Boris Computational Spintronics

User manual, *version 2.8*

Dr. Serban Lepadatu, 13th *July 2020*

Abstract

This manual describes Boris Computational Spintronics, a multi-physics and multi-scale research software designed to solve three-dimensional magnetization dynamics problems, coupled with a self-consistent charge and spin transport solver, heat flow solver with temperature-dependent material parameters, in arbitrary multi-layered structures and shapes. The computational routines run both on central processors and graphics processors using the CUDA platform. In addition to simple user control, advanced simulation configurations are made possible using Python scripts. The software is open source and currently runs on Windows 7, Windows 10, and Linux-based 64-bit operating systems, and was programmed using C++17, CUDA C, and Python.

boris-spintronics.uk/download
https://github.com/SerbanL/Boris2

Disclaimer

Boris Computational Spintronics is a freely available research, design and educational software. The author assumes no responsibility whatsoever for its use by other parties, and makes no guarantees, expressed or implied, about its quality, reliability, or any other characteristic. If using Boris for published research please use a relevant reference as given in the “Selected Publications using Boris” section (an article describing Boris, which may be used as a complete reference in the future, is pending).

Contents

Installation - Windows 6

Installation – Linux-based OS 7

Overview 9

Tutorial 0 – Quick-Start 10

Tutorial 1 – Introduction 24

Tutorial 2 – Data Output 31

Tutorial 3 – Further Data Output 39

Tutorial 4 – Domain Walls 43

Tutorial 5 – Domain Wall Movement and Data Processing 49

Tutorial 6 – ODE Control and Setting Shapes 53

Tutorial 7 – Magnetocrystalline Anisotropy 57

Tutorial 8 – Anisotropic Magneto-Resistance 60

Tutorial 9 – Scripting using Python 67

Tutorial 10 – Current-Induced Domain Wall Movement 69

Tutorial 11 – Oersted Fields 72

Tutorial 12 – Surface Exchange, Multi-Layered Demagnetization and CUDA ... 76

Tutorial 13 – Dzyaloshinskii-Moriya Exchange 82

Tutorial 14 – Simulations with non-zero temperature 85

Tutorial 15 – Thermal Fields 88

Tutorial 16 – Heat Flow Solver and Joule Heating 89

Tutorial 17 – Spin Transport Solver 94

Tutorial 18 – Spin Hall Effect 99

Tutorial 19 – Spin Pumping and Inverse Spin Hall Effect 102

Tutorial 20 – Ferromagnetic Resonance 105

Tutorial 21 – Ferromagnetic Resonance with Spin Torques 111

Tutorial 22 – CPP-GMR 115

Tutorial 23 – Skyrmion Movement with Spin Currents 118

Tutorial 24 – Roughness and Staircase Corrections 123

Tutorial 25 – Defects and Impurities 130

Tutorial 26 – Polycrystalline and Granular Films 133

Tutorial 27 – Periodic Boundary Conditions 136

Tutorial 28 – Ultrafast Demagnetisation 138

Tutorial 29 – Magneto-Optical Effect 142

Tutorial 30 – Spin-Wave Dispersion 143

Tutorial 31 – Two-Sublattice Model 146

Tutorial 32 – Exchange Bias 152

Tutorial 33 – Magneto-Elastic Effect 153

Tutorial 34 – Atomistic Modelling 156

User-Defined Text Equations **157**

Working with OVF2 Files **163**

Materials Database **165**

Differential Equations **168**

Modules **180**

Material Parameters **197**

Commands – Essential **203**

Commands – Important **204**

Commands – Useful **205**

Commands – All (Alphabetical) **206**

Selected Publications using Boris **255**

Installation - Windows

An installer has been provided with the program and instructions therein should be followed. The program is designed to run on Windows 7 and Windows 10, 64 bit versions, and requires Microsoft Visual C++ 2017 Redistributable (x64) – included with the installer. On Windows 10 the executable (Boris.exe) must be run in compatibility mode – the installer sets this.

CUDA

To enable CUDA computations Boris requires a CUDA-enabled graphics card with CUDA compute capability 5.0 or greater. The installer detects the CUDA compute capability of the graphics card and launches the required program version. **You should always run the installed Boris.exe program**, not the separate CUDA versions found in the same directory.

The following architectures are supported by the installer package: Maxwell (sm_50, sm_53), Pascal (sm_60, sm_61, and sm_62), Volta (sm_70, sm_72). The program has not been tested on the Turing architecture (sm_75). If you have problems with the program on this architecture please let me know (SLepadatu@uclan.ac.uk).

Known Issues

Running DiagTrack and DPS (Diagnostic Policy Service) Windows services can in certain cases result in very poor performance. In this case these services should be stopped and disabled.

Installation – Linux-based OS

Extract the archive. On Linux-based OS the program needs to be compiled from source using the provided *makefile* in the extracted BorisLin directory.

Make sure you have all the required updates and dependencies:

Updates:

1. Get latest g++ compiler: *$ sudo apt install build-essential*
2. Get OpenMP: *$ sudo apt-get install libomp-dev*
3. Get CUDA: *$ sudo apt install nvidia-cuda-toolkit*
4. Get SFML: *$ sudo apt-get install libsfml-dev*
5. Get FFTW3: Instructions at http://www.fftw.org/fftw2_doc/fftw_6.html

Before running *make* you need to manually set the CUDA architecture to the correct value in the *makefile* and in the *cuBLib_Flags.h* file:

Configuration:

1. Find *cuBLib_Flags.h* file in BorisLin/BorisCUDALib directory
2. Set __CUDA_ARCH__ to the correct value (500, 600, or 700: see below)
3. Edit the *makefile*: find the compilation flag *-arch=sm_50* on the last line. This value needs to match the __CUDA_ARCH__ value and the architecture of your NVidia GPU as:

   __CUDA_ARCH__ 500 needs *-arch=sm_50*
   __CUDA_ARCH__ 600 needs *-arch=sm_60*
   __CUDA_ARCH__ 700 needs *-arch=sm_70*

   - *-arch=sm_50* is required for **Maxwell** architecture
   - *-arch=sm_60* is required for **Pascal** architecture
   - *-arch=sm_70* is required for **Volta** (and **Turing**) architecture

For a list of architectures and more details see: https://en.wikipedia.org/wiki/CUDA.

Installation:

1. Open terminal and go to extracted BorisLin directory.
2. **$ make compile -j N**
   (replace N with the number of logical cores on your CPU for multi-processor compilation, e.g. **$ make compile -j 16**)
3. **$ make install**

Run:

$ ./BorisLin

Advanced Configuration:

- You can compile the CUDA code in single or double precision (default is single precision). Find *cuBLib_Flags.h* file in BorisLin/BorisCUDALib directory. Edit the SINGLEPRECISION value to 0 for double precision, 1 for single precision.
- You can disable CUDA compilation entirely, which will produce an executable for CPU computations only (with CUDA compilation both CPU and GPU computations can be executed, and you can switch between computation modes with the **cuda** console command in BorisLin: **cuda 0** or **cuda 1**). Find the *CompileFlags.h* file in the BorisLin/Boris directory. Set the COMPILECUDA value to 0 to disable CUDA compilation.

Notes:

In the current version Boris runs in text mode only on Linux-based OS, thus the GRAPHICS 0 value needs to be kept in *CompileFlags.h* file. In a future version a graphical interface will be ported to Linux also.

Overview

This manual contains a set of self-teaching tutorials that guide the user through most of its functionality. The tutorials contain a number of exercises designed for users without a background in micromagnetics, and may be skipped by more advanced users. A number of examples that accompany the tutorials have also been provided in the accompanying Examples folder.

You can use Tutorial 0 as a quick-start. This tutorial contains a number of Python scripts as examples but doesn't contain in-depth explanations.

Tutorials 1 to 7 cover the basics and it is recommended all users read through them. After these you can skip to the required tutorials as needed. Tutorial 9 covers the basics of automating simulations using Python and should be used as a starting point if required. For the transport solver Tutorials 8 and 10 should be used as a starting point. Tutorials 17 to 23 cover the spin transport solver. Tutorials 14 to 16 cover the heat solver.

The equations solved are given in the Differential Equations and Modules sections. All material parameters used in these equations have been given in the Material Parameters section.

A full list of commands has been provided in the Commands section in alphabetical order. The most commonly used commands have also been outlined.

Tutorial 0 – Quick-Start

This tutorial contains a set of Python scripts for a selected number of micromagnetics problems, and are intended as a crash-course for experienced users who want to get things going very quickly. No detailed comments are provided, except for comments in the Python scripts – if you want in-depth explanations you need to read the other tutorials. All scripts are found in the Examples/Tutorial 0 folder.

The Python scripts below can be run either locally (connect as 'localhost' – this is the default), or remotely (connect using ip address of machine running Boris).

For the Python scripts below you need the NetSocks.py module in the same directory. Before executing the script Boris needs to be running; scripted communication is enabled by default, but if you've disabled it you need to re-enable it (use **scriptserver** *1* command).

The default program start-up state is a *permalloy* rectangle with dimensions 80 nm × 80 nm × 10 nm, and cubic 5 nm cellsize. The *demag*, *exchange*, and *Zeeman* modules are enabled. The LLG equation is set with the RKF45 evaluation method. To restore the program to default state at any time use the **default** console command (or alternatively send it as ns.default() from a Python script).

The default simulation stage is a *Relax* stage with $|mxh| < 10^{-4}$ stopping condition and no data saving condition.

<u>Hysteresis Loop</u>

```
import os
import sys
from NetSocks import NSClient

#setup communication with server.
ns = NSClient('localhost')
```

```
##########################################

#the working directory : same as this script file
directory = os.path.dirname(sys.argv[0]) + "/"
#restore program to default state
ns.default()
ns.chdir(directory)

##########################################

#This is based on Exercise 2.1, done entirely using a Python script

ns.meshrect([160e-9, 80e-9, 10e-9])
ns.cellsize([5e-9, 5e-9, 10e-9])

#setup two stages to sweep field up and down between -100 kA/m and +100kA/m in 100 steps, slightly off-axis.
#setstage sets a single stage, replacing the default stage
ns.setstage('Hxyz_seq')
ns.editstagevalue(0, [-100e3, 1e3, 0, +100e3, 1e3, 0, 100])
#add new stage
ns.addstage('Hxyz_seq')
ns.editstagevalue(1, [100e3, 1e3, 0, -100e3, 1e3, 0, 100])

#stop each field step using |mxh| < 10^-5 condition
ns.editstagestop(-1, 'mxh', 1e-5)
ns.editdatasave(-1, 'step')

#output data : applied field and average magnetisation
ns.setdata('Ha')
ns.adddata('<M>')
ns.savedatafile('hysteresis.txt')

#solve using LLGStatic equation (damping set to 1 and no precession term)
ns.setode('LLGStatic', 'RKF45')

#run program
ns.Run()

#output file has field (x, y, z components) in columns 0, 1, 2, and average magnetisation (x, y, z components) in columns 3, 4, 5
hysteresis_data = ns.Get_Data_Columns('hysteresis.txt', [0, 3])
#plot Mx vs Hx
ns.Plot_Data(hysteresis_data[0], hysteresis_data[1], xlabel = 'H (A/m)', ylabel = 'M (A/m)', title = 'Hysteresis Loop')
```

Domain Wall Movement

```
import os
import sys
from NetSocks import NSClient
import matplotlib.pyplot as plt
import numpy as np

#setup communication with server
ns = NSClient('localhost')
##########################################
#the working directory : same as this script file
directory = os.path.dirname(sys.argv[0]) + "/"
#restore program to default state
ns.default()
ns.chdir(directory)
##########################################
#This is based on Exercise 5.1, done entirely using a Python script
ns.meshrect([320e-9, 80e-9, 20e-9])
ns.cellsize([5e-9, 5e-9, 5e-9])
#setup the moving mesh algorithm for a transverse domain wall along the x axis:
ns.preparemovingmesh()
#relax dw in zero field to |mxh| < 10^-5
ns.editstagestop(0, 'mxh', 1e-5)
ns.Run()
#setup 2 field stages, each 5 ns long, but only second one saves data at 1 ps time intervals
ns.setstage('Hxyz')
ns.editstagestop(0, 'time', 5e-9)
ns.addstage('Hxyz')
ns.editstagestop(1, 'time', 5e-9)
ns.editdatasave(1, 'time', 1e-12)
#save time (s) and dw shift (m) data
ns.setdata('stime')
ns.adddata('dwshift')
ns.savedatafile('dwmovement_temp.txt')
#set fixed time-step RK4 method with 500fs time step
```

```
ns.setode('LLG', 'RK4')
ns.setdt(500e-15)
#save setup simulation file (next time you can just load it using ns.loadsim('dwmovement'))
ns.savesim('dwmovement')
##########################################
Hrange = np.arange(100, 2400, 200)
dwvelocity = np.array([])

for H in Hrange:
    ns.reset()
    #first stage achieves steady state movement
    ns.editstagevalue(0, H)
    #second stage captures data
    ns.editstagevalue(1, H)
    ns.Run()
    #process data to extract domain wall velocity
    ns.dp_load('dwmovement_temp.txt', [0, 1, 0, 1])
    ns.dp_replacerepeats(1)
    dwdata = ns.dp_linreg(0, 1)
    dwvelocity = np.append(dwvelocity, dwdata[0])
    print('H (A/m) = %f, DW velocity (m/s) = %0.4f' % (H, dwdata[0]))

plt.axes(xlabel = 'H (A/m)', ylabel = 'DW Velocity (m/s)', title = 'DW Velocity and Walker Breakdown')
plt.plot(Hrange, dwvelocity, 'o-')
plt.show()
```

Anisotropic Magnetoresistance

```
import os
import sys
from NetSocks import NSClient
import matplotlib.pyplot as plt
import numpy as np

#setup communication with server
ns = NSClient('localhost')

##########################################

#the working directory : same as this script file
directory = os.path.dirname(sys.argv[0]) + "/"
#restore program to default state
ns.default()
ns.chdir(directory)

##########################################
#This is based on Exercise 8.3, done entirely using a Python script

ns.meshrect([160e-9, 80e-9, 10e-9])
ns.cellsize([5e-9, 5e-9, 5e-9])
#amr loop angle (deg.)
direction_deg = 5

ns.addmodule('permalloy', 'transport')
#set electrodes at x-axis ends with a 1 mV potential drop
ns.setdefaultelectrodes()
ns.setpotential(1e-3)

ns.setstage('Hpolar_seq')
ns.editstagevalue(0, [-100e3, 90, direction_deg, 100e3, 90, direction_deg, 200])
ns.editstagestop(0, 'mxh', 1e-7)
ns.editdatasave(0, 'step')
```

```
ns.addstage('Hpolar_seq')
ns.editstagevalue(1, [100e3, 90, direction_deg, -100e3, 90, direction_deg, 200])
ns.editstagestop(1, 'mxh', 1e-7)
ns.editdatasave(1, 'step')

ns.setangle(90, 180.0 + direction_deg)

#set amr percentage of 2%
ns.setparam('permalloy', 'amr', 2.0)

#save applied field (A/m) and resistance (Ohms)
ns.setdata('Ha')
ns.adddata('R')
ns.savedatafile('amr_rawdata.txt')

ns.setode('LLGStatic', 'SDesc')

ns.Run()
#load all columns from file (0, 1, 2, 3) into internal arrays (0, 1, 2, 4)
ns.dp_load('amr_rawdata.txt', [0, 1, 2, 3, 0, 1, 2, 4])
#get field strength along loop direction and save it in internal array 3
ns.dp_dotprod(0, np.cos(np.radians(direction_deg)), np.sin(np.radians(direction_deg)), 0, 3)
#save field strength and resistance in processed file, then plot it here
ns.dp_save('amr_loop.txt', [3, 4])
amr_data = ns.Get_Data_Columns('amr_loop.txt', [0, 1])
plt.axes(xlabel = 'H (A/m)', ylabel = 'R (Ohms)', title = 'AMR Loop')
plt.plot(amr_data[0], amr_data[1])
plt.show()
```

RKKY Simulation (multi-mesh demonstration)

```
import os

import sys

from NetSocks import NSClient

import matplotlib.pyplot as plt

import numpy as np


#setup communication with server

ns = NSClient('localhost')

##########################################

#the working directory : same as this script file

directory = os.path.dirname(sys.argv[0]) + "/"

ns.default()

ns.chdir(directory)

##########################################

direction_deg = 1.0

ns.meshrect([320e-9, 160e-9, 10e-9])

#shape mesh as an ellipse (mask file is stretched to mesh aspect ratios)

ns.loadmaskfile('Circle')

#add a new ferromagnetic mesh (permalloy by default) above the first one with 1 nm separation

#the 2 meshes still retain a cubic 5 nm cellsize

ns.addmesh('permalloy2', [0.0, 0.0, 11e-9, 320e-9, 160e-9, 31e-9])

ns.meshfocus('permalloy2')

ns.loadmaskfile('Circle')

#enable multilayered demag field calculation allowing exact and efficient calculation of demag fields, even though
the separation between meshes is 1 nm

ns.addmodule('supermesh', 'sdemag')

#enable RKKY coupling (surface exchange coupling) keeping default J1 (bilinear) and J2 (biquadratic) values

ns.addmodule('permalloy', 'surfexchange')

ns.addmodule('permalloy2', 'surfexchange')

#set field sequence to apply to both meshes (so set it to the supermesh)

ns.setstage('Hpolar_seq', 'supermesh')

ns.editstagevalue(0, [-300e3, 90, direction_deg, 300e3, 90, direction_deg, 300])

ns.editstagestop(0, 'mxh', 1e-5)

ns.editdatasave(0, 'step')
```

```
ns.addstage('Hpolar_seq', 'supermesh')
ns.editstagevalue(1, [300e3, 90, direction_deg, -300e3, 90, direction_deg, 300])
ns.editstagestop(1, 'mxh', 1e-5)
ns.editdatasave(1, 'step')

ns.setdata('Ha')
ns.adddata('<M>', 'permalloy')
ns.adddata('<M>', 'permalloy2')
ns.savedatafile('rkky_hysteresis.txt')

ns.setode('LLGStatic', 'SDesc')
ns.cuda(1)

ns.Run()

##########################################
#process raw data into a hysteresis loop for the entire bilayer
u = [np.cos(np.radians(direction_deg)), np.sin(np.radians(direction_deg)), 0]
ns.dp_load('rkky_hysteresis', [0, 1, 2, 3, 4, 5, 6, 7, 8, 0, 1, 2, 3, 4, 5, 6, 7, 8])
ns.dp_dotprod(0, u[0], u[1], u[2], 10)
ns.dp_dotprod(3, u[0], u[1], u[2], 11)
ns.dp_dotprod(6, u[0], u[1], u[2], 12)
ns.dp_mul(11, 1.0/3)
ns.dp_mul(12, 2.0/3)
ns.dp_adddp(11, 12, 13)
ns.dp_save('rkky_hysteresis_loop.txt', [10, 13])
loop = ns.Get_Data_Columns('rkky_hysteresis_loop.txt', [0, 1])
plt.axes(xlabel = 'H (A/m)', ylabel = 'M (A/m)', title = 'RKKY Hysteresis Loop')
plt.plot(loop[0], loop[1])
plt.show()
```

Setting Shapes Programatically

```
import os

import sys

from NetSocks import NSClient

import numpy as np

from itertools import product

ns = NSClient('localhost')

##########################################

directory = os.path.dirname(sys.argv[0]) + "/"

ns.default()

ns.chdir(directory)

##########################################

#Mesh with Nxy cells along x and y, and Nz cells along z.

#Easier to generate mesh this way but the OVF2 file can be loaded in an arbitrarily shaped mesh in Boris
(mapped to dimensions).

Nxy, Nz = 32, 16

#M list to write in OVF2 file : has Nxy*Nxy*Nz cells, initialised with empty cells

M = [[0,0,0]] * Nxy**2*Nz

#setup hollow hemisphere values

origin = [Nxy/2, Nxy/2, Nz]

inner_ratio, outer_ratio = 0.75, 1.0

rad_inner, rad_outer = Nxy * inner_ratio / 2, Nxy * outer_ratio / 2

for i, j, k in product(range(Nxy), range(Nxy), range(Nz)):

    rad = np.sqrt((i - origin[0])**2 + (j - origin[1])**2 + (k - origin[2])**2)

    if rad >= rad_inner and rad <= rad_outer:

        #Mark these cells as non-empty

        M[i + j*Nxy + k*Nxy*Nxy] = [1,0,0]


#Write M to OVF2 file ready to load into Boris

fileName = 'HHemi.ovf'

#cellsize used to generate mesh rectangle

h = 5e-9

ns.Write_OVF2(fileName, M, [Nxy, Nxy, Nz], [0.0, 0.0, 0.0, Nxy*h, Nxy*h, Nz*h])

ns.meshrect([0.0, 0.0, 0.0, Nxy*h, Nxy*h, Nz*h])

ns.loadovf2mag(8e5, fileName)
```

Exchange Bias

```
import os
import sys
from NetSocks import NSClient
import matplotlib.pyplot as plt

#setup communication with server
ns = NSClient('localhost')

##########################################

#the working directory : same as this script file
directory = os.path.dirname(sys.argv[0]) + "/"
#restore program to default state
ns.default()
ns.chdir(directory)

##########################################

ns.setafmesh('Antiferromagnet', [320e-9, 320e-9, 10e-9])
ns.cellsize([5e-9, 5e-9, 5e-9])
ns.addmodule('Antiferromagnet', 'aniuni')
ns.addmodule('Antiferromagnet', 'surfexchange')
#set sub-lattice A magnetisation to result in biasing towards +ve side
ns.setangle(90, 180)
#Add Fe mesh on top of the antiferromagnet
ns.addmaterial('Fe', [0, 0, 10e-9, 320e-9, 320e-9, 12e-9])
ns.meshfocus('Fe')
#need smaller cellsize for Fe (in Boris meshes can be independently discretised)
ns.cellsize([2.5e-9, 2.5e-9, 2e-9])
ns.pbc('Fe', 'x', 10)
ns.pbc('Fe', 'y', 10)
ns.addmodule('Fe', 'anicubi')
#Now both the Antiferromagnet and Fe meshes have surfexchange module enabled, so exchange bias field will
be included in computations
```

```
ns.addmodule('Fe', 'surfexchange')
#set bilinear surface exchange coupling value - exchange bias is proportional to this
ns.setparam('Fe', 'J1', 0.2e-3)

ns.setode('LLGStatic', 'RKF45')

ns.setstage('Hpolar_seq', 'supermesh')
ns.editstagevalue(0, [-50e3, 90, 5, 100e3, 90, 5, 50])
ns.editstagestop(0, 'mxh', 1e-4)
ns.editdatasave(0, 'step')
ns.addstage('Hpolar_seq', 'supermesh')
ns.editstagevalue(1, [100e3, 90, 5, -50e3, 90, 5, 50])
ns.editstagestop(1, 'mxh', 1e-4)
ns.editdatasave(1, 'step')

ns.setdata('Ha')
ns.adddata('<M>', 'Fe')
ns.savedatafile('exchangebias.txt')

ns.cuda(1)
ns.Run()

#we should really project along the 5 degree direction, but will keep this simple
data = ns.Get_Data_Columns('exchangebias.txt', [0, 3])

plt.axes(xlabel = 'H (A/m)', ylabel = 'M (A/m)', title = 'Exchange bias')
plt.plot(data[0], data[1])
plt.show()
```

Ultrafast Demagnetisation and Skyrmion Creation in a Co/Pt/SiO_2 Trilayer (Advanced)

```
import os
import sys
from NetSocks import NSClient
import numpy as np
import matplotlib.pyplot as plt
ns = NSClient('localhost')
##########################################
directory = os.path.dirname(sys.argv[0]) + "/"
ns.default()
ns.chdir(directory)
##########################################
ns.setode('sLLB', 'TEuler')
#Co layer
ns.setmaterial('Co/Pt', [512e-9, 512e-9, 2e-9])
ns.cellsize([1e-9, 1e-9, 2e-9])
ns.scellsize([4e-9, 4e-9, 2e-9])
ns.setdtstoch(20e-15)
ns.addmodule('Co/Pt', 'iDMexchange')
ns.addmodule('Co/Pt', 'aniuni')
ns.addmodule('Co/Pt', 'heat')
ns.tcellsize([2e-9, 2e-9, 2e-9])
ns.tmodel(2, 'Co/Pt')
ns.curietemperature(500)
ns.pbc('Co/Pt', 'x', 10)
ns.pbc('Co/Pt', 'y', 10)
#Pt layer
ns.addmaterial('Pt', [0.0, 0.0, -8e-9, 512e-9, 512e-9, 0.0])
ns.meshfocus('Pt')
ns.addmodule('Pt', 'heat')
ns.tcellsize([4e-9, 4e-9, 4e-9])
ns.tmodel(2, 'Pt')
#SiO2 layer
ns.addmaterial('SiO2', [0.0, 0.0, -48e-9, 512e-9, 512e-9, -8e-9])
```

```
ns.meshfocus('SiO2')

ns.addmodule('SiO2', 'heat')

ns.tcellsize([8e-9, 8e-9, 8e-9])

#general settings

ns.setangle(0, 0)

ns.setfield(100e3, 0, 0)

ns.temperature(300)

#simulation stage

ns.setstage('Qequation', 'Co/Pt')

ns.editstagevalue(0, 'Q0 * exp(-sqrt((x/Lx - 0.5)^2 + (y/Ly - 0.5)^2) / ((d0/Lx)^2/(4*ln(2)))) * exp(-(t-
2*tau)^2/(tau^2/(4*ln(2))))')

ns.equationconstants('d0', 400e-9)

ns.equationconstants('tau', 100e-15)

ns.equationconstants('Q0', 4e21)

#output data

ns.setdata('time')

ns.adddata('Q_topo', 'Co/Pt')

ns.adddata('<T>', 'Co/Pt', [255e-9, 255e-9, 0.0, 256e-9, 256e-9, 2e-9])

ns.editdatasave(0, 'time', 10e-15)

ns.savedatafile('ufsky.txt')

#set-up display

ns.meshfocus('Co/Pt')

ns.display('M', 'Co/Pt')

ns.vecrep('Co/Pt', 3)

ns.cuda(1)

#next time you can just load this and run it

ns.savesim('ufsky_fm')


#0 to 10ps

ns.editstagestop(0, 'time', 10e-12)

ns.setdt(1e-15)

ns.setheatdt(1e-15)

ns.Run()


#10ps to 20ps

ns.editstagestop(0, 'time', 20e-12)

ns.setdt(2e-15)
```

```
ns.setheatdt(2e-15)

ns.Run()


#20ps to 60ps

ns.editstagestop(0, 'time', 60e-12)

#mid-simulation re-meshing! 1nm cellsize only needed around the Curie temperature.

#finely tuned simulations of this type means you can run thousands of these events in a reasonable time-scale
and still keep accuracy

ns.cellsize([2e-9, 2e-9, 2e-9])

ns.setdt(10e-15)

ns.setheatdt(2e-15)

ns.Run()


#60ps to 800ps

ns.editstagestop(0, 'time', 800e-12)

ns.tcellsize([4e-9, 4e-9, 1e-9])

ns.setdt(50e-15)

#could do with Crank-Nicolson method in next version

ns.setheatdt(5e-15)

ns.editdatasave(0, 'time', 250e-15)

ns.Run()


#now plot |Q| as a function of time

data = ns.Get_Data_Columns('ufsky.txt', [0, 1])

time_ps = [t/1e-12 for t in data[0]]

Qmod = [np.abs(Qval) for Qval in data[1]]

plt.axes(xlabel = 'Time (ps)', ylabel = '|Q|')

plt.xscale('log')

plt.yscale('log')

plt.plot(time_ps, Qmod)

plt.xlim(0.1)

plt.ylim(1e-3)

plt.savefig('ufsky_plot.png', dpi = 600)

plt.show()
```

Tutorial 1 – Introduction

Basics

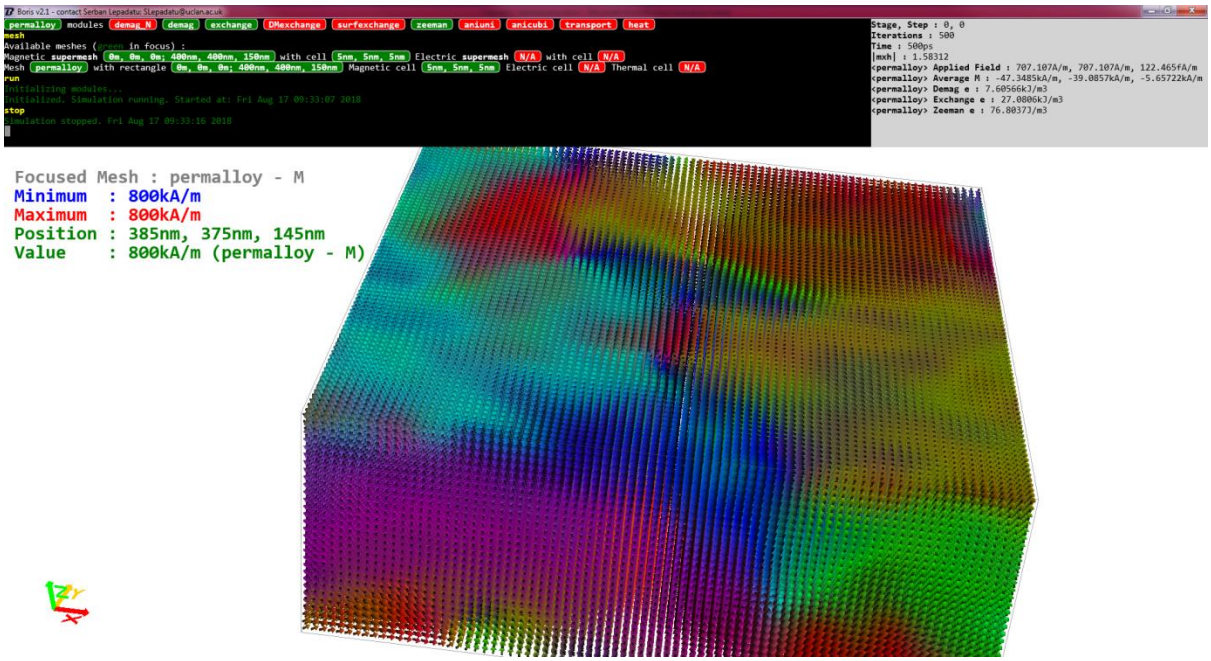
All commands are entered using the console (top-left black box) in Figure 1.1. Each console command has a case-sensitive syntax and may have a number of parameters separated by spaces. If a command is entered with wrong parameters a help prompt will be displayed explaining the command and full syntax. Alternatively a command may be immediately preceded by a question mark in order to display the command help. For example try it for the **run** command:

?run

Note, the program auto-completes commands entered out of the list of possible inputs – and equally stops any wrong inputs from being entered.

The main display shows the magnetization configuration (other vector and scalar quantities may be displayed, but magnetization is the default setting).

Figure 1.1 – Boris interface



The magnetization display may be controlled using the mouse: left-click and drag to re-position the display mesh, middle-click and move mouse to rotate the camera view about the center of the displayed (focused) mesh, right-click and move mouse up/down to zoom in/out, or left/right to rotate the camera about its axis; finally the wheel may be used to set the coarseness of magnetization representation: for large mesh dimensions, each arrow represents an average of the magnetization in that area.

Various simulation data may be displayed in the data box (top-right box) for convenience – more on these later. The displayed windows may be resized by dragging their outlines – the outline appears if you hover the mouse over the edge of a window.

The magnetization display can be reset to the default view using:

center

<u>Simulation Mesh Control</u>

To display the current problem size enter the following command (see Figure 1.2 for expected output):

mesh

Figure 1.2 – Default mesh configuration

```
mesh
Available meshes (green in focus) :
Magnetic supermesh 0m, 0m, 0m; 80nm, 80nm, 10nm with cell 5nm, 5nm, 5nm Electric supermesh N/A with cell N/A
Mesh permalloy with rectangle 0m, 0m, 0m; 80nm, 80nm, 10nm Magnetic cell 5nm, 5nm, 5nm Electric cell N/A Thermal cell N/A
```

The simulation space consists of one or more named meshes – the default configuration consists of a single ferromagnetic mesh named *permalloy*. This has a rectangle with lower-left corner coordinates of (0, 0, 0) and upper-right corner coordinates of (80 nm, 80 nm, 10 nm). The magnetic discretization cellsize is a rectangular prism with dimensions (d_x, d_y, d_z) = (5 nm, 5 nm, 5 nm) – a cubic cellsize by default. Thus the *permalloy* mesh is discretized with the integer number of cells (16, 16, 2).

