Towards *Unified Micromagnetic Modelling* with Ubermag

Hans Fangohr[*,1,2], Martin Lang[1,2], Sam Holt[1], Swapneel Pathak[1], Marijan Beg[3]

[1]Max Planck Institute Structure and Dynamics of Matter (Hamburg, Germany)
[2]University of Southampton (Southampton, UK)
[3]Imperial College (London, UK)

*hans.fangohr@mpsd.mpg.de, @ProfCompMod@fosstodon.org

SPP2137 Skyrmionics Retreat Hamburg, 13 Sept to 15 September 2023

slides at https://s.gwdg.de/7ExWfu

Outline

Introduction

Introduction and Motivation

- Micromagnetic simulations widely used
  - OOMMF > 3500 citations
  - mumax^3 > 2800 citations

- Many *micromagnetic calculators* available, for example: OOMMF, Magpar, Nmag, MicroMagnum, Fidimag, Finmag, mumax^3, Boris, Magnum.np, ...

Introduction and Motivation

- Micromagnetic simulations widely used
  - OOMMF > 3500 citations
  - mumax^3 > 2800 citations

- Many *micromagnetic calculators* available, for example: OOMMF, Magpar, Nmag, MicroMagnum, Fidimag, Finmag, mumax^3, Boris, Magnum.np, ...

Idea Unified Micromagnetic Modelling

- specify micromagnetic problem once
- can choose which micromagnetic calculator to use

Using a micromagnetic simulation

Computational research workflow (Example: OOMMF)

1. Decide what physics problem needs to be solved.
2. Translate physics into OOMMF syntax.
3. Run OOMMF simulation to compute results.

Using a micromagnetic simulation

Computational research workflow (Example: OOMMF)

1. Decide what physics problem needs to be solved.
2. Translate physics into OOMMF syntax.
3. Run OOMMF simulation to compute results.
4. Read simulation output files.

Using a micromagnetic simulation

Computational research workflow (Example: OOMMF)

1. Decide what physics problem needs to be solved.
2. Translate physics into OOMMF syntax.
3. Run OOMMF simulation to compute results.
4. Read simulation output files.
5. Analyse and visualise results.

Using a micromagnetic simulation

Computational research workflow (Example: OOMMF)

1. Decide what physics problem needs to be solved.
2. Translate physics into OOMMF syntax.
3. Run OOMMF simulation to compute results.
4. Read simulation output files.
5. Analyse and visualise results.

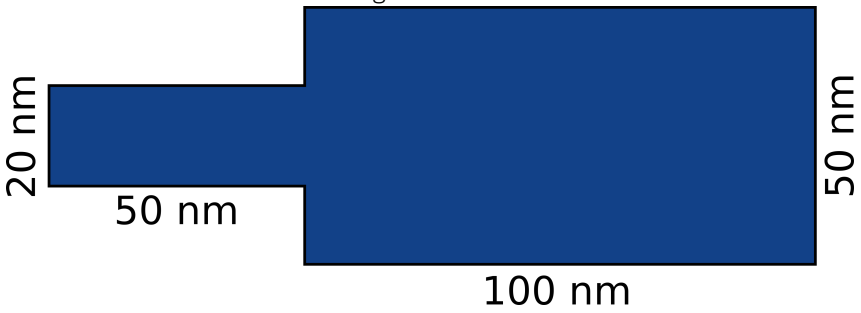
Preview: Ubermag

Ubermag automates steps 2 to 4.

