Computational Micromagnetics – OpenDreamKit demonstrator

Hans Fangohr

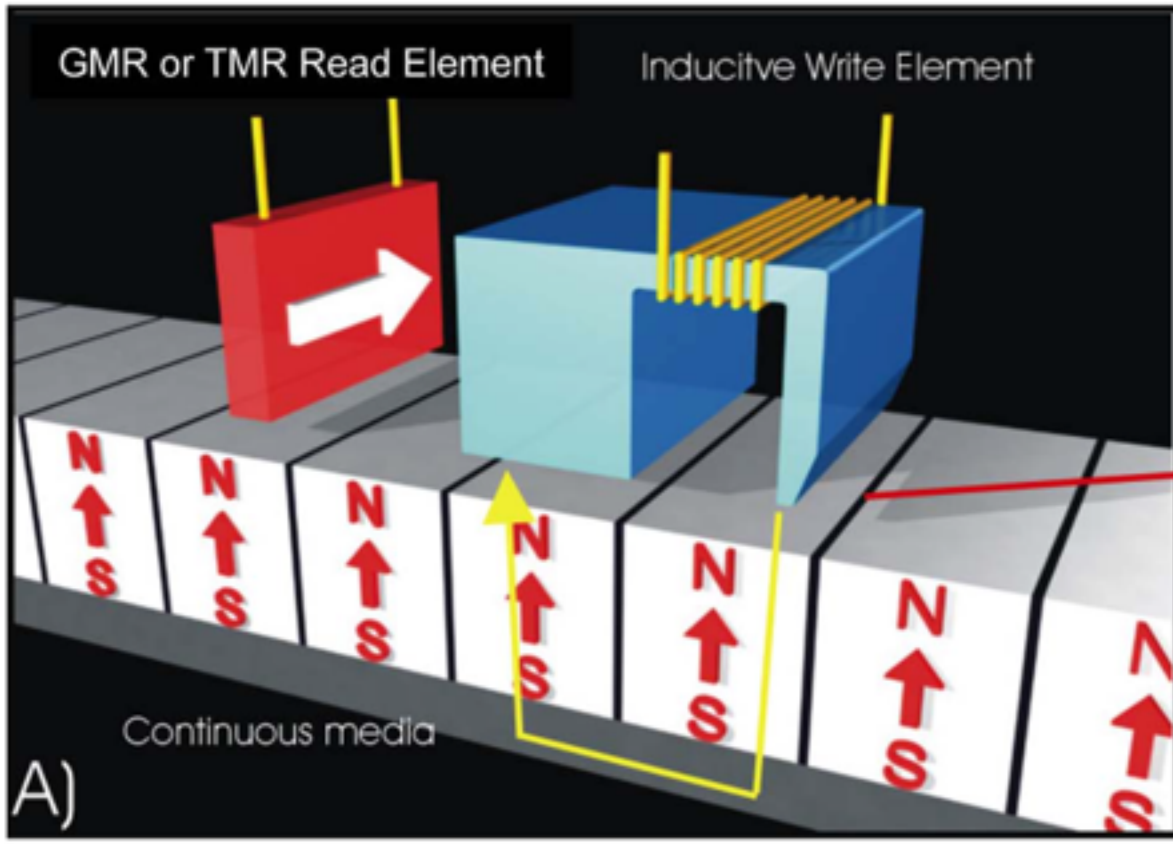
University of Southampton

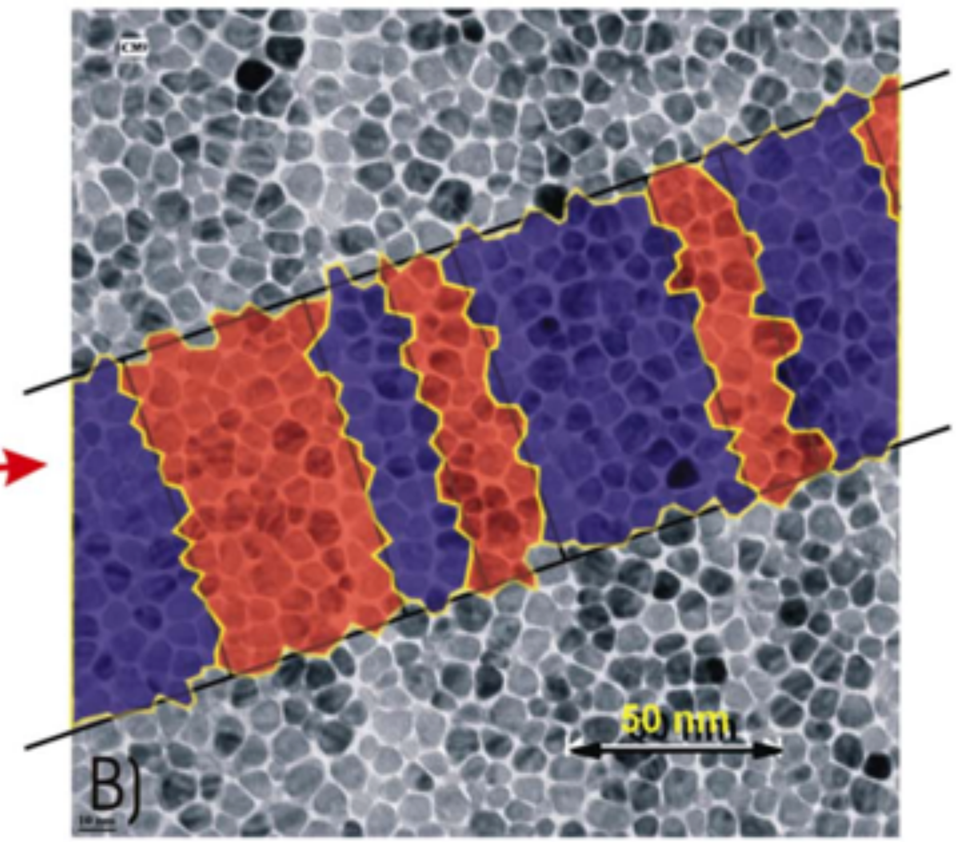
What is micromagnetics?

- Study of magnetism at length scales of ~1 nm to ~10 μm, timescales 10 ps to 100 ns
- Physics comes from simplified Maxwell Equations
- Nanotechnology applications
