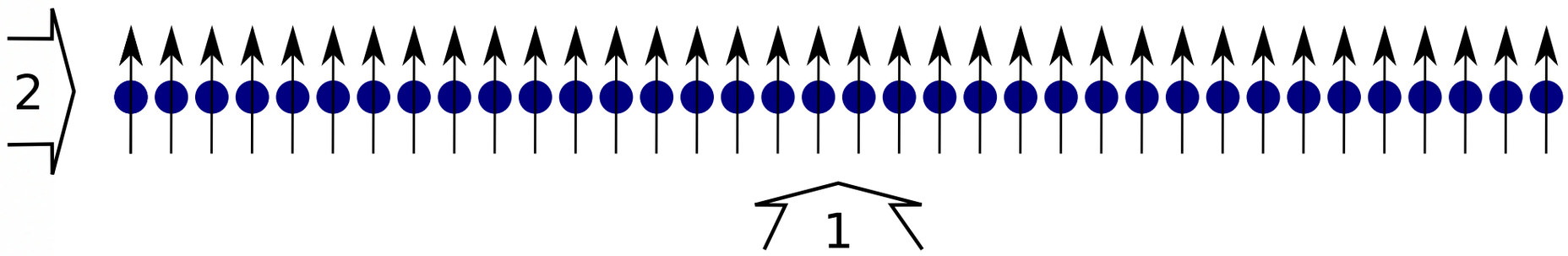
Zeeman energy

Align the magnetic moment to an external field.

$$w_z = -\mu_0 \mathbf{H} \cdot \mathbf{m}$$



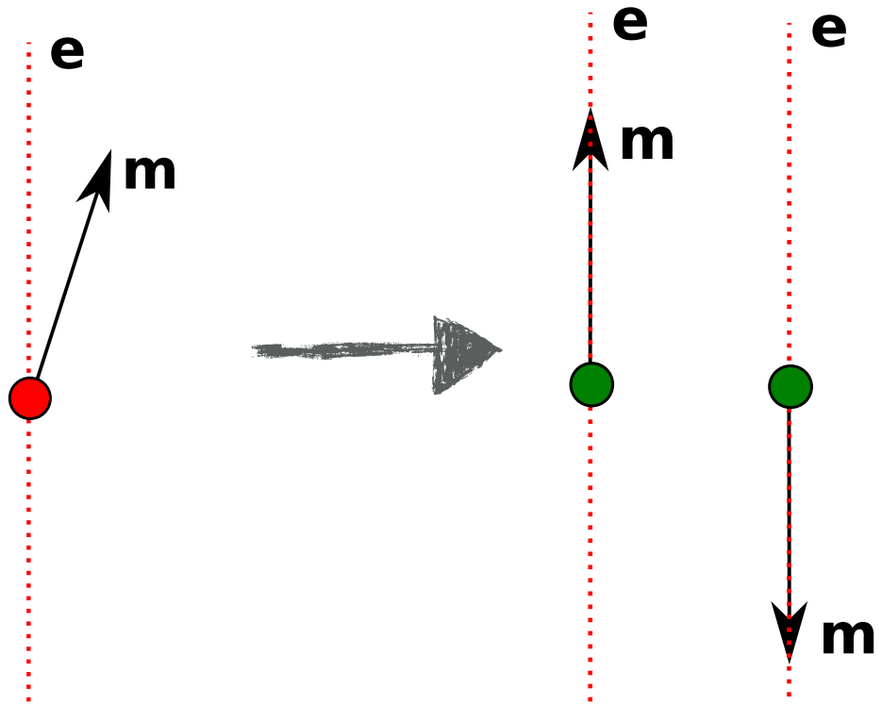
local



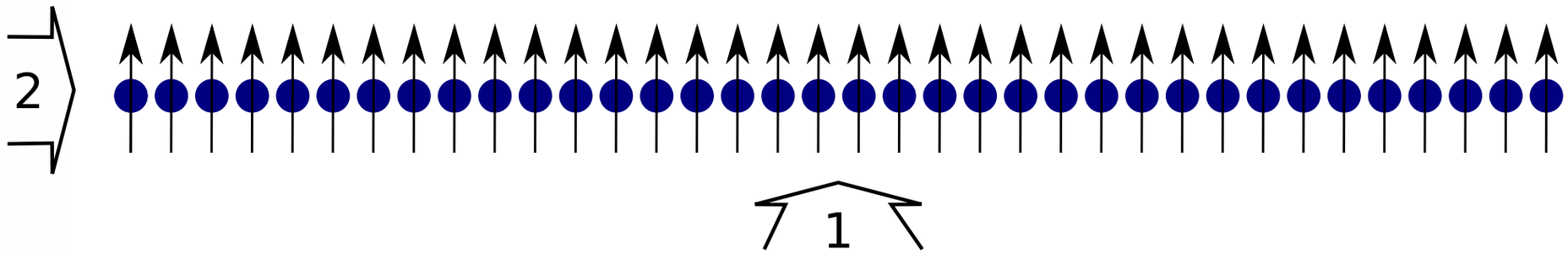
Anisotropy energy

Align the magnetic moment to an easy axis.

$$w_{\mathrm{a}} = -K(\mathbf{m}\cdot\mathbf{e})^2$$



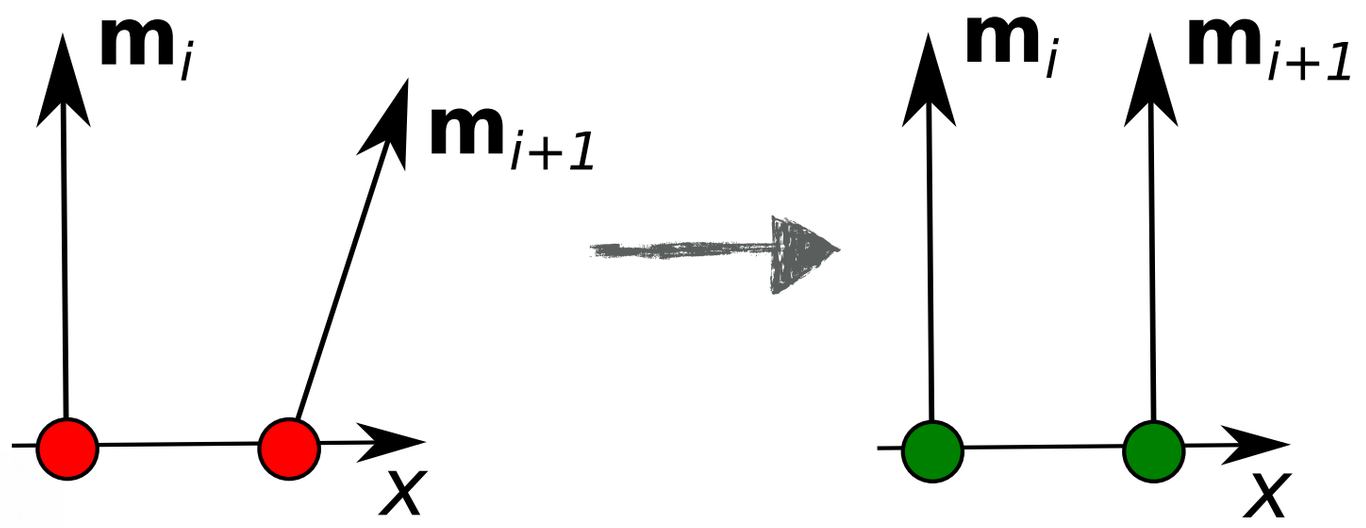
local



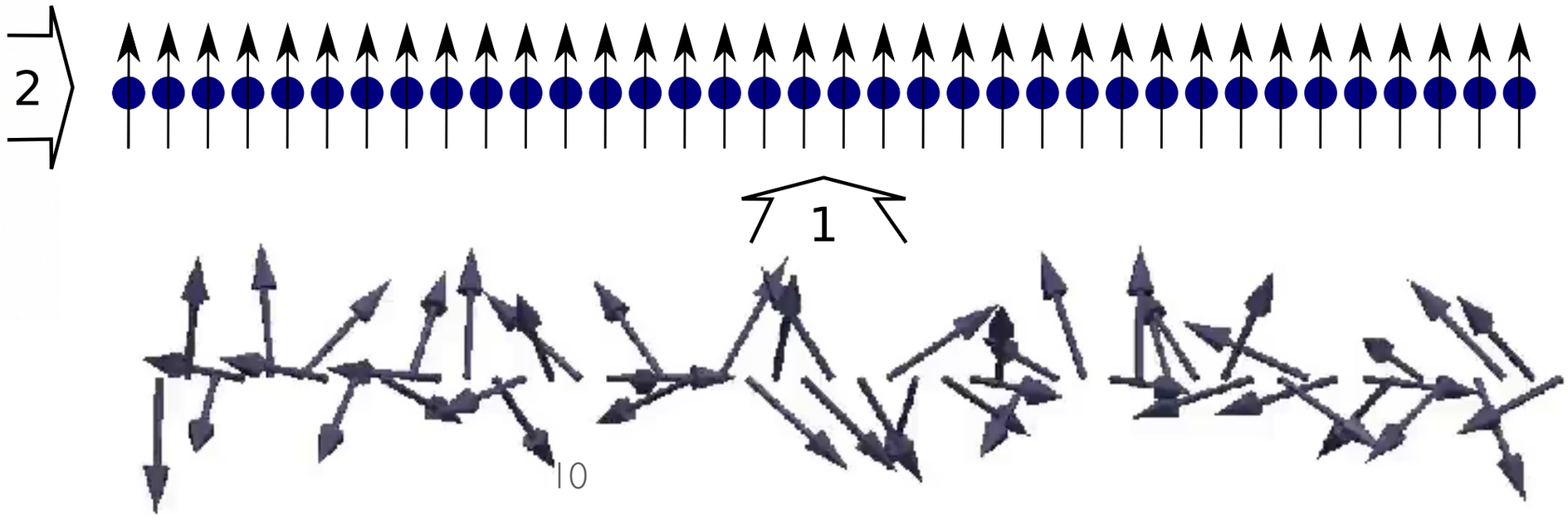
Exchange energy

Align all magnetic moments parallel.