Ubermag

Example: Study domain wall skyrmion conversion (Zhou and Ezawa, 2014)

1nm thick Cobalt on a substrate inducing DMI



Zhou, Y., & Ezawa, M. 2014 Nature Communications 5, 8.
`https://doi.org/10.1038/ncomms5652`

Define the problem: physics model

[2]:
```
system = mm.System(name='dw_pair_conversion')

system.energy = (mm.Exchange(A=15e-12)
                 + mm.DMI(D=3e-3, crystalclass='Cnv_z')
                 + mm.UniaxialAnisotropy(K=0.5e6, u=(0, 0, 1)))

system.energy
```

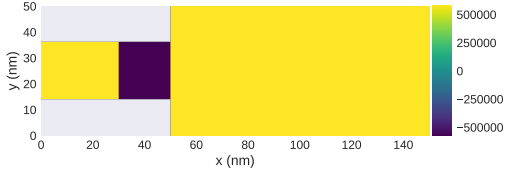
[2]:
$$-A\mathbf{m}\cdot\nabla^2\mathbf{m} + D(\mathbf{m}\cdot\nabla m_z - m_z\nabla\cdot\mathbf{m}) - K(\mathbf{m}\cdot\mathbf{u})^2$$

[3]:
```
system.dynamics = (
    mm.Precession(gamma0=mm.consts.gamma0)  # gyromagnetic ratio (m/As)
    + mm.Damping(alpha=0.3)  # Gilbert damping
    + mm.ZhangLi(u=400, beta=0.5)  # STT parameter
)

system.dynamics
```

[3]:
$$-\frac{\gamma_0}{1+\alpha^2}\mathbf{m}\times\mathbf{H}_{\text{eff}} - \frac{\gamma_0\alpha}{1+\alpha^2}\mathbf{m}\times(\mathbf{m}\times\mathbf{H}_{\text{eff}}) - \frac{1+\alpha\beta}{1+\alpha^2}\mathbf{m}\times(\mathbf{m}\times(\mathbf{u}\cdot\boldsymbol{\nabla})\mathbf{m}) - \frac{\beta-\alpha}{1+\alpha^2}\mathbf{m}\times(\mathbf{u}\cdot\boldsymbol{\nabla})\mathbf{m}$$

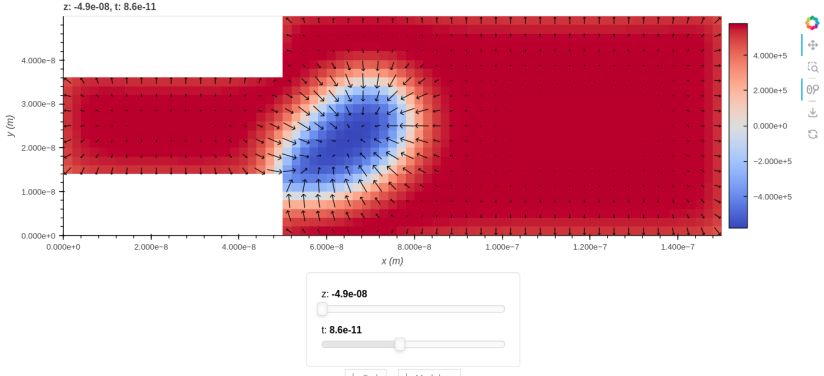
Define the problem: geometry and initial magnetisation

[4]:
```python
Ms = 5.8e5  # saturation magnetisation (A/m)

region = df.Region(p1=(0, 0, -50e-9), p2=(150e-9, 50e-9, 0))
mesh = df.Mesh(region=region, cell=(2e-9, 2e-9, 2e-9))

def Ms_fun(pos):  # Sample shape
    x, y, z = pos
    if x < 50e-9 and (y < 15e-9 or y > 35e-9):
        return 0
    else:
        return Ms

def m_init(pos):  # Magnetisation direction
    x, y, z = pos
    if 30e-9 < x < 50e-9:
        return (0.1, 0.1, -1)
    else:
        return (0.1, 0.1, 1)

system.m = df.Field(mesh, nvdim=3, value=m_init, norm=Ms_fun, valid='norm')

system.m.z.sel('z').mpl.scalar()
```