To adjust the mesh dimensions you can use the **meshrect** command. An easy way to bring up this command, with current fields already entered, is to double-click on the outlined text containing the mesh rectangle dimensions: `0m, 0m, 0m; 80nm, 80nm, 10nm`

This type of outlined text is a special console text called an interactive console object, allowing a number of user interactions depending on the particular object, including left or right click, double-click, or drag. The text is also automatically updated to display currently set values. You can find out what an interactive object does by using shift-click.

Try to resize the *permalloy* mesh so it has the dimensions (300 nm, 100 nm, 15 nm). Values may be entered without specifying the units, in which case the applicable S.I. unit is assumed, or the applicable unit may be entered together with its magnitude specifier (e.g. for a meter the currently available units are designated as *am, fm, pm, nm, um, mm, m, km, Mm, Gm, Tm, Pm*). If entering the unit, do not leave a space between the number and unit.

The magnetic cellsize may be adjusted by double-clicking on the magnetic cell interactive object, which brings up the **cellsize** command. Try to adjust the cellsize so it has dimensions (6 nm, 6 nm, 5 nm). After changing the cellsize its dimensions are automatically adjusted in order to satisfy the requirement of integer number of discretization cells in each dimension.

Similarly the *permalloy* mesh may be renamed by double-clicking on the mesh name interactive object, which brings up the **renamemesh** command.

In Figure 1.2 you can also see an entry for the *supermesh*. Its rectangle is not controlled directly, but depends on the currently set meshes (however you can adjust the supermesh cellsize). The magnetic supermesh is the smallest simulation space containing all the currently set magnetic meshes and is useful to compute long-range interactions over several independently discretized meshes (e.g. supermesh demagnetizing field) – more on this in a dedicated tutorial.

Basic Simulation Control

The simulation is started and stopped using:

run
stop

The stop command simply pauses the simulation without resetting it. To continue from the stop point simply type **run** again. To reset the simulation use:

reset

The display refresh frequency can be set using (*iter* is the number of iterations – remember you can query to command for full details: **?iterupdate**):

iterupdate *iter*

The simulation may be saved at any point using (do not use a termination, the .bsm termination is added by default):

savesim *filename*

If a directory path is not specified, the default directory path is used. To set a default directory use:

chdir *directory*

To load a previously saved simulation use:

loadsim *filename*

Alternatively a simulation file may be dragged into the console area. At any point you can return to the default program state by using:

default

A uniform magnetization configuration can be set using the following (theta is the polar angle, phi is the azimuthal angle in spherical polar coordinates):

setangle theta phi

A uniform magnetic field can be set using (again use spherical polar coordinates):

setfield Hmag Htheta Hphi

Simulation Modules

Simulation modules typically correspond to effective field terms. These can be managed using interactive objects by typing the following command:

modules

The default configuration includes the demagnetizing field (*demag*), direct exchange interaction (*exchange*), and applied field (*zeeman*) – see Figure 1.3.

Figure 1.3 – Default simulation modules

```
Available modules (green added, red not added) :
supermesh   modules  sdemag  strayfield  Oersted
permalloy   modules  demag_N  demag  exchange  DMexchange  surfexchange  zeeman  aniuni  anicubi  transport  heat
```

Currently added modules are displayed in green. To add or remove a module left or right-click on the respective interactive object. Individual modules will be explored in future tutorials.

Material Parameters

Default simulation parameters are set for permalloy ($Ni_{80}Fe_{20}$), as M_s = 8e5 A/m, A = 1.3e-11 J/m, and α = 0.02. To see a list of currently set parameter values use the command:

params

Values may be modified by double-clicking on the respective interactive objects. By default parameters are constant for each mesh, however they can be assigned temperature and spatial dependence for advanced simulations (more on this later).

Simulation Flow

A basic simulation flow can be programmed by setting a number of *stages*. Each stage has an identifier, parameters depending on the identifier, a stopping condition and data save condition. To show the currently set simulation stages use:

stages

By default the *Relax* stage type is used (no simulation values changed), with a stopping condition based on the normalized torque |*mxh*| < 10^{-4} (*mxh: 0.0001*), and no data saving configured. New stages may be added by double-clicking on the interactive objects at the bottom – see Figure 1.4.

Figure 1.4 – Simulation stage types

You can delete added stages by right-clicking on them, change the stage type by double-clicking and editing, and re-arrange the stage order by dragging.

Available stopping conditions are: *nostop, iter* (stop after a number of iterations), *mxh* (stop when |*mxh*| falls below the set threshold), *dmdt* (stop when the normalized |∂**m**/∂t| value falls below the set threshold), or *time* (stop after an elapsed simulation time). When a stage reaches the stopping condition the next stage starts or the simulation finishes.

Some stage types are broken down into several sub-stages (referred to as *steps*), for example a field sequence using *Hxyz_seq*. Try to add a field sequence stage by

double-clicking on the *Hxyz_seq* interactive object. The default parameters for a field sequence are shown in Figure 1.5. This consists of a field sequence starting from a field of -100 kA/m along the x-axis, and stopping at +100 kA/m along the x-axis. The sequence consists of 100 field steps, thus the field step is 2 kA/m. The simulation proceeds to the next *step* when the $|mxh| < 10^{-4}$ stopping condition is satisfied.

Figure 1.5 – Default parameters for the *Hxyz_seq* stage type

```
-100kA/m, 0A/m, 0A/m; 100kA/m, 0A/m, 0A/m; 100
```

Exercise 1.1

Set a 160×80×5 nm permalloy mesh (with cubic cellsize of 5 nm) starting from a saturated magnetization state along the negative x-axis direction. Set a field sequence from -60 kA/m to +60 kA/m along the x-axis using a step of 2 kA/m and *mxh* stopping condition of 10^{-4} (typically this threshold is too high, it should be 10^{-5} or even lower for an accurate simulation depending on the problem, but this will speed up the exercise).

Display the applied field and average magnetization in the data box. To do this use the command:

data

You will see a list of interactive console objects representing possible output data which can be displayed in the data box or saved to a file. For now just display the applied field (*Ha*) and average magnetization (<*M*>) by right-clicking on the interactive objects (or dragging them to the data box).

Run the simulation - the magnetization should switch during this field sequence.

Tutorial 2 – Data Output

Saving Numerical Data

In order to save numerical simulation data (automated saving of images will be explored in a future tutorial) you need to set a data saving file, a list of output data, and a saving schedule. To set output data and a save file use the **data** command.

Figure 2.1 – Default output data

The default output data file is called *out_data.txt* and its name may be modified by double-clicking on the interactive object. The default saving directory is the path to the program executable file and may be modified by double-clicking on the interactive object.

The default output data includes *sstep* (stage and step), *iter* (iteration), *time* (simulation time), *Ha* (applied field), and <*M*> (average magnetization). This is the order the output data will appear in the output file as numerical columns. The order may be modified by dragging the respective interactive objects in the list of output data. New output data may be added by double-clicking on the interactive objects at the bottom, and set output data may be deleted by right-clicking on the respective interactive objects in the output list.

Some output data (such as *ha* and <*M*>) may be saved in a particular mesh – in this case the *permalloy* mesh which is specified using the notation <permalloy>, whilst other output data do not depend on any particular mesh. Some output data (such as

<*M*>) may also be saved in a particular rectangle of the named mesh (the rectangle is relative to the named mesh) – by default the entire mesh rectangle is saved, but this can be modified by double-clicking on the respective interactive object and editing.

Finally, a saving schedule may be set in the simulation stages: use the **stages** command. Each stage has a list of possible saving conditions: *none* (default – do not save), *stage* (save at the end of the stage), *step* (save at the end of each step in the current stage), *iter* (save every n iterations), and *time* (save every t simulation seconds); for *iter* and *time* the parameters may be edited by double-clicking the respective interactive objects.

Exercise 2.1

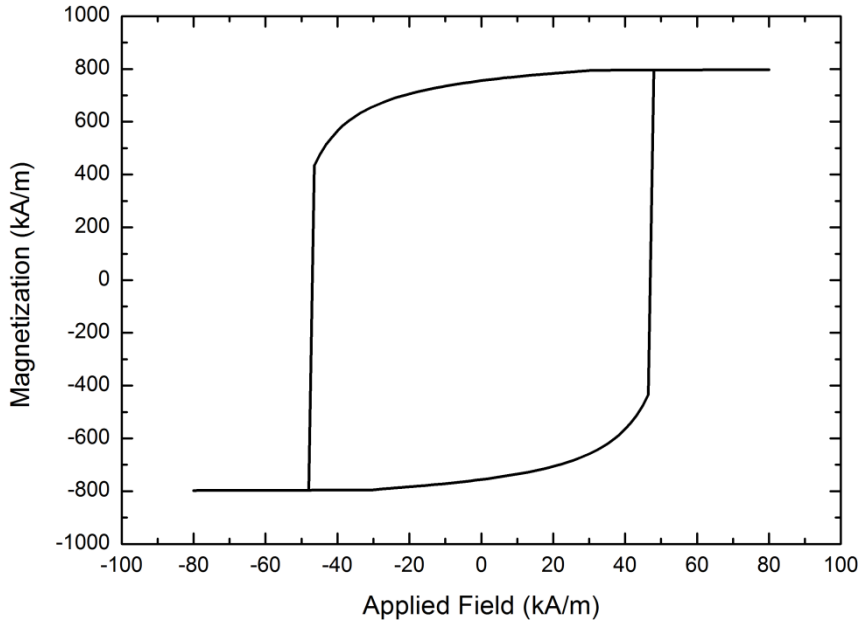
Set a 160×80×10 nm permalloy mesh (with cubic cellsize of 5 nm) starting from a saturated magnetization state along the negative x-axis direction. Set a field sequence from -100 kA/m to +100 kA/m and then back to -100 kA/m along the x-axis using a step of 2 kA/m and *mxh* stopping condition of 10^{-4}.

Configure the simulation so it saves output data for a hysteresis loop (applied field and average magnetization saved after every step).

Before running the simulation save the simulation file. Once the simulation file is saved using a specified name (and directory path if needed), the next time you don't need to specify the file name – simply use **savesim** without a file name and the previously used file name will be saved. Note, correct commands previously entered in the console can be recalled using the arrow keys (invalid commands are not saved). You can also use Ctrl^v to paste text in the console.

Run the simulation and plot the hysteresis loop (magnetization along the applied field direction) at the end.

Figure 2.2 – Hysteresis loop obtained in exercise 2.1



Further Data Box Control

As introduced in the previous tutorial, output data may also be displayed in the data box for convenience. The possible output data may be listed as interactive objects by using the **data** command, and the listed interactive objects may be displayed in the data box by dragging them there, or right-clicking on them. Data box entries may be removed by right-clicking on them in the data box, and they may be re-arranged by dragging them.

If you just want to quickly see the current values of particular data without displaying them in the data box, bring up an interactive object list using the **showdata** command and double-click on the respective interactive objects.

Exercise 2.2

In this exercise you will run the μMAG standard problem #4: https://www.ctcms.nist.gov/~rdm/std4/spec4.html

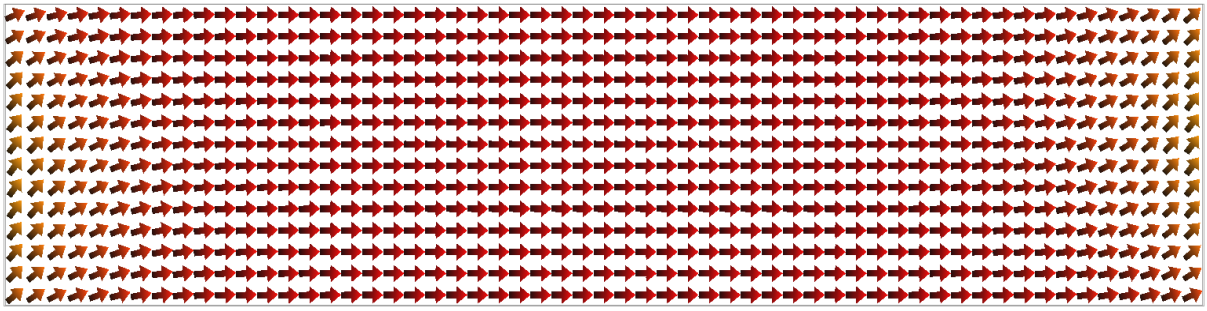
a) For this problem we need a permalloy mesh with dimensions 500x125x3 nm. First initialize the magnetization configuration to a so-called s-state: this may be obtained by reducing a large applied field to zero along the [1,1,1] direction. For example set a field sequence starting from 1 MA/m along the [1,1,1] direction, reducing to zero in 20 steps – you should use the stricter $|mxh| < 10^{-5}$ condition this time. Save an image of the obtained magnetization configuration – see Figure 2.3.

The easiest way to setup Exercise 2.2a is to use a polar field sequence: *Hpolar_seq*. This specifies the starting and ending field values using polar coordinates: magnitude, polar angle and azimuthal angle – thus a starting field of 1 MA/m along the [1,1,1] direction would be specified (roughly) as *1MA/m, 55, 45*. Make sure to specify the ending field value as *0, 55, 45* to keep the field values in the sequence along the same direction. You should also set the starting magnetization state along the [1,1,1] direction: **setangle** *55 45*.

To save an image of the currently displayed mesh, use the command:

savemeshimage *(directory\)filename*

Figure 2.3 – Starting s-state for micromagnetics standard problem #4



Exercise 2.2 continued

b) Starting from the s-state, apply a fixed field with magnetic flux density (B-field) of 25 mT directed 170° counterclockwise from the x-axis in the x-y plane. Simulate the switching event for a duration of 5 ns, saving output data (in particular the average magnetization and simulation time are required) every 5 ps. Plot the 3 components of magnetization against time. Remember to save the simulation before starting it. How do these results compare with published solutions ?

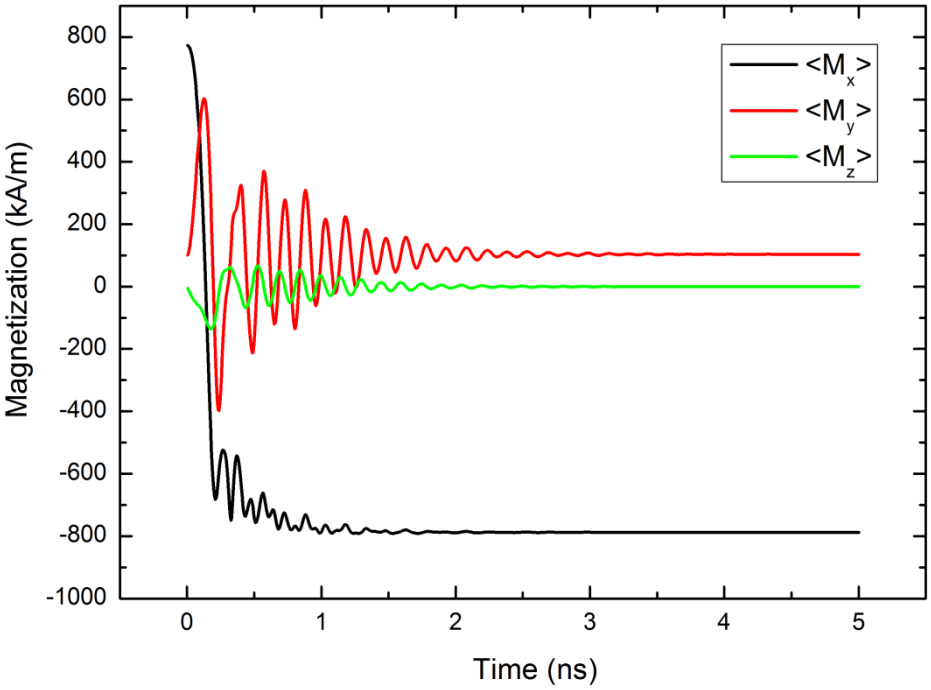
   (see https://www.ctcms.nist.gov/~rdm/std4/results.html)

c) Repeat part b) but this time for a B-field of 36 mT directed 190 degress counterclockwise from the x-axis in the x-y plane.

d) For parts b) and c) obtain images of the magnetization configuration when the average magnetization (x component) first crosses zero – use the output data to determine the time when this occurs, then run the simulation to stop at this particular time.

Figure 2.4 – a) Results obtained after running the μMAG standard problem #4 with field 1, and b) magnetization configuration when the average magnetization (x component) first crosses zero.

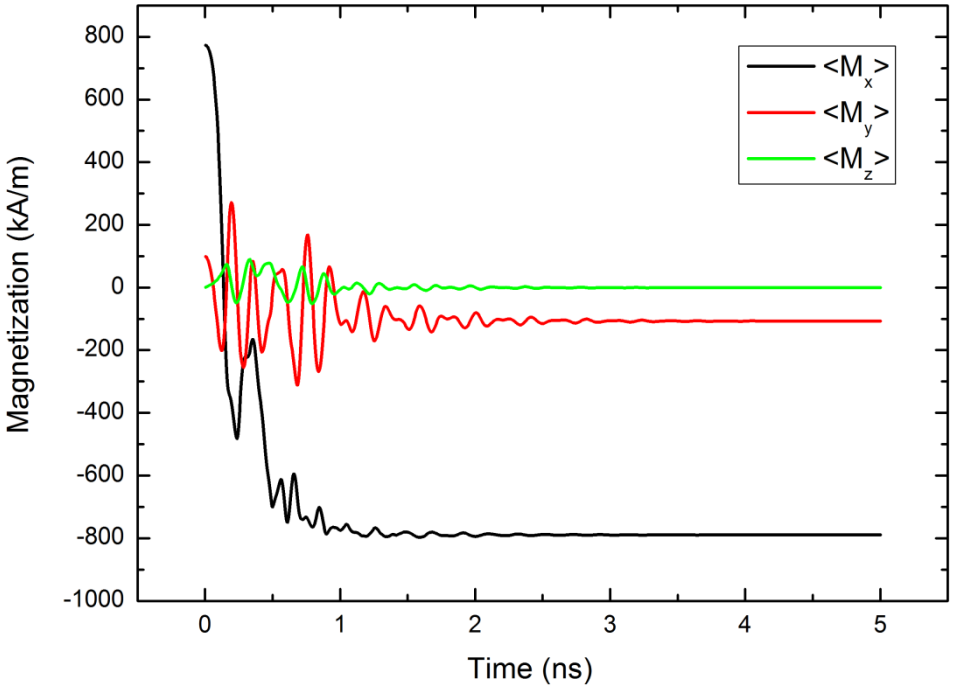
a)



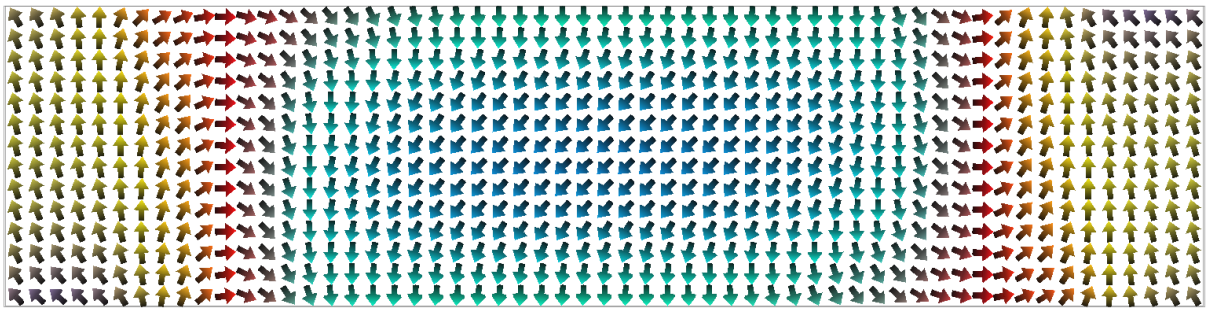
b)

Figure 2.5 – a) Results obtained after running the μMAG standard problem #4 with field 2, and b) magnetization configuration when the average magnetization (x component) first crosses zero.

a)



b)



<u>Making a video from an image sequence</u>

A video may be encoded from a sequence of .png files (e.g. as produced from a simulation with an image saving schedule) – this functionality is built into the program for convenience; for advanced image processing you should use an external program. To produce a video file from an image sequence, use:

makevideo (directory\)*filebase fps quality*

This makes a video from all .png files which start with the *filebase* name, including the directory, at the given *fps* (frames per second). The *quality* parameter sets the bit-rate of the output video: 0 for worst quality but smallest size, 5 for best quality but largest size. For the **makevideo** command, the files are sorted by their creation time, not alphabetically.

To enable mesh image saving, use the **data** command then click on the respective interactive object. You can also edit the mesh image filebase name. The mesh images are saved using the same save conditions as output data.

Exercise 2.3

For the switching event in exercise 2.2b, set the problem to save mesh images every 10 ps; also reduce the simulation time to 3 ns and disable data saving. Make a video of the switching event (40 fps and quality level 3 works well).

If you want to capture only a part of the mesh display window you can set cropping factors. These are specified as normalized values and are applied whenever an image is saved (either manually or during a simulation). This is set using:

imagecropping *left bottom right top*

The left-bottom of the mesh display is (0, 0), whilst the right-top of the mesh display is (1,1).

Tutorial 3 – Further Data Output

Exercise 3.1

In this exercise you will compare coercive fields obtained using the full micromagnetics model with the predictions of the simpler Stoner-Wohlfarth model.

a) Simulate an in-plane hysteresis loop along a 10° direction in a permalloy rectangle with dimensions 250x50x5 nm using the full micromagnetics model (*demag, exch*, and *zeeman* modules for permalloy) and obtain the coercive field. In order to speed up the simulation you only need to simulate one branch of the hysteresis loop, e.g. negative to positive field only, and you should also set a coarse field step up to zero field (e.g. 10 kA/m), then a fine field step in order to obtain a more accurate switching field value (use a 500 A/m fine field step or less); use the *mxh* stopping condition with a 10^{-5} threshold.

   Solution: use two polar field sequences (Hpolar_seq) along the (polar, azimuthal) = (90°, 10°) direction. Start at -200 kA/m and finish at 60 kA/m. This range is just enough to start from a saturated state and capture the switching field.

   i.e. : 1) Hpolar_seq -200kA/m 90 10 0kA/m 90 10 20, 2) Hpolar_seq 0kA/m 90 10 60kA/m 90 10 120

b) Compute the anisotropy energy density (shape anisotropy) and hence obtain the switching field predicted by the Stoner-Wohlfarth model.

Note, the Stoner-Wohlfarth model predicts the switching field:

$$H_S = \frac{2K_u}{\mu_0 M_S}\frac{\sqrt{1-t^2+t^4}}{1+t^2}, \; where \; t = \sqrt[3]{\tan(\theta)}$$

and θ is the angle between the applied field and anisotropy easy axis (formula applicable for θ between 0 and 45°).

To compute the energy density for a given magnetization orientation, set the magnetization orientation (**setangle**) and calculate the energy density terms using the command:

computefields

This command runs the simulation for a single iteration and does not advance the simulation time – only the currently set simulation modules are refreshed. After running this command the required energy density term (*e_demag*) will be available.

c) For the same geometry and applied field direction obtain the hysteresis loop using the Stoner-Wohlfarth model. Does the coercive field agree with that obtained in part b) ?

   To run this you will need to use the *demag_N* module instead of the full *demag* module; the exchange module (*exch*) is not needed.

   The *demag_N* module computes the demagnetizing field using the simple approximation $H_{d,\,i} = -N_i\, M_i$ ($i = x, y, z$). You will need to enter correct values for the demagnetizing factors N_x and N_y (remembering that $N_x + N_y + N_z = 1$). These can be entered using the command **params**, then editing the values under the N_{xy} interactive object.

   Calculate N_x, N_y and N_z directly from the demagnetizing field (obtained using the full micromagnetics model) and also using the demagnetizing energy density values obtained in part b). Do the values agree, and does the relationship $N_x + N_y + N_z = 1$ hold?

To obtain the demagnetizing field you will need to update the field using the **computefields** command with only the *demag* module enabled. After this you can display the demagnetizing field using the command:

display

Using this command brings up a list of interactive objects with display options. Click on the *Heff* option under the *permalloy* mesh. This will display the computed effective field. Using the average effective field value you can obtain a value for the demagnetizing factor along the set magnetization direction using the expression H_d = -N M.

In order to obtain the average value of the demagnetizing field you can use the command:

dp_averagemeshrect (*sx sy sz ex ey ez*)

This command returns the average value for the displayed quantity in the currently focused mesh (the *permalloy* mesh in this case) when used without parameters. The parameters specify a mesh rectangle (start and end Cartesian coordinates) which is relative to the currently focused mesh.

d) Repeat this exercise using the 100x25x5 nm.
e) Repeat this exercise using the 50x25x3 nm.

Another way to calculate the demagnetizing factors is to use the formula:

$$\varepsilon_d = -\frac{\mu_0}{2}\mathbf{M}.\mathbf{H}_d$$

Thus for **M** along *i* = x, y, z, you can obtain *e_demag*, then use:

$$\varepsilon_{d,i} = \frac{\mu_0}{2} N_i M_S^2 \quad (i = x, y, z)$$

Exercise 3.2

Here you will obtain the magnetization dynamics during a switching event and investigate the effect of the cellsize value on the simulation.

a) Set a 320x160x10 nm permalloy rectangle with magnetization along the length of the rectangle (set the magnetization towards the left, thus blue coloured). Obtain the stable magnetization configuration at zero field by reducing the magnetic field from a large saturation value along x to zero in a number of steps. (e.g. from -50 kA/m to 0). For now use a cellsize of 5 nm.

b) Starting from the magnetization configuration set-up in part a) set a single stage where you apply a large field along the x direction, opposing the magnetization – use 50 kA/m. As stopping condition using a time interval of 4 ns. Set a saving schedule to save the simulation time and average magnetization components every 10 ps. For now use a cellsize of 5 nm. From the saved data plot $<M_x>$ and $<M_y>$ versus time.

c) Repeat the simulation in b) with cellsize values of 10 nm and 2.5 nm. Plot $<M_x>$ and $<M_y>$ versus stage time for the 3 cellsize values. How do the results compare ? Which cellsize would you recommend to use ?

Tutorial 4 – Domain Walls

Generating Domain Walls

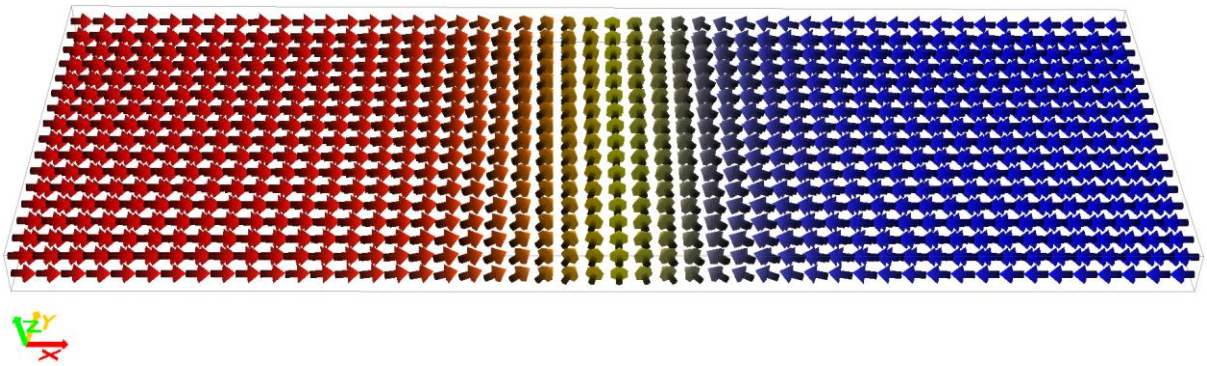
An idealized domain wall (using a tanh profile) along the x direction can be generated using:

dwall *longitudinal transverse width position*

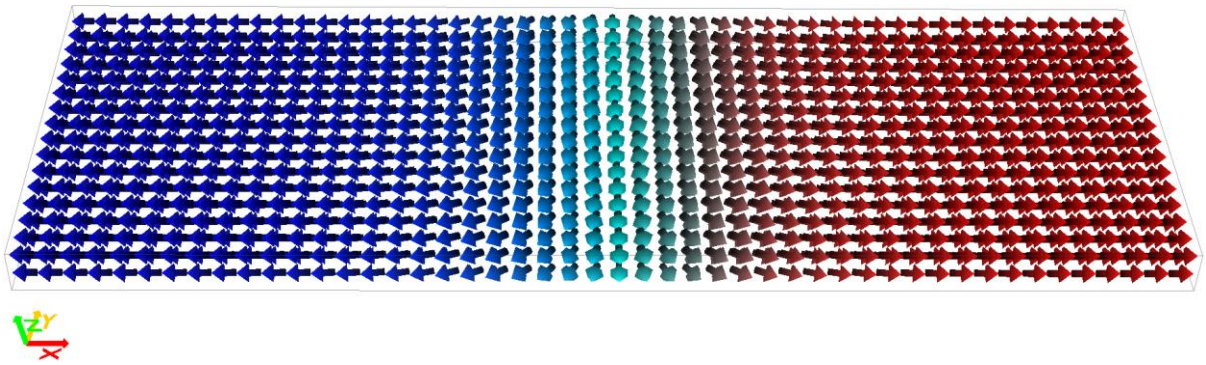
For an in-plane domain wall the longitudinal parameter determines if the wall is head-to-head (*longitudinal* = *x*) or tail-to-tail (*longitudinal* = -*x*); Bloch walls may be generated using *z* or –*z* for the longitudinal component. The transverse parameter determines the rotation direction through the wall – see Figure 4.1 for examples. The w*idth* value is the total domain wall width and *position* is the starting left-hand-side coordinate of the wall (along the x axis), relative to the focused mesh rectangle.

Figure 4.1 – Domain walls generated using the **dwall** command for a mesh with 240nm length, and:

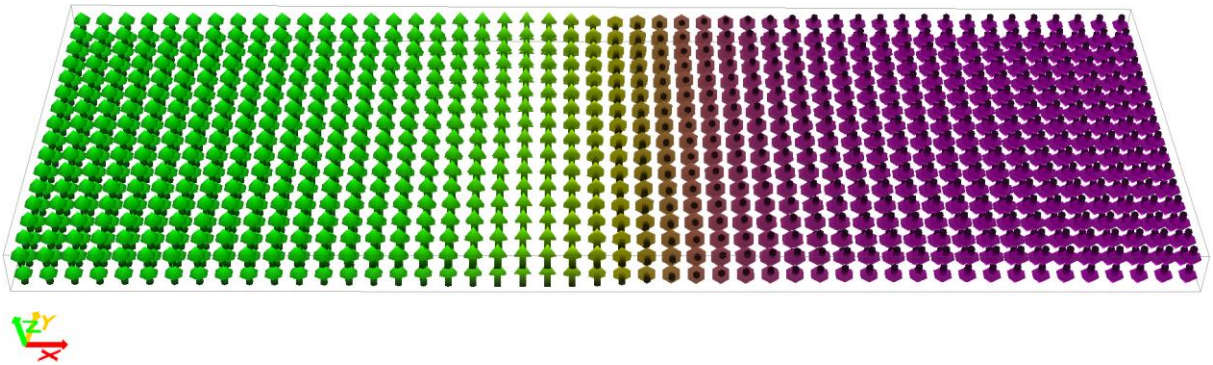
a) in-plane head-to-head transverse domain wall as **dwall** x y 240nm 0



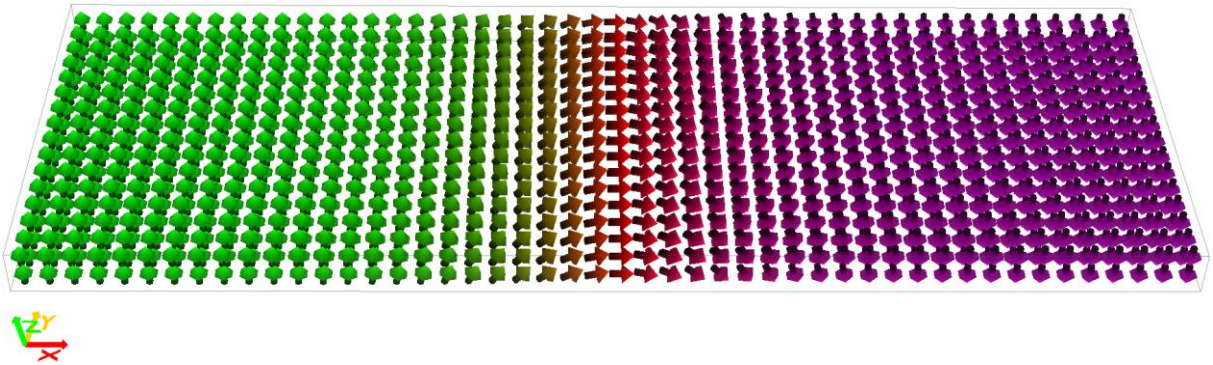
b) in-plane tail-to-tail transverse domain wall as **dwall** -x -y 240nm 0



c) Bloch domain wall as **dwall** z y 240nm 0



d) Néel domain wall as **dwall** z x 240nm 0



Exercise 4.1

Set a mesh for a permalloy rectangle of dimensions 640x80x5 nm with 5 nm cellsize. Generate a head-to-head domain wall over this wire. Run the simulation without a stopping condition and observe how the domain wall is relaxed. You will observe the domain wall does not remain in the center, but eventually drifts towards one side until it is expelled at one of the edges. Can you explain why this happens?