$$w_{\mathrm{e}} = -J\mathbf{m}_i \cdot \mathbf{m}_{i+1}$$



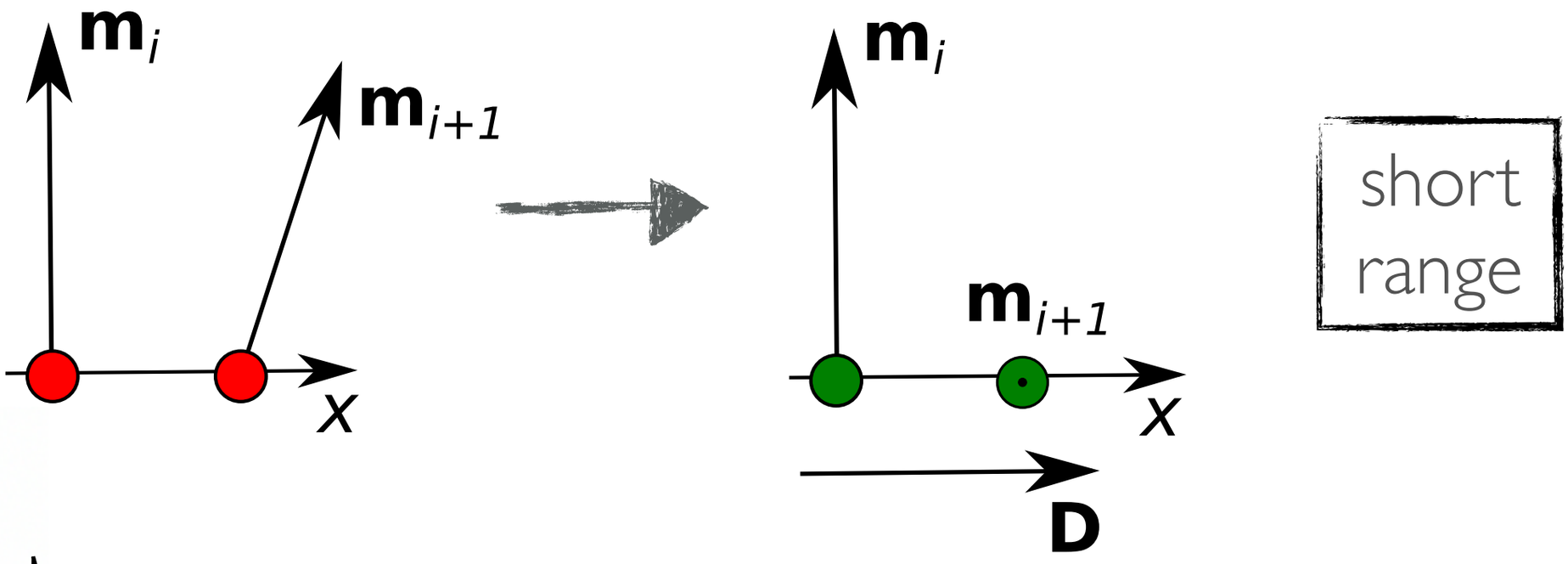
short range



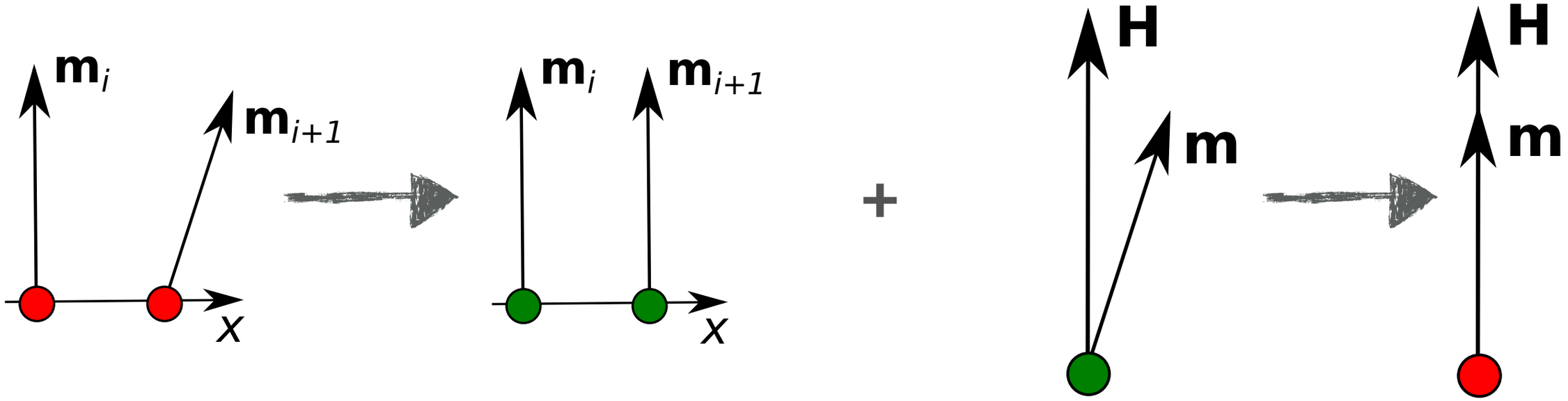
Dzyaloshinskii-Moriya energy (DMI)

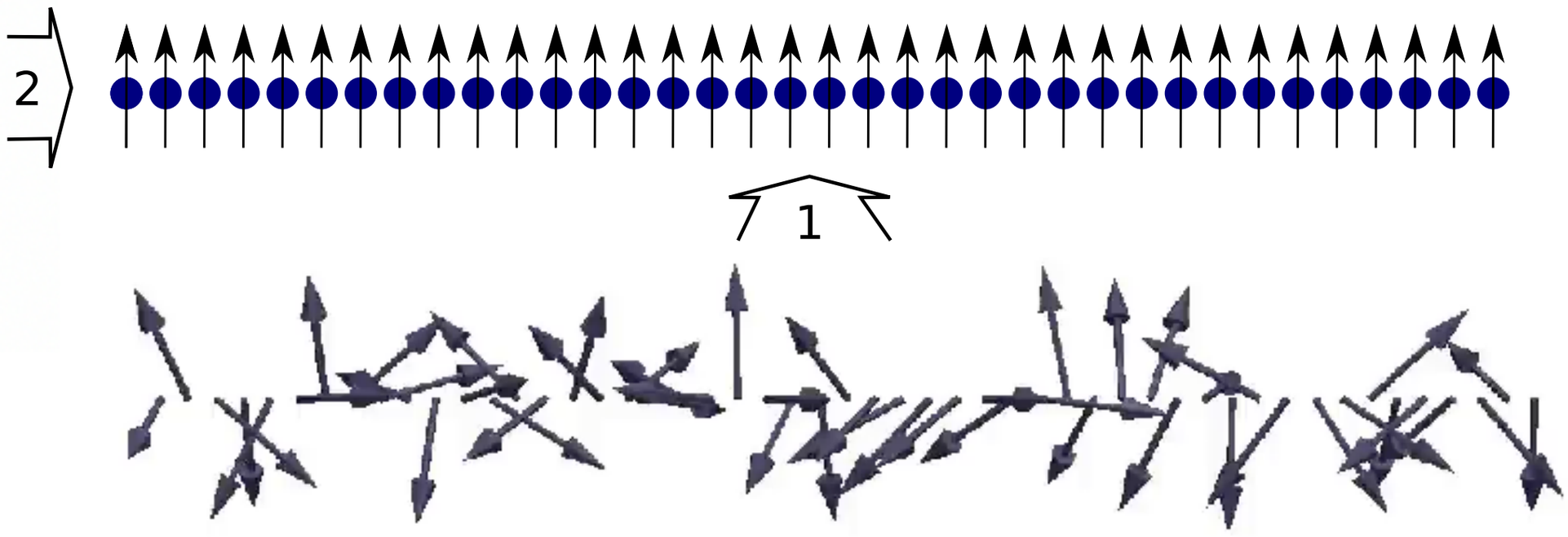
Align neighbouring moments perpendicular.