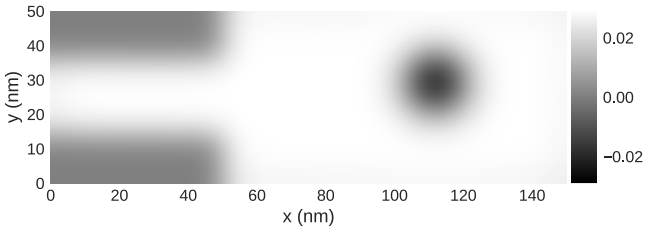
(Interactive) data analysis

[6]:
```
import micromagneticdata as mdata
data = mdata.Data(system.name)

data[-1].hv(kdims=['x', 'y'],
            vector_kw={'n' : (45, 15)},
            scalar_kw = {'clim':(-Ms, Ms), 'cmap': 'coolwarm'})
```



Simulate experimental signature (mag2exp)

```
[7]: import mag2exp
holo_image = mag2exp.x_ray.holography(system.m, fwhm=(10e-9,10e-9))
holo_image.mpl.scalar(cmap='gray', interpolation='spline16', symmetric_clim=True)
```



→ See poster Samuel Holt et al. for **mag2exp**.

Ubermag overview (very short)

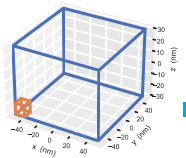
Researcher

$$E = \int_V \left[-A\mathbf{m} \cdot \nabla^2 \mathbf{m} - \mu_0 M_\mathrm{s} \mathbf{m} \cdot \mathbf{H} + w_\mathrm{d} \right] \mathrm{d}V$$

Ubermag

Automated backend

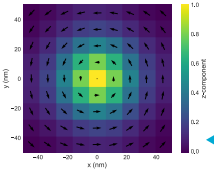
```
In [1]: system = mm.System(name='vortex')
        system.energy = mm.Exchange(A=1e-11) + mm.Demag()
        system.dynamics = (mm.Precession(c.gamma0)
                           + mm.Damping(alpha=0.2))
```




```
In [2]: mesh = df.Mesh(p1=(-50e-9, -50e-9, -30e-9),
                       p2=(50e-9, 50e-9, 30e-9),
                       cell=(10e-9, 10e-9, 10e-9))
        system.m = df.Field(mesh, dim=3, norm=4e5,
                            value=vortex_fun)
```

```
In [3]: md.drive(system)
```

```
In [4]: system.m.orientation.sel('z').mpl()
```



Ubermag design

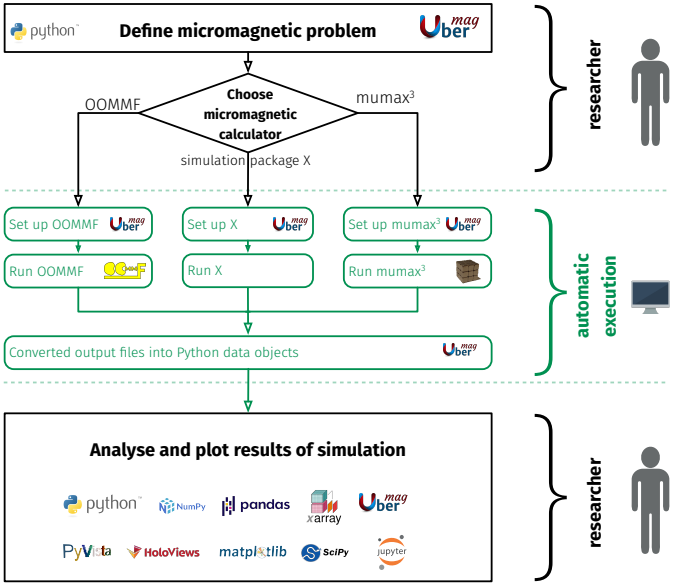
Domain-specific language for *micromagnetic problems*

- express physics problem through Python library `micromagneticmodel`
- cannot *solve* the problem (but is *machine readable*)
- *separates* the problem definition from the (numerical) solving of the problem
  - problem description is independent of simulation package syntax

Beg, Pepper, Fangohr: *User interfaces for computational science: a domain specific language for OOMMF embedded in Python*, DOI 10.1063/1.4977225, eprint pdf (2017)