Setting Stray Fields

For domain wall mobility calculations and domain wall configuration relaxation problems in very long wires, it is possible to extend the wires outside of the ferromagnetic mesh by using external uniformly magnetized magnetic bodies, and to calculate the stray field inside the mesh, thereby allowing a smaller mesh size – this eliminates the domain wall drift problem noted in Exercise 4.1. This is done by adding dipole meshes at the left and right-hand-side of the *permalloy* mesh with magnetization direction set as a continuation of the magnetization inside the *permalloy* mesh, and enabling the *strayfield* module.

To add a dipole mesh use the following:

adddipole *name rectangle*

A dipole mesh has a uniform magnetization orientation which is not evolved by the ODE solver but may be handled in a similar manner to ferromagnetic meshes (such as the *permalloy* mesh). Thus modules may be added for computation (**modules**), mesh parameters edited (**params**), quantities displayed (**display**), etc.

You should also exchange couple the ends of the wires to the dipole meshes, thus completing the approximation of a long wire with a domain wall in the center. To enable this use the command:

coupletodipoles

Click on the interactive object to enable exchange coupling to the On state – with this flag turned on all magnetic cells in a ferromagnetic meshes, at the interface with a dipole mesh, are exchange coupled to the fixed dipole magnetization direction.

Exercise 4.2

a) Set two dipole meshes to the left and right of the *permalloy* mesh from the previous exercise, with lengths of 2.56 µm (but same width and thickness). These dipole meshes should now be visible when using the **mesh** command – see Figure 4.2. Set their magnetization orientation in order to extend the head-to-head domain wall configuration from Exercise 4.1 (use **setangle** remembering to specify the mesh name – see **?setangle** for details) – note, the dipole meshes do not display anything by default, you will need to use the **display** command and click the *M* interactive object in order to see their magnetization orientation.

   Run the simulation to relax the domain wall configuration to |mxh| < 10^{-5}. What does the stray field from the dipole meshes look like? (use the **display** command)

Figure 4.2 – Configuration of dipole meshes for exercise 4.2

```
Magnetic supermesh (0m, 0m, 0m; 640nm, 80nm, 5nm) with cell (5nm, 5nm, 5nm) Electric supermesh (N/A) with cell (N/A)
Mesh (permalloy) with rectangle (0m, 0m, 0m; 640nm, 80nm, 5nm)   Magnetic cell (5nm, 5nm, 5nm) Electric cell (N/A) Thermal cell (N/A)
Mesh (dip_left) with rectangle (-2.56um, 0m, 0m; 0m, 80nm, 5nm)  Magnetic cell (N/A)           Electric cell (N/A) Thermal cell (N/A)
Mesh (dip_right) with rectangle (640nm, 0m, 0m; 3.2um, 80nm, 5nm) Magnetic cell (N/A)           Electric cell (N/A) Thermal cell (N/A)
```

b) Change the *permalloy* mesh from part a) to a new size of 640x160x30 nm, making sure the dipole meshes are also scaled accordingly (to 160nm width and 30 nm thickness). For the purposes of this exercise you may use a 2D approximation by setting a cellsize of 5x5x30 nm.

   Run the simulation to relax the domain wall configuration to |mxh| < 10^{-5}. What type of domain wall results?

When changing mesh dimensions with multiple meshes added to the simulation, there are two options available: i) change just the mesh rectangle required, ii) change the mesh rectangle required and resize/translate all other meshes in proportion. To change the behaviour of the program use the following command and click the interactive object to the On state:

scalemeshrects

With multiple meshes, clicking on the mesh name interactive object (e.g. as displayed using the **mesh** command) changes the display focus to that mesh and resets camera orientation – try it. To quickly focus on a mesh without changing the camera orientation you can double-click on a mesh in the display window.

Exercise 4.3

In this exercise you will calculate the domain wall width for a symmetric transverse domain wall as a function of wire width and compare it to the values obtained using the domain wall formula (A is the exchange stiffness – see the **params** command – and K_u is the anisotropy energy density).

$$\Delta_{dw} = \pi \sqrt{\frac{A}{K_u}}$$

a) Relax domain walls as a function of wire width for a permalloy mesh with dimensions 640 x Width x 5nm, where Width ranges from 40 nm up to 160 nm (simulate at least 4 different values of width). Obtain the domain wall width defined as the half-M_s width value – i.e. the distance it takes for the longitudinal component to change from $+M_S/2$ to $-M_S/2$ for a head-to-head domain wall – and compare it to the value obtained using the formula above (when calculating K_u remember the longitudinal demagnetizing energy is assumed to be negligible as for a very long wire)

To obtain the domain wall profile you can use the following command:

dp_getprofile *start end dp_index*

The above command saves numerical data from the currently displayed mesh quantities (magnetization in this case) along a line starting from the *start* up to the *end* Cartesian coordinates (absolute position values, i.e. not relative to any mesh). The data is saved in internal data processing arrays – more on these in a separate tutorial. For now just obtain a magnetization profile through the middle of the wire as (e.g. for the 80nm wire width): **dp_getprofile** *0 40nm 0 640nm 40nm 0 0.*

The magnetization components are saved in the data processing arrays with indexes starting at 1 (so 1, 2, and 3), whilst data processing array 0 contains the position value. These can be saved to a file as numerical columns using the **dp_save** command as **dp_save** *(directory\)filename.txt 0 1 2 3*. The file will contain the 4 columns as position along the profile (so x coordinate), Mx, My, Mz.

b) Repeat the exercise for a thickness of 10nm using both a 3D simulation (cubic cellsize of 5x5x5 nm) and a 2D approximation (cellsize of 5x5x10 nm). How do the width values compare to those predicted by the formula and are the results obtained using the 2D and 3D model similar?

Tutorial 5 – Domain Wall Movement and Data Processing

In this tutorial you will learn how to obtain a domain wall field-driven mobility curve.
In order to simulate domain wall movement, in addition to setting up a domain wall and dipole meshes as in the previous exercise, the moving mesh algorithm must be enabled by setting a “triggering mesh” as:

movingmesh *mesh_name*

When moving mesh is enabled the magnetization is shifted either to the left or to the right by one notch at a time in order to keep the average x component of magnetization in the triggering mesh, <Mx>, within set boundaries. This keeps the domain wall roughly in the centre of the mesh. When the mesh is shifted to the left or to the right, the data parameter *dwshift* is changed. This data parameter is available for saving to file – see list of data parameters using the **data** command, as discussed previously. Note, this can also be displayed in the console using **showdata** *dwshift*, or displayed in the data box. By saving the simulation time and domain wall shift, the domain wall velocity can be calculated using linear regression.

Since this type of computation is common, there is a shortcut command which sets-up everything required (adding dipole meshes with exchange coupling to the ferromagnetic mesh, enabling stray field computation, setting a domain wall and a triggering mesh):

preparemovingmesh (meshname)

Exercise 5.1

a) Set a 320 × 80 × 20 nm permalloy rectangle with 5x5x20 nm cellsize (2D problem). Enable the moving mesh algorithm and let the domain wall relax in zero field.

b) Set a simulation stage with a field sequence starting from 500 A/m to 2000 A/m in 500 A/m steps, keeping each field step for exactly 2 ns. Set a data

save file, and make sure you save the stage time and dwshift parameters every 50 ps.

c) For each field step extract the gradient of the dwshift vs time and plot the wall velocity as a function of field. How does the velocity compare with that predicted by the formula below? (K_u is the in-plane anisotropy energy density)

$$v_{dw} = H\frac{\gamma}{\alpha}\sqrt{\frac{A}{K_u}},\ \ where\ \gamma = \mu_0|\gamma_e| \cong 221276\ (m/As)$$

<u>Console Data Processing</u>

There are a number of built-in commands which allow for a number of operations to be performed on data processing arrays.

First of all, it is possible to load tab-spaced data from a file (such as the data files produced by a Boris simulation) into the internal data processing arrays. This is done using the following command:

dp_load *filename filecol1 … dp_arr1 …*

The above command loads entire columns from the specified file. Thus if the file has a number of tab-spaced data columns, we can load the column with number *filecol1* from the file into the internal data processing array with number dp_*arr1* (these indexes are numbered from 0 up). Multiple columns can be loaded in one command.

Some common data processing commands are listed below.

To multiply a data processing array by a constant value use the following command:

dp_mul *dp_source value dp_dest*

The above command multiplies the data processing array with index *dp_source* by the specified value and stores the result in the *dp_dest* data processing array (this can be the same as *dp_source*). This can be used to normalize data (e.g. a hysteresis loop).

We can use the built-in linear regression command to extract the velocity values:

dp_linreg *dp_x dp_y (dp_z dp_out)*

The above command performs linear regression on the data stored in *dp_x* and *dp_y* data processing arrays and outputs the extracted gradient values and intercepts together with their uncertainties. If *dp_z* is specified multiple linear regressions are performed by using the values in the *dp_z* array to identify adjacent points to be included in a single linear regression; e.g. *dp_z* would contain the applied field values. In this case the outputs are placed in 5 data processing arrays starting at *dp_out* as follows: 1) unique *dp_z* values, 2) gradient, 3) gradient error, 4) intercept, 5) intercept error.

We can also save our processed data, e.g. the domain wall velocity curve, using:

dp_save *filename dp_arr1 …*

Exercise 5.2

Use the console data processing commands to process the output data from Exercise 5.1 and save a domain wall velocity curve.

Other notable commands include:

dp_coercivity *dp_x dp_y*
dp_remanence *dp_x dp_y*

These commands can be used on data from a simulated hysteresis loop in order to extract coercivity and remanence values.

The data processing arrays may be cleared using:

dp_clear *dp_arr1* …

This clears data in the specified data processing arrays. If no parameters are included all data processing arrays are cleared.

Exercise 5.3

Continuing Exercise 1, find the Walker breakdown threshold with a resolution of 100 A/m starting at 100 A/m. Compare the velocity values with that predicted by the formula in Exercise 5.1c for the steady domain wall movement regime. What is the Walker breakdown threshold?

Exercise 5.4

Calculate the field-driven domain wall mobility curve for permalloy, with a resolution of 100 A/m, as a function of Gilbert damping, for values of damping 0.005, 0.01 and 0.015. How does the Walker breakdown threshold compare with the value predicted by the formula below? ($K_{u,op}$ is the out-of-plane anisotropy energy density)

$$H_W = \frac{\alpha}{2}\frac{K_{u,op}}{\mu_0 M_S} \quad (A/m)$$

Tutorial 6 – ODE Control and Setting Shapes

Setting an ODE solver

The differential equation to solve and its evaluation method is configured using the following command:

ode

The default equation is the Landau-Lifshitz-Gilbert (LLG) equation which you have been using so far. Other equations may be set, e.g. LLB for temperature-dependent simulations, which will be covered in other tutorials.

There are a number of evaluation methods which you can select. The fixed-step methods available are: *Euler* (1st order), trapezoidal Euler (*TEuler* – 2nd order) and Runge-Kuta (*RK4* - 4th order). The adaptive time-step methods are the adaptive Heun (*AHeun* – 2nd order), the multi-step Adams-Bashforth-Moulton (*ABM* – 2nd order), Runge-Kutta-Bogacki-Shampine (*RK23* – 3rd order with embedded 2nd order error estimator), Runge-Kutta-Fehlberg (*RKF45* – 4th order with embedded 5th order error estimator), Runge-Kutta-Cash-Karp (*RKCK45* – 4th order with embedded 5th order error estimator), and Runge-Kutta-Dormand-Prince (*RKDP54* – 5th order with embedded 4th order error estimator).

There is also a Steepest Descent solver (*SDesc*) used for static problems where we don't need the magnetization dynamics, but are merely interested in calculating the ground state (e.g. relaxing a magnetization configuration and hysteresis loops). The steepest descent solver is only enabled for the *LLGStatic* equation, which is the LLG equation with the precession term disabled and damping value set to 1. In this case the *SDesc* solver is typically at least an order of magnitude faster in computing the ground state compared to the other methods.

For most methods the calculated *mxh* and *dmdt* stopping condition values are the maximum |**m**x**h**| value (normalized torque) in any given iteration. The exceptions are

the *Euler*, *TEuler*, and *AHeun* methods, which are only ever used in practice for stochastic equations when including a thermal field (more on this later). For this reason the |**m**x**h**| values for these are the mesh averages in any given iteration. This allows using these methods with an *mxh* stopping condition for stochastic equations.

As an exercise we will briefly investigate here the stability of the fixed-step methods as the time step is changed. The time step may be set using:

setdt *dt*

Here *dt* is the time value in seconds. For the adaptive time step methods this command sets the starting time step.

Exercise 6.1

a) Set a 320 × 160 x 20 nm permalloy mesh with a 5 nm cubic cell and relax the magnetization to |mxh| < 10^{-5}.

b) Set a stage with a magnetic field with components (40 kA/m, 5 kA/m, 0) for 5 ns and save data every 10 ps. Use the *RKF45* method. Record the actual computation time required to complete the simulations. Plot the magnetization switching dynamics.

c) Repeat the simulation using the *RK4* method for fixed time steps of 0.5 ps, 0.7 ps, 0.9 ps and 1.1 ps. Record the actual computation time required to complete the simulations. Compare the results with the reference results from the *RKF45* method. How do the results change and why?

d) Compare the computation times. Which method is more efficient whilst still maintaining accuracy?

e) Investigate the computation time required to complete the same problem with *TEuler* with a time step of 50fs and *Euler* with a time step of 30as.

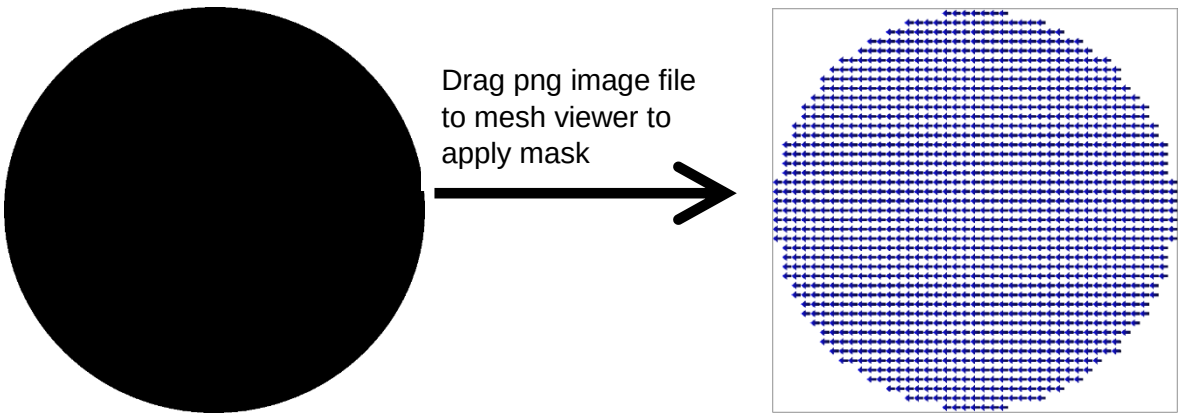
Setting shapes

Until now we've mostly considered meshes which are filled with magnetic cells. In general, complex shapes may be generated by masking the mesh using a shape in an image file. A more advanced method of setting shapes using numerical ovf2 files is covered in a later tutorial. This is achieved using the following command:

loadmaskfile *(zDepth) (directory\)maskfile*

In the simplest case the image file defines a 2D shape in black and the void cells in white – see Figure 6.1 for an example. Instead of using the command you may also drag and drop the image in the mesh viewer window. The *zDepth* value defines the depth the mesh is voided to from top down (if *zDepth* > 0), or the height the mesh is voided to from bottom up (if *zDepth* < 0); this may be used to define 3D shapes with the maskfile being a grayscale image.

Figure 6.1 – Setting a mesh shape using a mask from a png file



Exercise 6.2

a) Load an ellipse into a 320×160×10 nm mesh. Try not to leave void cells at the sides. You should use a circle mask as this will be stretched over the defined mesh rectangle.

b) Obtain hysteresis loops between -50 kA/m and 50 kA/m for the ellipse along the x axis and along the y axis separately using a cellsize of 5x5x10 nm (2D simulations). You may use the relaxation condition $|mxh| < 10^{-4}$ in order to speed up the exercise. Use the *RKF45* evaluation method. How do the two hysteresis loops compare ?

c) Obtain further hysteresis loops for ellipses with dimensions 260x160x10 nm and 200x160x10 nm. Compare results for all the simulated ellipses and explain the changes in the hysteresis loops.

There is a modifier for how shapes are applied to a mesh, accessed using the **individualshape** command. By default this flag is Off, and any shape applied to the mesh is set for all relevant computational quantities. Thus **M** (magnetisation) is a computational quantity which may be shaped, but also *T* (temperature), and σ (electrical conductivity) may be shaped for the relevant solvers (micromagnetics, heat equation, and transport solver respectively). Note that in order to shape *T* and σ you need the relevant solvers enabled. If instead you only want to apply the shape to one of these quantities, or even have different shapes for all of them, you need to enabled the **individualshape** flag, then apply the mask. This is useful for example if you want to include non-magnetic components (e.g. contact leads) in the magnetic mesh.

You may reset the mesh back to its solid shape using:

resetmesh

This command is also useful to recover the mesh following a wrongly-posed computation – e.g. if too large a time-step is used the magnetization values will become *NaN* (not a number) and must be reset. Other methods to shape a mesh include setting and deleting rectangles using:

delrect *rectangle (meshname)*
addrect *rectangle (meshname)*

Tutorial 7 – Magnetocrystalline Anisotropy

In this tutorial you will learn how to use the anisotropy module and simulate hysteresis loops for different magneto-crystalline anisotropy configurations. You need to be familiar with all the basic tutorials.

There are two options available for adding magneto-crystalline anisotropy to the computations: uniaxial or cubic. These are enabled by choosing the *aniuni* or *anicubi* modules from the list displayed using the **modules** command. The modules are mutually exclusive, thus enabling one will delete the other one from the list of active modules.

The strength of the anisotropy is controlled using the *K1* and *K2* parameters (K1 and K2 are the anisotropy energy density constants) from the list displayed using the **params** command. These are the constants that appear in the anisotropy energy formulas:

Uniaxial anisotropy:

$$\varepsilon = K_1\left[1-(\mathbf{m}.\mathbf{e}_a)^2\right]+K_2\left[1-(\mathbf{m}.\mathbf{e}_a)^2\right]^2$$

Cubic anisotropy:

$$\varepsilon = K_1\left[\alpha^2\beta^2+\alpha^2\gamma^2+\beta^2\gamma^2\right]+K_2\left[\alpha^2\beta^2\gamma^2\right], \textit{where}$$
$$\alpha = \mathbf{m}.\mathbf{e}_1,\ \beta = \mathbf{m}.\mathbf{e}_2,\ \gamma = \mathbf{m}.(\mathbf{e}_1\times\mathbf{e}_2)$$

We also need to define the anisotropy symmetry axes. For uniaxial anisotropy we only have one symmetry axis and this is set using the *ea1* parameter by giving the Cartesian components of the unit vector $\mathbf{e}_a$ (e.g. default is 1, 0, 0 for easy axis along the x-axis). For cubic anisotropy we need two symmetry axes directions, *ea1* and *ea2* which should normally be orthogonal.

In the following you will investigate the effect of magnetocrystalline anisotropy on hysteresis loops in circular dots.

Exercise 7.1

a) Set a 160 × 160 × 5 nm permalloy circle with 5 nm cellsize using a mask file. Set a uniform magnetization along the x direction towards the left (blue state). Set uniaxial anisotropy with K1 = 10 kJ/m^3, K2 = 0 J/m^3 and easy axis along x direction.
b) Simulate hysteresis loops along the x-axis (easy axis), y-axis (hard axis) and in between along a 45° in-plane direction (remember you will need to use a polar field sequence for this). You will need to determine appropriate field sweep ranges so the loops start from a saturated magnetization state.

c) Plot the hysteresis loops using the normalized magnetization (divide by M_S value – the saturation magnetization constant). What are the coercivity and normalized remanence values? Explain the difference between the loops.

Exercise 7.2

Repeat the simulations in Exercise 7.1, but this time set cubic anisotropy with K1 = 20 kJ/m^3 and K2 = 0 J/m^3, with two perpendicular easy axes in the plane (e.g. x-axis and y-axis).

Plot the resulting hysteresis loops and compare them with the previous results.

Hints:

For the 45° direction you will need to project the magnetization along the applied field direction. You can do this by taking the dot product of **M** with the applied field direction unit vector:

$$M_H = \hat{\mathbf{h}}.\mathbf{M}$$

You can do this using console data processing with the command:

dp_dotprod *dp_vector ux uy uz dp_out*

Here *dp_vector* is the dp array index such that the dp arrays *dp_vector, dp_vector + 1, dp_vector + 2* hold the x, y, and z components of the magnetization, (*ux, uy, uz*) are the components of a vector (dot product taken with this vector), and *dp_out* is the dp array where the output is placed.

Finally, you can normalize the magnetization using the **dp_div** command where you will need to divide by M_S. To see the value of M_S you can look it up in the list displayed using the **params** command.

Remember you will first need to load into dp arrays the appropriate columns for the saved hysteresis loop data file using the **dp_load** command. You can save the contents of dp arrays after processing data using the **dp_save** command. As always you can find more details about a command by preceding it with the ? symbol, e.g. **?dp_load**.

Tutorial 8 – Anisotropic Magneto-Resistance

In this tutorial you will learn how to simulate magneto-resistance loops and calculate charge current densities using the *transport* module.

Transport Module Basics

The *transport* module is a complex spin and charge current solver (electron transport), allowing for a number of physical effects to be included in the magnetization dynamics problem, including Zhang-Li spin-transfer torques based on calculated charge currents, spin torques based on computed spin accumulations in multilayers, direct and inverse spin Hall effects (SHE and ISHE), spin pumping torques, anisotropic magneto-resistance (AMR), current-perpendicular-to-plane giant magneto-resistance (CPP-GMR), Oersted fields and Joule heating.

Here we will look at how a simple charge current density may be computed and AMR included in the simulation.

You will first need to enable the *transport* module from the list displayed using the **modules** command for the meshes where you want a charge current density to be computed. In the simplest case the computation is reduced to obtaining **J**, the charge current density, using Ohm's law : $\mathbf{J} = \sigma\mathbf{E}$, where σ is the electrical conductivity and $\mathbf{E} = -\nabla V$ is the electrical field with V being the electrical potential. If σ is constant this reduces to a Laplace equation for V.

The electrical conductivity, potential and charge current density are available as display outputs under the **display** command (*elC, V* and *Jc*).

The base electrical conductivity value may be changed by editing the *elC* mesh parameter displayed using the **params** command.

Similarly AMR may be enabled by editing the *amr* mesh parameter (0% by default which disables it). A typical value for permalloy is 2%. Enabling AMR results in a

non-uniform electrical conductivity and now the equation for V becomes a Poisson equation.

Before starting a computation you will need to define at least 2 electrodes – these set Dirichlet boundary conditions for V (fixed potential values) in the Laplace/Poisson solvers. The most common electrode configuration is to define two electrodes at the x-axis ends of the mesh (so in the y-z plane). You can do this using the command:

setdefaultelectrodes

To see which electrodes have been defined use the command:

electrodes

You will see two electrode rectangles. The electrode rectangles are in absolute values, so not relative to any particular mesh. You can add new electrodes but they must always be placed at the edges of a mesh rectangle – when initializing the simulation Dirichlet boundary conditions will be flagged for the boundary cells of the mesh intersecting the electrode rectangle. Each electrode has a fixed potential which may be edited. Exactly one of the electrodes has to be designated as the ground – this is the electrode where the outgoing total electrical current is calculated.

You can edit the individual electrode potential values, however a more common scenario is the set a single electrical potential drop from the ground to the other electrodes using:

setpotential *potential*

This sets a single inversely-symmetrical potential drop (i.e. +*potential/2 to -potential/2*). The inversely-symmetrical potential drop minimizes floating point errors (as opposed to setting a potential drop of *potential* to 0).

Simulations may use the constant-voltage or the constant-current mode (the interactive object displayed when using the **electrodes** command may be toggled

between these two states). Normally you would use the constant-voltage mode; with constant-current the electrode potentials are adjusted during the simulation to maintain a constant current (which may be set using the **setcurrent** command). In this tutorial we will be using the constant-voltage mode.

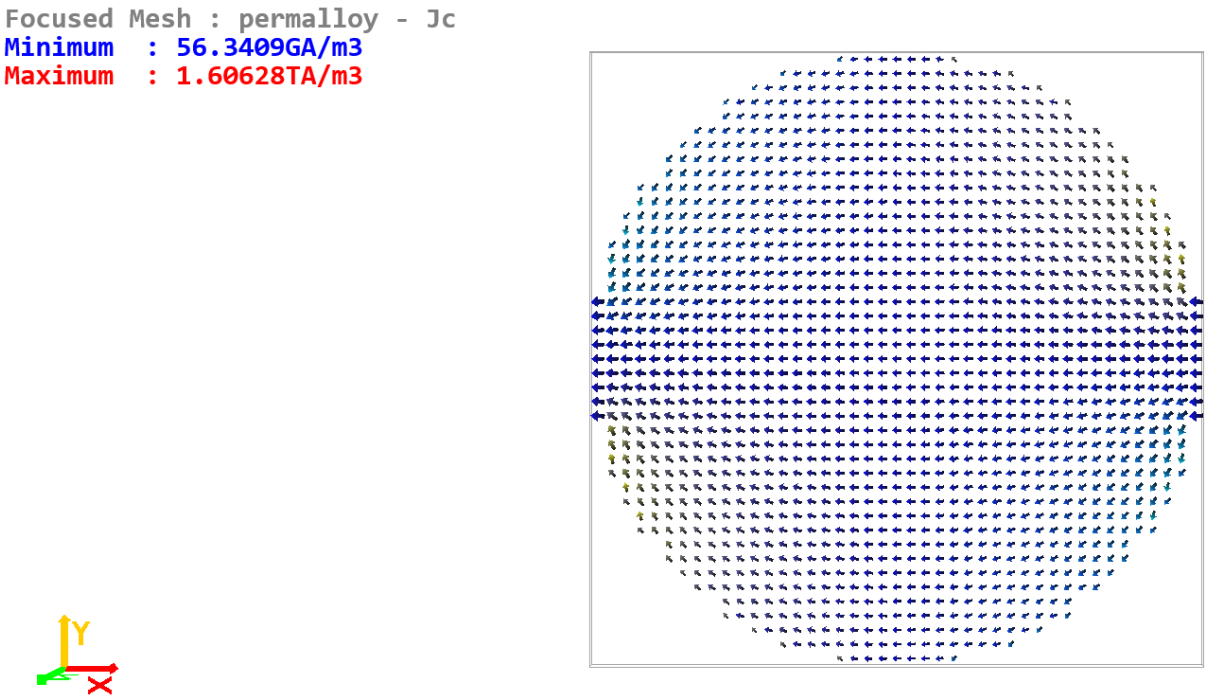
Exercise 8.1

Set a permalloy mesh with dimensions 320x320x10 nm and mask it using a circle shape (in the image file make sure to not leave any white spaces at the left and right sides).

Enable the transport module, set the default electrode configuration and a potential of 10 mV.

In the data box display the average current density (*Jc*), set potential (*V*), total current (*I*) and resistance (*R*). In the mesh display the current density. Run the simulation. The calculated current density should look similar to that in Figure 8.1.

Figure 8.1 – Computed charge current density for Exercise 8.1



For more advanced simulations you can add electrodes using the **addelectrode** command. Electodes may be deleted using the **delelectrode** command (or more simply by right-clicking on an existing electrode in the list displayed by the **electrodes** command. All electrodes may be deleted by using the **clearelectrodes** command.

Further info:

When displaying the meshes (use the **mesh** command) you will now notice a value for the electric cell. This is the discretization cellsize used by the transport solvers. Normally this should be equal to the magnetic cellsize (default setting) but can be controlled separately for more advanced simulations (decrease computation time or increase computation accuracy as required). Multiple meshes with transport modules enabled may be configured. If the meshes are touching, composite media boundary conditions will automatically be inserted in the computation, however we still typically require 2 electrodes for a well-posed problem.

Exercise 8.2

Set a permalloy rectangle with dimensions 300x100x20 nm. Calculate the current density for the default electrodes setting by using a potential drop of 1 V.

What is the computed sample resistance and does it agree with that predicted by the formula:

$$R = \frac{\rho l}{A},$$

where l is the length, A the cross-sectional area and ρ is the resistivity.

What is the total current, and does it agree with the expected value for a 1 V potential drop?

What is the average current density and does it agree with the expected value for the total current?

Further info:

For advanced simulations the accuracy of the transport solver may be controlled by using the command:

tsolverconfig

This command displays the set convergence error (a value around 10^{-6}) is normally a good compromise between accuracy and computational speed. In this case the Laplace/Poisson solvers stop iterating when the maximum change in *V* from one iteration to another, normalized to the set potential drop, drops below this set convergence value. You can display the *ts_iter* (current number of transport solver iterations) and *ts_err* (current transport solver error) data in the data box. If the convergence error threshold is too low the transport solver will take a large number of iterations during computations – if AMR, GMR, ISHE or temperature-dependent transport parameters are enabled the transport solver must update after every magnetization and/or heat solver time step.

The transport solver may be used to calculate magneto-resistance loops when an AMR (anisotropic magneto-resistance) value is set. Since this is a static problem it is best to set the *LLGStatic* equation with the *SDesc* solver. When using the *SDesc* solver the stopping condition (*mxh* or *dmdt*) should be much lower than usual, typically at least 10^{-6} or even lower. You should also set the static transport solver flag to On (see **tsolverconfig** command output). Setting this flag to On will only iterate the transport solver at the end of a schedule step, thus resulting in faster computation. In this case you should also increase the iterations timeout to at least 5000 or more to ensure the transport solver converge threshold is met.

For problems where the current density is uniform a transport solver converge threshold of 10^{-6} is sufficient, however this may have to be decreased to 10^{-8} or even 10^{-9} if the current density is highly non-uniform.

In the following problem you should use both the static LLG equation and static transport solver as explained above.

Exercise 8.3

Here you will obtain longitudinal and transverse magneto-resistance loops.

a) Set a permalloy rectangle with dimensions 160x80x10 nm. You can use a 2D simulation with magnetic cellsize 5x5x10 nm, but the transport solver should be left with cubic electric cellsize of 5 nm.

   Sweep the field from -100 kA/m to +100 kA/m strength and back, using a field step of at most 1 kA/m, along a nearly longitudinal direction (use 5° from the x-axis). You should use a *Hpolar_seq* sequence, and an *mxh* stopping condition of 10^{-7}.

   Set the transport solver with default electrodes, a non-zero potential drop (e.g. 1 mV), and *amr* = 2% (see **params**).