  - magnetic data storage (hard disk)
  - cancer therapy
  - non destructive testing
  - electromagnetic wave generator & magnonics
  - low energy magnetic logic (spintronics)
- Interesting complex system with tuneable parameters and experiments
- Large community (annual magnetism meetings ~2000 participants)

HGST 4TB

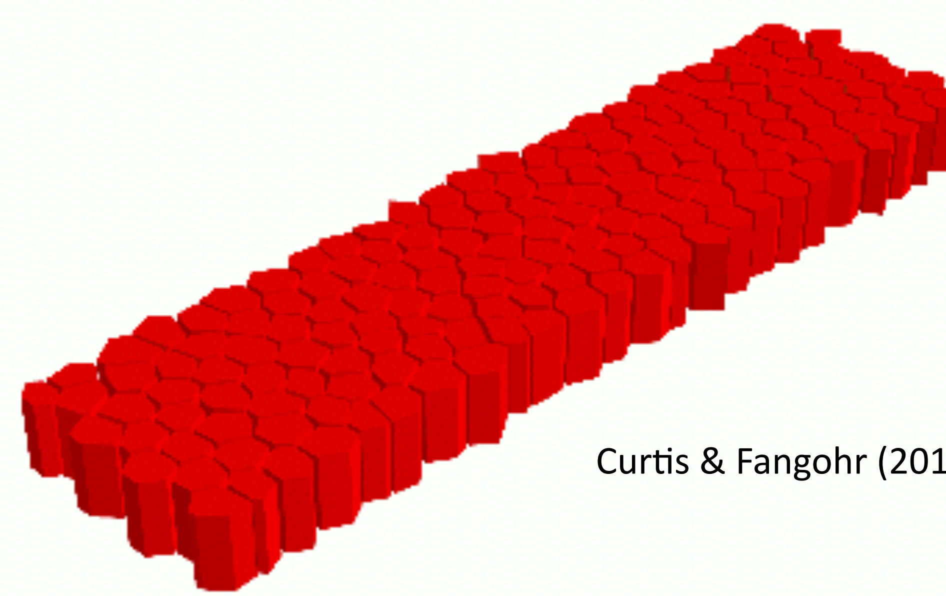
Example 1: Magnetic recording simulation