$$w_{\mathrm{e}} = -\mathbf{D} \cdot (\mathbf{m}_i \times \mathbf{m}_{i+1})$$

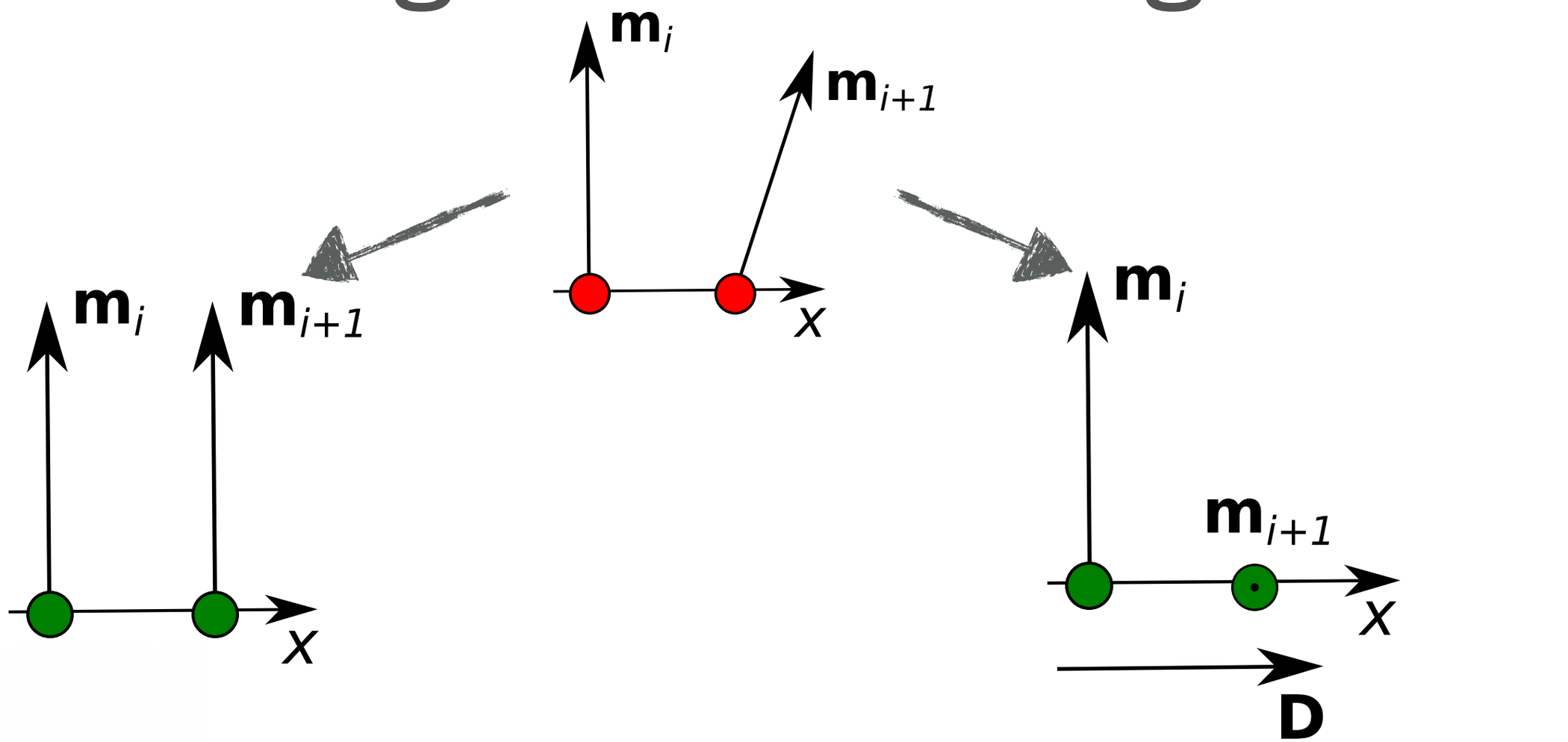


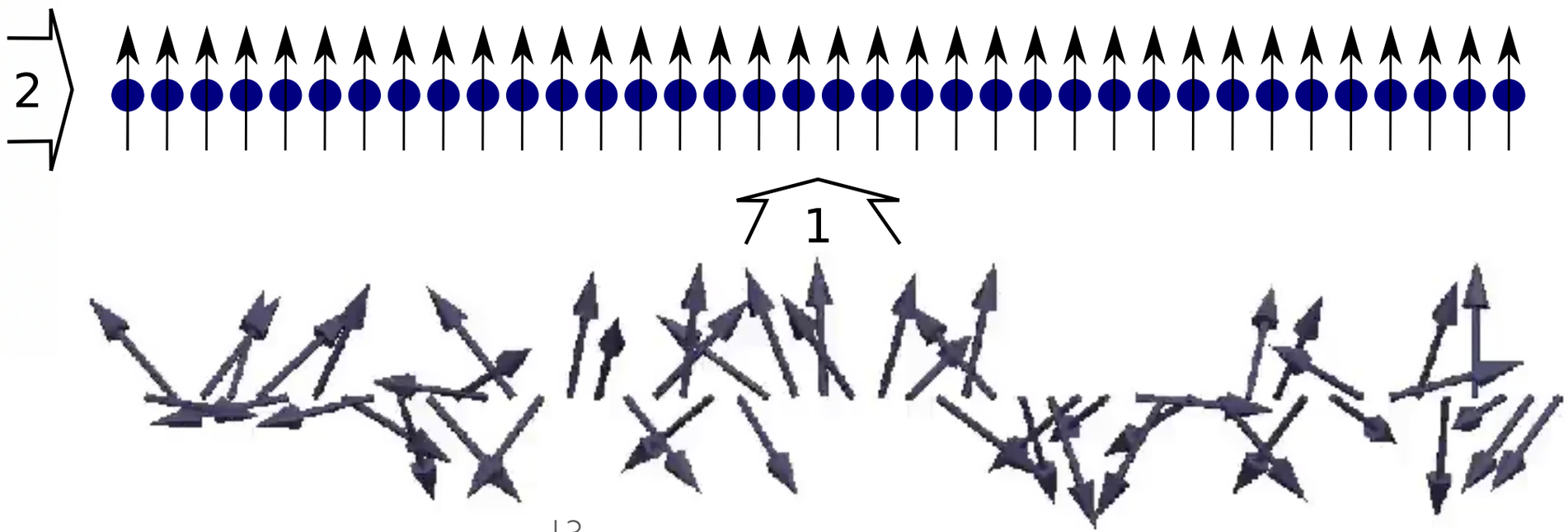
Exchange + Zeeman energies





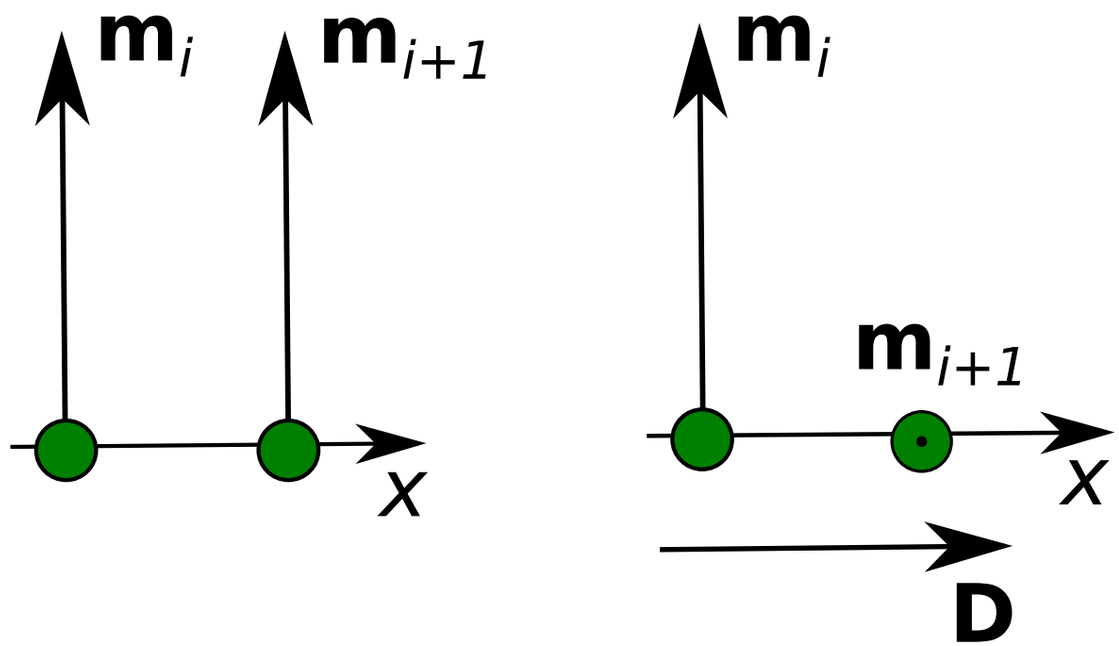
Exchange + DMI energies

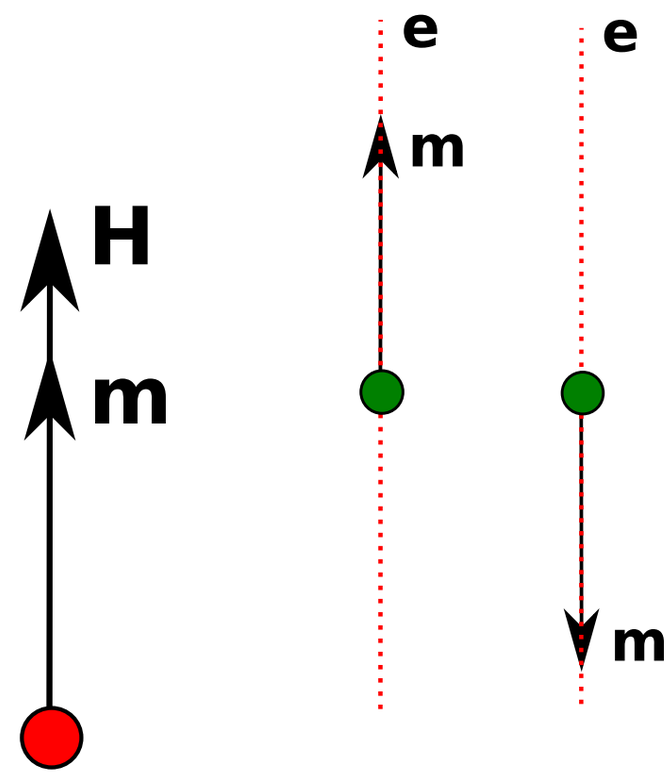


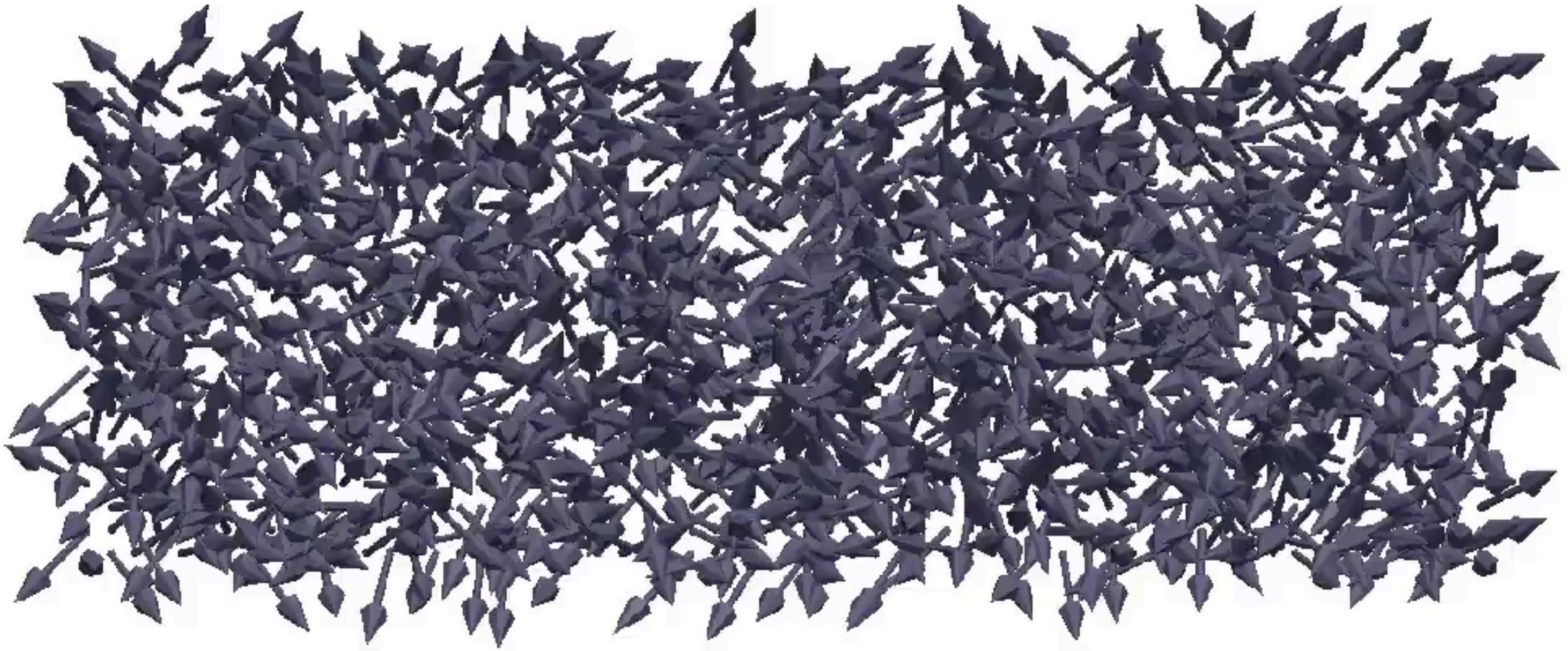


More complicated case

- All four energies included:
  - Exchange
  - Zeeman
  - Anisotropy
  - Dzyaloshinskii-Moriya energy (DMI)
- Two-dimensional sample.
- Particular set of material parameters *J*, *D*, and *K*.
- Non zero applied field.