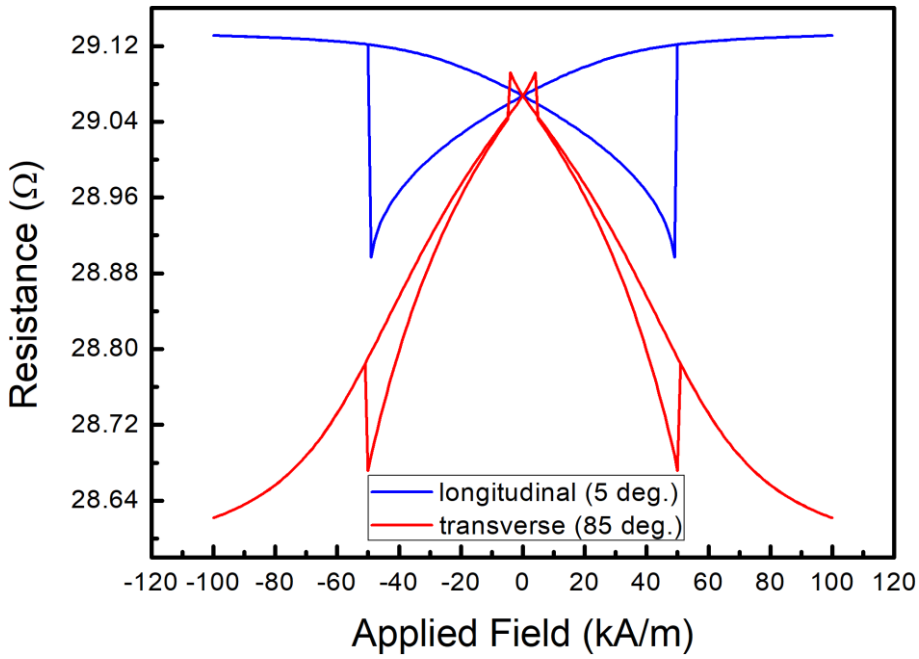
   In the output data make sure to save the applied field and sample resistance every step. You should also save the transport solver error to check the converge threshold has been met at the end of each step (*ts_err*).

   Run the simulation and plot the obtained MR loop. Explain your results.

b) Repeat the simulation along the in-plane nearly transverse direction (use 85° from the x-axis) by sweeping the field between -100 kA/m and +100 kA/m strength with a step of 1 kA/m.

Estimate the AMR percentage from the obtained H vs R loops from parts a) and b) – does it agree with the set 2% value?

Figure 8.2 – Longitudinal and transverse MR loops due to AMR as calculated in Exercise 8.3.



Tutorial 9 – Scripting using Python

In this tutorial you will learn how to use scripts to automate multiple simulations. Boris can communicate with an external program via network sockets, so both local and remote script-based control is possible. For this purpose a Python module (NetSocks) is provided, allowing use of Python scripts to communicate with Boris. This is contained in the NetSocks.py file and requires **Python 3.7**.

The scripts work by sending console commands, with syntax identical to that you would use when typing commands directly in the console. Some console commands also return values, which can be read by Python scripts. You can find out if a console command is set to return values by looking at the USAGE help for it. For example type the following in the console:

dp_coercivity

In the USAGE help you can see this command will return the calculated coercivity value (see the description "<script return values: firstcoercivityvalue>").

<u>NetSocks Python Commands Usage</u>

Look through the examples in Tutorial 0. To create a new simulation script:

1. Place the NetSocks.py file in the same directory as your Python simulation script file and import NSClient from NetSocks.

2. Next a NSClient object, *ns*, must be created as:

   ns = NSClient('localhost')

   If running the script remotely the *localhost* entry needs to be replaced by the IP address (port 1542 is used, which must not be blocked in the firewall) of the computer running the Boris executable.

Now all communication is done through methods in the *ns* object.

All commands can be sent through the ns object as ns.command(parameters…), for example *ns.setfield(90, 0)* sets a magnetic field along the x axis.

You can obtain returned parameters as data = ns.command(parameters…), for example *Mav = ns.showdata('<M>')* returns the average magnetisation as a list with 3 elements.

In many cases the script must wait for a simulation to finish before proceeding. This is achieved by using the Run function:

- *ws.Run()*

This is a blocking function call, which sends the **run** command, then waits for the simulation to finish by listening to messages from the running program.

NSClient also has a useful method for writing data to a file:

- ns.SaveDataToFile('Results.txt', [data1, data2, …])

This command appends a new line in the Results.txt file (in the script directory) which contains two numbers: the x and y components of magnetization.

Look through the examples in Tutorial 0 and run them.

Examples of other scripted simulations will be contained in further tutorials.

Tutorial 10 – Current-Induced Domain Wall Movement

In this tutorial you will learn how to obtain current-induced domain wall velocity curves. First we will simulate these using only console commands, including processing of output data, then we will use a Python script to more accurately determine the domain wall velocity.

The simplest available method for enabling spin torques is to use the Zhang-Li spin-transfer-torque (STT) formulation. In this formulation the calculated charge current density is used to calculate the following spin torques on the magnetization (included as additive terms in the normalised LLG equation):

$$T_{STT} = (\mathbf{u}.\nabla)\mathbf{m} - \beta\mathbf{m} \times [(\mathbf{u}.\nabla)\mathbf{m}]$$

where

$$\mathbf{u} = \mathbf{J}_{\mathbf{C}} \frac{P}{M_S} \frac{\mu_B}{e} \frac{1}{1+\beta^2}$$

To enable this use the **ode** command and select the *LLG-STT* equation with the *RK4* evaluation method. You will also need to enable the *transport* module (use the **modules** command and select the *transport* module). As before you need to set two electrodes (**setdefaultelectrodes**) and enable the moving mesh algorithm (**preparemovingmesh**).

In the above equation we have two new parameters: P, the charge current spin polarisation, and β, the STT non-adiabaticity parameter. These can be edited by using the **params** command and double-clicking on the respective interactive console objects; the default values are set for permalloy.

Exercise 10.1

a) Set a 320 × 80 × 10 nm permalloy rectangle with 5x5x5 nm cellsize (3D problem). Enable the moving mesh algorithm and let the domain wall relax in

zero field. Enable the transport module, set the default electrode configuration and enable the LLG-STT equation with the RK4 evaluation method.

b) Set a simulation schedule to vary the current density from 10^{11} A/m^2 up to 10^{12} A/m^2 in 10 steps, saving the domain wall shift (*dwshift*), current (*I*), voltage (*V*), and charge current density (*Jc*) output data. Each current density value should be maintained for 5 ns with a data saving schedule every 10 ps.

Hint: You should use the V_seq simulation schedule stage. This sets a sequence of voltage values with given start and stop values in a number of steps. Since we have a simple geometry you can calculate the required voltage values using the formula:

$$V = \frac{l|\mathbf{J}_\mathrm{C}|}{\sigma}$$

where *l* is the distance between electrodes (length of the magnetic mesh) and σ is the electrical conductivity (see the set value using the **params** command). *Solution: use V_seq with -4.57mV; -45.7mV; 10.*

Note: You can also vary the current density using the I_seq schedule stage – this defines a sequence of current values. Setting current values directly enables the *constant current* mode, i.e. the voltage drop between electrodes is adjusted during the simulation to keep the current constant (the opposite of this is the *constant voltage* mode).
You can see the current settings using the **electrodes** command.
You can also set individual values in the console using the **setpotential** or **setcurrent** commands.

c) Obtain and plot the domain wall velocity, *v*, using the method introduced in Tutorial 5 with **dp_linreg**, as a function of current density. Convert the current density to spin drift velocity using the formula:

$$u = |\mathbf{J}_\mathrm{C}| \frac{P}{M_S} \frac{\mu_B}{e} \frac{1}{1+\beta^2}$$

Verify that the following formula holds:

$$\frac{v}{u} = \frac{\beta}{\alpha}$$

Domain wall velocity curves may also be obtained using a Python script, allowing more accurate determination of the velocity. The problem with the **dp_linreg** method, it also includes in the regression domain wall displacement points at the start of each step. Since the domain wall requires some time to reach a steady state velocity (the acceleration is not zero at the start of each step) these initial displacement values should ideally be discarded.
With a Python script you can set a single voltage stage without saving any data (e.g. for 3 ns); this is followed by another voltage stage (e.g. again for 3 ns) during which data is saved, then a linear regression is performed to extract the velocity value for the set voltage value. The script then proceeds to set the next voltage value and so on.

Exercise 10.2

Repeat the simulation in Exercise 10.1 but using a Python script to control the simulation flow and data output.
(*Solution: see the attached Python script*)
Note, in general you may need to adjust the stage duration times (1 – to achieve steady state motion and 2 – to generate enough data to obtain a representative velocity value using linear regression) for best results, but the suggested 3 ns, 3 ns breakdown will be sufficient for this exercise.

When setting up the Python script you will need to use the appropriate commands to edit the stage stopping and saving conditions, as well as the stage value. These commands are:
edistagevalue
editstagestop
editdatasave

Tutorial 11 – Oersted Fields

In addition to spin-transfer torques, electrical currents may also interact with magnetization via the generated Oersted fields. In this tutorial we will set-up a simple bilayer mesh structure, consisting of a magnetic wire and a non-magnetic metallic capping layer, then repeat the domain wall velocity simulations from the previous tutorial but also taking into account the generated Oersted field. Since the bilayer structure has a broken mirror symmetry in the z direction we might expect the domain wall speed to be different depending on the current direction.

To enable the Oersted field you need to enable the *Oersted* module (use the **modules** command). The *Oersted* module is an electric super-mesh module, i.e. it is calculated on the electric super-mesh with a separately controlled cell-size. After enabling the *Oersted* and *transport* modules bring up the configured meshes using the **mesh** command. You should now notice the electric super-mesh rectangle with its cellsize; the electric super-mesh is simply the smallest rectangle containing all meshes with the *transport* module enabled.

To set a non-magnetic metallic capping layer you will need to add a new mesh, in particular an electrical conductor mesh. This is done using the command:

addconductor *name rectangle*

Exercise 11.1

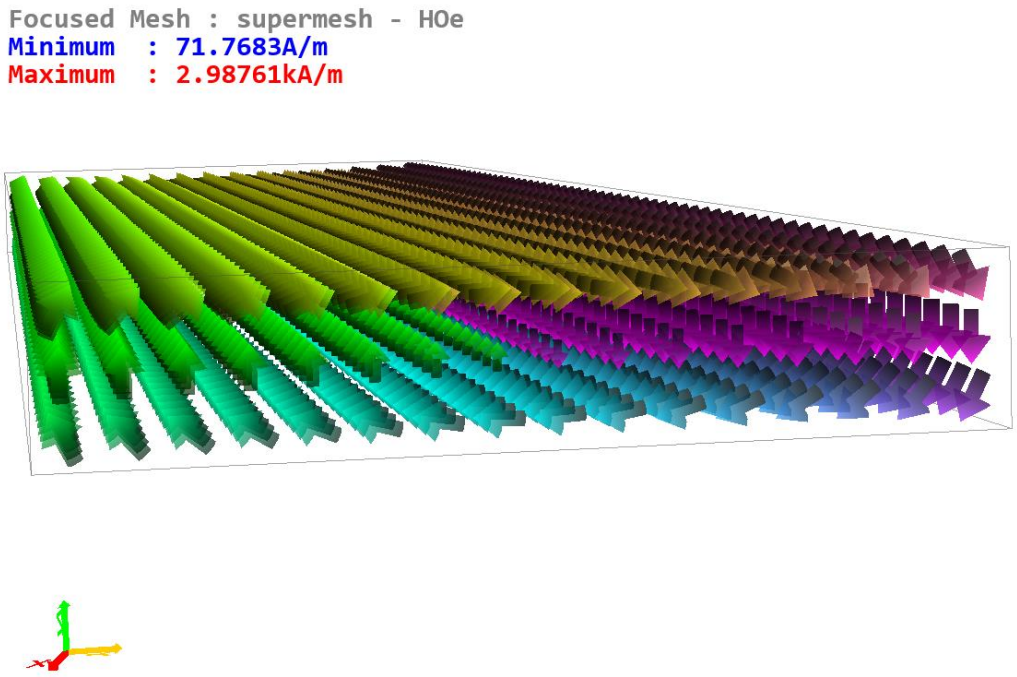
Set-up a simulation space as for Exercise 10.1. Add a metallic capping layer on top of the magnetic layer with a 5 nm thickness and enable its *transport* module (solution: use **addconductor** cap 0 0 10nm 320nm 80nm 15nm).

Set the electrodes to contact both meshes (use **setdefaultelectrodes** after both meshes had their *transport* module enabled).

Enable the Oersted field module and set a potential of 10mV (**setpotential** 10mV).

Compute a single iteration (**computefields**) and display the Oersted field (use the **display** command and select the *Oersted* display option on the super-mesh display line).

Figure 11.1 – Computed Oersted field for Exercise 11.1.



Exercise 11.2

Continuing from Exercise 11.1, use a Python script to simulate the domain wall velocity for both positive and negative currents in the magnitude range 10^{11} A/m^2 to 10^{12} A/m^2 in 10 steps. Plot the two ve
locity curves and compare them to the expected $v = (\beta/\alpha)u$ relation, as well as the simulations without an Oersted field. (Solution: see the Python script in the tutorial resources).

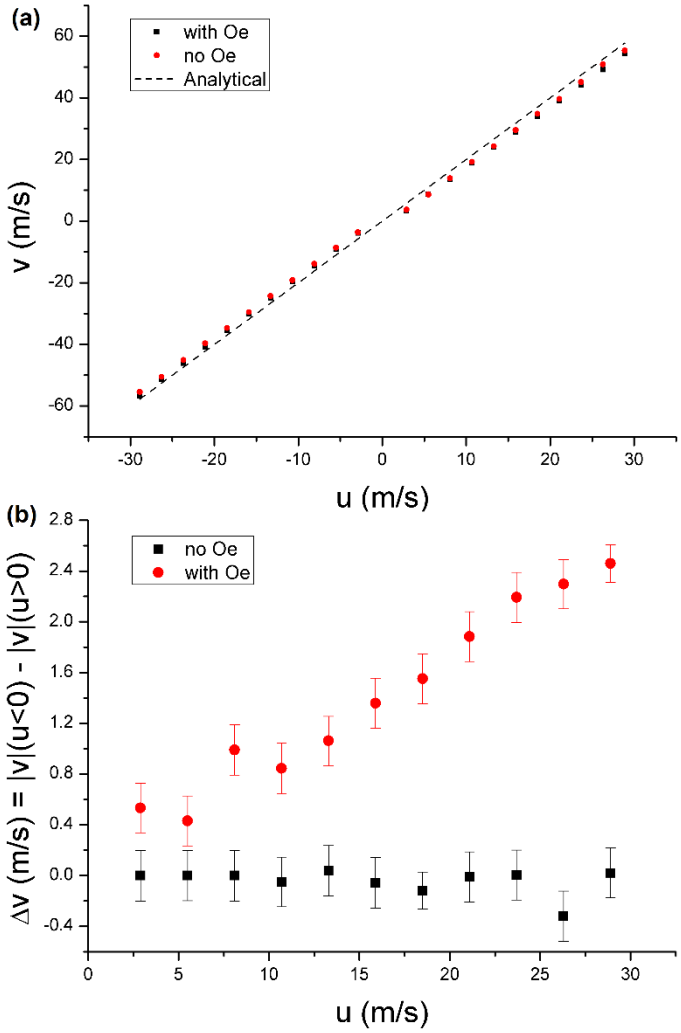
Note: when plotting *v* vs *u* you need to set the sign of *u* to be in the direction of electron drift, i.e. opposite sign to that of the charge current density.

In the above exercise the default electrical conductivity value of the capping layer is the same as for permalloy (this can be edited using the **params** command). In this case the current densities are the same in both layers. If you want to save output data for a particular mesh (e.g. Jc, the charge current density, which is mesh dependent) then you must focus on the required mesh first before adding that particular data to the output list.

You can focus on a particular mesh by clicking on the mesh name (bring up the list of meshes with **mesh** then click on a mesh name). Alternatively you can double click in the mesh graphical viewer on the required mesh.

Results from Exercise 11.2 are shown in Figure 11.2. The data obtained in Figure 11.2 could be improved further. The problem with using linear regression directly on the raw *dwshift* data, it contains steps due to the mesh discretisation. This is particularly problematic at low domain wall velocities where only a few steps are contained in the raw data, which can make the extracted velocity inaccurate.

Figure 11.2 – Simulation results from Exercise 11.2 showing a) domain wall velocity plotted against spin drift velocity for cases with and without Oersted fields and also compared with the analytical formula, and b) domain wall speed difference plotted against spin drift speed for cases with and without Oersted fields.



One possibility is to collect data for a longer time, but this is inefficient. Another possibility is to assume the domain wall displacement is linear with time (in this exercise this is a valid assumption). You can then either get rid of the repeated points before using linear regression, or replace the repeated points using linear interpolation. There is a built-in command for this, and is covered in a later tutorial on skyrmion movement (**dp_replacerepeats**).

Tutorial 12 – Surface Exchange, Multi-Layered Demagnetization and CUDA

Surface Exchange

Using the *surfexchange* module, two or more ferromagnetic meshes can be surface exchange coupled, allowing simulations of magnetic multilayers with RKKY interaction. The strength of the surface exchange coupling is controlled using the *J1* and *J2* material parameters: negative values result in anti-ferromagnetic coupling, positive values in ferromagnetic coupling. The *J1* parameter controls the strength of bilinear surface exchange, and *J2* controls the strength of biquadratic surface exchange.

Type **params** and have a look at *J1* and *J2*. For two meshes in surface exchange coupling, it is the top mesh *J1* and *J2* values that are used. This allows setting different coupling strength and types for the bottom and top of a mesh in a multi-layered structure. Boris allows surface exchange coupling only for xy planes, thus a multi-layered structure should be designed with the layers stacking along the z direction. Two ferromagnetic meshes will be surface exchange coupled if they both have the *surfexchange* module enabled and there's no other ferromagnetic mesh with the *surfexchange* module enabled in between them along the z direction. The coupling will only be calculated for cells which overlap in the xy plane.

Multi-Layered Demagnetization

To add another ferromagnetic mesh use the **addmesh** command. The *demag* module for each mesh only calculates the demagnetizing field for that ferromagnetic mesh alone. When 2 or more ferromagnetic meshes are being used, if you want to compute the overall demagnetizing field you need to use the supermesh *sdemag* demagnetizing field module. In this case the individual *demag* modules are disabled and the overall demagnetizing field is computed for the collection of individual ferromagnetic meshes, including all stray field contributions. There are two ways to

do this. The default method is called multi-layered convolution, and you can see the settings for this by using the command:

multiconvolution

This is an exact method of computing demagnetizing fields for a collection of computational meshes, which is able to handle arbitrary spacing and relative positioning between the layers without sacrificing accuracy or computational performance – see S. Lepadatu, Journal of Applied Physics 126, 103903 (2019) for details. In many cases you can rely on the default settings for multi-layered convolution, but for more advanced control you should read the information below.

Further info:

You can force the demagnetizing field to be computed in a 2D approximation in each mesh by clicking on the respective interactive console object (see output of the **multiconvolution** command, Force 2D : 2D meshes). This allows each mesh to have an arbitrary thickness, and is appropriate if the meshes are thin enough for the 2D approximation to hold. This option is turned off by default (Force 2D : Off), resulting in a 2D or 3D convolution in each mesh depending on their cellsizes. If the 2D approximation for each mesh is not appropriate you should use this option as long as the z cellsizes are the same in all meshes. If the z cellsizes differ the computation with Force 2D : Off may not be accurate, and in this case you should use the 2D layering option (Force 2D : 2D layered). This is similar to the 2D meshes option, however instead of forcing each mesh to be 2D, it is instead split into multiple layers, with each layer thickness set by the z cellsize in each respective mesh. This option is thus the most general case and allows exact computation of demagnetizing fields without any approximations, however it is in general more computationally expensive and should only be selected if the other 2 options are not appropriate.

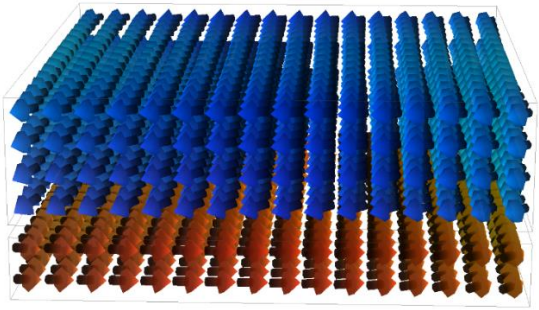
For multiple computational meshes with unequal sizes, the algorithm works by first transferring the magnetization values to scratch spaces with a common discretization. You can specify what this discretization should be, but by default it is calculated for you (see output of the **multiconvolution** command).

Another method of calculating demagnetizing fields for a collection of computational meshes is to use the so-called supermesh demagnetization. This is achieved by disabling the multi-layered convolution algorithm (see output of the **multiconvolution** command). This method calculates the demagnetizing field on the ferromagnetic supermesh by transferring magnetization and demagnetizing field values to and from the ferromagnetic supermesh using a local averaging smoother. Remember the ferromagnetic supermesh is the smallest rectangle containing all the ferromagnetic meshes and can be viewed using the **mesh** command. Computations on the ferromagnetic supermesh are done using its independent cellsize as can be seen in the console output of the **mesh** command. This cellsize will need to be carefully determined in each case to ensure accuracy of results. As a rule you should set the cellsize to be the minimum value out of the individual mesh cellsize values required to compute the demagnetizing field accurately separately. In most cases of interest, if full accuracy is required, this method is much slower than multi-layered convolution. It is also far less flexible, as it cannot accurately handle spacing between layers which cannot be exactly discretized.

Another use for the *sdemag* module without multi-layered convolution, is to calculate the demagnetizing field in an individual mesh with a different cellsize to that used for the exchange interaction. The exchange interaction typically requires a smaller cellsize to ensure accuracy, thus this method can be used to improve computational speed for larger simulations.

Starting from the **default** state, add another ferromagnetic mesh with dimensions of 80 nm × 80 nm × 20 nm, separated from the first mesh by 2 nm along the z direction. Enable the *surfexchange* module for both meshes, as well as the *sdemag* module. Now **run** the simulation and observe the result – see Figure 12.1.

Figure 12.1 – Anti-ferromagnetic surface exchange coupled ferromagnetic meshes



You can also add a metallic spacer layer in between, using the **addconductor** command, however this will not affect magnetic computations directly; it will be needed however if charge or spin transport computations are also enabled.

Exercise 12.1

a) Set-up a synthetic ferrimagnetic (SyF) Ni80Fe20 bilayer with elliptical shape of 320 nm × 160 nm, thickness values of 20 nm and 10 nm respectively, and with a separation of 2 nm between layers. Simulate a hysteresis loop for this SyF structure along 1° to the x-axis direction, plotting the average magnetization for the entire bilayer against field (you will need to calculate this from the magnetization saved for the two layers separately – see notes below).

   You should use the *SDesc* with *LLGStatic* solver and set a low *mxh* threshold value, in this case it is recommended to set it to 10^{-7} (use **ode** command). The field step should not be greater than 1 kA/m.

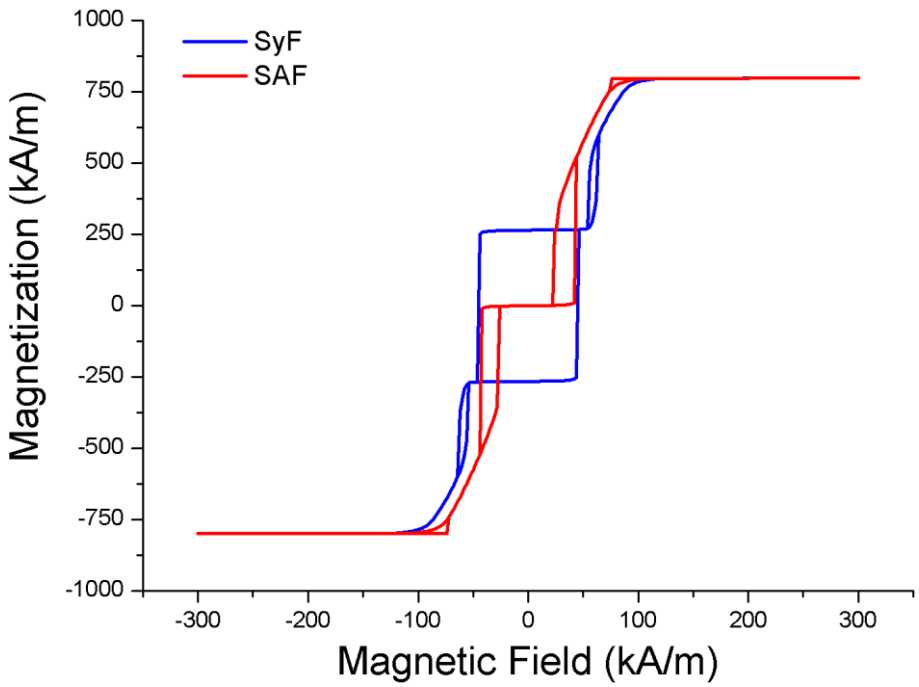
b) Repeat the same exercise but this time set-up a synthetic antiferromagnetic bilayer (SAF) with the same overall thickness as above – i.e. 15nm thick layers with 2 nm spacing.

When adding data to the output list, you can select the mesh for which it applies, if applicable, e.g. magnetization output. You can do this either by adding data when

the required mesh is in focus, or editing that data entry later to change the applicable mesh name. For the exercise above you will need to add to the output the magnetization for both meshes as two separate entries (use **data** command and follow instructions therein).

You will need to apply the field for the hysteresis loop to both meshes, not just the first *permalloy* mesh. Use the **stages** command and add a *Hpolar_seq* stage. You will see the field is set to be applied only to the current mesh in focus, e.g. *Hpolar_seq* <*permalloy*>. Instead, you will need to edit this by double-clicking on the added stage entry, and changing the name from *permalloy* to *supermesh*. After this the entry should read *Hpolar_seq* <*supermesh*>. The field sequence will now be applied to both ferromagnetic meshes.

Figure 12.2 – Hysteresis loops obtained for the SyF and SAF bilayers in Exercise 12.1



When working with multiple ferromagnetic meshes, all the commands that affect changes in a ferromagnetic mesh have an optional parameter which specifies which mesh to use. If only one ferromagnetic mesh is created the mesh name doesn't need to be specified explicitly. If multiple meshes are used, unless the name is specified

the settings are applied either to the current mesh in focus, or to the supermesh, depending on the command.

For example the **setangle** and **setfield** commands will make changes to all ferromagnetic meshes unless a specific name is specified – see the help for these commands (**?setangle**, **?setfield**). On the other hand the **loadmaskfile** command (remember you can just drag a .png file to the mesh viewer instead of typing this command) only applies the shape to the current mesh in focus.

CUDA Computations

If you have a CUDA-enabled graphics card you can enable GPU computations using the **cuda** command:

cuda 1

If CUDA computations are not available for your computer, typing the **cuda** command will show an N/A status. You can also see how much CPU and GPU-addressable memory you have by using the **memory** command.

When using CUDA computations, for optimum efficiency you will want to limit the display update frequency (use the **iterupdate** command). Boris is designed to limit memory transfers between GPU and CPU-addressable memory to an absolute minimum, as this is a critical bottleneck in performance. To display mesh data in the viewer, an average display data is computed on the GPU, then transferred to CPU-addressable memory so it can be used by graphics routines. This transfer can slow down computations, especially if the display viewing coarseness is small (remember you can use the mouse wheel to change viewing coarseness).

If the simulation is very large you should be careful about setting too small a viewing coarseness, as this will make the program slow and less responsive when working with the console.

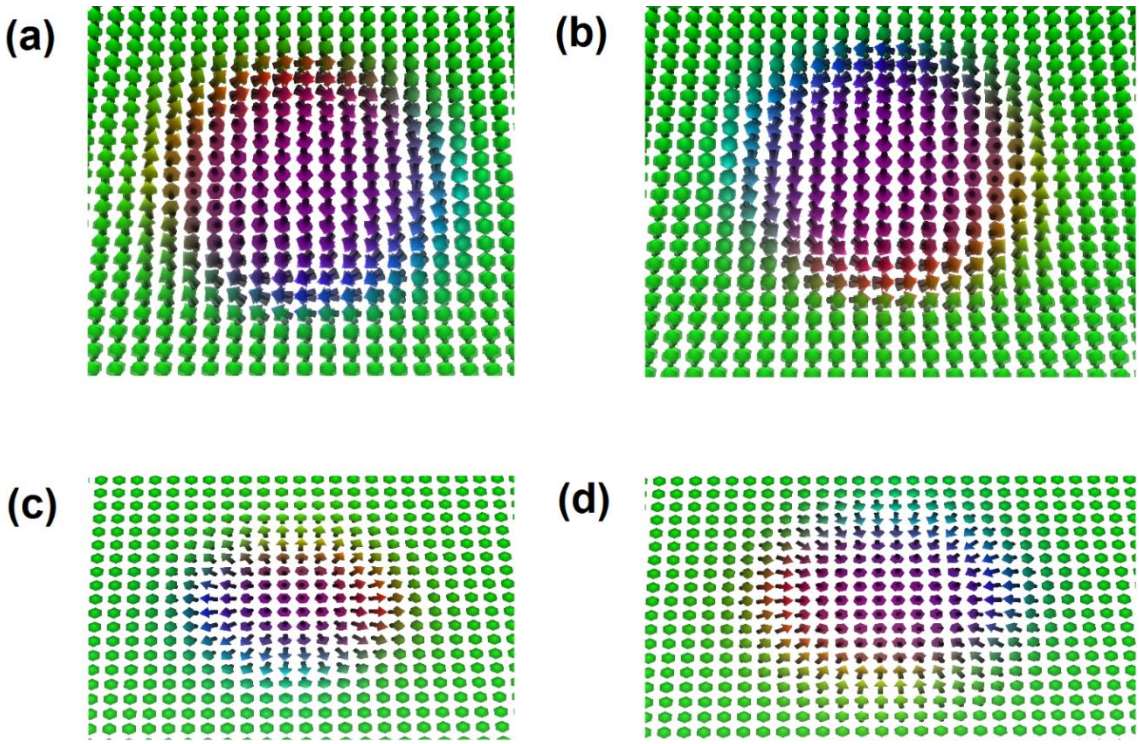
Tutorial 13 – Dzyaloshinskii-Moriya Exchange

Dzyaloshinskii-Moriya interaction (DMI) may be included in the simulation by enabling the *DMExchange*, or the *iDMExchange* module. The former is used for bulk DMI, whilst the latter is used for interfacial DMI. The strength of the DMI interaction is controlled using the *D* material parameter (use **params** command). Néel skyrmions may be generated in the xy plane using the **skyrmion** command:

skyrmion *core chirality diameter position*

In the above command the *core* parameter sets the z direction of the skyrmion core (-1 or 1), the *chirality* parameter sets the radial direction rotation (-1 for away from core, 1 for towards core). The *diameter* and *position* may be specified using metric units, with the *position* requiring 2 components. **Figure 13.1** shows examples of relaxed Néel (*iDMExchange*) and Bloch (*DMExchange*) skyrmions for both D > 0 and D < 0.

Figure 13.1 – a) Bloch skyrmion for D < 0, b) Bloch skyrmion for D > 0, c) Néel skyrmion for D < 0, and d) Néel skyrmion for D > 0.



For the following exercises, when computing a relaxed magnetization state you should use the *SDesc* with *LLGStatic* solver, remembering to set a low *mxh* convergence value (at least 10^{-6}).

Exercise 13.1

a) Obtain relaxed Néel skyrmions for both D > 0 and D < 0 in an ultrathin (1 nm) Co layer with perpendicular magnetization. You will need to use the *iDMExchange* module. Use material parameters as Ms = 600 kA/m, A = 10 pJ/m, |D| = 1.5 mJ/m^2, K1 = 380 kJ/m^3 for uniaxial anisotropy with easy axis along z direction. To reduce the skyrmion diameter apply an out-of-plane magnetic field opposing the skyrmion core, e.g. 15 kA/m along the 0, 0 (polar coordinates) direction.

b) Obtain Bloch skyrmions for both D > 0 and D < 0. You may use the same parameters as above, but this time set a thickness of 10 nm and use the *DMExchange* module. You will need to set a larger out-of-plane magnetic field to control the skyrmion diameter.

c) For part a) compute the skyrmion diameter and topological charge.