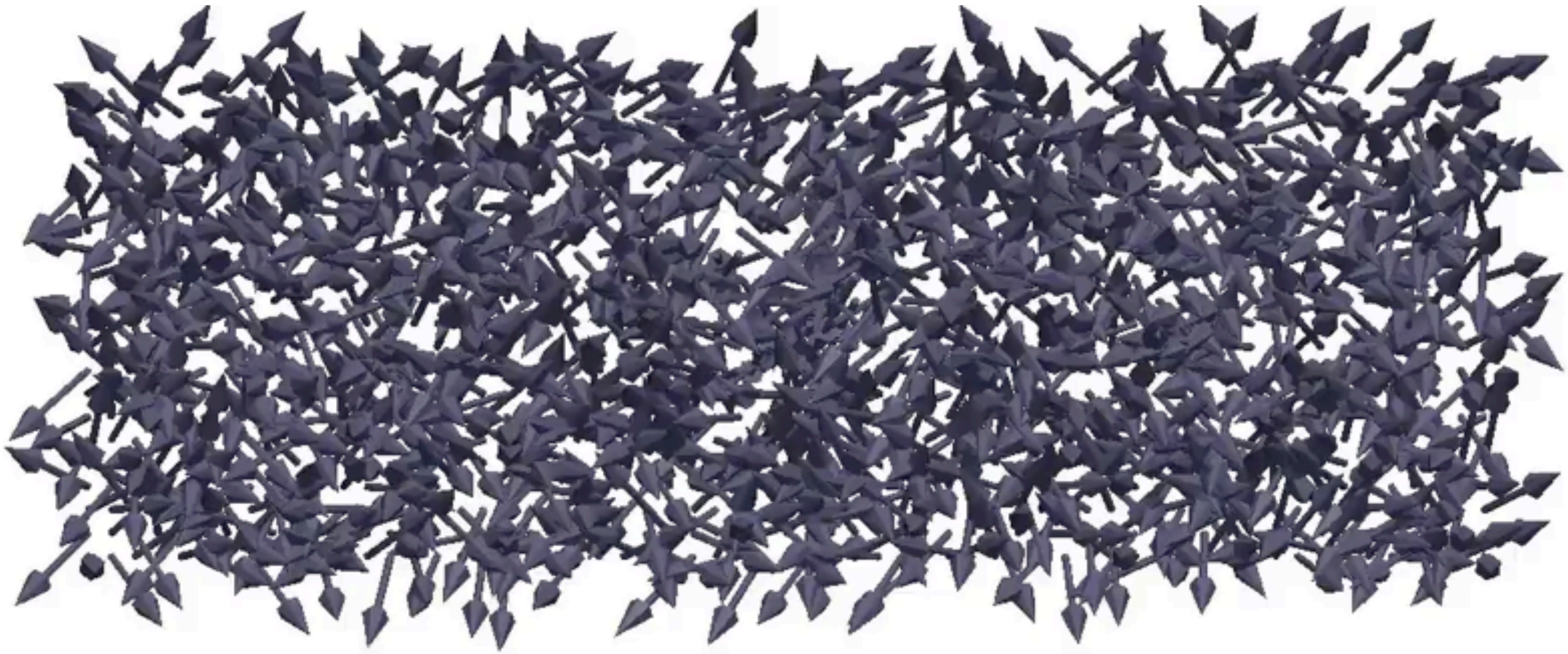
E. Dobisz et. al., Proceedings of IEEE TransMag 96, 1836 (2008)