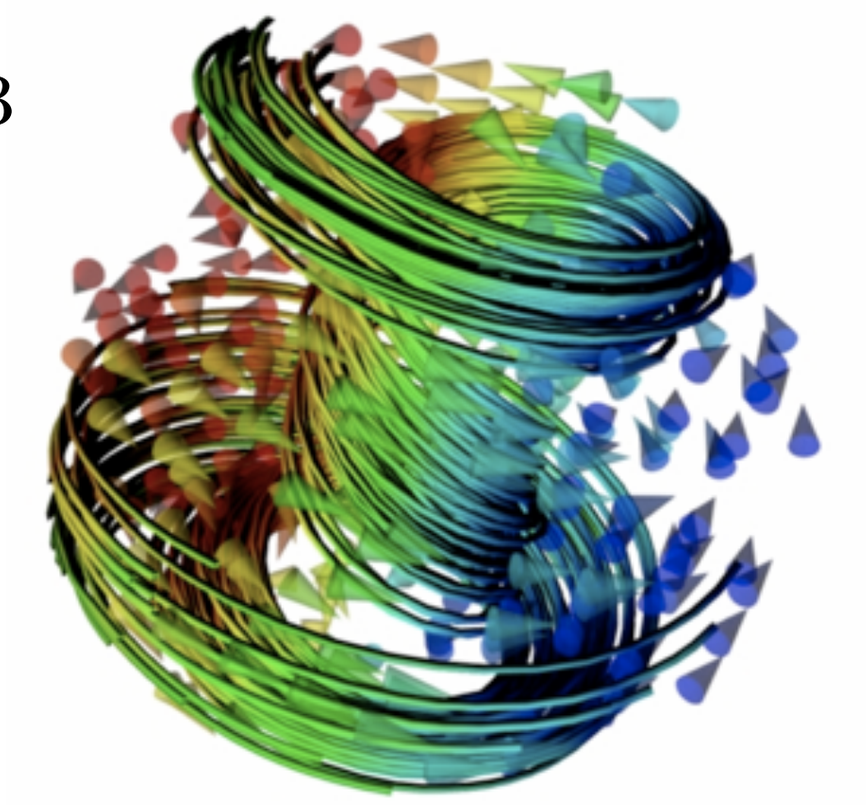




Micromagnetic model

- Magnetisation in sample V is described by a continuous *vector field* **m**(**r**):

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



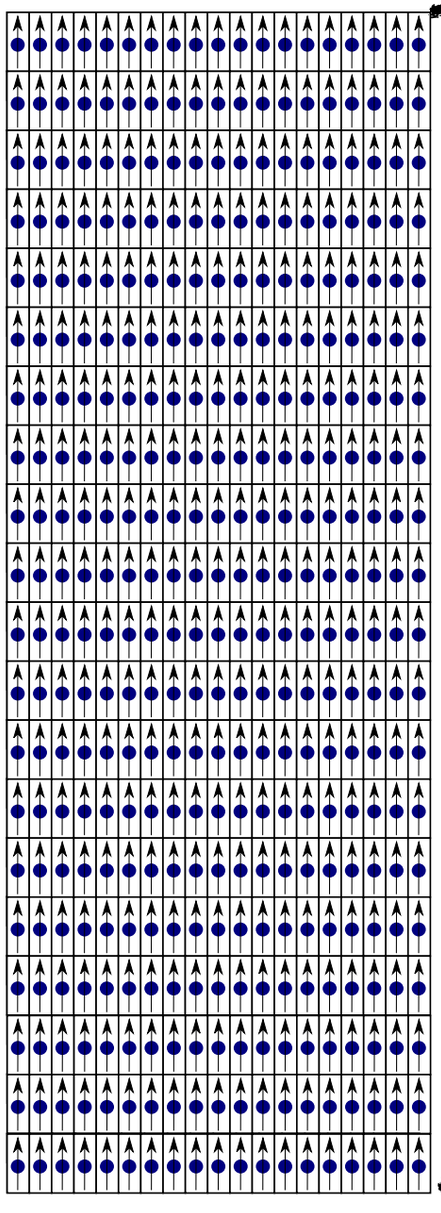
- We have an equation of motion

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

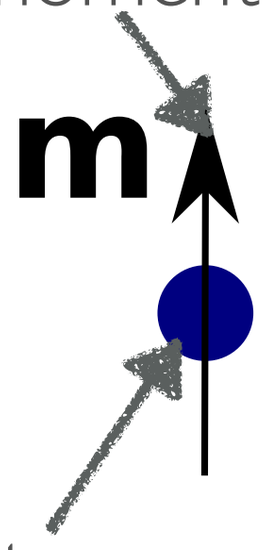
- f is complicated, involves PDEs

Coarse graining

atomistic model

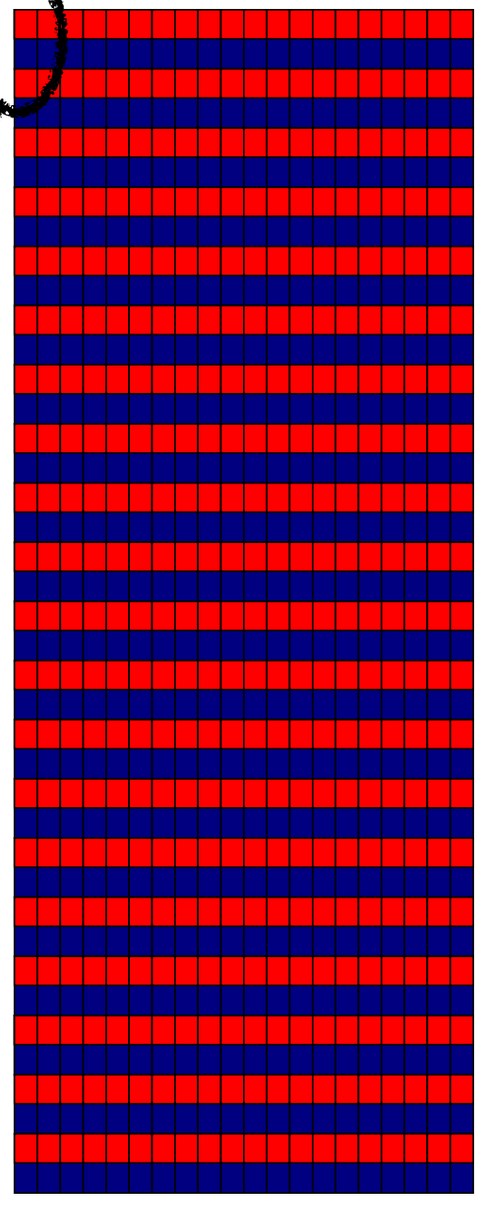


magnetic moment

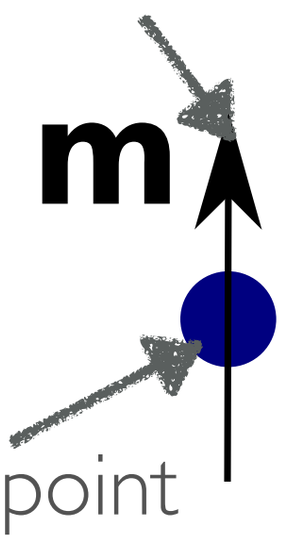


atom

continuous (=micromagnetic) model



magnetic moment



mesh point

unknown:
magnetisation vector field

$$\mathbf{m} = ?$$

interaction expressions

	atomistic model	continuous model	effective field
Zeeman energy	$-\mu_0 M_\mathrm{s}\mathbf{m}\cdot\mathbf{H}$	$-\mu_0 M_\mathrm{s}\mathbf{m}\cdot\mathbf{H}$	$\mathbf{H}$
Anisotropy energy	$-K(\mathbf{m}\cdot\mathbf{e})^2$	$-K(\mathbf{m}\cdot\mathbf{e})^2$	$\frac{2K}{\mu_0 M_\mathrm{s}}(\mathbf{m}\cdot\mathbf{e})\mathbf{e}$
Exchange energy	$-J\mathbf{m}_i\cdot\mathbf{m}_{i+1}$	$A(\nabla\mathbf{m})^2$	$\frac{2A}{\mu_0 M_\mathrm{s}}\nabla^2\mathbf{m}$
DMI energy	$\mathbf{D}\cdot(\mathbf{m}_i\times\mathbf{m}_{i+1})$	$D\mathbf{m}\cdot(\nabla\times\mathbf{m})$	$-\frac{2D}{\mu_0 M_\mathrm{s}}(\nabla\times\mathbf{m})$

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

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

Micromagnetic computational problem

- semi-discretised: PDE in every timestep, coupled set of ODEs to integrate time

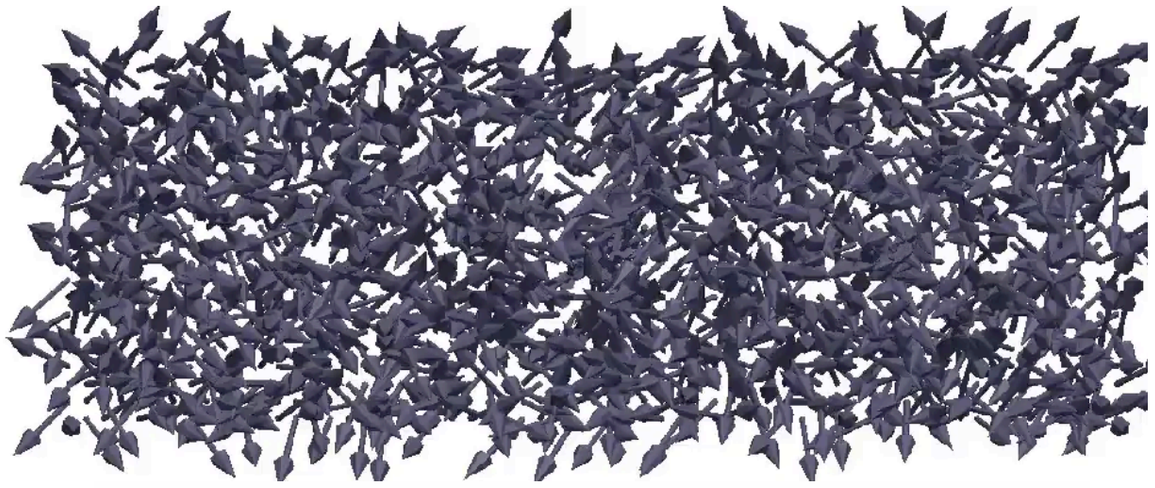
- Dynamics of magnetisation is defined by a stiff set of equations (different timescales)