To complete Exercise 13.1 c), you will need to extract a profile along the skyrmion diameter, then fit it using an analytical model for a skyrmion. Boris provides this functionality through the **dp_fitskyrmion** command. The z component of **M** is fitted using the following formula (see X.S. Wang et al., Commun. Phys. 1, 31 (2018)):

$$M_Z(x) = M_S \cos\left(2\arctan\left(\frac{\sinh(R/w)}{\sinh(x-x_0)/w)}\right)\right), \quad w = \frac{\pi D}{4K}, \quad K = K_u - \mu_0 M_S^2/2$$

Here M_S (saturation magnetization), R (skyrmion radius), x_0 (skyrmion center position), and w are used as fitting parameters, and in particular the M_s and w values after the fit should match the expected values from the material parameters set.

Thus to obtain the skyrmion radius first use **dp_getprofile** to extract a profile through the center of the skyrmion, then use **dp_fitskyrmion** on the z component of **M**. The fitting works for both skyrmion topological charges, but the fitted value of M_S will change sign depending on the sign of the topological charge.

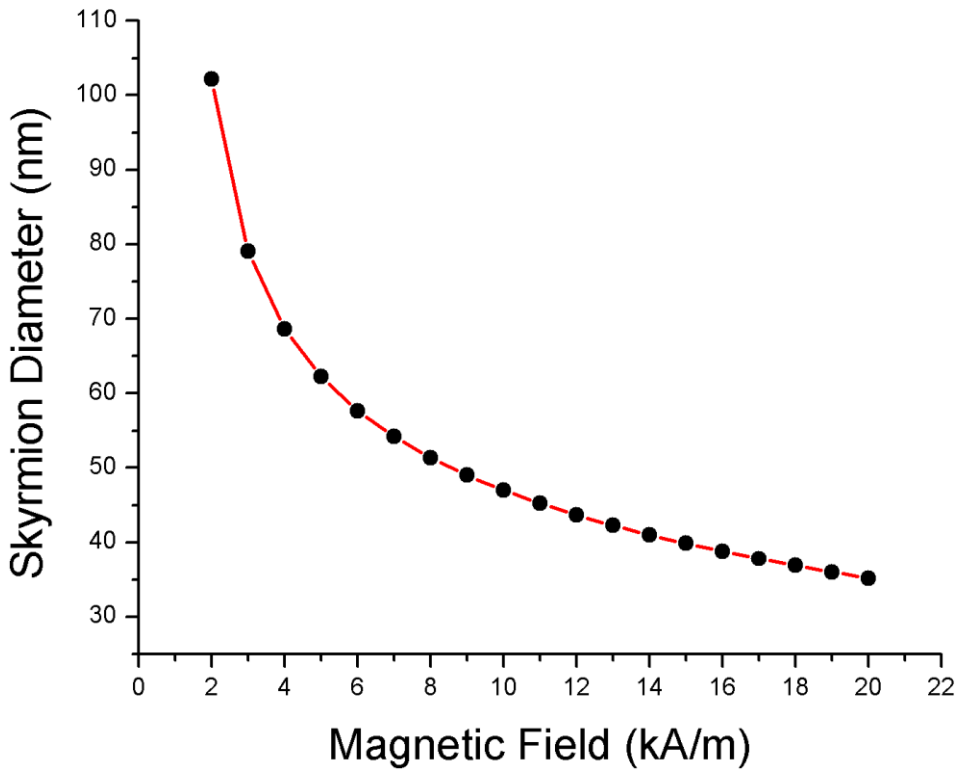
To calculate the topological charge, there is a built-in command, **dp_topocharge**. This command solves the following formula over the xy plane *S*:

$$Q = \frac{1}{4\pi}\int_S \mathbf{m}.\left(\frac{d\mathbf{m}}{dx}\times\frac{d\mathbf{m}}{dy}\right)dxdy$$

Exercise 13.2

For the same parameters as in Exercise 13.1, compute the skyrmion diameter as a function of out-of-plane field strength from 2 kA/m to 20 kA/m in 1 kA/m steps. You should set-up a Python script to automate this simulation.

Figure 13.2 – Skyrmion diameter as a function of out-of-plane field strength obtained in Exercise 13.2.



Tutorial 14 – Simulations with non-zero temperature

Landau-Lifshitz-Bloch equation and Curie temperature

Non-zero temperature simulations may be performed by using the Landau-Lifshitz-Bloch (LLB) equation – use the **ode** command then select the appropriate equation to solve.

With a non-zero temperature the magnetization length (saturation magnetization) is no longer a constant, but depends on the applied field strength. This is modelled via a longitudinal susceptibility included in the LLB equation. Instead, we talk about the *equilibrium magnetization*, which is the stable magnetization length at a given field. Thus at zero temperature the equilibrium magnetization coincides with the saturation magnetization. With a non-zero temperature the equilibrium magnetization gradually decreases, reaching zero at the Curie temperature. Other parameters which change with temperature include the exchange stiffness and magnetization damping. With a non-zero temperature the damping is now divided into two terms: transverse damping (coincides with the Gilbert damping at zero temperature) and longitudinal damping. For further information these articles can be used as a starting point: S. Lepadatu, Journal of Applied Physics 120, 163908 (2016) and S. Lepadatu & M.M. Vopson, Materials 10, 991 (2017).

In the simplest case the temperature inside the mesh is uniform, and is controlled using the **temperature** command, which sets the mesh *base temperature*:

temperature *value (meshname)*

When enabling the LLB equation you will also need to set appropriate temperature dependences for some material parameters, including *Ms*, *damping*, *A,* and *susrel* (the relative longitudinal susceptibility). The longitudinal damping used in the LLB equation is not available as a separate material parameter, but is automatically calculated based on the transverse damping parameter (*damping*). Default temperature dependences for these parameters may be generated based on the Curie temperature of the material – for details see S. Lepadatu, Journal of Applied

Physics 120, 163908 (2016). You can do this using the **curietemperature** command:

curietemperature *value (meshname)*

Setting material parameters temperature dependences

Exercise 14.1

Set a Curie temperature of 870 K (appropriate for Ni80Fe20) and obtain plots of the temperature dependences of the *Ms*, *damping*, *A,* and *susrel* material parameters (see below).

Almost all material parameters available in Boris can be assigned a temperature dependence. This is achieved by specifying a scaling law, *t*. The value of a parameter at a temperature T is then obtained as value_at_T_K = value_at_0_K × *t*(T) – any computational routine in Boris for a which a parameter is used, obtains an updated value in this way where appropriate, where T is either the base temperature (uniform temperature mode) or the local temperature (non-uniform temperature mode). To see the currently set temperature dependences use the **paramstemp** command. You can set a temperature dependence by supplying a text equation, where *T* is the temperature value (see console output for **paramstemp**), or by loading an array using the **setparamtemparray** command. Further details on using text equations are given in a dedicated tutorial. Once a temperature dependence array has been set (e.g. after using the **curietemperature** command), you can load the set temperature dependence into an internal data processing array using the **dp_dumptdep** command:

dp_dumptdep *meshname paramname max_temperature dp_index*

For example to see the set temperature dependence for *Ms* use the following:

dp_dumptdep *permalloy Ms 870 0*

dp_save Ms_scaling 0

In the file Ms_scaling.txt (saved under the current working directory – see **data** command) you will see a single column with the scaling coefficients. These are saved in increments of 1 K, from 0 K up to 870 K. Internally the scaling coefficients are obtained from the user loaded array at 1 K increments, irrespective of how the user specified the temperature dependence – missing temperature points are filled in using interpolation. During computations the scaling coefficients are obtained by interpolating the nearest 2 temperature scaling points. To reset all parameters temperature dependences you can use the **clearparamstemp** command.

Field dependence of material parameters temperature dependences

With non-zero temperature simulations, the equilibrium magnetization also depends on the strength of the applied magnetic field. This dependence is enabled by setting the strength of the net atomic moment of the material, specified in Boris as multiples of the Bohr magneton. This is done using the command:

atomicmoment *(value)*

If this value is not zero, whenever the applied magnetic field changes, the temperature dependences of all the parameters affected by the Curie temperature setting (see above) are recalculated. Moreover, the longitudinal susceptibility is directly proportional to this value so must be set correctly whenever the LLB equation is used.

Non-zero temperature simulations

Exercise 14.2

Simulate the hysteresis loops in a 160 × 160 × 5 nm permalloy circle at zero temperature (using the LLG equation), as well as at room temperature (297 K, using the LLB equation) and compare the two loops. With the LLB equation the time step for numerical stability is usually lower – you might need to investigate this.

Tutorial 15 – Thermal Fields

When non-zero temperature modelling is considered, an additional effect that can be included is lattice thermal agitation. This gives rise to fluctuations in magnetic moments, and may be modelled by introducing appropriate stochastic fields and torques. In Boris thermal fields may be enabled by selecting a stochastic magnetization dynamics equation, e.g. sLLB – use the **ode** command then select the appropriate equation to solve. For further information see S. Lepadatu & M.M. Vopson, Materials 10, 991 (2017).

When solving stochastic equations, the choice of available ODE evaluation methods is more limited since they must be able to handle the stochasticity introduced. Currently the best fixed time-step method available in Boris is the trapezoidal Euler (*TEuler*) evaluation method, also known as Heun's method. Since this method is a fixed time step method you will need to investigate the time step required for numerical stability. You can use the default time step as a starting point.

There is an adaptive time-step version of this called *AHeun* which you can also use.

Exercise 15.1

Simulate out-of-plane hysteresis loops at room temperature in a 256 nm × 256 nm Co rectangle with perpendicular magnetization, with 4 nm thickness, and cubic 4 nm cellsize. Use material parameters as Ms = 600 kA/m, A = 10 pJ/m, and K1 = 380 kJ/m^3 for uniaxial anisotropy with easy axis along z direction. You should simulate hysteresis loops with and without thermal fields for comparison.

Tutorial 16 – Heat Flow Solver and Joule Heating

Heat equation

Non-uniform temperature simulations may be enabled by selecting the *heat* module. Any mesh with this module enabled will solve the heat equation as a function of time. If any two meshes with the *heat* module enabled are in contact, then heat flow across the interface (also referred to as a composite media boundary) is automatically calculated based on the continuity of heat flux and temperature perpendicular to the composite media boundary.

There is a special type of mesh, referred to as an *insulator* mesh in Boris, which can be used to model substrates. You can add an insulator mesh using:

addinsulator *name rectangle*

When the *heat* module is enabled, the *thermal cell* discretisation cellsize becomes available in the mesh descriptions (use the **mesh** command). This can be controlled independently of the magnetic and electric cellsize (if enabled), and can also be set independently of other cellsize values in other meshes.

The heat equation time step may be set using:

setheatdt *value*

This value shouldn't be larger than the magnetization dynamics equation time step (**setdt**), since during computations the heat equation time is incremented only up to the current magnetization equation time (the global time, or total time – see the *time* output data). If this value is lower, the heat equation will be iterated multiple times until it catches up to the magnetization equation time.

The mesh temperature may be set as before using the **temperature** command. This sets a uniform mesh temperature as a starting point, but depending on the simulation

configuration the mesh temperature can change. This is true particularly if the mesh ambient temperature is different. For the heat equation, boundary conditions for cells not at a composite media boundary are set based on Newton's law of cooling – i.e. Robin boundary conditions are used. These require an ambient temperature (the surrounding temperature) and a heat transfer coefficient (the Robin coefficient). To adjust these values you may use the **ambient** command, then double click on the respective interactive objects to modify their values. Note, when the **temperature** command is used, setting a mesh temperature automatically sets the ambient temperature to the same value too. You may also choose to have thermally insulating boundary conditions by selecting the appropriate options displayed by the **ambient** command.

Parameters for heat transport

A few parameters are used to specify the thermal properties of the material, in particular the *thermK* (thermal conductivity) and the *shc* (specific heat capacity) material parameters – see these by using the **params** command. Additionally the *density* (mass density) parameter also enters the heat equation.

Note, all material parameters with a temperature dependence enabled will now also vary non-uniformly throughout the mesh (if *heat* module enabled), taking on the value set by the local cell temperature value.

You may obtain the mesh average temperature through the <*Temp*> output data – use **data** command. The mesh temperature may also be displayed (*Temp*) – use the **display** command.

Joule heating

If the *transport* module is also enabled in the same mesh as the *heat* module, Joule heating is taken into account. This results in a heat source term in the heat equation due to the charge current density, **J** as:

$$C\rho \frac{\partial T(\mathbf{r},t)}{\partial t} = \nabla.K\nabla T(\mathbf{r},t) + \frac{\mathbf{J}^2}{\sigma}$$

In the next exercise you will investigate the effect of a voltage pulse on a $Ni_{80}Fe_{20}$ nanowire placed on a SiO_2 substrate, similar to the work in S. Lepadatu, Journal of Applied Physics 120, 163908 (2016). To model a very long wire on a substrate (say the wire is oriented along the x-axis) you should set the x-axis ends of both the magnetic wire and substrate as insulating since in this case the heat flux is oriented only along the y and z directions. Similar considerations apply to the substrate if you want it to be effectively infinite in the x-y plane and depth – set insulating boundary conditions in the required directions. Note in this latter case the modelled substrate must still be large enough for the temperature evolution to be correct for the required duration – for details see S. Lepadatu, Journal of Applied Physics 120, 163908 (2016). The x-axis ends of the magnetic wire should have electrodes so a uniform current density is achieved – remember you can use the **setdefaultelectrodes** command. You can leave the Robin heat transfer coefficient (see **ambient** command output) to the default value as this is appropriate for ventilated air surrounding.

The SiO_2 substrate may be added by using the **addinsulator** command and enabling its *heat* module. You will also need to enter appropriate values for thermal conductivity, specific heat capacity and density, and similarly for the $Ni_{80}Fe_{20}$ magnetic wire.

Note, typically the *thermal cellsize* can be greater than the magnetic (or electric) cellsize, and again can be set independently in different meshes (composite media boundary conditions do **not** require the discretizations to match on the two contacting meshes, for any computational routines used in Boris; generic composite media boundary computational routines are used which are second order accurate in space for all meshes). For the purposes of the next exercise you can use a simple cubic cellsize with a 10 nm side for all the thermal discretization lengths (note, if the mesh thickness is 10 nm then the z cellsize will be adjusted so there are at least 2 computational cells along the z direction – 3D solver used).

Exercise 16.1

Create a $Ni_{80}Fe_{20}$ nanowire with 160 nm width, 10 nm thickness and 640 nm length, centered on a SiO_2 substrate with 800 nm width, 640 nm length and 150 nm depth. Enable heat equation computations in both meshes and transport module in the $Ni_{80}Fe_{20}$ nanowire. By setting appropriate insulating boundary conditions for heat conduction, define the nanowire to be effectively infinite along the x axis, and the substrate elongated in the x-y plane and depth. Set the ambient temperature (as well as the base temperature – the starting temperature) to be the room temperature value (297 K). The default thermal conductivity, specific heat capacity and mass density values are appropriate for $Ni_{80}Fe_{20}$. For SiO_2 you should edit these as K = 1.4 W/mK (*thermK*), C = 730 J/kgK (*shc*), and ρ = 2200 kg/m^3.

a) Set a voltage step with 50 ns duration which results in a current density of 10^{12} A/m^2 at T = 297 K. Use a temperature dependence for the electrical conductivity σ such that:

$$\sigma = \frac{\sigma_0}{1 + 0.025T}$$

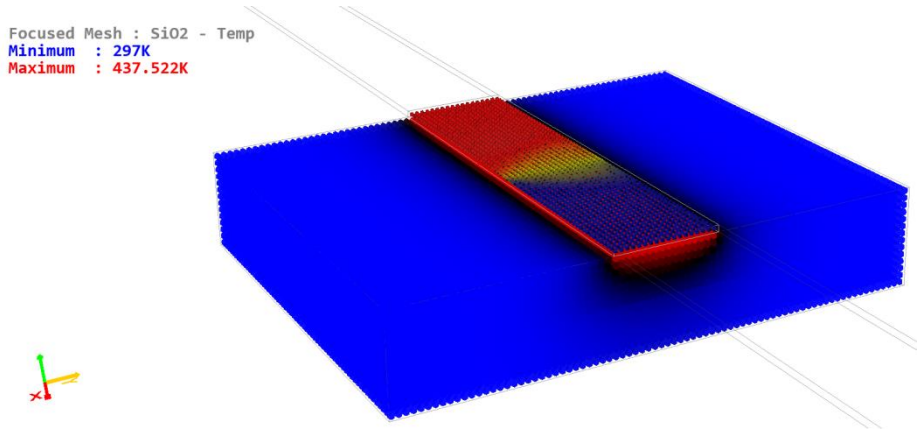
The above formula represents the default temperature dependence set for electrical conductivity - see **paramstemp** command. Obtain the average temperature and current density as a function of time in the $Ni_{80}Fe_{20}$ mesh both for the heating cycle (first 50 ns), as well as the next 50 ns of the cooling cycle when the voltage is set to zero. (Note, just for this part, to speed up the computations you may want to disable any magnetic computations in the $Ni_{80}Fe_{20}$ nanowire by disabling the *demag*, *exchange*, and *zeeman* modules).

b) Set a transverse domain wall in the center of the nanowire through the **preparemovingmesh** command and relax it. Redo the simulation in part a), but this time also obtain the domain wall displacement as a function of time when using the LLG-STT equation (use the **ode** command).

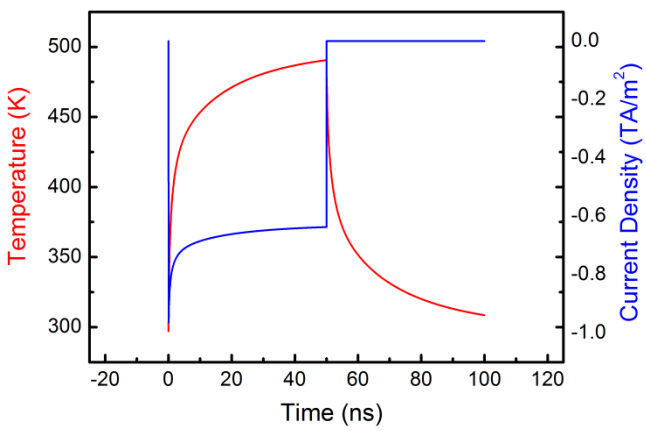
c) Repeat part b) but this time use the LLB-STT equation with a Curie temperature of 870 K. Note, you will need to reduce the time step significantly for the LLB equation for numerical stability.

Figure 16.1 – a) Geometry used for Exercise 16.1, showing a magnetic wire on a SiO_2 substrate with Joule heating computations enabled – heat is generated in the magnetic wire due to an applied charge current density. b) Average temperature in the permalloy nanowire, also showing the current density during and after the applied voltage pulse. c) Domain wall displacement simulated using the LLG-STT and LLB-STT equations. For the latter a Curie temperature of 870 K was set.

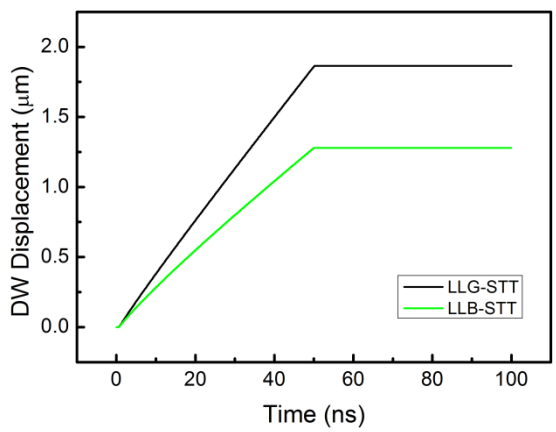
a)



b)



c)



Tutorial 17 – Spin Transport Solver

In addition to the simple Ohm's law used to obtain the charge current density with the *transport* module, Boris also integrates a 3D spin current solver based on the spin drift-diffusion equations – see S. Lepadatu, Scientific Reports 7, 12937 (2017). This solver allows for a number of effects to be computed self-consistently in arbitrary multi-layered geometries and integrated with the magnetization dynamics solver. These include the spin Hall effect (SHE), inverse SHE, CPP-GMR, spin diffusion and non-local spin transport effects, spin pumping, as well as bulk and interfacial spin torques calculated from the spin accumulation and composite media boundary conditions.

The spin transport solver computes both the charge and spin polarisation current densities, $\mathbf{J}_C$ and $\mathbf{J}_s$ respectively, together with the charge potential, V, and spin accumulation, **S**:

$$\mathbf{J}_C = \sigma\mathbf{E} + \beta_D D_e \frac{e}{\mu_B}(\nabla\mathbf{S})\mathbf{m} + \theta_{SHA} D_e \frac{e}{\mu_B}\nabla\times\mathbf{S} + P\sigma\frac{\hbar}{2e}\mathbf{E}^\sigma - P\frac{\sigma^2\hbar}{e^2 n}\mathbf{E}\times\mathbf{B}^\sigma$$

$$\mathbf{J}_S = -\frac{\mu_B}{e}P\sigma\mathbf{E}\otimes\mathbf{m} - D_e\nabla\mathbf{S} + \theta_{SHA}\frac{\mu_B}{e}\boldsymbol{\varepsilon}\sigma\mathbf{E} + \frac{\hbar\mu_B\sigma}{2e^2}\sum_i \mathbf{e}_i\otimes(\dot{\mathbf{m}}\times\partial_i\mathbf{m}) + \frac{\hbar\mu_B\sigma^2}{e^3 n}(\mathbf{z}\times\mathbf{E})\otimes(\partial_x\mathbf{m}\times\partial_y\mathbf{m})$$

where:

$$E_i^\sigma = (\dot{\mathbf{m}}\times\partial_i\mathbf{m}).\mathbf{m}$$

$$\mathbf{B}^\sigma = \mathbf{z}(\partial_x\mathbf{m}\times\partial_y\mathbf{m}).\mathbf{m}$$

Here $\mathbf{E}^\sigma$ and $\mathbf{B}^\sigma$ are the directions of the emergent electric field due to charge pumping, and emergent magnetic field due to topological Hall effect respectively. $\mathbf{J}_s$ is a rank-2 tensor such that $\mathbf{J}_{Sij}$ signifies the flow of the *j* component of spin polarisation in the direction *i*. The electric field is given by **E** = -∇V and **S** satisfies the equation of motion:

$$\frac{\partial \mathbf{S}}{\partial t} = -\nabla.\mathbf{J}_S - D_e\left(\frac{\mathbf{S}}{\lambda_{sf}^2} + \frac{\mathbf{S}\times\mathbf{m}}{\lambda_J^2} + \frac{\mathbf{m}\times(\mathbf{S}\times\mathbf{m})}{\lambda_\varphi^2}\right)$$

In the above equations we have a number of material constants which can be controlled via the **params** command:

- D_e is the electron diffusion constant
- β_D is the diffusion spin polarisation - this term leads to CPP-GMR
- P is the charge current spin polarisation – this terms leads to Zhang-Li spin transfer torques, among other effects
- θ_{SHA} is the spin Hall angle (unitless) – the term in the equation for $\mathbf{J}_s$ leads to SHE, whilst the term in the equation for $\mathbf{J}_C$ leads to the inverse SHE; Note there are two related parameters available in Boris: *SHA* and *iSHA*. These represent the spin Hall angle, but may be set to different values, allowing the SHE or inverse SHE to be turned on or off in the computations by setting one or the other to zero.
- λ_{sf} is the spin flip length
- λ_J and λ_φ are the exchange rotation and spin dephasing lengths respectively, describing the absorption of transverse spin components (transverse to **m**, the magnetization direction) within a ferromagnetic material
- n is the carrier density (m^{-3})
- Charge pumping and topological Hall effect may be turned on or off by setting the *pump_eff* and *the_eff* values to 1 or 0 (disabled by default).

Bulk spin torques are included in the computations as:

$$\mathbf{T}_\mathbf{S} = -\frac{D_e}{\lambda_J^2}\mathbf{m}\times\mathbf{S} - \frac{D_e}{\lambda_\varphi^2}\mathbf{m}\times(\mathbf{m}\times\mathbf{S})$$

This term is included in the implicit LLG (or LLB) equation as (in practice this term results in an effective field which is added to $\mathbf{H}_{eff}$):

$$\frac{\partial \mathbf{m}}{\partial t} = -\gamma\mathbf{m}\times\mathbf{H}_{eff} + \alpha\mathbf{m}\times\frac{\partial \mathbf{m}}{\partial t} + \frac{1}{M_S}\mathbf{T}_\mathbf{S}$$

There is a parameter in Boris, called *ts_eff* (see **params**): This is a unitless constant which multiplies $\mathbf{T}_S$ and is termed the spin torque efficiency, allowing bulk spin torques to be turned off (*ts_eff* = 0) or fully on (*ts_eff* = 1).

There are two possibilities for treating composite media boundaries. The simplest approach is to assume continuity of a flux and potential – for the spin transport solver these are $\mathbf{J}_C$ and V for charge transport and $\mathbf{J}_S$ and **S** for spin transport. The continuity conditions are used when modelling interfaces between two normal metals (N) or two ferromagnets (F); they may also be used to model interfaces between a normal metal and ferromagnet (N/F) but in this case typically the second approach is more appropriate, based on interface spin conductances:

$$\mathbf{J}_C.\mathbf{n}\big|_N = \mathbf{J}_C.\mathbf{n}\big|_F = -\left(G^{\uparrow} + G^{\downarrow}\right)\Delta V + \left(G^{\uparrow} - G^{\downarrow}\right)\Delta\mathbf{V}_S.\mathbf{m}$$

$$\mathbf{J}_S.\mathbf{n}\big|_N - \mathbf{J}_S.\mathbf{n}\big|_F = \frac{2\mu_B}{e}\left[\mathrm{Re}\left\{G^{\uparrow\downarrow}\right\}\mathbf{m}\times\left(\mathbf{m}\times\Delta\mathbf{V}_S\right) + \mathrm{Im}\left\{G^{\uparrow\downarrow}\right\}\mathbf{m}\times\Delta\mathbf{V}_S\right]$$

$$\mathbf{J}_S.\mathbf{n}\big|_F = \frac{\mu_B}{e}\left[-\left(G^{\uparrow} + G^{\downarrow}\right)\left(\Delta\mathbf{V}_S.\mathbf{m}\right)\mathbf{m} + \left(G^{\uparrow} - G^{\downarrow}\right)\Delta V\mathbf{m}\right]$$

In the above equations $G^{\uparrow}$, $G^{\downarrow}$ are interface conductances for the majority and minority spin carriers respectively, and $G^{\uparrow\downarrow}$ is the complex spin mixing conductance. Also ΔV is the potential drop across the N/F interface ($\Delta V = V_F - V_N$) and $\Delta\mathbf{V}_S$ is the spin chemical potential drop, where $\mathbf{V}_S = \left(D_e/\sigma\right)\left(e/\mu_B\right)\mathbf{S}$. These interface conditions describe the absorption of transverse spin components at the interface, giving rise to interfacial spin torques:

$$\mathbf{T}_S^{interface} = \frac{g\mu_B}{ed_h}\left[\mathrm{Re}\left\{G^{\uparrow\downarrow}\right\}\mathbf{m}\times\left(\mathbf{m}\times\Delta\mathbf{V}_S\right) + \mathrm{Im}\left\{G^{\uparrow\downarrow}\right\}\mathbf{m}\times\Delta\mathbf{V}_S\right]$$

There is also an associated interfacial spin torque efficiency constant – *tsi_eff*. The above term is also included in the magnetization dynamics equation, much in the same way as $\mathbf{T}_S$ is. The main difference is this torque is only included in the computational cells at the interface, where d_h is the cellsize normal to the interface – this allows correct computation of interfacial spin torques for a ferromagnetic layer of

given thickness *t*, independent of its computational discretization, since the effect on magnetization of the interfacial spin torque is averaged over its thickness.

Spin pumping is generated at an N/F interface as:

$$\mathbf{J}_S^{pump} = \frac{\mu_B}{2\pi}\left[\mathrm{Re}\left\{g^{\uparrow\downarrow}\right\}\mathbf{m}\times\frac{\partial\mathbf{m}}{\partial t} + \mathrm{Im}\left\{g^{\uparrow\downarrow}\right\}\frac{\partial\mathbf{m}}{\partial t}\right]$$

Here $g^{\uparrow\downarrow} = (h/e^2)G^{\uparrow\downarrow}$ and the pumped spin current is used in the calculation of composite media boundary conditions by including it on the normal metal side of the equations. As with the spin torques, there's an associated spin pumping efficiency parameter – *pump_eff* – which allows spin pumping to be turned on or off in the computations.

When modelling N/F interfaces you may need to have different interface conductances on different sides of a ferromagnetic layer (e.g. a Pt/Co/Ta multilayer, where the Pt/Co and Co/Ta interfaces may need different spin mixing conductances). For this reason, the interface conductances ($G^{\uparrow}$, $G^{\downarrow}$, and $G^{\uparrow\downarrow}$) are not associated just with a ferromagnetic mesh, but also appear in the list of parameters for normal metal meshes (*conductor* meshes - **addconductor**). When an N/F interface is defined by the contact of two meshes, the interface conductances stored in the *upper* mesh are used – e.g. if the meshes are arranged in a multilayer structure along the z direction, the upper mesh is that with a higher z coordinate. To turn off the interface conductance approach to modelling composite media boundaries you need to set the $G^{\uparrow\downarrow}$ values to zero for the appropriate mesh. In this case the computations revert to using the continuity approach described above.

To enable the spin transport solver you need to have the *transport* module active in the mesh you want spin transport computations and you must also select a magnetization dynamics equation with spin accumulation (e.g. LLG-SA, LLB-SA, etc.) – see the **ode** command. Using the **display** command you can select to display a number of associated quantities in the mesh viewer, including **S**, bulk and interfacial spin torques, x, y, and z directions for the spin current.

With the spin transport solver enabled you will need to pay attention to the convergence constant for the spin accumulation solver. Similarly to the charge potential solver, which solves a Poisson equation to obtain V within the set convergence constant, the spin accumulation **S** is obtained by solving a vector Poisson equation – this equation is obtained from the equation of motion for **S** in the "steady state", i.e. when $\partial\mathbf{S}/\partial t = 0$. The response time-scales of **m** and **S** are separated typically by 3 orders of magnitude (ps vs fs time-scales respectively) thus we only require to obtain the "steady state" values for **S** for a given magnetization configuration. The vector Poisson equation also uses a convergence constant and a timeout for the maximum number of allowed sequential iterations, and these values may be changed by using the **tsolverconfig** command.

Further info:

Whilst each iteration taken for the Poisson equations for V and **S** is relatively cheap, typical problems may require a large number of iterations to reach convergence, which significantly slows down computations. This is especially true in the initialization stage when the timeout number of iterations may be reached for the first few iterations; after this, small steps in **m** should result in relatively few steps in the solution of **S** (and where appropriate V). A recommended general approach is to solve for the steady state V and **S** values with all spin torques turned off and for a relaxed starting magnetization configuration. After this, save the simulation (which also saves the computed V and **S**), and re-enable the spin torques as required. From this point initialization should be quicker, with any further iterations in V and **S** triggered by changes in **m** (changing set electrode potential values can also trigger the Poisson solvers).

Setting the convergence factors too low may result in very slow simulations as the solvers will require a large number of iterations. You will need to determine the best compromise between computational speed and accuracy. The default normalised convergence values of 10^{-6} for V and 10^{-5} for **S** Poisson equations are set on the side of accuracy, having been found to give accurate results in all test cases, but you should still verify this for your particular simulation.

Tutorial 18 – Spin Hall Effect

Exercise 18.1

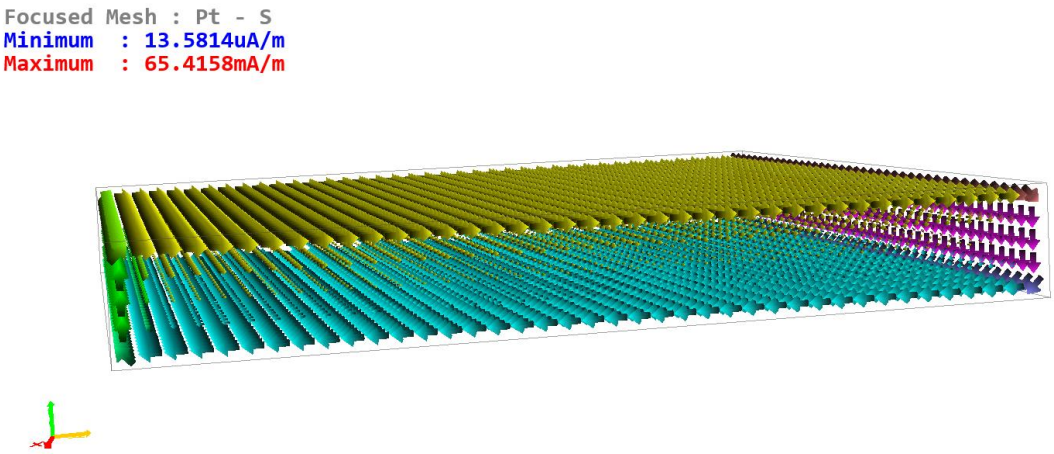
Consider a single Pt mesh with dimensions 320 nm x 320 nm and 40 nm thickness. Compute the spin accumulation and z-direction spin current density in response to a set potential of 10 mV with electrodes placed at the x-axis ends. Verify that **S** obeys the right-hand-rule with respect to the charge current direction.