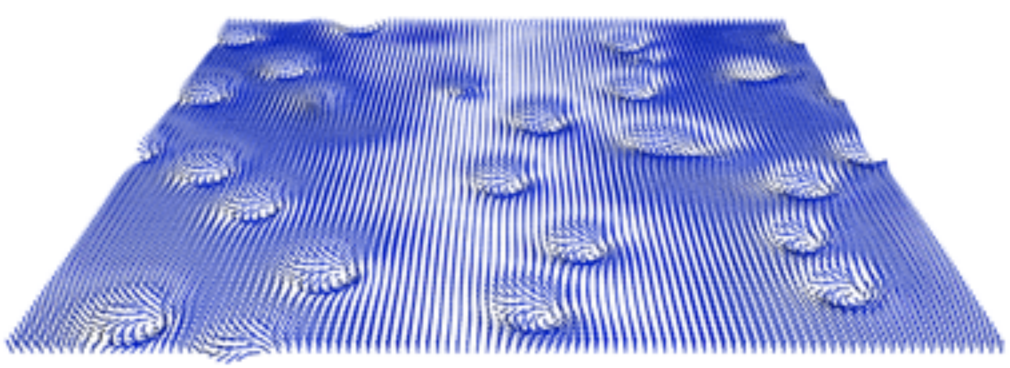
- Grains can be magnetised
- Grain size is between 5 nm – 7nm diameter
- ~100 grains per bit

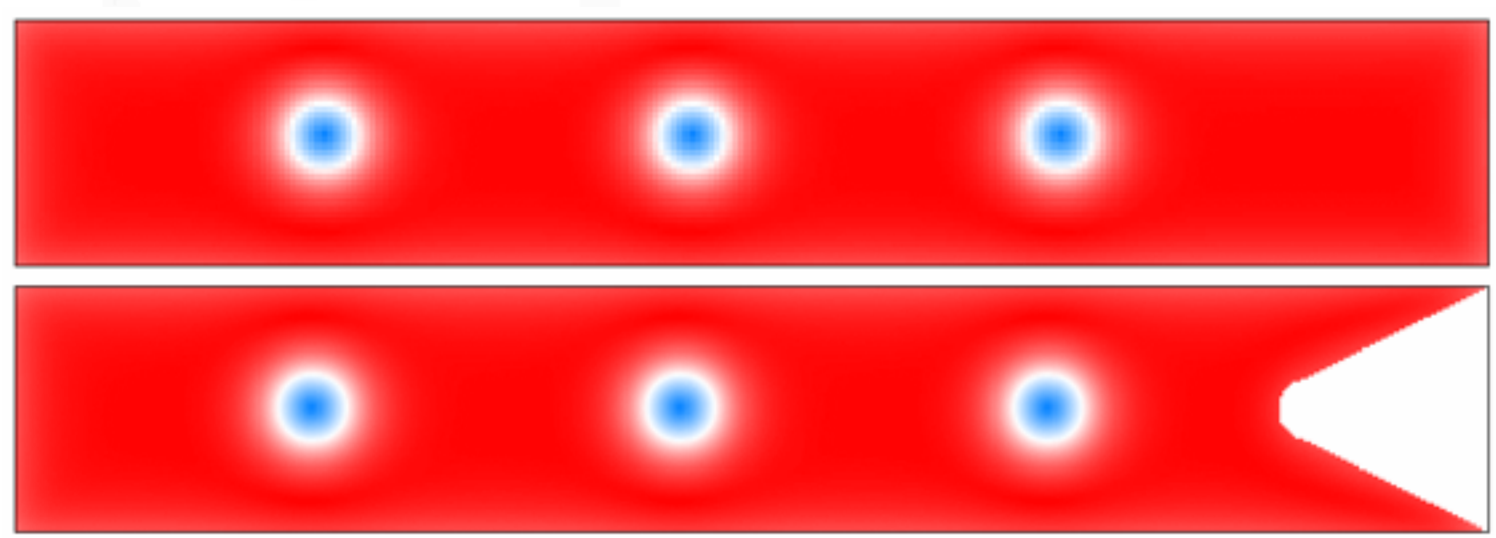


Curtis & Fangohr (2010)

Example 2: Skyrmions (future magnetic recording system?)