Unified Micromagnetic Modelling (UMM)

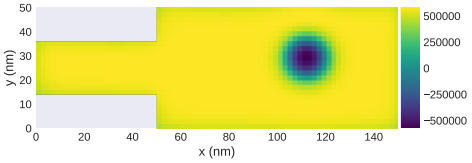
Unified Micromagnetic Modelling (UMM) with Ubermag



Ubermag can use different calculators (OOMMF and mumax^3)

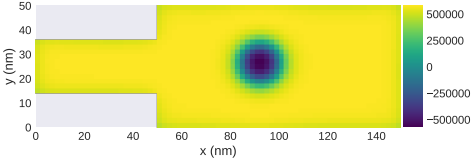
[5]:
```
import oommfc as mc
md = mc.MinDriver()
md.drive(system)

td = mc.TimeDriver()
td.drive(system, t=0.2e-9, n=200, verbose=2)
system.m.z.sel('z').mpl.scalar()
```

```
Running OOMMF (ExeOOMMFRunner)[2023/08/31 15:33]... (1.5 s)
Running OOMMF (ExeOOMMFRunner)[2023/08/31 15:33] took 25.8 s
```



[5]:
```
import mumax3c as mc
md = mc.MinDriver()
md.drive(system)

td = mc.TimeDriver()
td.drive(system, t=0.2e-9, n=200, verbose=2)
system.m.z.sel('z').mpl.scalar()
```

```
Running mumax3 (ExeMumax3Runner)[2023/09/05 14:59]… (0.7 s)
Running mumax3 (ExeMumax3Runner)[2023/09/05 14:59] took 3.4 s
```



Advantages Unified Micromagnetic Modelling

1. Can *choose* micromagnetic calculator depending on available hardware:
   - NVIDIA GPU $\rightarrow$ mumax^3
   - CPU $\rightarrow$ OOMMF

Advantages Unified Micromagnetic Modelling

1. Can *choose* micromagnetic calculator depending on available hardware:
   - NVIDIA GPU $\rightarrow$ mumax^3
   - CPU $\rightarrow$ OOMMF
2. Can *compare and validate* different micromagnetic calculators easily

Advantages Unified Micromagnetic Modelling

1. Can *choose* micromagnetic calculator depending on available hardware:
   - NVIDIA GPU $\rightarrow$ mumax^3
   - CPU $\rightarrow$ OOMMF
2. Can *compare and validate* different micromagnetic calculators easily
3. Can *combine* unique feature of calculator 1 with unique feature of calculator 2

Advantages Unified Micromagnetic Modelling

1. Can *choose* micromagnetic calculator depending on available hardware:
   - NVIDIA GPU $\rightarrow$ mumax^3
   - CPU $\rightarrow$ OOMMF
2. Can *compare and validate* different micromagnetic calculators easily
3. Can *combine* unique feature of calculator 1 with unique feature of calculator 2
4. Can *extend* micromagnetic calculations in (Python-based) data science environment

Advantages Unified Micromagnetic Modelling

1. Can *choose* micromagnetic calculator depending on available hardware:
   - NVIDIA GPU $\rightarrow$ mumax^3
   - CPU $\rightarrow$ OOMMF
2. Can *compare and validate* different micromagnetic calculators easily
3. Can *combine* unique feature of calculator 1 with unique feature of calculator 2
4. Can *extend* micromagnetic calculations in (Python-based) data science environment
5. Scientist does not need to learn the syntax of each micromagnetic calculator

Issues

Design of a unified micromagnetic modelling interface

difficult if the different calculators (e.g. OOMMF and mumax^3) have different features