For Pt you may use $\sigma = 7\times10^6$ S/m, $\lambda_{sf} = 1.4$ nm and $\theta_{SHA} = 0.1$. You may use a cubic cellsize with 5 nm side.

Plot the y components of the z-direction spin current density and spin accumulation along the z-axis, through the center of the Pt slab (remember the **dp_getprofile** command). Verify that the following relation holds, using the plotted value of the spin current at the center of the Pt slab:

$$\theta_{SHA} = -\frac{J_{sz,y}}{J_{cx}} / \frac{\mu_B}{e}$$

Figure 18.1 – Computed spin accumulation for Exercise 18.1, where the charge current density is along the negative x direction.



Exercise 18.2

a) Continuing from Exercise 18.1, now add a $Ni_{80}Fe_{20}$ layer with 20 nm thickness on top of the Pt layer. Make sure to reset to default electrodes (**setdefaultelectrodes**) so a uniform charge current density is obtained in each layer. Plot the y components of the z-direction spin current density along the z axis for both the continuous and spin-mixing conductance interface models for a) magnetization direction along the injected spin current, i.e. along the y axis, and b) magnetization direction transverse to the injected spin current, i.e. along the x axis. Explain the differences between these cases.

Note, for this exercise you will have to use a smaller cellsize along the z direction. This is due to the large gradients involved, and is normally the case when N / F multilayers are used. You should use a cellsize of (5 nm, 5 nm, 1 nm).

Remember you can use the **computefields** command for this exercise, instead of using **run**, since you don't want to relax the magnetization configuration. It helps to monitor the transport solver number of iterations and convergence error (in the data box display the following using the **data** command and righ-clicking on the respective interactive objects: *v_iter, s_iter, ts_err*).

b) Does the relation in Exercise 18.1 hold at the N/F interface, and why not? Investigate this again with a spin flip length in Pt of 8 nm, checking the relation both at the center of the Pt layer and at the interface.

Figure 18.2 – Spin current density in the z direction for a Pt/$Ni_{80}Fe_{20}$ bilayer, where the magnetization in the permalloy mesh is along the y axis (longitudinal).

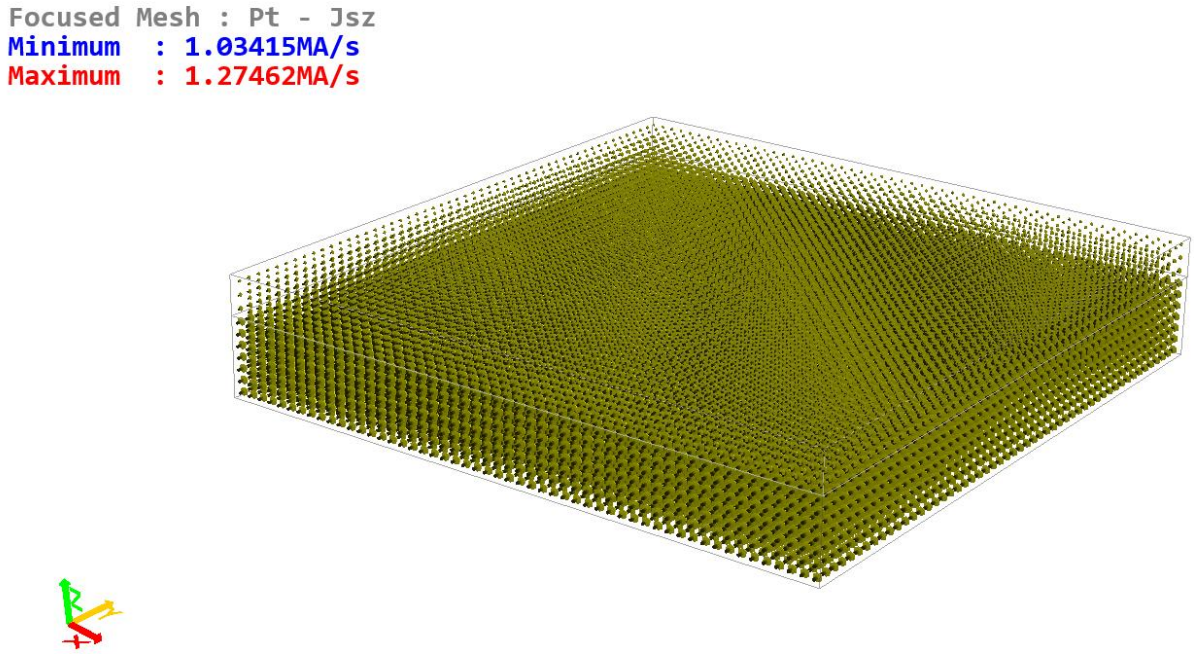
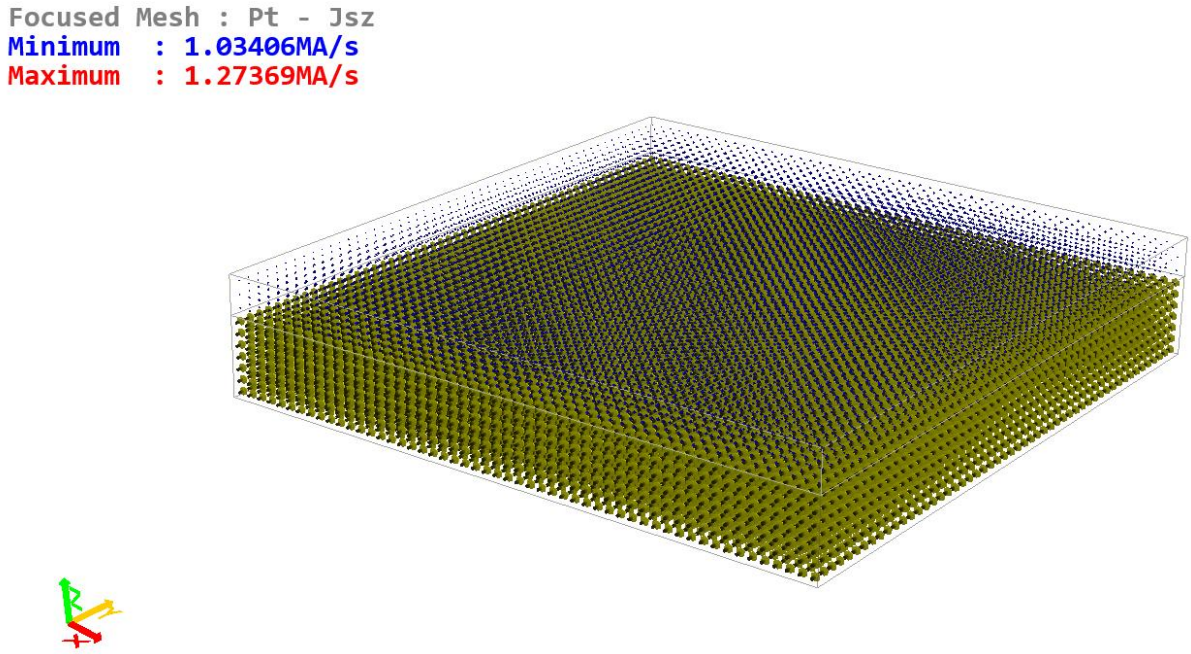


Figure 18.3 – Spin current density in the z direction for a Pt/$Ni_{80}Fe_{20}$ bilayer, where the magnetization in the permalloy mesh is along the x axis (transverse).



Tutorial 19 – Spin Pumping and Inverse Spin Hall Effect

In this tutorial you will set-up a ferromagnetic resonance in a magnetic dot, then investigate the generated spin Hall voltage in a Pt underlayer. Due to the motion of magnetic moments in the ferromagnetic layer a spin current is pumped in the Pt underlayer, where an electrical current is generated due to the inverse SHE. This leads to charge accumulation at opposing sides of the Pt underlayer, and thus an electrical potential is generated.

Exercise 19.1

Setup a ferromagnetic resonance (FMR) at 20 GHz excitation frequency in a $Ni_{80}Fe_{20}$ circle with 80 nm diameter and 10 nm thickness, with a bias field applied in the plane of the circle along the y axis.

First, find the demagnetizing factor in the plane of the circle and set the *demag_N* module with demagnetizing factors $N_x = N_y = N$. Remember you can calculate the demagnetizing energy for uniform magnetization (*e_demag*), and the demagnetizing factor is then related to it by:

$$\varepsilon_{demag} = \frac{\mu_0}{2} N M_S^2$$

For this exercise, since you are effectively using the Stoner-Wohlfarth model you can turn off the *exchange* module. Calculate the FMR bias field required for resonance at an r.f. frequency of 20 GHz. You can use Kittel's formula applicable for elliptical shapes for a bias field H_0 along the y direction:

$$f = \frac{\mu_0 |\gamma_e|}{2\pi} \sqrt{(H_0 + (N_x - N_y)M_s)(H_0 + (N_z - N_y)M_s)}$$

Using the *Hfmr* stage type, apply the excitation r.f. field (together with the calculated orthogonal bias field) in the plane of the circle for a number of cycles, and record the average magnetization. An r.f. field amplitude of 100 A/m is normally sufficient. If the r.f. field is applied for a sufficient number of cycles, the magnetization will achieve a

steady state precession at resonance. Determine the number of cycles required by examining the output average magnetization data, then **reset** and save the simulation – the next time you load the simulation the FMR precession will start directly in the steady state.

The *Hfmr* stage consists of the following parameters: H_{0x}, H_{0y}, H_{0z}; H_{rfx}, H_{rfy}, H_{rfz}; *r.f. steps; r.f. cycles*.

The bias field (H_0) and r.f. field amplitude (H_{rf}) are specified using Cartesian coordinates. The *r.f. steps* is the number of discretisation steps in each r.f. cycle, and the *r.f. cycles* is the number of sinusoidal oscillations the r.f. field will be applied for. To set the required 20 GHz frequency you will need to set the correct combination of *r.f. steps* and time stopping condition for each step. For example, since at 20 GHz each period takes 50 ps, if you use 20 r.f. steps per cycle, the time stopping condition for each step should be 2.5 ps (the default time stopping condition of 50 ps results in a 1 GHz frequency with 20 r.f. steps per cycle, so you will need to edit this).

Exercise 19.2

Using the prepared simulation from Exercise 19.1, add a Pt underlayer with dimensions 160 nm × 160 nm with the magnetic dot centered, and 20 nm depth.
Enable spin pumping (*pump_eff* = 1 in the permalloy mesh), and inverse SHE (*SHA* = *iSHA* = 0.1) in the Pt mesh. Do not set any electrodes but make sure the transport module is enabled in both the permalloy and Pt meshes, and the LLG-SA equation is selected so the spin transport solver is enabled. You will need to refine the electric cellsize in both meshes along the z direction to 1 nm. You should also relax the transport solver convergence criteria to 10^{-4} for both the charge and spin solvers.

Obtain the induced spin Hall voltage at the opposing y-axis sides of the Pt mesh and plot them as a function of time for a few FMR precessions. Note, in the output data (**data**) you will have to add <*V*> (the average calculated voltage) for the Pt mesh two times, editing the respective rectangles to correspond to the required two sides of the Pt mesh.

Figure 19.1 – Inverse spin-Hall effect voltage in a Pt underlayer generated through spin pumping from a ferromagnetic dot at ferromagnetic resonance.

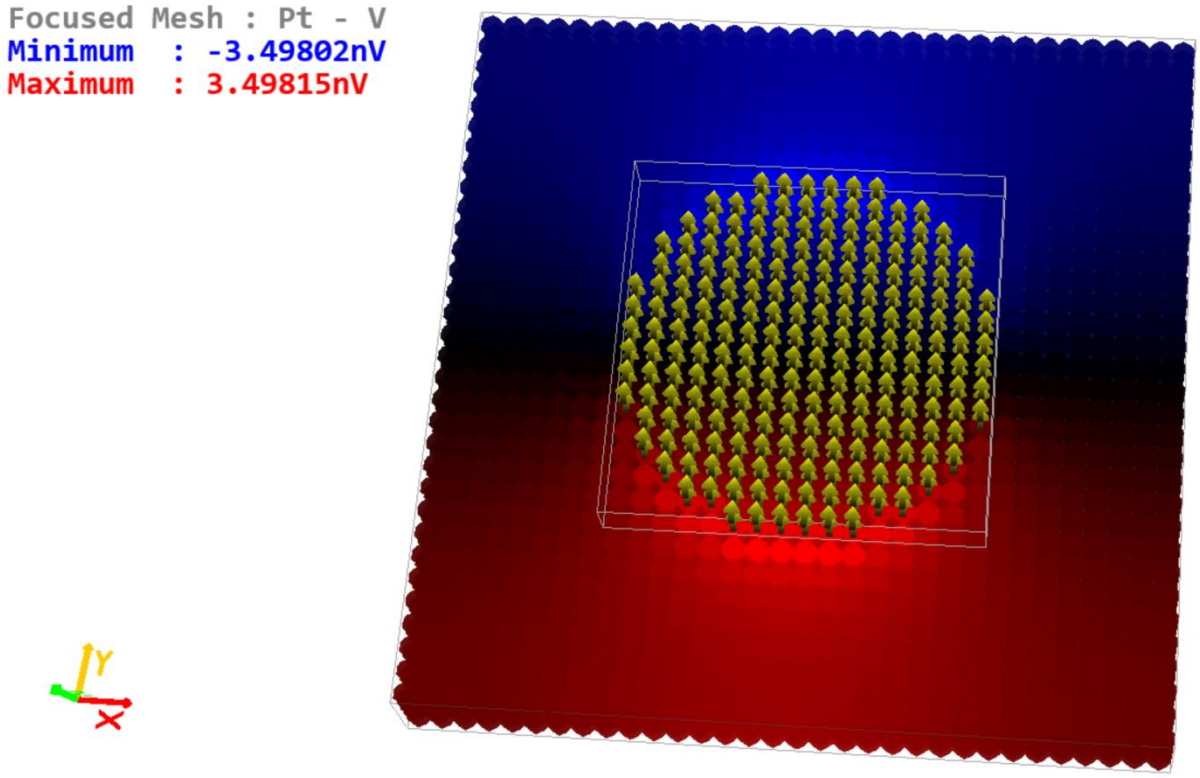
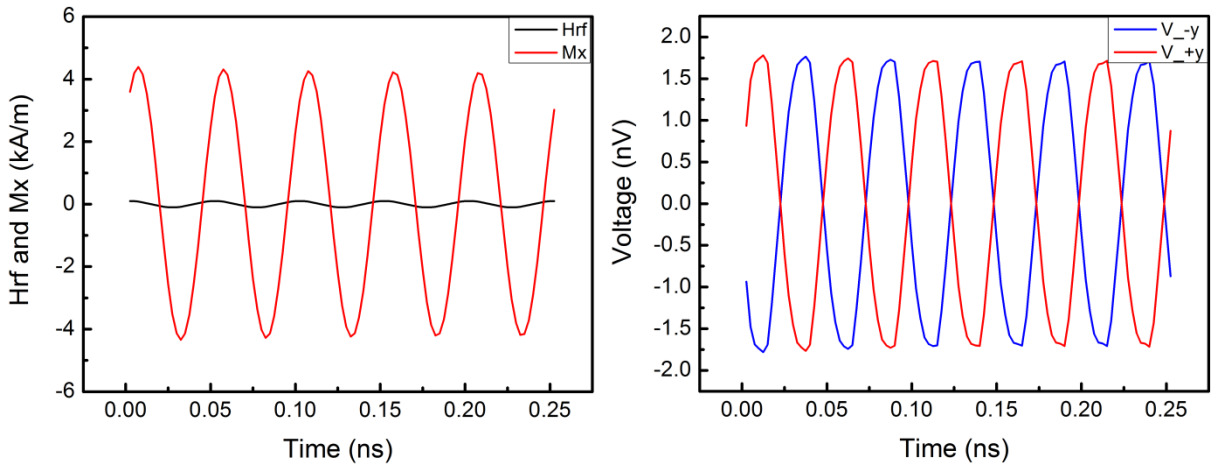


Figure 19.2 – a) Magnetization precession at ferromagnetic resonance with a 20 GHz r.f. field, b) inverse spin-Hall effect voltage at resonance on opposing sides of the Pt mesh – see Figure 19.1.

a) b)



Tutorial 20 – Ferromagnetic Resonance

In this tutorial you will learn how to simulate ferromagnetic resonance (FMR), and reproduce input material parameters. Following this, in the next tutorial you will investigate how the spin torques due to spin pumping and the SHE affect the effective damping observed. An introduction to FMR simulations was given in the preceding Tutorial, and you must complete it before proceeding. Here we will investigate both field-swept FMR (fixed excitation frequency) and frequency-swept FMR (fixed bias field).

<u>Field-Swept FMR</u>

In Boris, FMR simulations are best done using a Python script. As you will note from the previous tutorial, applying an r.f. field excitation requires a number of cycles for the magnetization precession to reach steady state. For a field-swept FMR simulation, after changing the bias field you must ensure the magnetization precession is stable before obtaining output data. The simulation procedure is as follows:

1) Set bias field value and run the simulation for a fixed number of r.f. cycles (the "chunk" – e.g. 20 or more), but do not save any output data.
2) After the chunk has completed, run the simulation for a single r.f. cycle and save the output data (<*M*>).
3) From the saved data obtain the magnetization oscillation amplitude along the r.f. field direction.
4) Compare the oscillation amplitude against the previous oscillation amplitude (which is zero if this is the first chunk). If the change exceeds a set threshold (e.g. 0.1%) then repeat from step 1), otherwise proceed.
5) Record the oscillation amplitude and bias field. Increase bias field value and start again from step 1) until the field sweep range is completed.

A general-purpose FMR simulation Python script has been prepared and saved in the examples folder for this Tutorial.

Exercise 20.1

Using a $Ni_{80}Fe_{20}$ square of 80 nm side and 10 nm thickness simulate an FMR peak around the resonance bias field and plot the resulting magnetization oscillation amplitude against bias field data. Set the bias field along the –y direction, i.e. at 270° azimuthal angle. Use a Python script to simulate this as described above. (*You may use $N_x = N_y = 0.12$, with the predicted resonance field of $H_0 \cong 367$ kA/m; aim for at least 50 kA/m either side of resonance*).

From the simulated oscillation amplitude versus bias field data, you will need to obtain a quantity proportional to the absorbed FMR power. The simplest way to do this is to square the oscillation amplitude data. The FMR power absorption peak is described by a Lorentz peak function, and you will need to fit this to your squared amplitude data.

Boris has built-in data processing command to help with processing FMR simulation data. You will need the following commands:

First load bias field and oscillation amplitude from the raw output data file (e.g. named 'fmr_fieldsweepFMR_data.*txt*'):

dp_load *fmr_fieldsweepFMR_data 0 1 0 1*

Next square the magnetization oscillation amplitude data:

dp_muldp *1 1 1*

Finally fit a Lorentz peak function to the data:

dp_fitlorentz *0 1*

The Lorentz peak function is given as:

$$f(x) = y_0 + S\frac{w}{4(x - x_0)^2 + w^2}$$

In the above equation *w* is the full-width half-maximum (FWHM), and x_0 is the peak center. You can obtain these values from the **dp_fitlorentz** command, including fitting uncertainties (Boris has a built-in generic Levenberg-Marquardt algorithm for curve fitting).

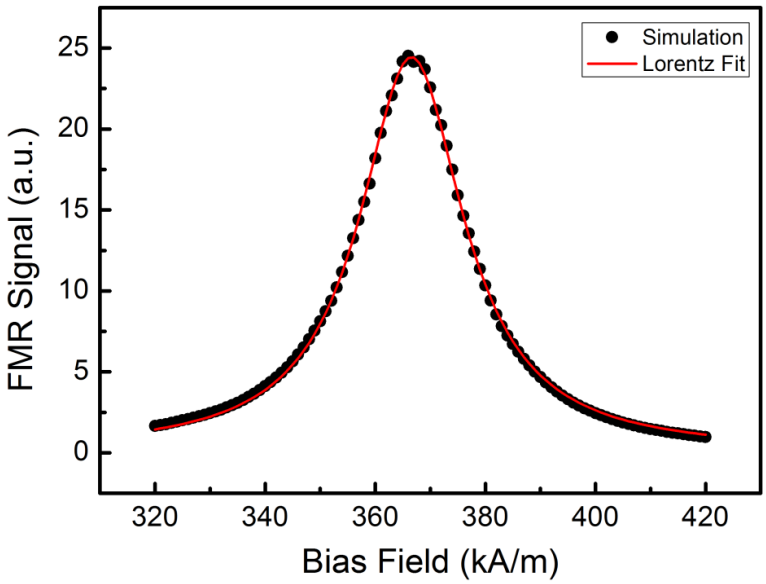
The magnetization damping value is related to the full-width half-maximum (ΔH) by:

$$\alpha = \frac{\mu_0 |\gamma_e|}{4\pi}\frac{\Delta H}{f}$$

Exercise 20.1 continued

Process the output FMR data and verify the damping obtained from the FWHM matches the set damping value (*damping* = 0.02).

Figure 20.1 – Simulated FMR peak with Lorentz peak function fit for Exercise 20.1.



Frequency-Swept FMR

Frequency-swept FMR simulations are also possible and are typically much more efficient to compute than field-swept FMR. This type of simulation is closely related to the method used to investigate spin-wave dispersion. Here we apply a sinc pulse in the time domain and capture the magnetisation response, which we then transform to the frequency domain using a Fourier transform. The reason for using a sinc pulse is its Fourier transform is a symmetric hat function with a defined frequency cut-off. Thus in order to capture the required resonance (and also any required higher resonance modes) we simply need to set the cut-off to a large enough value.

The sinc pulse excitation is given as:

$$H(t) = H_e \sin\left(2\pi f_c\left(t - t_0\right)\right) / \left(2\pi f_c\left(t - t_0\right)\right)$$

Here f_c (Hz) is the cut-off frequency, H_e is the excitation amplitude, with $H(t)$ taking on the value H_e at $t = t_0$, the sinc pulse centre. We also need a fixed bias field, H_0, which must be orthogonal to the excitation field, and in general we must capture the average magnetisation (for now we'll assume the magnetisation is uniform) and use the magnetisation component along the excitation field. We need to capture the magnetisation at a time step set by the Nyquist criterion:

$$t_S = \frac{1}{2 f_C} \quad (s)$$

Capturing magnetisation data at other time steps is sub-optimal and will result in lost information if larger than the above formula, or increased noise in the Fourier transform spectrum if smaller than the above formula. The sinc pulse must be simulated for a total time of $2t_0$: simulating for longer or shorter than this time will result in increased noise in the Fourier transform spectrum. If you want to increase the spectrum resolution (number of points) then you must instead increase the t_0 value, but still simulate for a total time of $2t_0$, saving data at a time step of t_s.

We can set this type of excitation using a text equation stage, namely *Hequation*. To set the above equation, edit the stage value for the added *Hequation* stage to: *He*sinc(2*PI*fc*(t-t0)), H0, 0*. This will set the excitation field along the x axis, and bias field along the y axis, with zero z axis field. A dedicated chapter in the manual is given for details on using text equations. In the above equation *t* is a reserved parameter, namely the stage time, and the equation provided is evaluated internally every iteration. The remaining parameters, *He*, *fc*, *t0*, *H0* must be defined in order for the equation evaluation to function. Equation constants may be given as:

equationconstants *name value,*

e.g. **equationconstants** fc 200e9.

Other related commands are **delequationconstant**, and **clearequationconstants**.

For frequency-swept FMR after taking the Fourier transform, similarly to field-swept FMR, the output data must be squared. The resulting FMR peak in the frequency domain is also described by a Lorentz peak, where the FWHM, Δf, is related to the resonance frequency f_0, and Gilbert damping as:

$$\alpha = \frac{\Delta f}{2 f_0}$$

By capturing a set of frequency-domain FMR peaks as a function of bias field H_0 the following Kittel formula can be verified:

$$f = \frac{\mu_0 |\gamma_e|}{2\pi} \sqrt{\left(H_0 + \left(N_y - N_z + k\right) M_S\right)\left(H_0 + \left(N_x - N_z + k\right) M_S\right)}$$

Here N_x, N_y, and N_z are demagnetizing factors such that the bias field is applied along the z direction. The above formula also includes uniaxial anisotropy, with easy axis along the z direction, where:

$$k = \frac{2K_1}{\mu_0 M_S^2}$$

For the following exercise you'll simulate a thin film Co material interfaced with Pt (this is stored in the materials database and can be loaded as **setmaterial** *Co/Pt* – more on the materials database in a dedicated chapter. You will also need to simulate a thin film, thus must set periodic boundary conditions in the xy plane as: **pbc** x 10, **pbc** y 10 – more on periodic boundary conditions in a dedicated tutorial.

Exercise 20.2 (Advanced)

Simulate the frequency-swept FMR response of a thin-film Co/Pt material with 2 nm thickness using the method described above.

You can use a cut-off frequency of 200 GHz, with excitation field amplitude of 1000 A/m, and vary the bias field between 100 kA/m and 1 MA/m. Simulate both out-of-plane FMR (anisotropy easy axis and bias field out of plane), and in-plane FMR (anisotropy easy axis and bias field in the plane, e.g. both along the y direction).

In both cases use the Kittel relation and damping formula to verify the input simulation parameters using fitting procedures (damping and magneto-crystalline anisotropy).

Tutorial 21 – Ferromagnetic Resonance with Spin Torques

Continuing from the previous tutorial, you will now investigate the effect of spin torques due to the spin-Hall effect on the magnetization damping using a Pt/$Ni_{80}Fe_{20}$ bilayer. The spin current generated in the Pt underlayer is absorbed by the $Ni_{80}Fe_{20}$ layer, resulting in a combination of damping-like and field-like torques. Depending on the current direction a decrease or increase of the effective damping is obtained. First the full spin transport solver is used, and following this a simpler method using an analytical form for the spin-orbit torques is introduced. Using the full spin transport solver we can also consider the effect of spin pumping on the effective damping, and this is investigated at the end of this tutorial.

The interfacial spin orbit torque added to the implicit LLG equation, as explained in Tutorial 17, is given by:

$$\mathbf{T}_S^{interface} = \frac{g\mu_B}{ed_h}\left[\mathrm{Re}\left\{G^{\uparrow\downarrow}\right\}\mathbf{m}\times\left(\mathbf{m}\times\Delta\mathbf{V}_S\right) + \mathrm{Im}\left\{G^{\uparrow\downarrow}\right\}\mathbf{m}\times\Delta\mathbf{V}_S\right]$$

For an N/F interface in the x-y plane with uniform current densities we can obtain an analytical expression for this interfacial torque as (see S. Lepadatu, Scientific Reports 7, 12937 (2017)):

$$\mathbf{T}_S^{interface} = \theta_{SHA,\,eff}\,\frac{\mu_B}{e}\frac{|J_c|}{d_h}\left[\mathbf{m}\times\left(\mathbf{m}\times\mathbf{p}\right) + r_G\mathbf{m}\times\mathbf{p}\right]$$

Here $\mathbf{p} = \mathbf{z} \times \mathbf{e}_{Jc}$, where $\mathbf{e}_{Jc}$ is the charge current direction, and:

$$\theta_{SHAeff} = \theta_{SHA}\left(1 - \frac{1}{cosh(d_N/\lambda_{sf}^N)}\right)\frac{\mathrm{Re}\left\{\tilde{G}\right\}^2 - \mathrm{Im}\left\{\tilde{G}\right\}^2 + N_\lambda Re\left\{\tilde{G}\right\}}{\left(N_\lambda + Re\left\{\tilde{G}\right\}\right)^2 + Im\left\{\tilde{G}\right\}^2},$$

$$r_G = \frac{N_\lambda\,\mathrm{Im}\{\tilde{G}\} + 2\,\mathrm{Re}\{\tilde{G}\}\,\mathrm{Im}\{\tilde{G}\}}{N_\lambda\,\mathrm{Re}\{\tilde{G}\} + \mathrm{Re}\{\tilde{G}\}^2 - \mathrm{Im}\{\tilde{G}\}^2}$$

In the above, $N_\lambda = tanh\left(d_N / \lambda_{sf}^N\right) / \lambda_{sf}^N$, $F_\lambda = tanh\left(d_F / \lambda_{sf}^F\right) / \lambda_{sf}^F$, and $\tilde{G} = 2G^{\uparrow\downarrow} / \sigma_N$. Thus the interfacial torque has a damping-like and a field-like component. In the limit of abrupt interface ($\lambda_\varphi \to 0$ or equivalently Re{$G^{\uparrow\downarrow}$} $\to \infty$) the field-like component tends to zero and we obtain the following expression for the torque:

$$\mathbf{T}_{SOT} = \theta_{SHA,eff} \frac{\mu_B}{e} \frac{|J_c|}{d_h} \mathbf{m} \times (\mathbf{m} \times \mathbf{p})$$

and

$$\theta_{SHA,eff} = \theta_{SHA} \left(1 - \frac{1}{cosh(d_N / \lambda_{sf}^N)} \right)$$

This approximation can also be used when the damping-like torque is much larger than the field-like torque. This expression is commonly used in the literature to model the spin-orbit torque resulting from the spin-Hall effect. Note however the spin-Hall angle in this expression is not the bulk (or intrinsic) spin Hall angle, but an effective spin Hall angle, scaled by transport parameters; if further the N layer thickness is many times larger than its spin flip length, we can use the approximation $\theta_{SHA,eff} \cong \theta_{SHA}$. In many cases this may not be true, and moreover the abrupt interface approximation may not be good either, thus to model the effect of the damping-like torque with the analytical form of the spin-orbit torque, in the expression for $\mathbf{T}_{SOT}$ you should use the full expression for the effective spin-Hall angle given above.

In Boris you can include this analytical spin-orbit torque using the *SOTField* module. Enabling this module in a ferromagnetic mesh introduces an additional effective field into the LLG equation which results in the $\mathbf{T}_{SOT}$ torque given above (as it appears in the implicit LLG equation). To use it you still need to have the *transport* module enabled in order to calculate the charge current density, but instead of selecting LLG-SA (enabling the full spin transport solver) you should select just the LLG equation (**ode** command). In the material parameters for the ferromagnetic mesh (**params** command) you need to enter the correct effective spin-Hall angle (*SHA*) to use with the *SOTField* module.

Exercise 21.1

Using the $Ni_{80}Fe_{20}$ layer from the previous tutorial, now add a Pt underlayer with the same dimensions (80 nm × 80 nm × 10 nm), using the Pt parameters from Tutorial 18. Set default electrodes (resulting in current flow along the x direction), enabling the spin transport solver both in the $Ni_{80}Fe_{20}$ and Pt meshes (add *transport* modules and set the **ode** solver to LLG-SA). Make sure to disable spin pumping (*pump_eff* = 0) and the inverse SHE (*iSHA* = 0). You should also disable bulk spin torques (*ts_eff* = 0), only leaving interfacial spin torques enabled (*tsi_eff* = 1). As before you may need to decrease the z-direction electrical cellsize to ensure accuracy (and numerical convergence!).

Obtain FMR peaks for charge current densities in the Pt layer of Jc = $\pm 10^{12}$ A/m^2. How does the damping change with current density direction?

In this case, even though you are using the Stoner-Wohlfarth model (*demag_N* module), you should still enabled the *exchange* module. The reason for this, the spin torques may not be perfectly uniform (e.g. if the permalloy and Pt layers have the same width, the spin torques will not be uniform since the spin accumulation has gradients at the sample edges), thus you do need to take the exchange interaction into consideration.

