# OOMMF Python interface and Jupyter integration: the JOOMMF project

AD-02 Hans Fangohr, Ryan Pepper, Marijan Beg

2015-12-07

# Overview – status quo and vision

Current state:

- OOMMF is fantastic tool, widely used
- Jupyter notebook (`http://jupyter.org`) is next generation workflow tool for computational science

The Jupyter-OOMMF (J-OOMMF) project will:

- develop a Python interface for OOMMF
- make OOMMF usable in the Jupyter notebook

Talk introduces the project (and Jupyter).

# OOMMF interface

- Simulation definition via
  - Tk-Graphical User Interface
  - Tcl (mif-files)
- Essentially one simulation per mif file, carrying out:
  - time integration (dynamics)
  - energy minimisation
  - hysteresis loop calculation
- Possible (further) improvement
  - driving OOMMF through mif files results in multiple runs, multiple mif files, bash scripts, for some kind of problems

# OOMMF Python Interface

- Imagine we could write arbitrary Python code that uses OOMMF:

```
import oommf                          # Access oommf as Python module
Py = oommf.materials.permalloy        # permalloy (Py) parameters
my_geometry = oommf.geometry.Cuboid((0,0,0), (30, 30, 100))
sim = oommf.Simulation(my_geometry, cellsize=5e-9, material=Py)
sim.m = [1, 1, 0]                     # initialise magn. uniformly
sim.advance_time(1e-9)                # solve LLG for 1 nano second
```

- Then we can
  - create and run multiple OOMMF simulations from a single (python) file
  - carry out (embedded) data analysis in same environment
  - benefit from the existing scientific Python libraries and tools
  - achieve better science in less time
- J-OOMMF project will provide Python interface for OOMMF

# OOMMF Python interface: example application

Scan parameter space: does the mesh size affect results?

- `for-loop` over cell size:

```
for cellsize in [1e-9, 2e-9, 3e-9, 4e-9, 5e-9]:
    sim = oommf.Simulation(my_geometry, cellsize, Py)
    sim.advance_time()
    # <compute and save some entity for this cell size>

# <plot entity as function of cellsize>
```

  - self contained study
  - flexible use of multiple oommf simulations

# History: micromagnetics embedded in Python language

- Nmag[1], released 2007, is prototype for embedding micromagnetics into Python
- The Nmag manual[2] provides many more examples
- Subsequently followed by MicroMagnum, Magnum.fe, Magnum.fd, Finmag, (Magpar)
- Common approach in computational science (outside micromagnetics)

[1] IEEE TransMag 43, 6, 2896-2898 (2007);
[2] `http://nmag.soton.ac.uk`

# Jupyter Notebook: Introduction 1/2

- Jupyter is as web-browser hosted notebook that combines
  - text and equations (with $\LaTeX$)
  - executable code and outputs
  - graphics

  through "cells", that can be executed in any order.

# Interactive example Jupyter Notebook (output 1/2)

In [1]:
```
%matplotlib inline
from numpy import exp, cos, linspace
import pylab
```

## Visualise $f(t) = \exp(-\alpha t)\cos(\omega t)$

In [2]:
```
def f(t, alpha, omega):
    """Computes and returns exp(-alpha*t) * cos(omega*t)"""
    return exp(-alpha * t) * cos(omega * t)
```

We can execute the function for valuel of $\alpha$ and $\omega$:

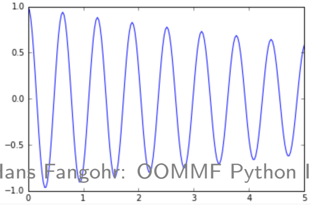
In [3]:
```
f(t=1, alpha=1, omega=10)
```

Out[3]: -0.30867716521951294

Although sometimes a plot is more instructive:

In [4]:
```
def plot_f(alpha, omega):
    xs = linspace(0, 5, 200)     # 100 points in the interval [0, 5]
    ys = f(xs, alpha, omega)
    pylab.plot(xs, ys, '-')
```

In [5]:
```
plot_f(alpha=0.1, omega=10)
```



# Interactive Example Jupyter Notebook (output 2/2)

A wide range of convenience tools, for example graphical interaction elements called *Widgets*:

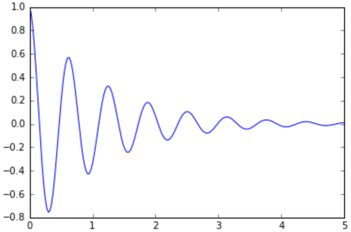
In [6]:
```
from ipywidgets import interact
```

In [7]:
```
interact(plot_f, alpha=(0, 2, 0.1), omega=(0, 20, 0.5))
```

alpha 0.9

omega 10

Out[7]:
```
<function __main__.plot_f>
```




## Conclusion

So we observe: Parameter $\alpha$ is responsible for damping, and $\omega$ for the frequency.

# Jupyter Notebook: Introduction 2/2

- Notebooks can be shared and communicated
  - saved, reloaded and re-executed
  - exported to html, latex, pdf
- Useful
  - for interactive computational analysis and exploration
  - documentation of process and results
  - reproducibility
  - creating and communicating reports / publications / theses / ...
- $\rightarrow$ Jupyter is a productivity tool

# OOMMF in the Jupyter Notebook (output 1/2)

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
```

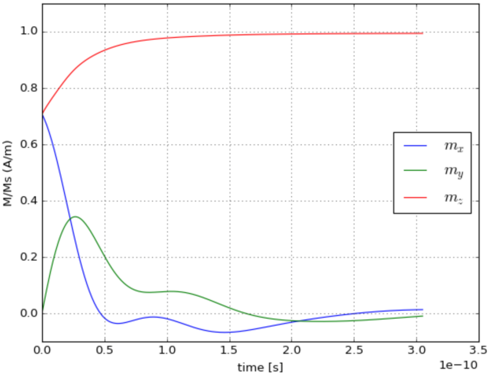
In [4]:
```
sim.advance_time(3e-9)               # Solve LLG for another 0.2 ns
```

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

# OOMMF in the Jupyter Notebook (output 2/2)

In [5]:
```
data = sim.odt()                    # access data from 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
```

# Jupyter notebook OOMMF: J-OOMMF

Vision:

- use OOMMF through python interface in Jupyter notebook
- combining widgets and GUI like elements with
- powerful scriptability
- convenient documentation and reproducibility
- $\rightarrow$ faster and better science

Managing expectations:

- project about to start in February 2016
- J-OOMMF notebook example shown today is a mock-up

# Summary

## Jupyter-hosted OOMMF → JOOMMF

- will also provide OOMMF Python interface
- Project home page: `http://joommf.github.io`
- get in touch with ideas, concerns, questions, ...


## Acknowledgements

- financial support from the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541), `http://opendreamkit.org`
- Support from UK's EPSRC Centre for Doctoral Training in Next Generation Computational Modelling (`http://ngcm.soton.ac.uk`)