Issues

Design of a unified micromagnetic modelling interface

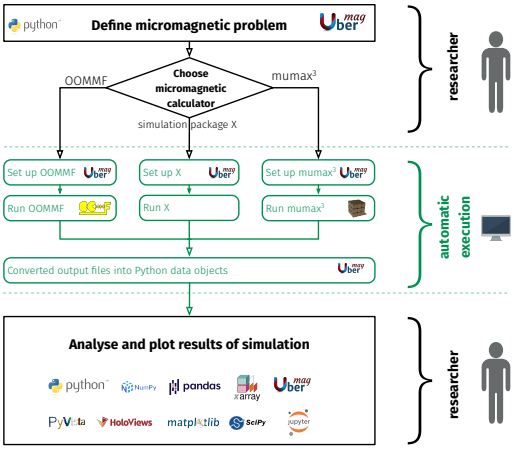
difficult if the different calculators (e.g. OOMMF and mumax^3) have different features

Discovering inconsistencies

ability to easily compare different simulation packages reveals discrepancies: are these important?

Summary

Summary Unified Micromagnetic Modelling (UMM)



1. UMM makes research more efficient

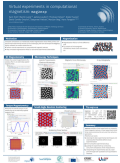
- automate what is possible
- define physics problem once, re-use in all simulation packages

2. Ubermag provides initial implementation for UMM

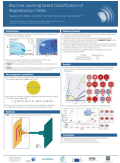
- supports* OOMMF and mumax^3
- contributions are welcome
  https://github.com/ubermag/ubermag

* compatibility overview: `https://ubermag.github.io/documentation/compatibility.html`

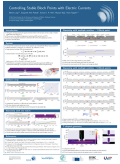
Ubermag applications in poster session



Sam Holt, Simulate experimental measurements with `mag2exp`



Swapneel Pathak, ML-based classification of magnetisation configurations



Martin Lang, Controlling stable Bloch points with electric currents

Open positions

MAgnetic Multiscale MOdelling Suite (MaMMoS)

- project January 2024 to December 2027
- modelling magnetic systems
- contribute to green power generation
- machine learning
- software engineering
- positions to be advertised in the coming months
- please get in touch if interested

Relevant Links

These slides are available at https://s.gwdg.de/7ExWfu

Ubermag software

https://ubermag.github.io/

Marijan Beg, Martin Lang and Hans Fangohr, *Ubermag: Toward More Effective Micromagnetic Workflows*, in IEEE Transactions on Magnetics, vol. 58, no. 2, pp. 1-5, Art no. 7300205, 10.1109/TMAG.2021.3078896 (2022)

Domain specific language for micromagnetic problems

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, 10.1063/1.4977225 (2017)

Acknowledgments

This work was supported by the Horizon 2020 OpenDreamKit Project (#676541) and the EPSRC Programme grant on Skyrmionics (#EP/N032128/1).

Contact

- hans.fangohr@mpsd.mpg.de
- @ProfCompMod@fosstodon.org

Appendix

Current status for unified micromagnetic modelling in Ubermag

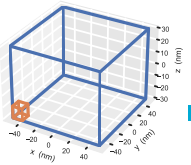
- Ubermag can use OOMMF and mumax^3 as micromagnetic calculator
  - not all features complete, see

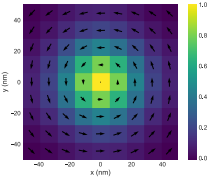
    `https://ubermag.github.io/documentation/compatibility.html`
- Ubermag design is open to include more micromagnetic calculators such as magnum.np, micromagnum, Boris, ...
- finite-difference-based discretisation is supported best

Ubermag overview (very short)

Researcher

$$E = \int_V \left[-A\mathbf{m} \cdot \nabla^2 \mathbf{m} - \mu_0 M_\mathrm{s} \mathbf{m} \cdot \mathbf{H} + w_\mathrm{d} \right] \mathrm{d}V$$