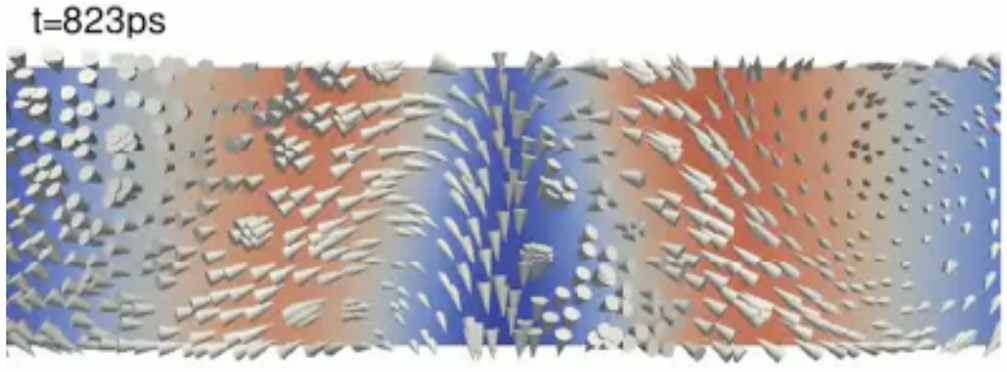
- Different length scales (exchange vs demagnetisation)

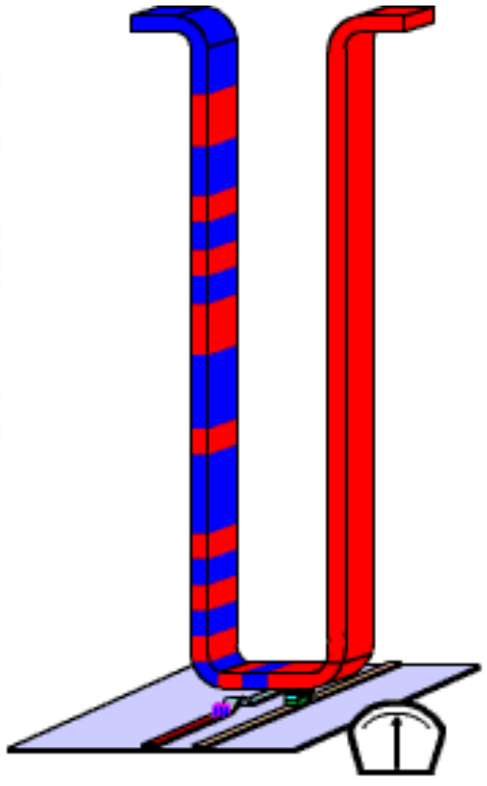
- "More difficult than CFD"

Micromagnetics

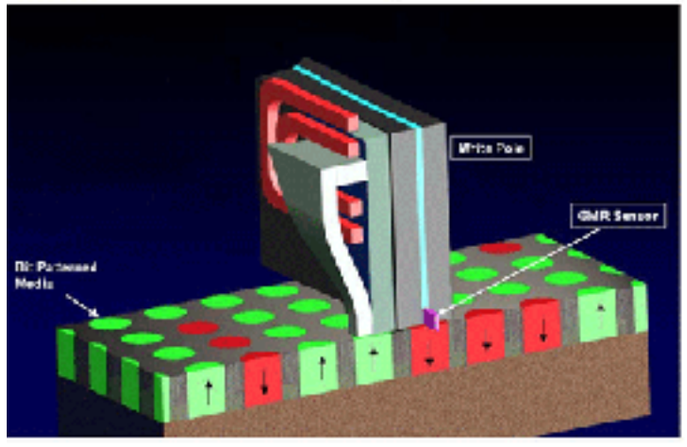
- The number of problems that can be solved analytically is very limited.
- Experimental techniques do not provide enough insight.
- Simulations allow to study the micromagnetic systems in much more detail, vary parameters, measure different observables, etc.







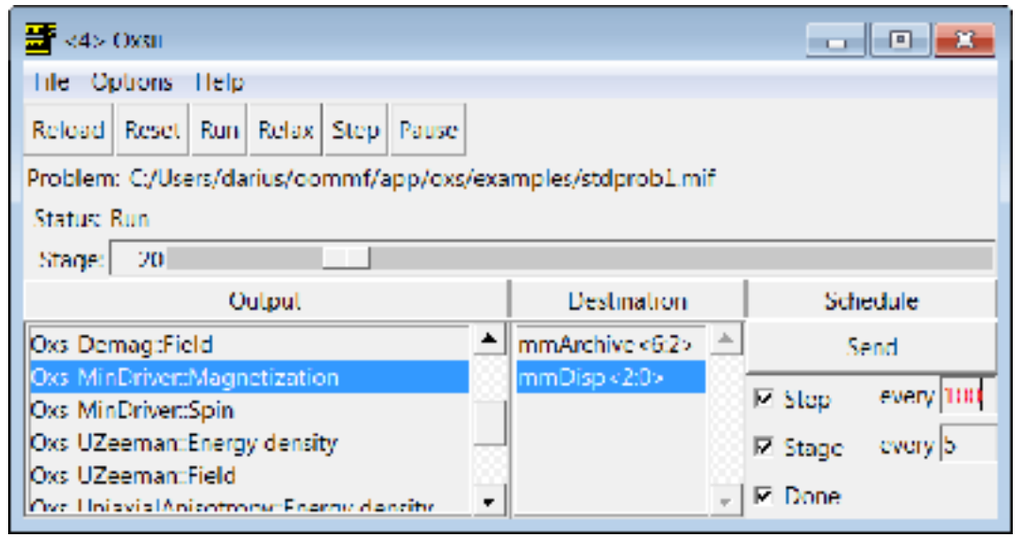
Parkin, Science, 320, 190 (2008)



Bit-patterned media (Seagate)