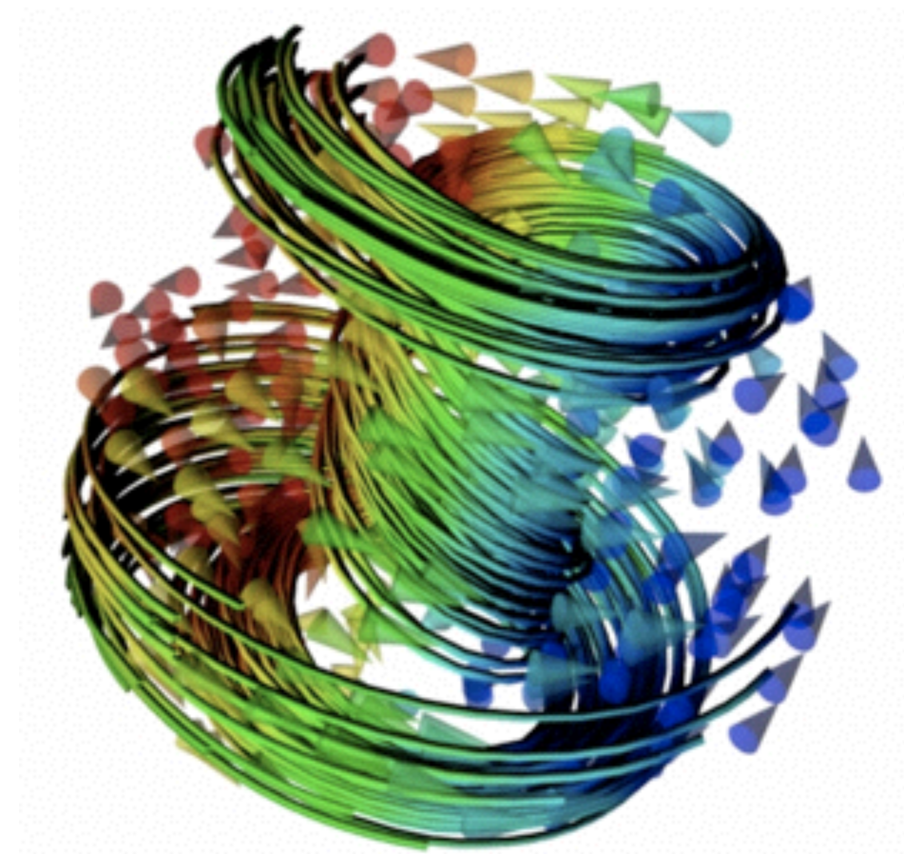




Micromagnetic model

- Magnetisation is described by a vector field m:

$$\mathbf{m}(\mathbf{r}) \in \mathbb{R}^3, \qquad \mathbf{r} \in \mathbb{R}^3$$



- We have an equation of motion

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

- **f** is complicated.
- Computing **f** involves solving PDEs
- Computationally hard: multiple length and time scales

Micromagnetic model

- **Energy density**

$$w(\mathbf{m}) = \underbrace{A(\nabla\mathbf{m})^2}_{\text{exchange}} + \overbrace{D\mathbf{m}\cdot(\nabla\times\mathbf{m})}^{\text{DMI}} - \underbrace{\mu_0 M_\mathrm{s}\mathbf{m}\cdot\mathbf{H}}_{\text{Zeeman}} + \overbrace{w_\mathrm{d}}^{\text{demagnetisation}}$$

- **Effective field**

$$H_\mathrm{eff}(\mathbf{m}) = -\frac{1}{\mu_o M_\mathrm{s}}\frac{\delta w(\mathbf{m})}{\delta\mathbf{m}}$$