The change in damping due to a damping-like spin-orbit torque may be roughly approximated by:

$$\Delta\alpha_{SHE} \cong \theta_{SHA,eff} \frac{\mu_B}{e} \frac{J_c}{2\pi f M_S d_F}$$

Verify the change in damping obtained from simulations with the above formula.

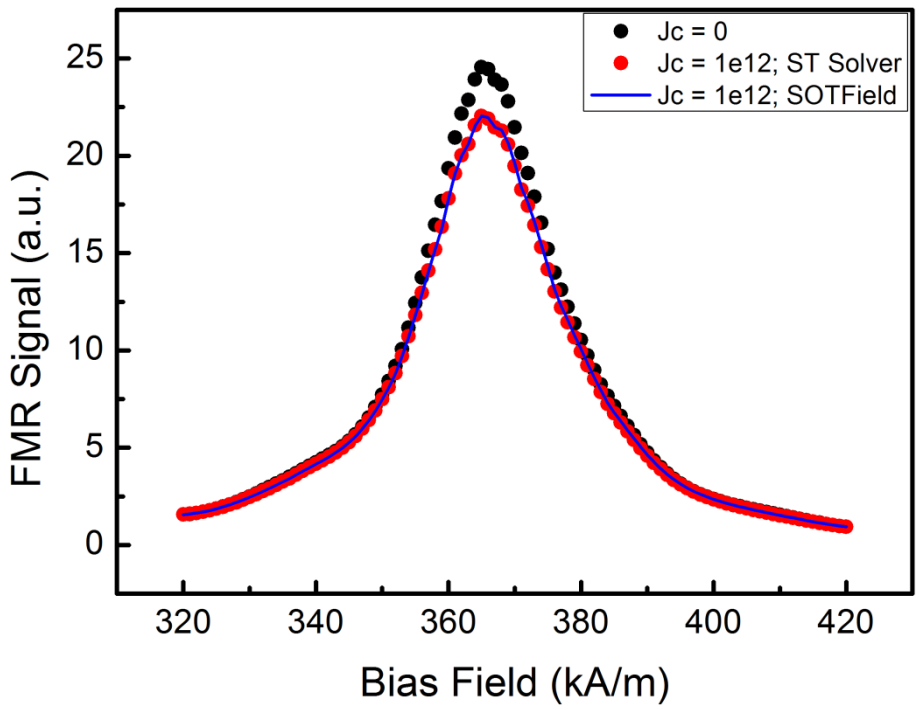
Exercise 21.2

Repeat Exercise 21.1 but this time without the spin transport solver, only using the analytical form for the spin-orbit torque (*SOTField* module). You should delete the Pt mesh and reset the electrodes and potential to give you the correct current density. Calculate an appropriate effective spin-Hall angle to use. Compare the results with the previous exercise.

Exercise 21.3

Repeat Exercise 21.1, using the full spin-transport solver, but now enable spin pumping (set *pump_eff* = 1). What is the increase in damping?

Figure 21.1 – FMR simulations with spin orbit torques for both the full spin transport solver (ST Solver) and effective field obtained from the analytical spin-orbit torque (SOTField).



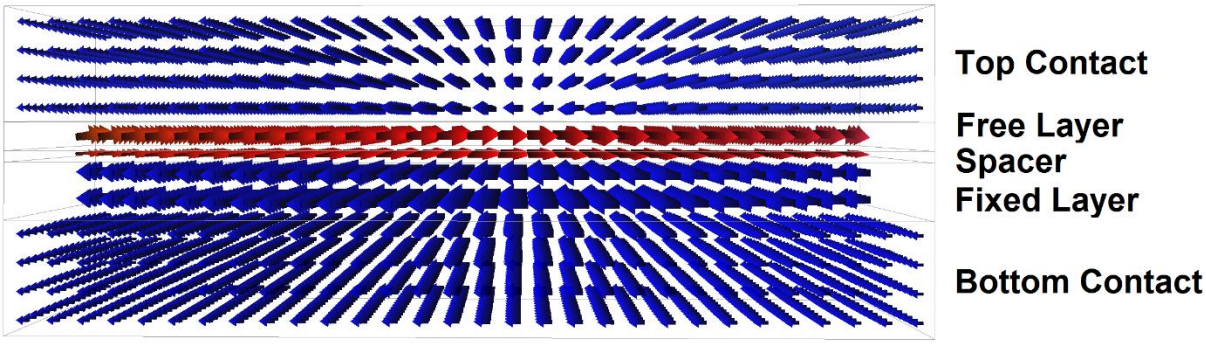
Tutorial 22 – CPP-GMR

The spin transport solver is also able to reproduce the spin torques in a current-perpendicular to plane (CPP) giant magneto-resistance (GMR) spin valve, in addition to its magneto-resistance. Here we will investigate the current-induced switching in a simple generic spin valve between the parallel and anti-parallel states, see Figure 22.1, and plot the resistance during these switching events.

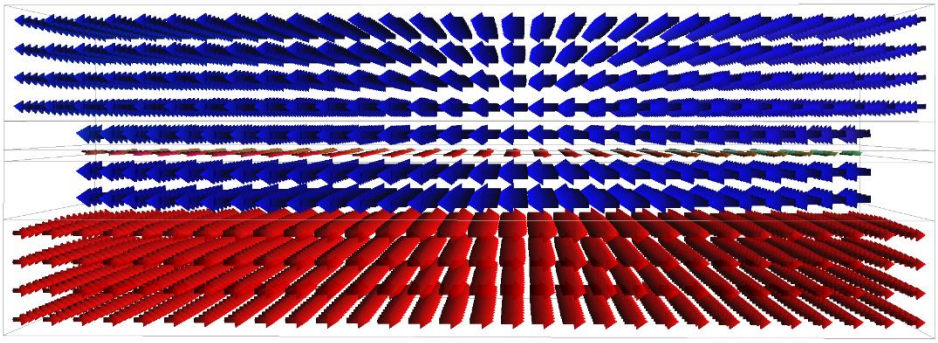
A spin valve, in its simplest form, consists of a fixed magnetic layer, a free magnetic layer which can be switched between an anti-parallel and parallel orientation with respect to the fixed layer, and a thin metallic spacer layer. The spacer layer thickness can be adjusted to give either a ferromagnetic or anti-ferromagnetic surface exchange coupling between the two magnetic layers. In the following simulation we will also add two metallic contacts, top and bottom.

Figure 22.1 – CPP-GMR spin valve showing the spin accumulation in the spacer layer, top, and bottom contacts, and the magnetization in the elliptically shaped fixed and free layers for a) anti-parallel state, and b) parallel state.

(a)



(b)



Exercise 22.1

Setup a generic spin valve structure (i.e. just use the default mesh parameters unless indicated otherwise) similar to that shown in Figure 22.1. This consists of:

- Bottom and top contacts (**addconductor**) with dimensions 160 nm × 80 nm × 20 nm. Disable spin-Hall effects in both (*SHA* = *iSHA* = 0).
- Spacer layer with dimensions 160 nm × 80 nm × 2 nm and set it to an elliptical shape (drag a .png file with a circle shape to the mesh viewer when the spacer layer mesh is in focus). Disable spin-Hall effects.
- Fixed layer (**addmesh**) with dimensions 160 nm × 80 nm × 10 nm and elliptical shape. Disable spin torques and spin pumping in this mesh (*ts_eff* = *tsi_eff* = *ts_pump* = 0). You should also disable magnetization dynamics in this mesh so the magnetization is fixed. You can do this by setting the relative gyromagnetic factor to zero (*grel* = 0 in **params**).
- Free layer with dimensions 160 nm × 80 nm × 5 nm and elliptical shape. Disable spin pumping and bulk spin torques only, in this mesh (i.e. keep *tsi_eff* = 1).

You will need to add the following modules:

- super-mesh multi-layered demagnetization (*sdemag*).
- *surfexchange* modules in both magnetic meshes. Edit the *J1* (bilinear surface exchange energy density) value in the free layer to give you a weak ferromagnetic coupling; set J1 = 0.1 mJ/m^2.
- *transport* modules in all meshes. Set the electrical cellsize to 5 nm × 5 nm × 1 nm everywhere except in the spacer layer where you should set it to 5 nm × 5 nm × 0.5 nm.

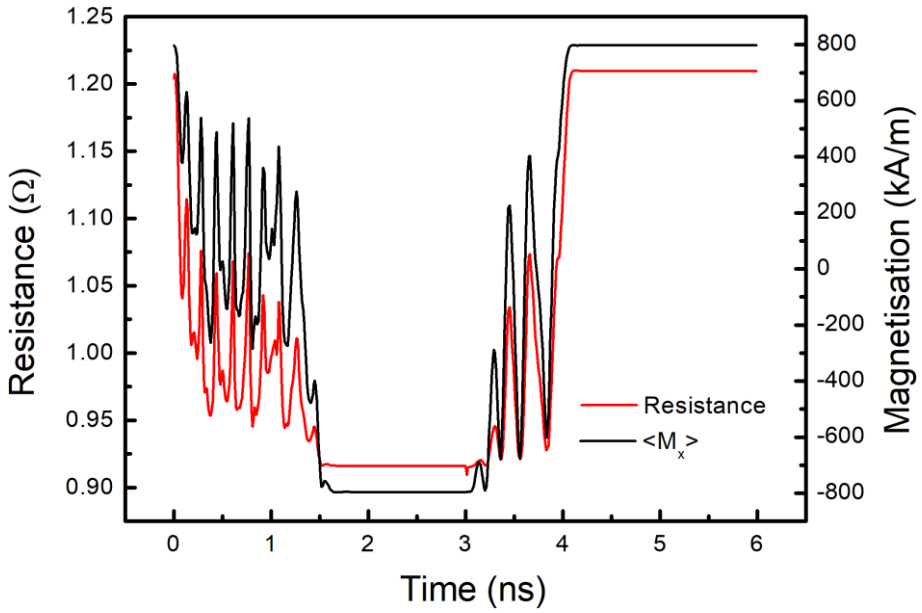
For the **ode** solver you should set the LLG-SA equation (thus enabling the spin-transport solver) with RKF45 evaluation. For output **data** you should have *time*, *R* (resistance), and <*M*> (average magnetization) in the free layer. Set electrodes top and bottom (**addelectrode**), designating the bottom electrode to be the ground electrode (**electrodes**). You need to simulate switching starting from the anti-parallel

state (see Figure 22.1) using a +15 mV pulse for 3 ns, then back to this state with a further -15 mV pulse for 3 ns. Save data every 10 ps for both stages. Before starting the simulation you should relax the starting state as follows:

1) Set all spin torques to zero and insert a *Relax* stage with *nostop* condition at the start.
2) First relax the magnetization in the anti-parallel state without the spin-transport solver (set **ode** to LLG).
3) Next enable the spin-transport solver and relax it (run it until the solver no longer iterates, monitoring *v_iter* and *s_iter* **data**).
4) Re-enable the appropriate spin torques (*tsi_eff* = 1 in the free layer only), **reset**, delete the *Relax* stage, then save the simulation (**savesim**).

Explain the resistance change observed by comparing it with the magnetization in the free layer as a function of time – see Figure 22.2 for expected results.

Figure 22.2 – Change in resistance for the CPP-GMR spin valve of Exercise 22.1, together with the magnetization along the longitudinal direction in the free layer.



Tutorial 23 – Skyrmion Movement with Spin Currents

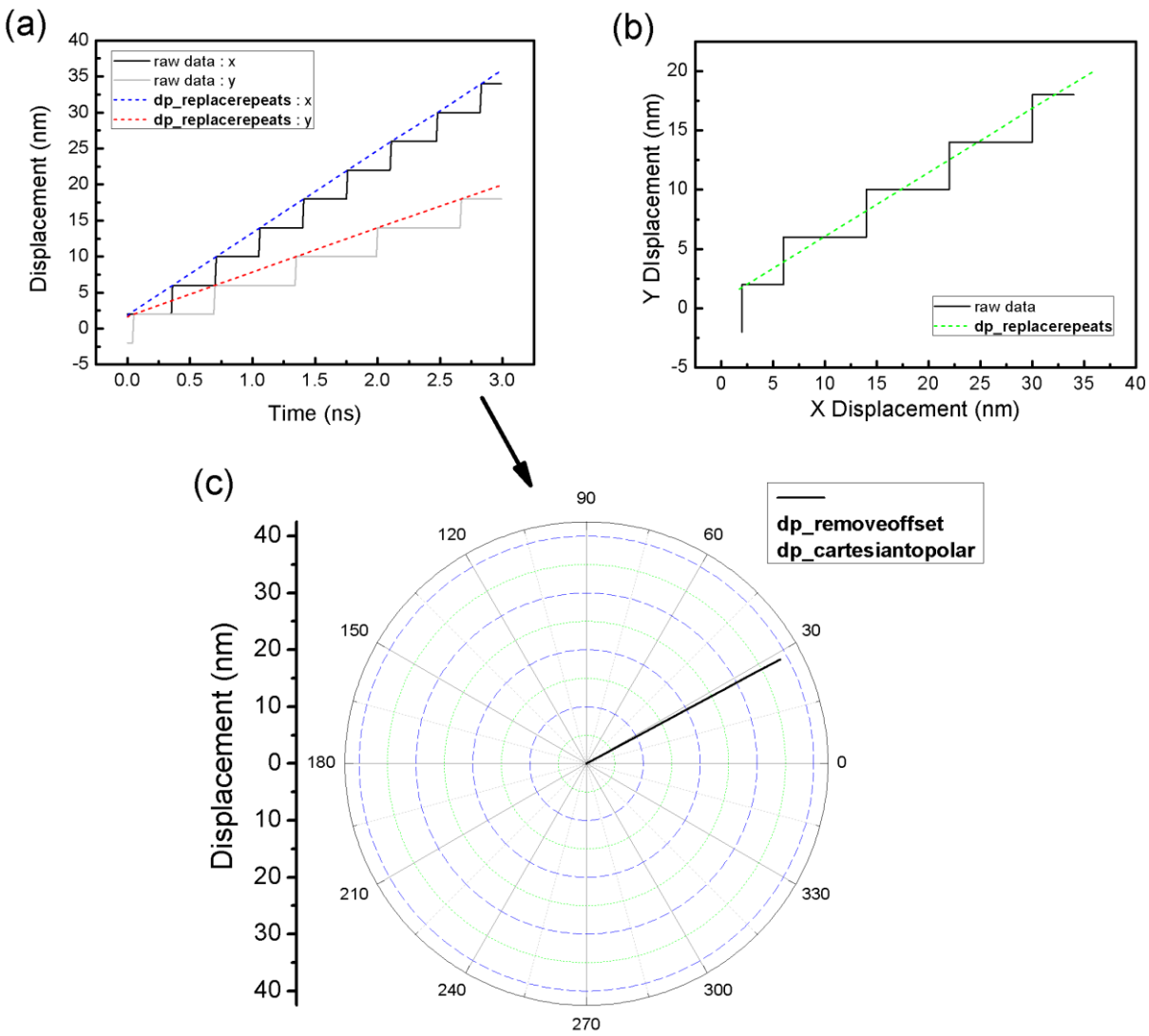
Skyrmions may be displaced efficiently using charge and spin currents. To study their movement a skyrmion tracking window can be used in Boris. There are two methods available for tracking a skyrmion, discussed below, both available as data outputs: *skyshift* and *skypos* (use **data** command).

The *skyshift* entry needs a rectangle defined, which should be set around the initial position of a skyrmion, making sure to fully contain it, but don't leave excessive space around it; the thickness of this rectangle should be set to the thickness of the ferromagnetic mesh containing the skyrmion. During a simulation a x-y shift is recorded and saved in the output data file. This shift is determined by comparing the average magnetization magnitude in the 4 quadrants of the skyrmion tracking window – e.g. if the skyrmion shifts to the right, the average magnetization magnitude in the 2 right-hand-side quadrants will decrease compared to the left, thus a right single-cell shift is recorded. Multiple *skyshift* entries can be defined, with different rectangles, to track multiple skyrmions. Note, the *skyshift* entry only works with data file output, and not in the data box or with the **showdata** command.

The raw output *skyshift* data will contain staircase steps due to mesh discretisation. It is possible to obtain a more natural skyrmion movement path by assuming linear displacement in between the staircase steps – this is illustrated in Figure 23.1. Here skyrmion displacement was simulated for 3 ns and the individual x and y *skyshift* raw data are shown in Figure 23.1(a). To remove the stair steps and replace them using linear interpolation you can use the **dp_replacerepeats** command on both the x and y *skyshift* data columns. The x, y data can then be plotted directly in Cartesian coordinates. You will notice this path doesn't start from (0, 0). To display the skyrmion displacement path relative to its starting position you should remove this offset using the **dp_removeoffset** command on both the x and y data. Finally, if you want to plot this path using polar coordinates you can use the **dp_cartesiantopolar** command, included in Boris for convenience. Note, especially when converting to polar coordinates you should check the processed data correctly represents the raw data. Problems may occur due to blips in the raw data, especially if the tracking

window was not defined well or the starting state is not sufficiently relaxed, thus the results from this procedure must be carefully compared with the raw data.

Figure 23.1 – Skyrmion movement raw data processing, showing (a) individual x and y displacements, (b) Cartesian coordinates path, and (c) polar coordinates path.



The second method uses the *skypos* data output. Again this needs an initial rectangle defined around the skyrmion as for the *skyshift* data output. *Skypos* uses a far more computationally expensive algorithm to track the skyrmion, but is able to obtain the exact skyrmion center position, as well as skyrmion diameters along the x and y axes, without being affected by staircase discretisation artifacts. It is also able to adjust the tracking window size if the skyrmion diameter changes significantly. The

algorithm works by fitting the skyrmion using an analytical function (the same one used for the **dp_fitskyrmion** command) in three steps: 1) Skyrmion is fitted along the x axis through the center of the skyrmion tracking window in order to find the center position. The tracking window x position is adjusted to center it. 2) Skyrmion is fitted along the y axis through the center of the skyrmion tracking window in order to find the center position. The tracking window y position is adjusted to center it. The y diameter is also recorded at this step. 3) The skyrmion is again fitted along the x axis to obtain its x diameter.

For the following exercises you should first use the *skyshift* method of tracking the skyrmion, the repeat them using the *skypos* method. Note, you should not use both methods simultaneously on the same skyrmion.

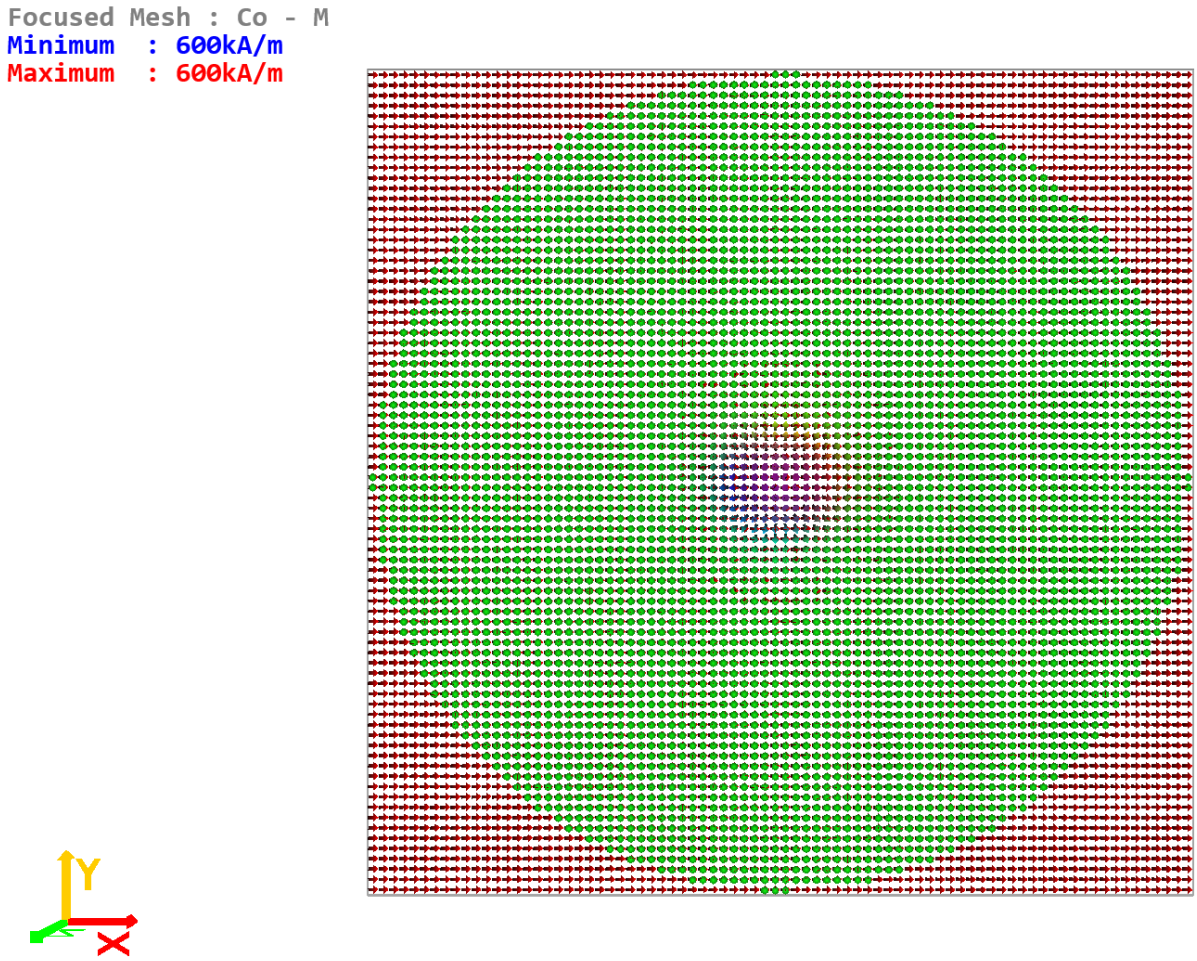
Exercise 23.1

a) Setup a Pt/Co bilayer with a skyrmion relaxed at the center of the Co layer under a 15 kA/m out-of-plane magnetic field as shown in Figure 23.2. The Pt layer should be a 320 nm × 320 nm × 3 nm rectangle, whilst the Co layer should be a 320 nm diameter disk with a 1 nm thickness. Relax this magnetization configuration.

   For Pt you should use σ = 7×10^6 S/m, λ_{sf} = 1.4 nm and θ_{SHA} = 0.19. Use a discretisation cellsize of (4 nm, 4 nm, 0.5 nm). Set *iSHA* to zero.

   For Co you should use σ = 5×10^6 S/m, λ_{sf} = 38 nm, λ_J = 2 nm, λ_φ = 4 nm, G_{mix} = 1.5 PS/m^2, g_{rel} = 1.3, α = 0.03, M_S = 600 kA/m, A = 10 pJ/m, D = -1.5 mJ/m^2, K_1 = 380 kJ/m^3 with uniaxial anisotropy perpendicular to the plane. You should also enable the interfacial DM exchange module. Use a discretisation cellsize of (4 nm, 4 nm, 1 nm) for magnetic computations and (4 nm, 4 nm, 0.25 nm) for spin transport computations. In the Co mesh only enable the interfacial spin torques, not the bulk spin torques or spin pumping.

Figure 23.2 – Skyrmion in a Co disk on a Pt underlayer.



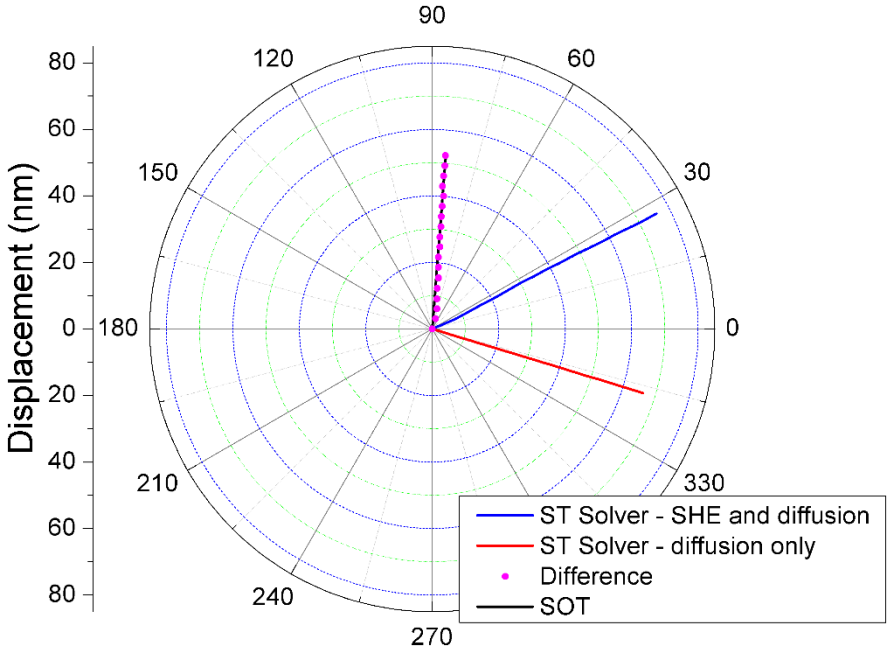
b) Enable the spin transport solver in both meshes and set electrodes at the x-axis ends of the Pt mesh only. Set a -20 mV potential for 3 ns and save the *time* and *skyshift* (or *skypos*) data every 10 ps. (for *skyshift* and *skypos* define a rectangle around the initial position of the skyrmion). Simulate the skyrmion movement path with SHE enabled (*SHA* = 0.19 in the Pt mesh), as well as without SHE (*SHA* = 0 in the Pt mesh), and plot them in polar coordinates.

c) Simulate the skyrmion movement path without the spin transport solver but with the *SOTfield* module enabled. Still keep the *transport* module enabled to calculate the charge current density. For the *SOTfield* module set a suitable effective spin Hall angle in the Co mesh (*SHA*). You can use the effective spin Hall angle formula from Tutorial 21, but note this is only strictly applicable for

uniform magnetization and spin currents. You can use this as a starting point, but will need to adjust the effective spin Hall angle.

Plot the skyrmion path in polar coordinates and compare it with the path obtained as the difference between the SHE and no SHE simulations above – see Figure 23.3 for expected results.

Figure 23.3 – Skyrmion movement paths obtained in Exercise 23.1.



Tutorial 24 – Roughness and Staircase Corrections

Staircase Corrections

With finite difference discretisation, errors can arise due to a staircase effect when discretising curved boundaries. In micromagnetics the largest errors arise in the demagnetizing field and may be reduced by decreasing the discretisation cellsize. This method is inefficient however since for most problems the results converge when the discretisation cellsize is close to the exchange length of the material – thus to further reduce this everywhere just to improve the discretisation accuracy at a boundary is very inefficient. Note, for materials where the demagnetizing energy dominates the exchange length may be defined as:

$$l_{ex} = \sqrt{\frac{2A}{\mu_0 M_S^2}}$$

For systems where the anisotropy energy dominates ($K_u > \mu_0 M_S^2/2$), the exchange length may be defined as:

$$l_{ex} = \sqrt{\frac{A}{K_u}}$$

Instead of refining this, a good approximation may be achieved by computing a correction field using a finely discretised demagnetization kernel, but applying it at run-time to the coarsely discretised mesh, as described in S. Lepadatu, Journal of Applied Physics 118, 243908 (2015). This correction field is typically similar to an uniaxial anisotropy field when averaged.

To enable staircase corrections in a particular magnetic mesh you must enable its *Roughness* module. When applying a mask shape to the mesh, staircase corrections will now automatically be taken into account. You must enable the *Roughness* module and reset the mesh shape before applying the mask to correctly enable

staircase corrections. You must also set the required refinement using the **refineroughness** command:

refineroughness m_x m_y m_z

When calculating the correction field factors, the shape is first discretised on a fine mesh as set by the **refineroughness** parameters. Thus if the coarse mesh has cellsize (h_x, h_y, h_z), the fine mesh used for correction field initialization has cellsize (h_x / m_x, h_y / m_y, h_z / m_z). Typically the improvement in accuracy is small above m > 10, so the refinement set should not be excessive. Since a demagnetizing kernel must be computed for the fine mesh, the initialisation time may become very long, and the available memory may be exceeded if the m factors are set too large. You must also set them before applying the mask shape.

With the *Roughness* module enabled, setting a mask shape may result in a slightly different shape than without. This is because a fine shape is internally obtained first, then the coarse mesh shape is calculated to be the smallest shape which includes the fine shape on the coarse mesh – this is a requirement of the corrections calculation method. To clear the staircase corrections, effectively setting the fine mesh shape to the coarse mesh shape you can use:

clearroughness

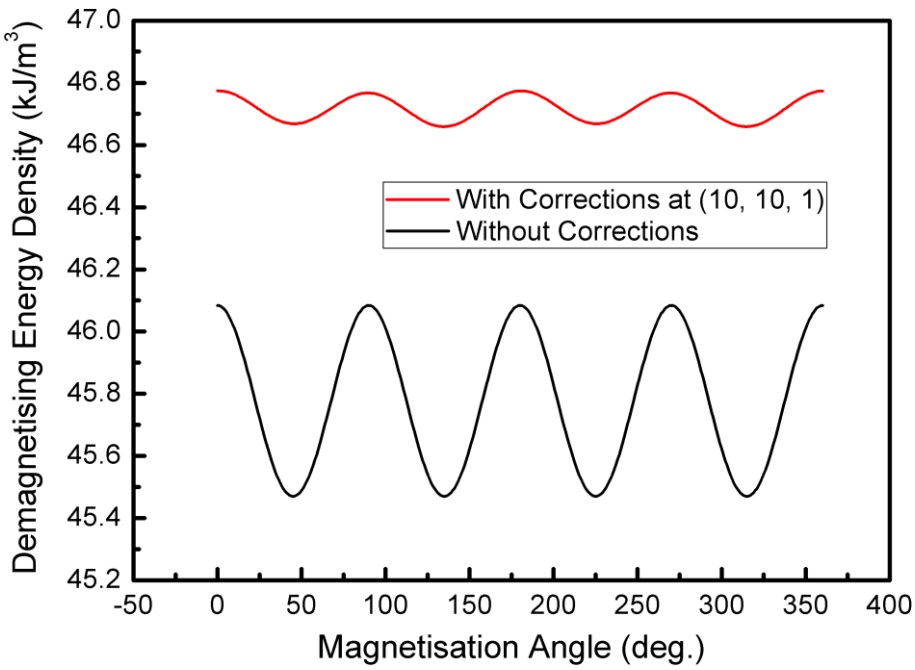
There's an energy density term associated with the correction fields, and this is available as a **data** parameter: *e_rough*. The demagnetizing energy density, *e_demag*, still corresponds to the coarse mesh shape; the sum of *e_rough* and *e_demag* is the approximated demagnetizing energy for the fine mesh shape.

Exercise 24.1

Set a 80 nm diameter $Ni_{80}Fe_{20}$ disk with 10 nm thickness. Calculate the demagnetizing energy density as a function of in-plane uniform magnetization orientation from 0° through 360° by saturating in a strong magnetic field (10^6 A/m). Repeat this computation but now set the shape with the *Roughness* module enabled and a refinement of (10, 10, 1) – **refineroughness**. Compare the demagnetizing energies for the coarse mesh and the approximated demagnetizing energy for the fine mesh (*e_rough* + *e_demag*).

For the above exercise, in theory the demagnetizing energy should be constant for a circle as the field rotates. In practice, due to discretisation errors a shape anisotropy effect is observed (the magnetization is not fully saturated even at 10^6 A/m, so some non-uniformity persists). With staircase corrections enabled this anisotropy should be significantly reduced, thus closer to the ideal uniform demagnetizing energy – see Figure 24.1. The refinement can be increased but further improvement is small.

Figure 24.1 – Demagnetizing energy computed for a circle with and without staircase corrections.



Edge and Surface Roughness

The same model used to reduce staircase corrections may be applied to compute the effect of topological roughness with variations below the exchange length of the material. As before, coefficients for a roughness field are computed at initialisation depending on the shape of a finely discretised mesh (the mesh with topological roughness applied), and that of the coarse mesh (the actual mesh used in computations but without roughness).

A roughness profile may be applied using a built-in algorithm, or alternatively a mask may be used. See Figures 24.2 – 4 for examples of real surface scans, processed into a grayscale image suitable for use as masks. These may be found in the Examples folder for this tutorial.