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



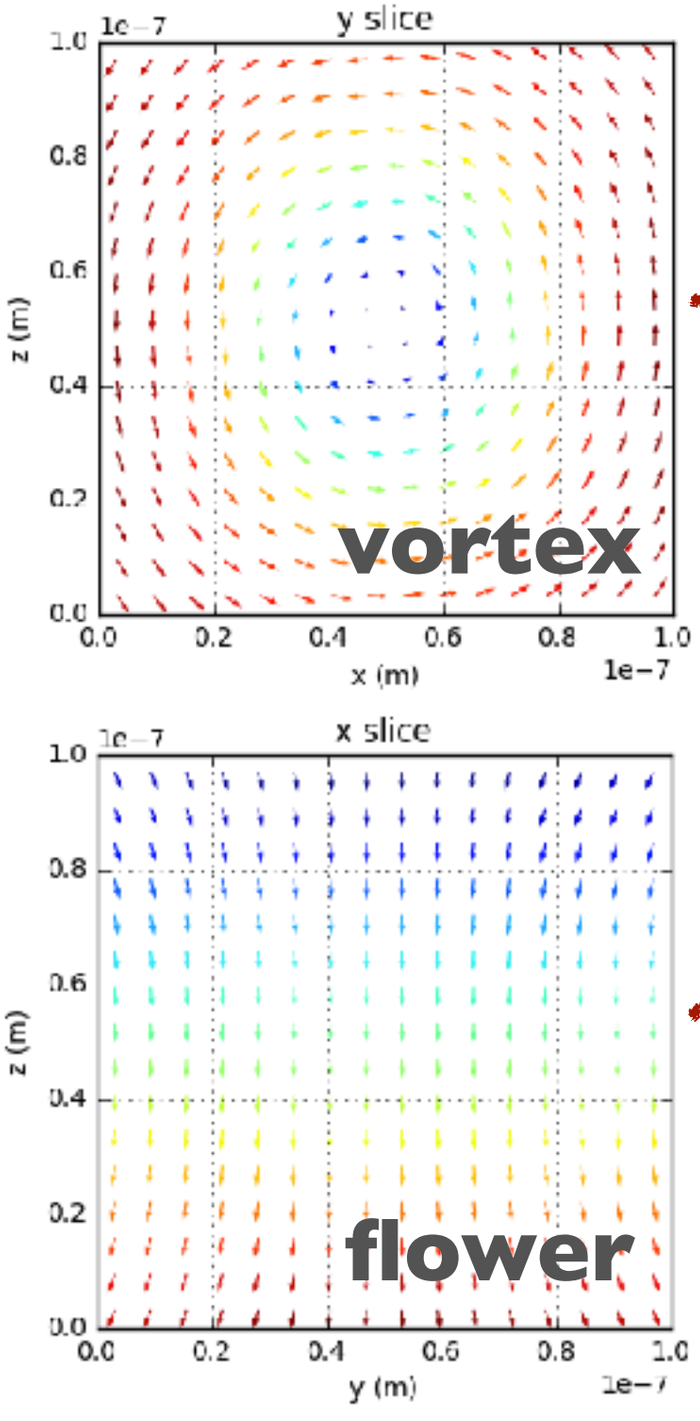
GUI

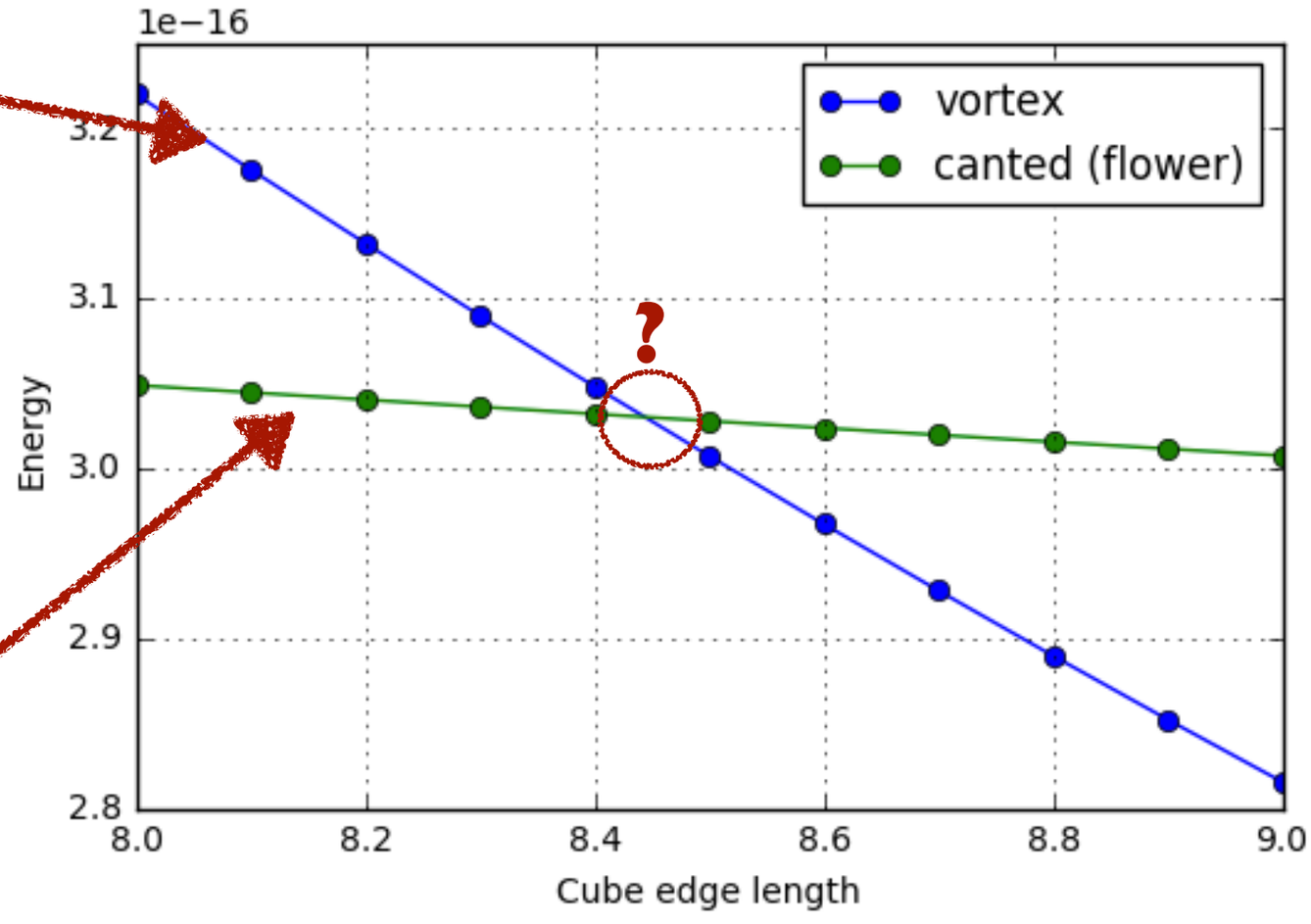
Tcl config file

OOMMF workflow

Research question

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

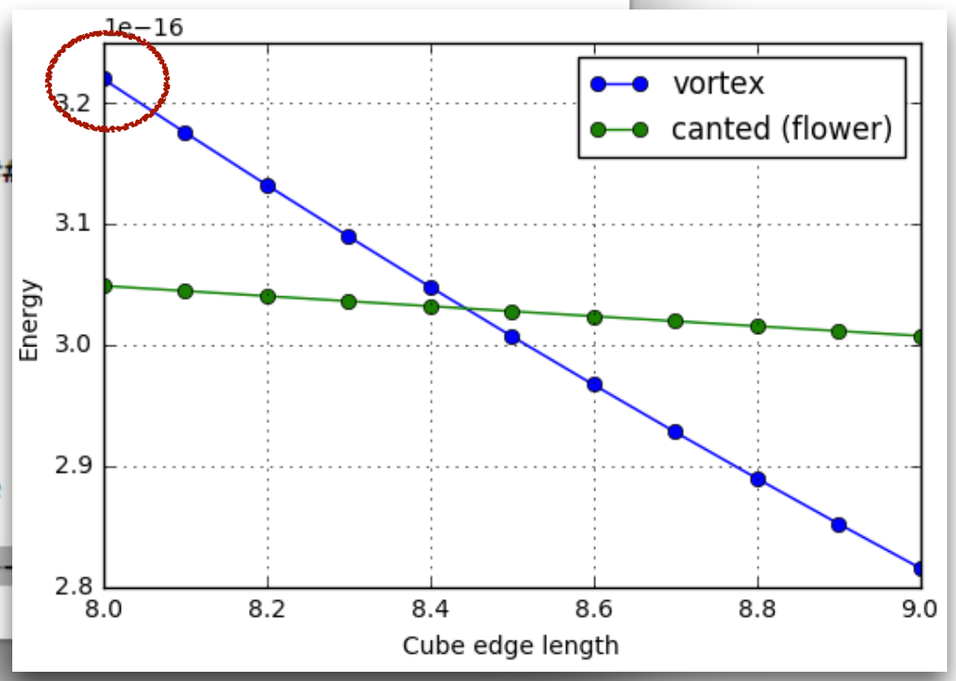




Step 1: write simulation configuration

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)--------------
```

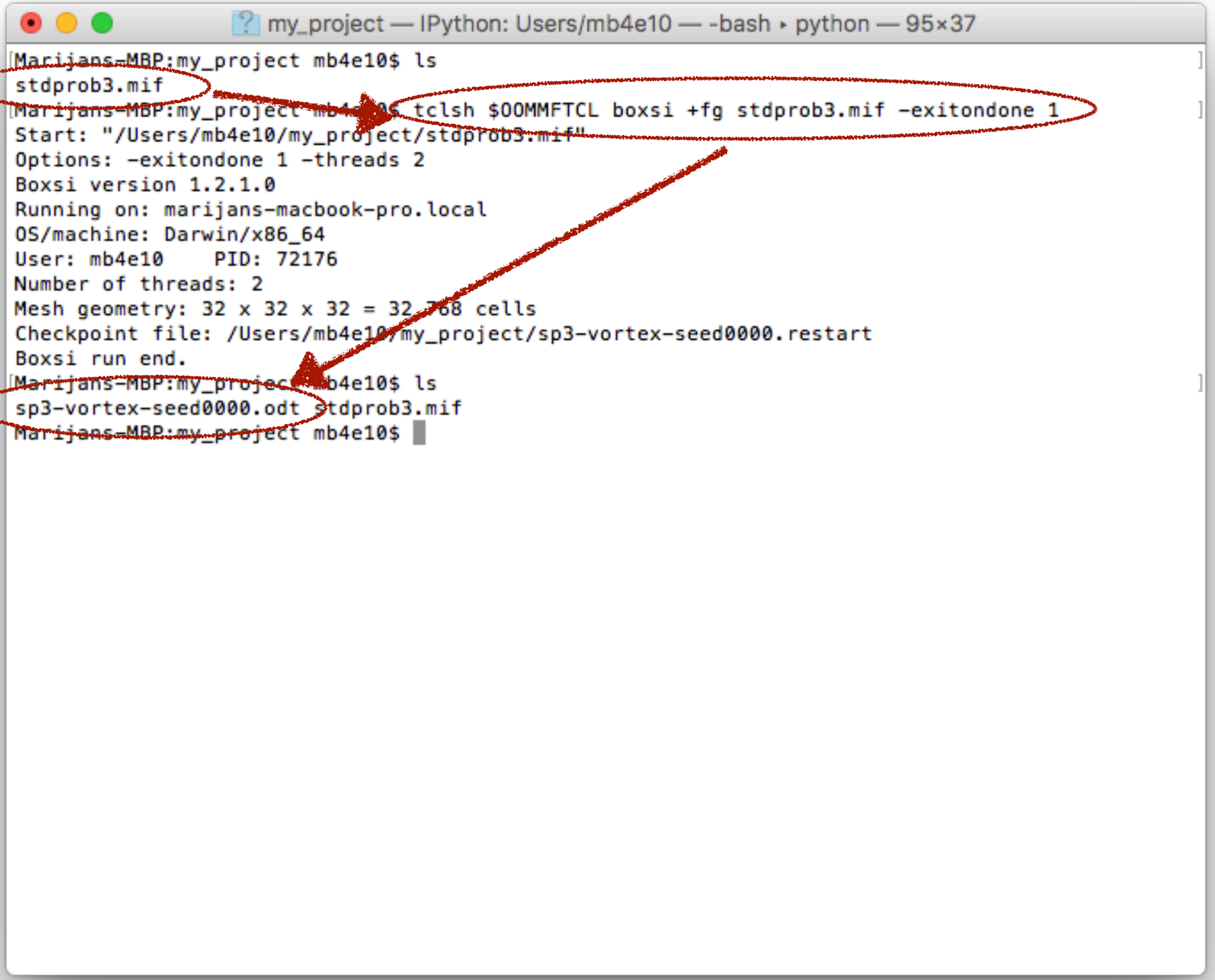


step 1: write simulation configuration

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: read data

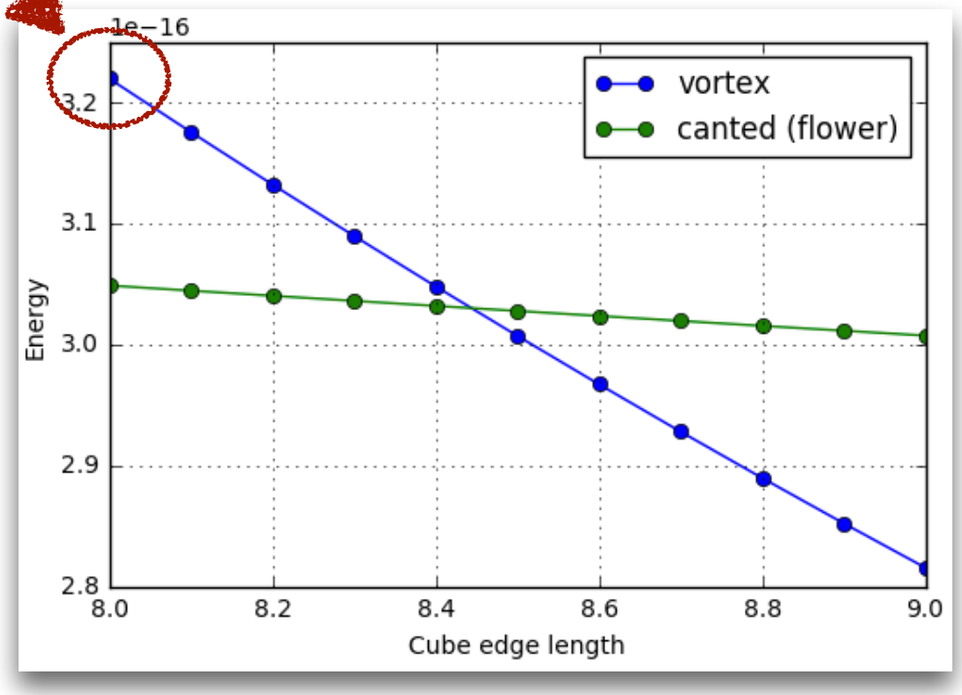
my_project — IPython: Users/mb4e10 — emacs -nw sp3-vortex-seed0000.odt — 95×37