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

- **LLG equation**

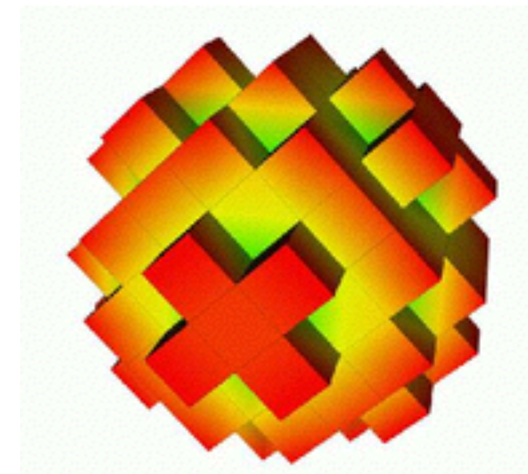
$$\frac{\partial\mathbf{m}}{\partial t} = \underbrace{\gamma^*\mathbf{m}\times\mathbf{H}_\mathrm{eff}}_{\text{precession}} + \overbrace{\alpha\mathbf{m}\times\frac{\partial\mathbf{m}}{\partial t}}^{\text{damping}} + \underbrace{u(|\mathbf{m}|-1)V(\mathbf{m})}_{\text{norm correction}}$$

What simulation tools are out there?

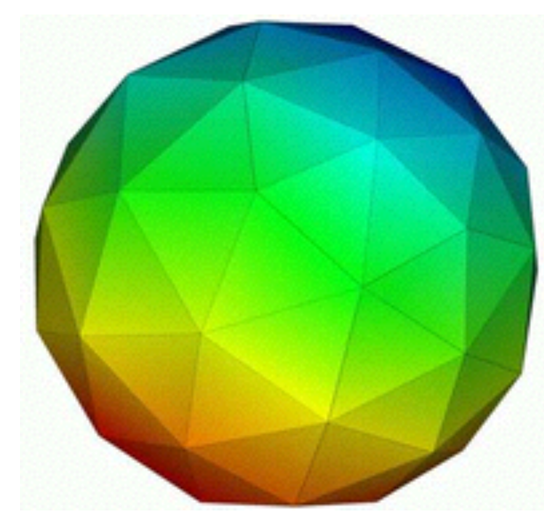
- Object Oriented MicroMagnetic Framework (OOMMF):

  finite differences, most widely used, with Tcl/Tk interface

- Several other packages, most of them using Python as the user interface, including Nmag (nmag.soton.ac.uk)