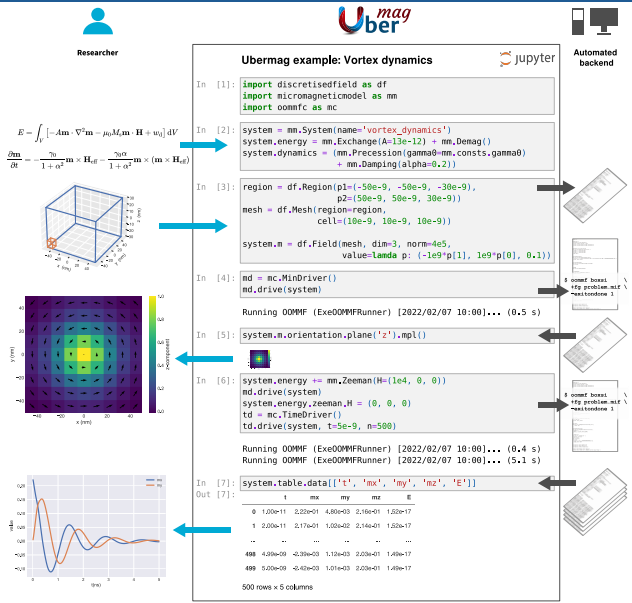


```
In [1]: system = mm.System(name='vortex')
        system.energy = mm.Exchange(A=1e-11) + mm.Demag()
        system.dynamics = (mm.Precession(c.gamma0)
                           + mm.Damping(alpha=0.2))

In [2]: mesh = df.Mesh(p1=(-50e-9, -50e-9, -30e-9),
                       p2=(50e-9, 50e-9, 30e-9),
                       cell=(10e-9, 10e-9, 10e-9))
        system.m = df.Field(mesh, dim=3, norm=4e5,
                            value=vortex_fun)

In [3]: md.drive(system)

In [4]: system.m.orientation.sel('z').mpl()
```

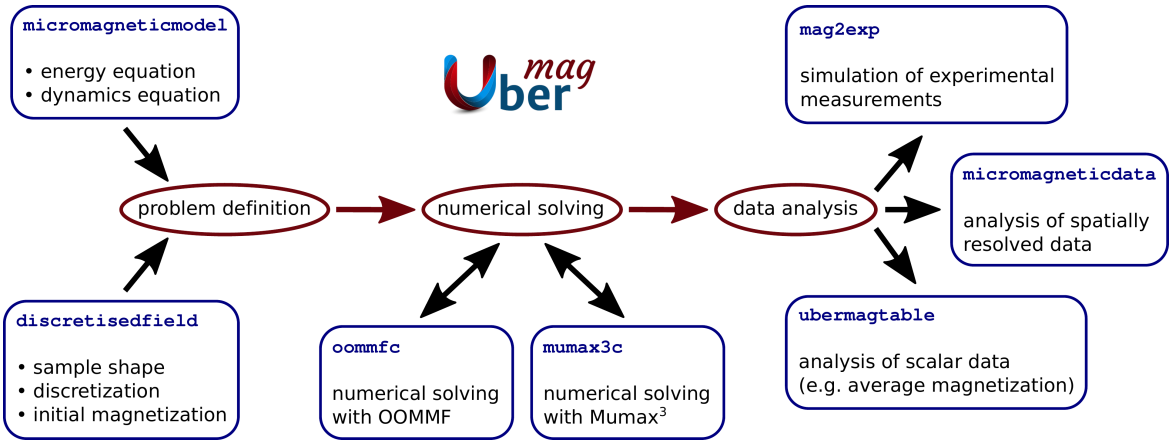
Automated backend

```
$ oommf boxsi  \
  +fg problem.mif \
  -exitondone 1
```

Ubermag overview (longer)



Ubermag workflow and packages



Installation options

- Ubermag:
  - `pip install ubermag` (Python only, needs separate install of OOMMF)
  - `conda install -c conda-forge ubermag` (contains OOMMF)
- OOMMF:
  - install binary package (Windows) from NIST
  - compile from source (Linux, OSX) from NIST
  - conda (`conda install -c conda-forge oommf`)
  - Docker (`docker pull oommf/oommf`)
  - Spack (`spack install oommf`)