```
# 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")
```

repeat simulations...

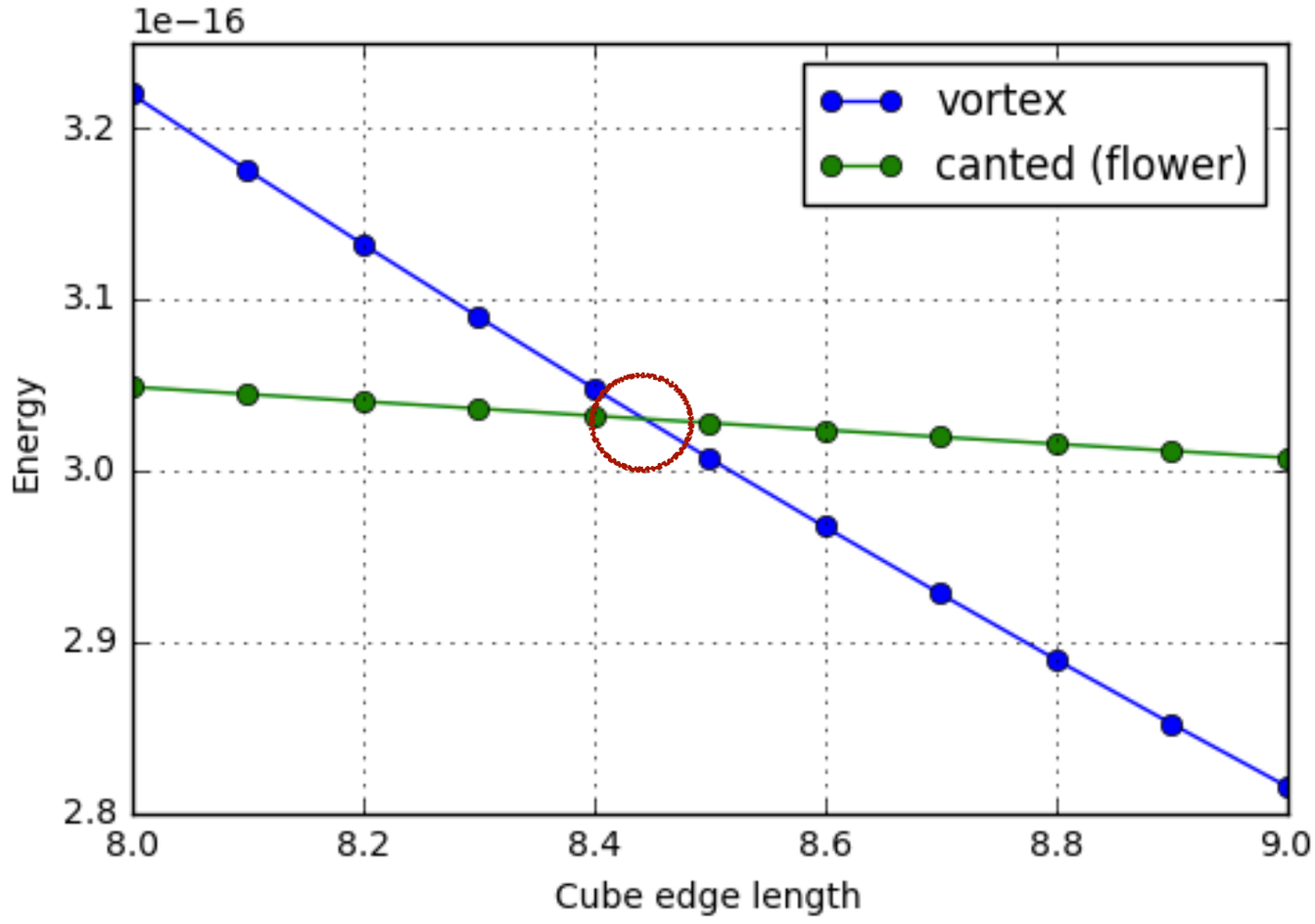
L	flower	vortex
8.0	?	3.23×10^{-16}
8.1	?	?
8.2	?	?
8.3	?	?
8.4	?	?
8.5	?	?
8.6	?	?
8.7	?	?
8.8	?	?
8.9	?	?
9.0	?	?



We have to repeat steps 1, 2, and 3 to obtain other 21 points

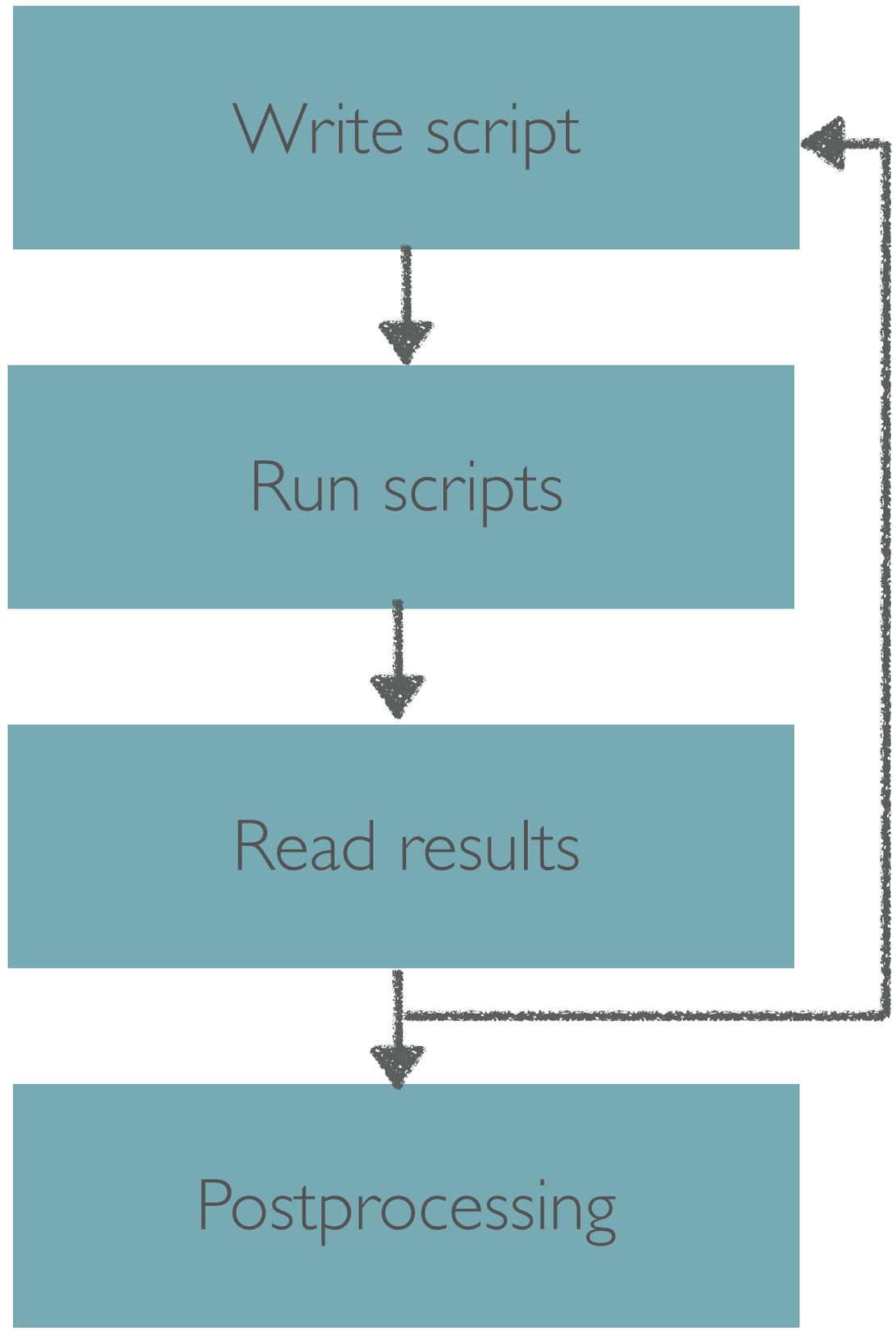
postprocessing

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



- Find crossing

Workflow



Issues with (OOMMF) workflow

- Time consuming
- Keeping log of all steps that were run and in what order
- Separate post processing scripts
- Reproducibility?

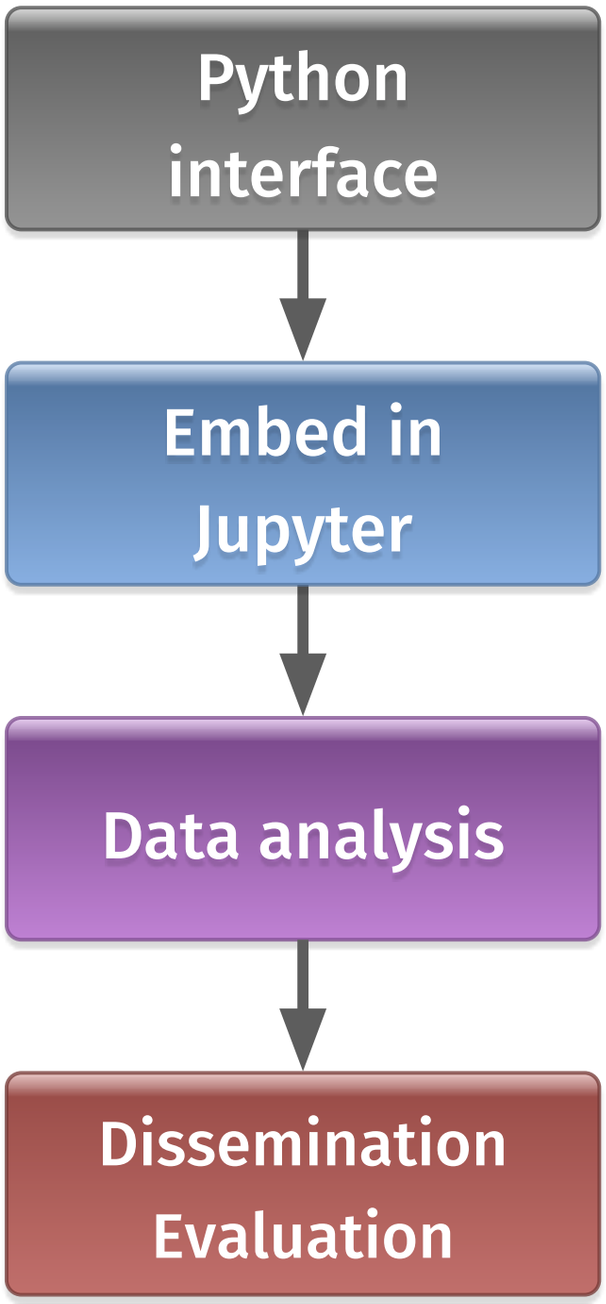
JOOMMF

JOOMMF

- Micromagnetic Virtual Research Environment
- Enable running OOMMF simulations in Jupyter notebook
- Jupyter + OOMMF = JOOMMF
- Communicate to OOMMF via Python interface (T3.8): Domain specific language embedded in general purpose language [1]

[1] Marijan Beg, Ryan A. Pepper, Hans Fangohr: *User interfaces for computational science: a domain specific language for OOMMF embedded in Python*, AIP Advances **accepted**, arxiv:1608.04876 (2017)

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

JOOMMF example

- in Jupyter -

[Live demo in Notebook]

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

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	8.53911 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