Finite Differences



Finite Elements

OOMMF:
Object-Oriented MicroMagnetic Framework

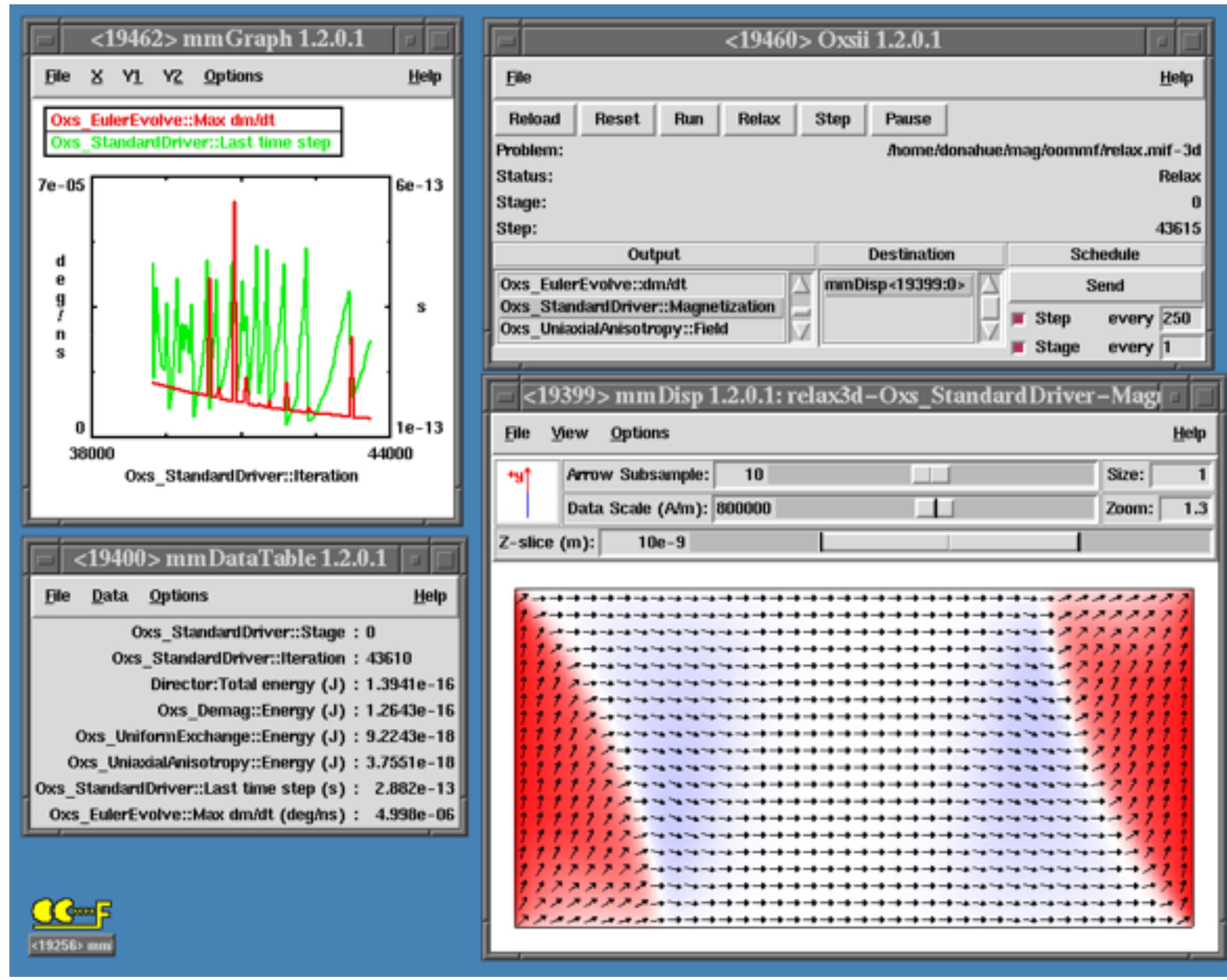
- Originates from the US's National Institute of Standards and Technology (NIST, Maryland), around 2000

- Code in Public Domain (http://math.nist.gov/oommf/)

- Nearly 2000 papers citing OOMMF, large user community

OOMMF Technology

- C++ core routines, linked to Tk/Tcl
- Tcl syntax used to configure simulation
- Tk provides graphical user interface (GUI)
- Can be fully scripted
- Modestly parallelised (OpenMP)



Tcl configuration file

```
# MIF 2.1

Specify Oxs_BoxAtlas:atlas {
  xrange {0 30e-9}
  yrange {0 30e-9}
  zrange {0 100e-9}
}

Specify Oxs_RectangularMesh:mesh {
  cellsize {2e-9 2e-9 2e-9}
  atlas Oxs_BoxAtlas:atlas
}

Specify Oxs_UniformExchange:exc {
  A 1.3e-11
}

Specify Oxs_Demag:demag { }

Specify Oxs_EulerEvolve:evolver {
  alpha 0.5
  gamma_G 2.211e5
}
```

```
Specify Oxs_UniformVectorField:m0Vec {
    norm  1
    vector { 1 0 1 }
}

Specify Oxs_TimeDriver {
  evolver :evolver
  mesh :mesh
  Ms 8.6e5
  m0 m0Vec
  stopping_time 5e-11
}
```

Proposed work for OpenDreamKit

- Wrap up C++ core of OOMMF with Python library
- Integrate Python-enabled OOMMF into IPython notebook
- Use of Widgets (GUI) to support problem definition and postprocessing
- Seamless integration of scripted (command driven) and GUI-based input / analysis
- Harvest benefits of the notebook: documentation, reproducibility, sharing, communication, ...

Mock up Notebook integration

In [1]:
```
import oommf                        # Access oommf as Python module
Py = oommf.materials.permalloy     # Material from database

# Define the geometry:
my_geometry = oommf.geometry.Cuboid((0,0,0), (30, 30, 100), unitlength=1e-9)

# Create a simulation object
sim = oommf.Simulation(my_geometry, cellsize=5e-9, material=Py)

sim.m = [1, 1, 0]                  # initialise magnetisation uniformly
```

In [2]:
```
sim                                # Show simulation info
```

Out[2]:
```
Simulation: Py(Fe80Ni20).
        Geometry: Cuboid corner1 = (0, 0, 0), corner2 = (30, 30, 100).
                  Cells = [6, 6, 20], total=720.
```

In [3]:
```
sim.advance_time(1e-9)             # Solve LLG for 0.1ns
```

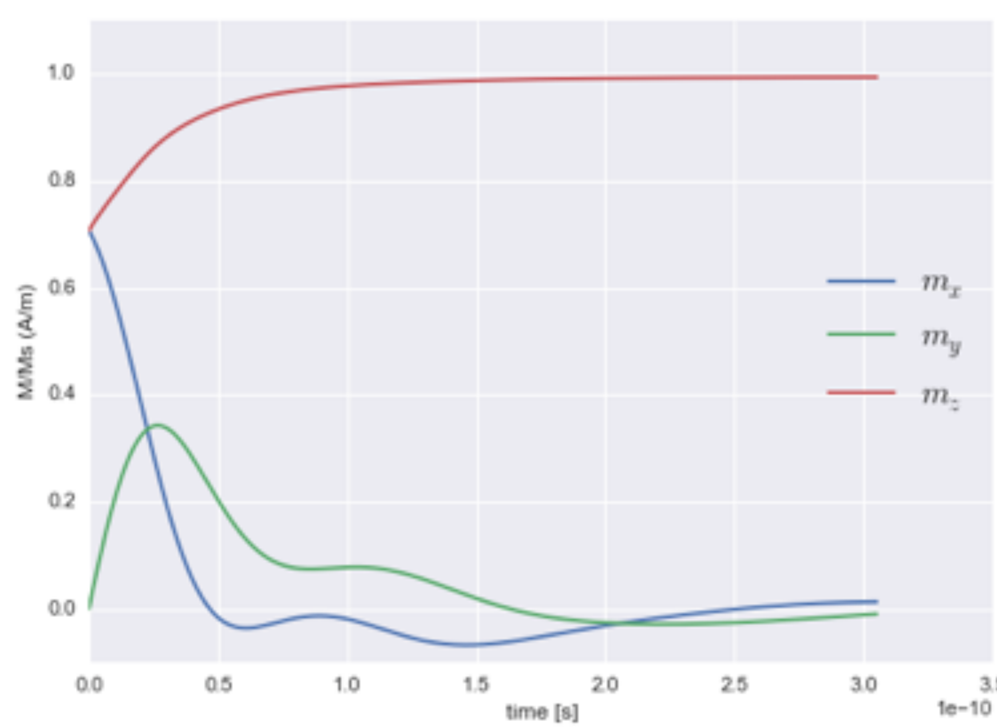
```
Integrating ODE from 0.0s to 1e-09s
```

In [4]:
```
sim.advance_time(3e-9)             # Solve LLG for another 0.2 ns
```

```
Integrating ODE from 1e-09s to 3e-09s
```

In [5]:
```
data = oommf.DataTable()           # Open ODT file
data.m_of_t()                      # and show m(t)
```

Out[5]:



In [6]:
```
sim                                # show simulation state again
```

Out[6]:
```
Simulation: Py(Fe80Ni20).
        Geometry: Cuboid corner1 = (0, 0, 0), corner2 = (30, 30, 100).
                  Cells = [6, 6, 20], total=720.
        Current t = 3e-09s
```

More details:

- Interactive visualisation and analysis in notebook
- 3d visualisation (OpenGL / VTK / Vispy?)
- Computational steering from Notebook?
- Executable documentation and tutorial for computational Micromagnetics
- Provide demo server similar to tmpnb.org
- Engage magnetics community, including delivery of OOMMF-Notebook training for scientists at international magnetism meetings