Benefits

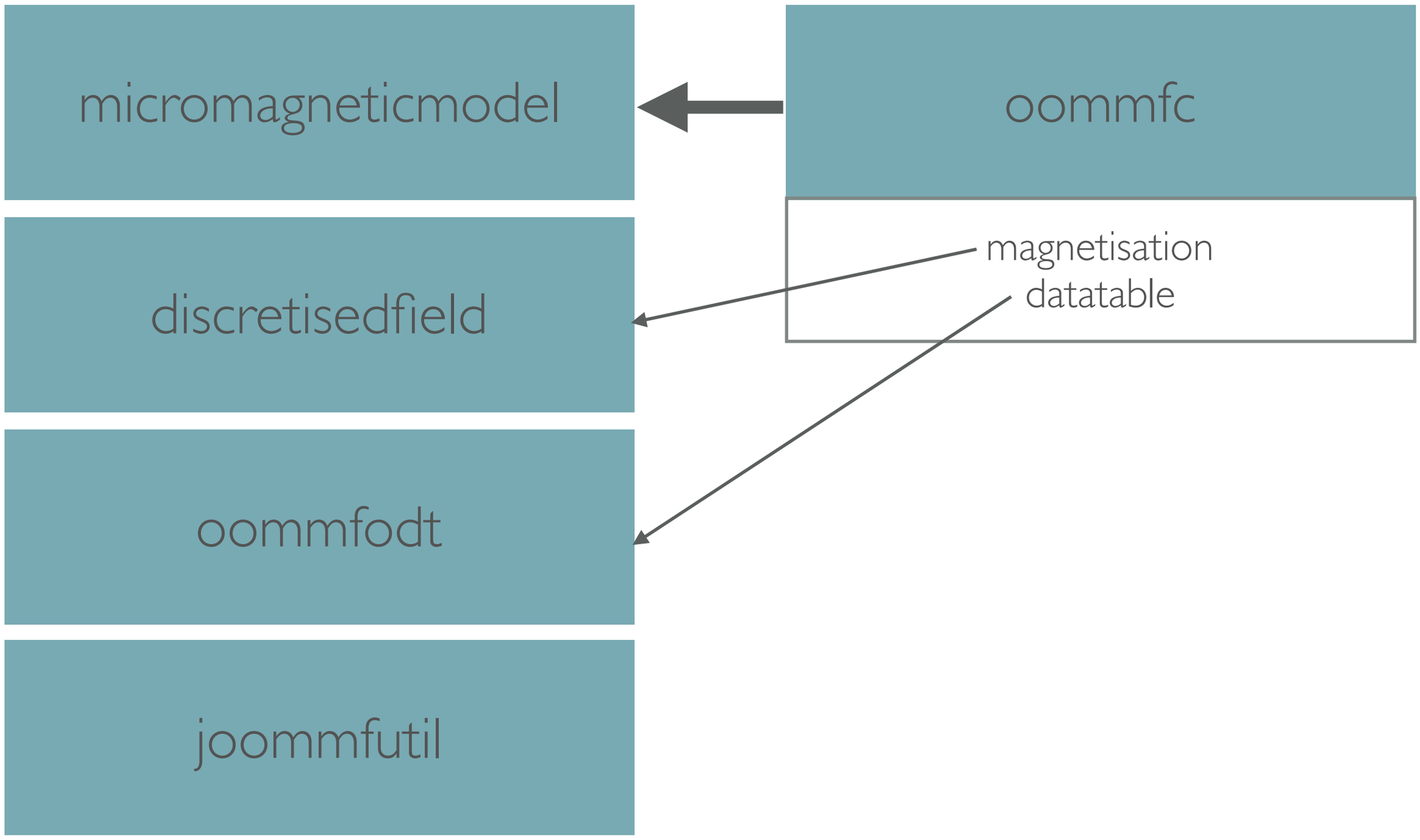
- Documentation, computation, visualisation in the same notebook
- The entire workflow is contained in a single document
- Self documenting, reproducible, NBVAL
- Easy to share, publish, and collaborate
- Executable tutorial for OOMMF Software

Micromagnetic model example

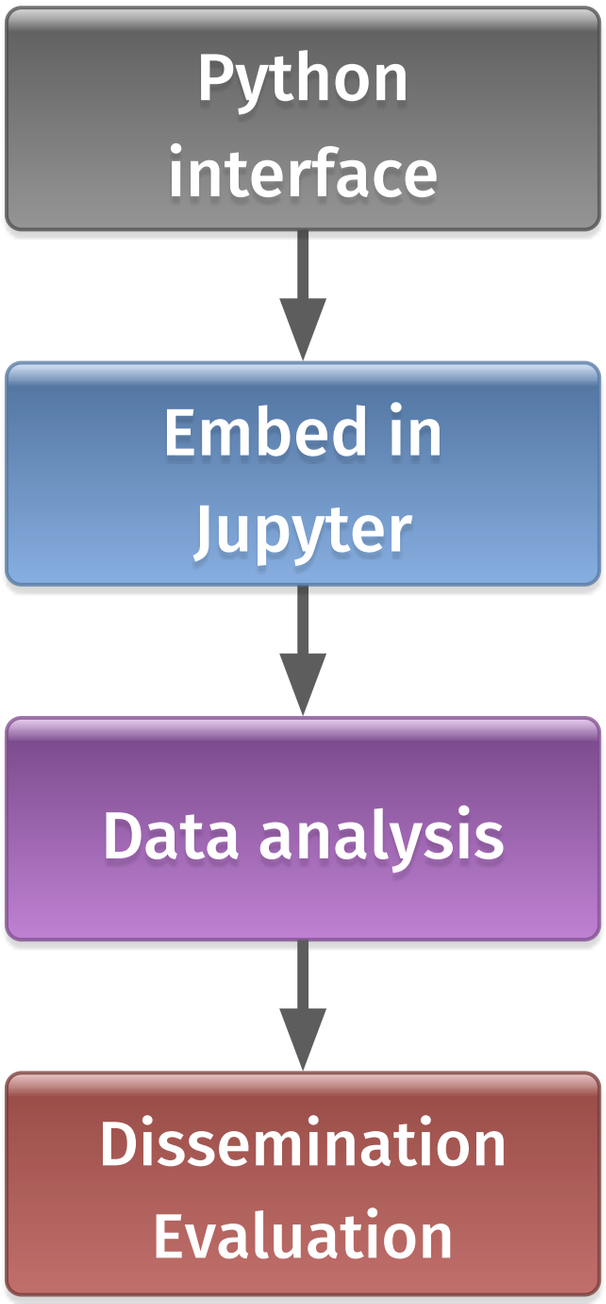
- in jupyter -

[Live demo in Notebook]

JOOMMF structure



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

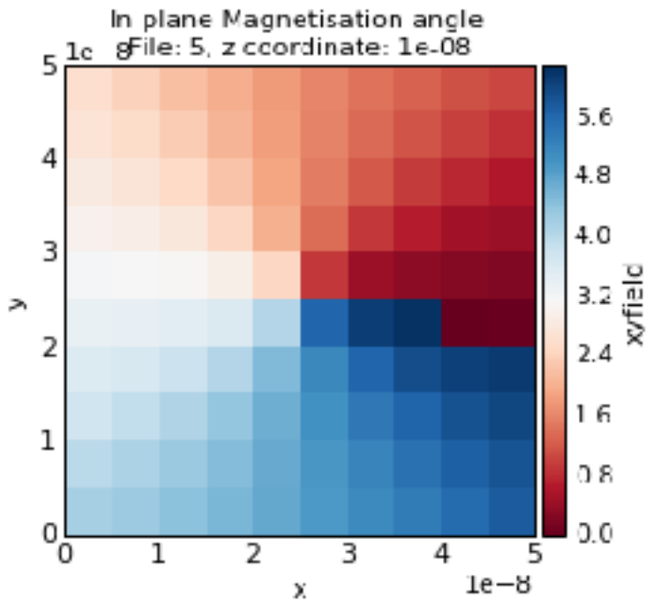
T2.8 - workshops

T7.4 - evaluation

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

Summary

- Virtual Research Environment allows us to have documentation (text and equations), code, code outputs (text, figures, tables) in a single file

- URL: joommf.github.io

- Domain specific language embedded in Python: benefit of using already existing language and libraries for data analysis and visualisation:
  Python interface and embedded Domain Specific Language:
  Beg *et al. arXiv* 1609.07432 (2016)

- email: h.fangohr@soton.ac.uk

Acknowledgements

- Contributors: Marijan Beg, Ryan Pepper, Thomas Kluyver, Hans Fangohr

- 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) and

  - The Gordon and Betty Moore Foundation through Grant GBMF #4856, by the Alfred P. Sloan Foundation and by the Helmsley Trust.