Figure 24.2 – Granular surface roughness profile

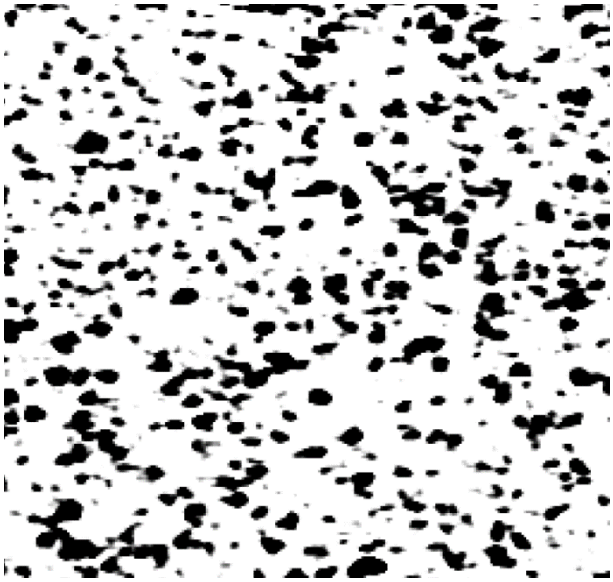


Figure 24.3 – Maze-like surface roughness profile.

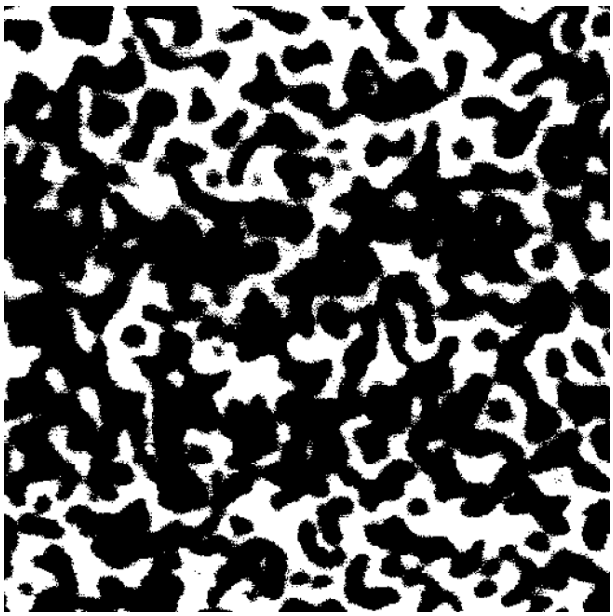
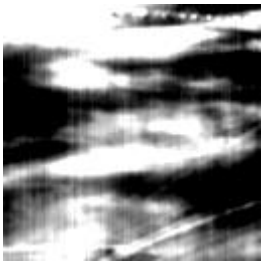


Figure 24.4 – Elongated defects, or stripes, surface roughness profile.



To apply a roughness profile using a mask, instead of simply dragging the file to the mesh viewer (as you would do when applying a shape), you should also specify the depth to which you want to apply the profile. For example, with the mask shown in Figure 24.4, to apply it to a 4.5 nm depth (the coarse discretisation cellsize is 5 nm so this keeps the coarse mesh shape intact) you need to use:

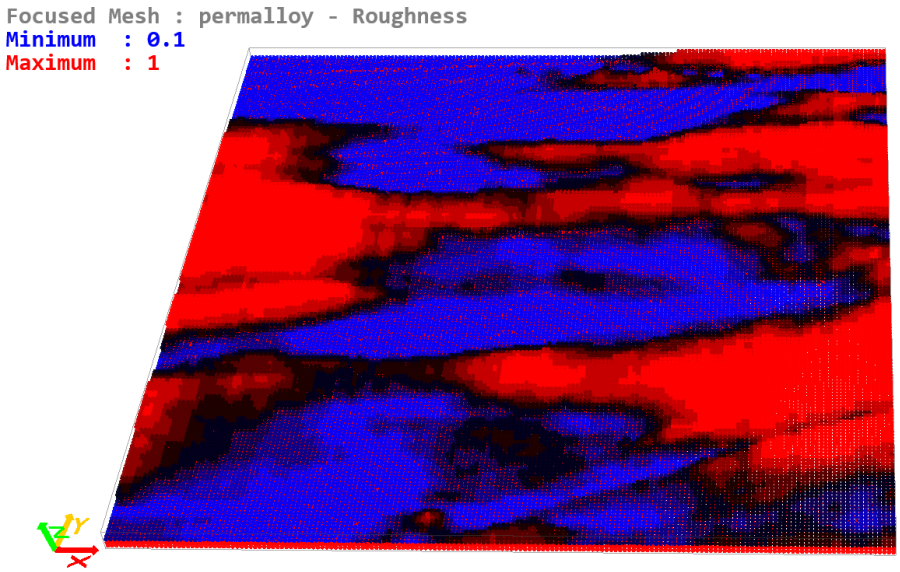
loadmaskfile *4.5nm (directory\)Stripes*

This will apply a surface roughness profile on the top face up to 4.5 nm depth – black results in 0 depth cut, whilst white results in full depth cut (i.e. 4.5 nm); values on the

greyscale in between are correlated linearly with the depth cut. For the actual surface roughness profile obtained see Figure 24.5.

In this example the starting mesh has dimensions of 320 nm × 320 nm × 10 nm, and the roughness refinement was set to (4, 4, 10) – **refineroughness**. To view the set roughness, under **display** select the *Roughness* option for the respective mesh. To apply the surface roughness to the bottom face, negative values need to be set for the depth value – see help for **loadmaskfile** command.

Figure 24.5 – Applied surface roughness using the mask in Figure 24.4.



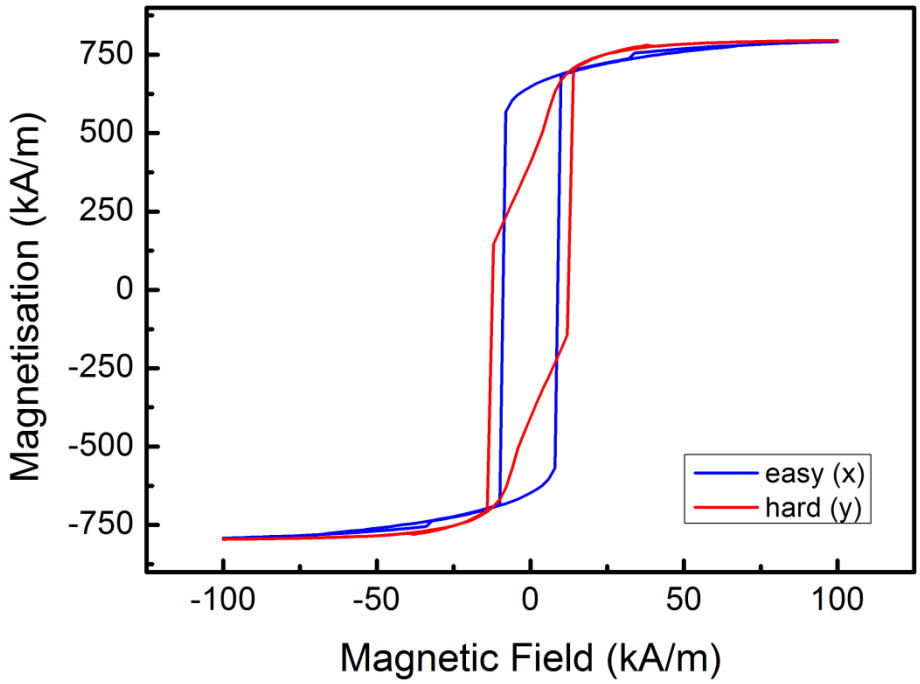
You can also apply edge and surface roughness using a built-in console command. Currently two methods are available: **roughenmesh**, **surfroughenjagged**. The **roughenmesh** command applies a completely random roughness profile to one of the 6 faces as indicated – see help for this command. The **surfroughenjagged** command applies a jagged profile to either the top, bottom, or both, faces – see help for this command.

Exercise 24.2

Set a 160 nm × 160 nm × 10 nm permalloy rectangle and apply the Stripes roughness profile from Figure 24.4 (use file in Examples folder) to a depth of 4.5 nm. Use a roughness refinement of (4, 4, 10). Simulate the hysteresis loops along the x and y directions and compare them. (*You should apply the field at a slight angle to the x and y directions to avoid artifacts associated with a finite geometry – in particular for the easy axis you want to avoid the "U" shape configuration at zero field which can happen if the field is perfectly along the x axis.*)

Since permalloy does not have a magneto-crystalline anisotropy, without roughness it is expected the two hysteresis loops will be identical. With roughness applied an effective anisotropy is observed, due to the orientation of the surface roughness stripes as seen in Figure 24.5.

Figure 24.6 – Hysteresis loops for Exercise 24.2, showing a roughness-induced anisotropy effect.



Tutorial 25 – Defects and Impurities

Material parameters in Boris may also be assigned a spatial variation, in addition to a temperature dependence. This spatial variation will be taken into account in all routines where the material parameters appear, allowing inclusion of material defects and impurities in simulations as appropriate.

To see the currently set parameters spatial variation use the command:

paramsvar

You can use a pre-defined method of generating defects by following the instructions displayed after using the **paramsvar** command. Currently these include: *random*, *jagged*, *defects*, *faults*. To see the spatial variation generated, under **display** select the *ParamVar* option, making sure to select the required parameter under the **paramvar** list. The generated spatial variation is stored as an array of coefficients, multiplying the base parameter value. For examples of these profiles see Figures 25.1 – 4.

Figure 25.1 – *Random* parameter variation between 0.9 and 1.1 with generator seed 1.

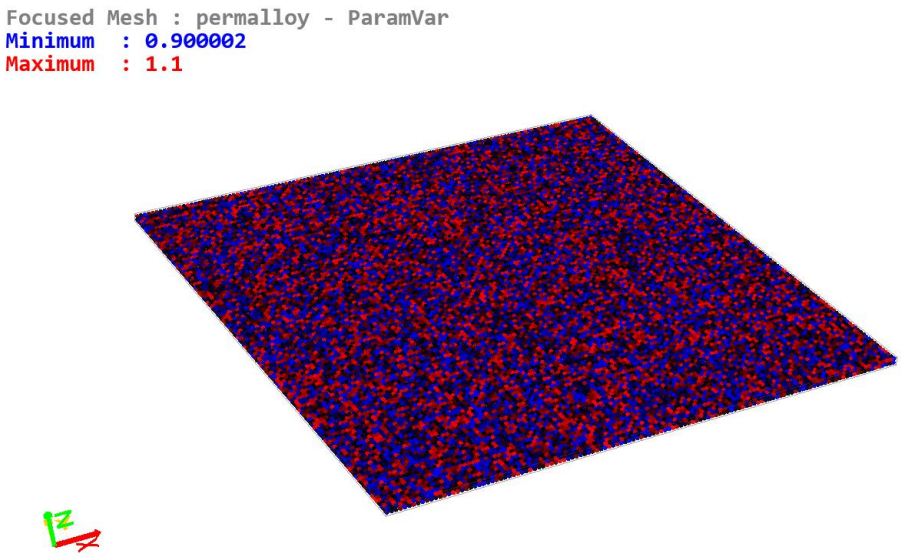


Figure 25.2 – *Jagged* parameter variation between 0.9 and 1.1 with 30 nm average spacing and generator seed 1.

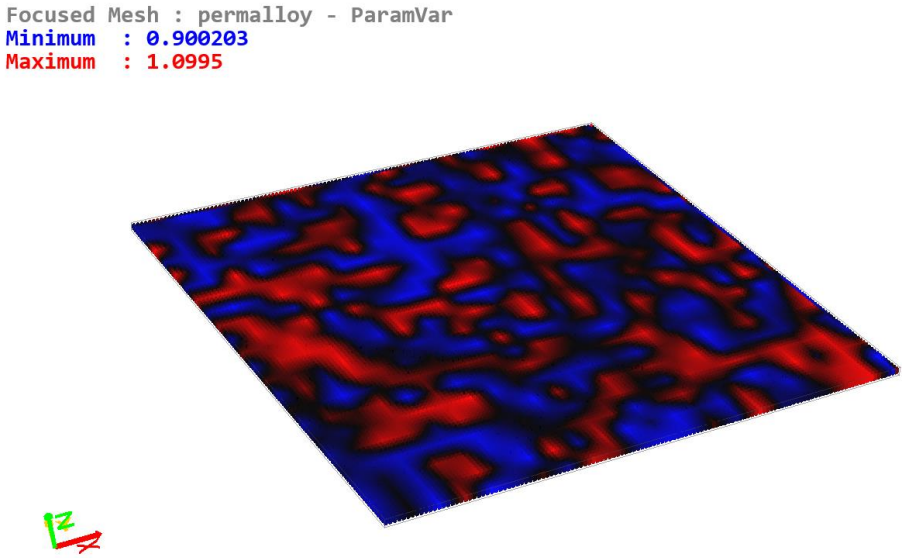


Figure 25.3 – *Defects* parameter variation between 0.9 and 1.1 with diameters in the range 20 nm to 50 nm, and 40 nm average spacing with generator seed 1.

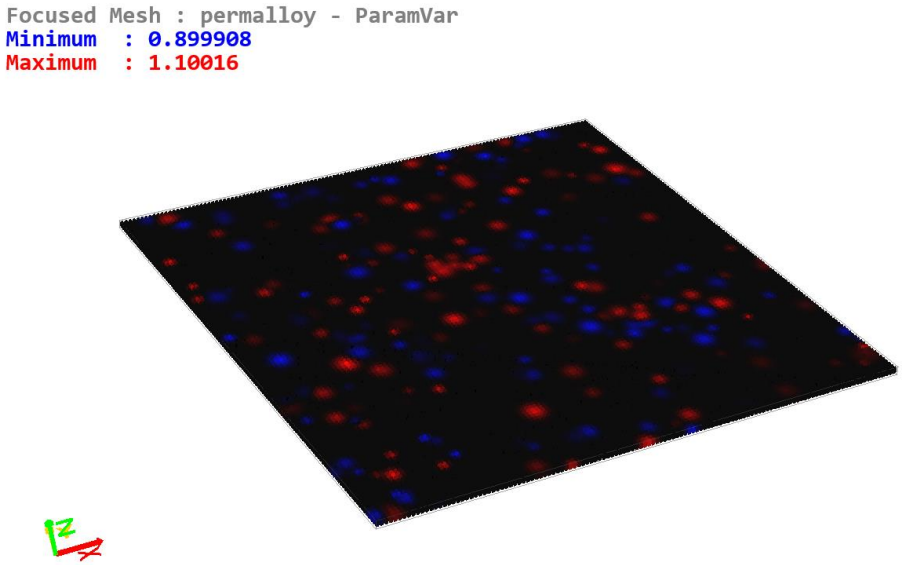
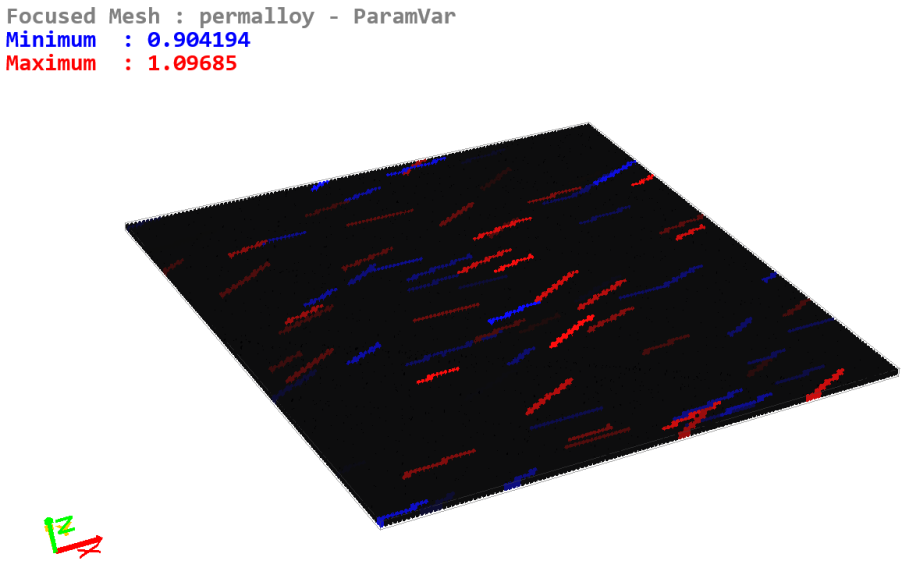


Figure 25.4 – *Faults* parameter variation between 0.9 and 1.1 with 20 nm to 50 nm fault length, -30° to 30° fault orientation and 50 nm average spacing with generator seed 1.



Exercise 25.1

Generate M_S defects in a 640 nm × 640 nm × 10 nm mesh as shown in Figures 25.1 – 4.

You can also set a custom parameter variation using an image as a mask file, although this is now a legacy option and documented in a previous version (v2.4). Instead if you want to set an arbitrary parameter variation you should use the ovf2 file option, which can be programmatically generated – a routine is included in the NetSocks module, allowing for easy generation of parameter spatial variation. This is covered in a separate chapter in the manual on working with ovf2 files. You can also set parameter variation using a text equation, which simultaneously allows for both spatial and temporal dependence. This is covered in a separate chapter in the manual on working with text equations.

Tutorial 26 – Polycrystalline and Granular Films

Boris includes a Voronoi tessellation generator, both 2D and 3D, which can be used to generate polycrystalline and granular films.

Figure 26.1 – Polycrystalline film showing *K1* parameter variation generated using **vor2D** generator between 0.9 and 1.1 with 40 nm spacing and generator seed 1.

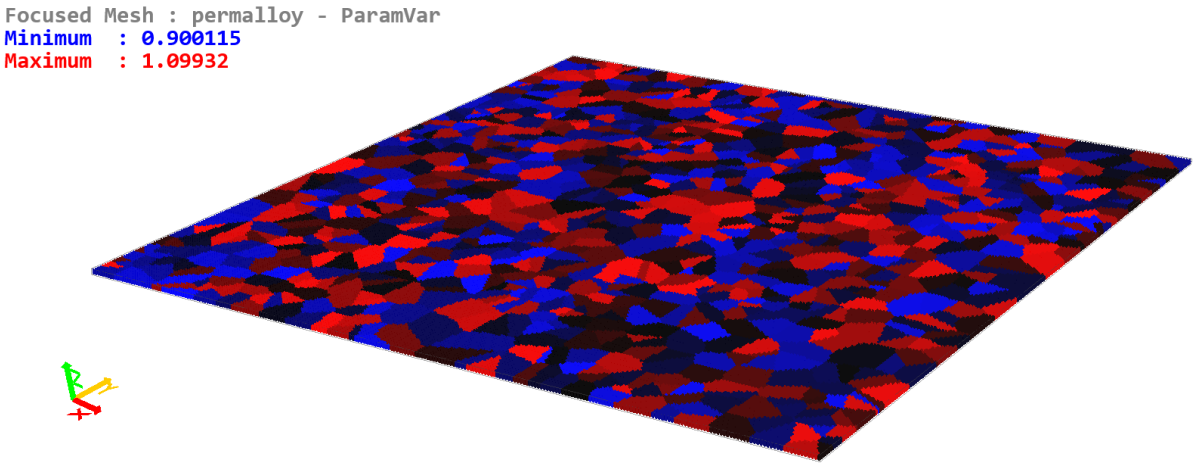
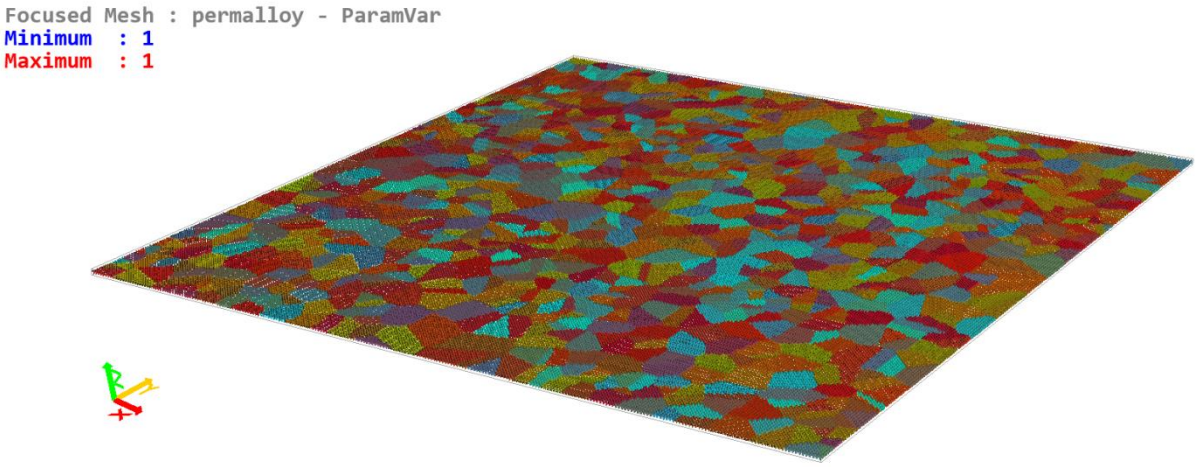


Figure 26.2 – Polycrystalline film showing easy axis (*ea1*) parameter variation generated using **vorrot2D** generator with polar angle range 70° to 110°, azimuthal angle range -90° to 90°, with 40 nm spacing and generator seed 1.



Polycrystalline films may be simulated by generating parameter variations using one of the Voronoi tessellation generators under the **paramsvar** command. These include:

vor2d *min, max; spacing; seed* – Used for 2d crystallites in the xy plane.
vor3d *min, max; spacing; seed* – Used for 3d crystallites.

vorbnd2d *min, max; spacing; seed* – Used for 2d crystallites in the xy plane, but parameter variation generated randomly only at Voronoi cell boundaries.
vorbnd3d *min, max; spacing; seed* – Used for 3d crystallites, but parameter variation generated randomly only at Voronoi cell boundaries.

vorrot2d *min_polar, max_polar; min_azimuthal, max_azimuthal, spacing; seed* – Used for 2d crystallites in the xy plane, specifically magneto-crystalline anisotropy easy axes.
vorrot3d *min_polar, max_polar; min_azimuthal, max_azimuthal, spacing; seed* – Used for 3d crystallites, specifically magneto-crystalline anisotropy easy axes.

Both 2D and 3D crystallites may be generated using the generators listed above. For an example of a 2D polycrystalline film with *K1* (magneto-crystalline anisotropy) variation see Figure 26.1. In order to generate crystallites with a varying magneto-crystalline anisotropy easy axis orientation you can use either the **vorrot2d** or **vorrot3d** generator – for example see Figure 26.2 for the *ea1* parameter having the same polycrystalline structure as in Figure 26.1. Figure 26.2 shows the rotation to be applied, as a vector quantity. For an easy axis base value set along the x-axis this coincides with the resulting easy axis orientation.

You can also generate a parameter variation at the Voronoi cell boundaries rather than in the cells themselves. This can be done using the **vorbnd2d** and **vorbnd3d** generators. This could be useful for example to modify the electrical conductivity at the grain boundaries only.

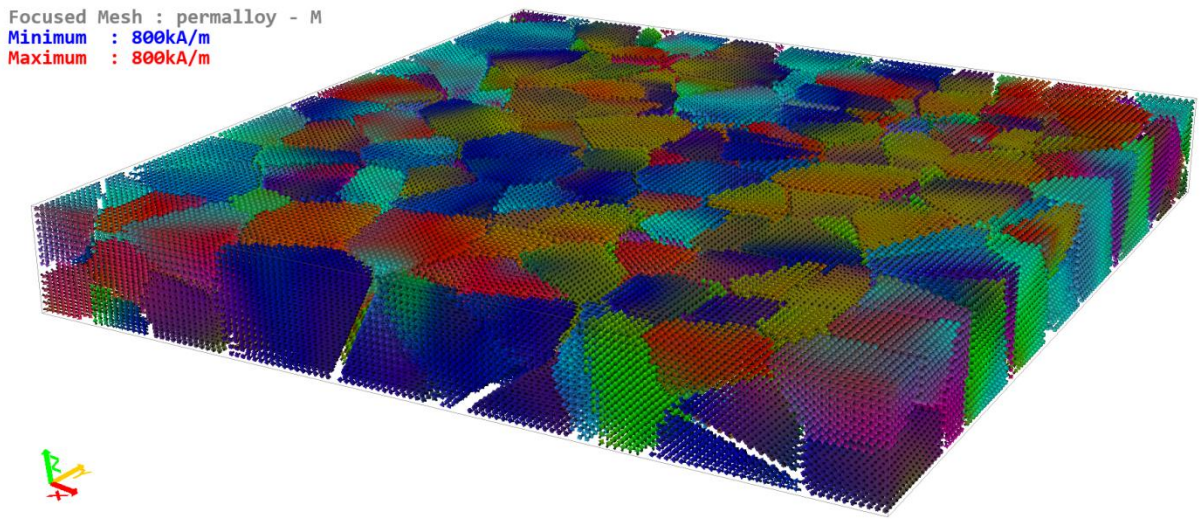
In order to generate a granular film with non-magnetic phase separation you can use one of the following commands:

generate2dgrains *spacing (seed)*
generate3dgrains *spacing (seed)*

These commands generate granular films directly on the magnetization mesh – see for example Figure 26.3. You can also combine this with parameter variations generated using the same generator seed and sizes (e.g. grains with varying M_S values).

When generating grains for the magnetic mesh you can choose to generate grains for the electrical conductivity mesh also. To carry the grain structure over you should generate the grains first without the *Transport* module enabled. After enabling the *Transport* module the granular structure is also applied to the electrical conductivity mesh. This could be useful for example in a multi-layered structure. If you apply the grain structure with the *Transport* module already enabled, the grains are not generated for the electrical conductivity also. You could combine this with a Voronoi generator for the *elC* material parameter however (e.g. **vorbnd2d** or **vorbnd3d**, or even **vor2d** or **vor3d**) as mentioned above.

Figure 26.3 – Granular film (80 nm thick) with non-magnetic phase separation, generated using **generate3dgrains** command with 50 nm spacing and generator seed 1. The image shows a magnetization configuration at zero field.



Tutorial 27 – Periodic Boundary Conditions

Periodic boundary conditions (PBC) may be applied when computing demagnetizing fields. Enabling this setting also affects exchange coupling (e.g. *Exchange*, *DMExchange*, *iDMExchange*), resulting in a wrap-around effect. PBCs are useful to simulate periodic arrays, or to approximate effectively infinite thin films or tracks using only a finite simulation window. PBCs are applicable to both single mesh demagnetization (*Demag*), as well as supermesh demagnetization (*SDemag*), including multi-layered demagnetization. Moreover, when enabling the *Roughness* module, PBCs are also taken into account when calculating an effective roughness field.

To enable PBCs use the command:

pbc

The configuration for all meshes, or the supermesh if applicable, will be shown. You can enable PBCs in any direction by setting a number of images using the interactive console objects (10 images are set by default when enabled, which can be edited if required; the default setting is for no PBCs). If the *SDemag* module is enabled you can edit the PBC settings on the supermesh, otherwise you need to edit them for the individual meshes.

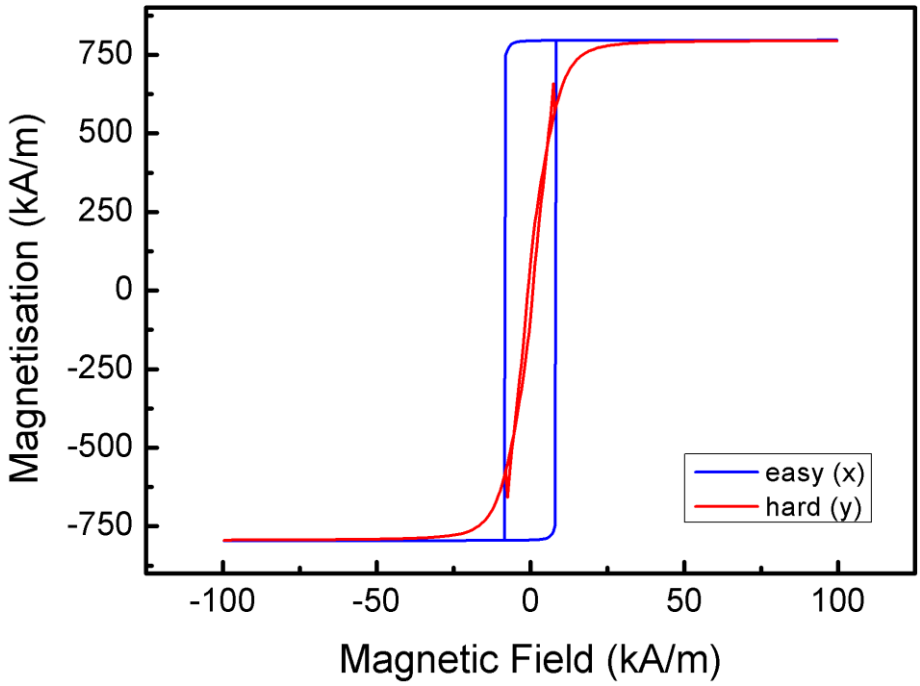
When using PBCs with multi-layered convolution you need to ensure the problem is physically meaningful. For example the recommended approach is to use a z direction stacking of layers, then either x or y PBCs are fine, but z PBCs will not lead to physically meaningful results in this case.

Exercise 27.1

Repeat Exercise 24.2, but this time set default PBCs along both x and y. Compare the hysteresis loops.

When using PBCs with the *Roughness* module enabled, you should be careful about setting a combination of a large number of PBC images and fine roughness refinement (**refineroughness**). This can result in excessive initialization time due to the large demag kernel calculated by the *Roughness* module.

Figure 27.1 – Hysteresis loops obtained for Exercise 27.1. The small peaks in the hard axis loop are artifacts resulting from the finite simulation mesh size, and could be decreased by increasing the simulation mesh rectangle.



Tutorial 28 – Ultrafast Demagnetisation

Ultrafast demagnetisation processes may be studied in Boris using the LLB equation (or sLLB) coupled to a two-temperature model. The default heat equation doesn't differentiate between lattice and electron temperature, i.e. it is a 1-temperature model. With the two-temperature model, two coupled equations for the electron and lattice temperatures are given as:

$$C_e \rho \frac{\partial T_e(\mathbf{r},t)}{\partial t} = \nabla . K \nabla T_e(\mathbf{r},t) - G_e(T_e - T_l) + S$$
$$C_l \rho \frac{\partial T_l(\mathbf{r},t)}{\partial t} = G_e(T_e - T_l)$$

Here C_e and C_l are the electron and lattice specific heat capacities, ρ is the mass density, K is the thermal conductivity, and G_e is the electron-lattice coupling constant, typically of the order 10^{18} W/m^{3}K.

To change the temperature model use the following command (or use the interactive console output from **tmodel** command):

tmodel *num_temperatures (meshname)*

e.g. **tmodel** 2 sets the two-temperature model for the currently focused mesh (nnot applicable to *insulator* meshes). The material parameters in the above equations are available as usual under the **params** command console output (also see list of parameters under the Material Parameters section in this manual).

In the above equation S is a heat source; a heat source may be specified in simulation using one of the following stage types: *Q*, *Q_seq*, *Qequation*, *Qfile*. These stages specify a heat source (W/m^3) in the heat equation: *Q* sets a constant value, *Q_seq* sets a sequence of values (similar to Hxyz_seq for external fields), *Qequation* sets a heat source using a text equation, thus allowing both spatial and temperature dependence, and finally *Qfile* sets a spatially uniform heat source, but with arbitrary time dependence as specified using a text file (see description of stage in console

help). Moreover a spatial variation may be assigned to the parameter *Q* under the **paramsvar** command.

A typical heat source from a focused laser pulse is given as:

$$S = P_0 \exp\left(\frac{-|\mathbf{r}-\mathbf{r}_0|}{d^2/4\ln(2)}\right)\exp\left(\frac{-(t-t_0)^2}{t_R{}^2/4\ln(2)}\right) \quad (W/m^3)$$

Here *d* and t_R are full-width at half-maximum (FWHM) values (pulse diameter and duration respectively) and P_0 is the maximum power density. This heat pulse may be simulated using a *Qequation* stage set to (pulse centre coincides with simulation mesh centre):

*Q0 * exp(-sqrt((x/Lx - 0.5)^2 + (y/Ly - 0.5)^2) / ((d0/Lx)^2/(4*ln(2)))) * exp(-(t-2*tau)^2/(tau^2/(4*ln(2))))*

For this equation we need to define the user constants (**equationconstants**): i) Q0, ii) d0, iii) tau.

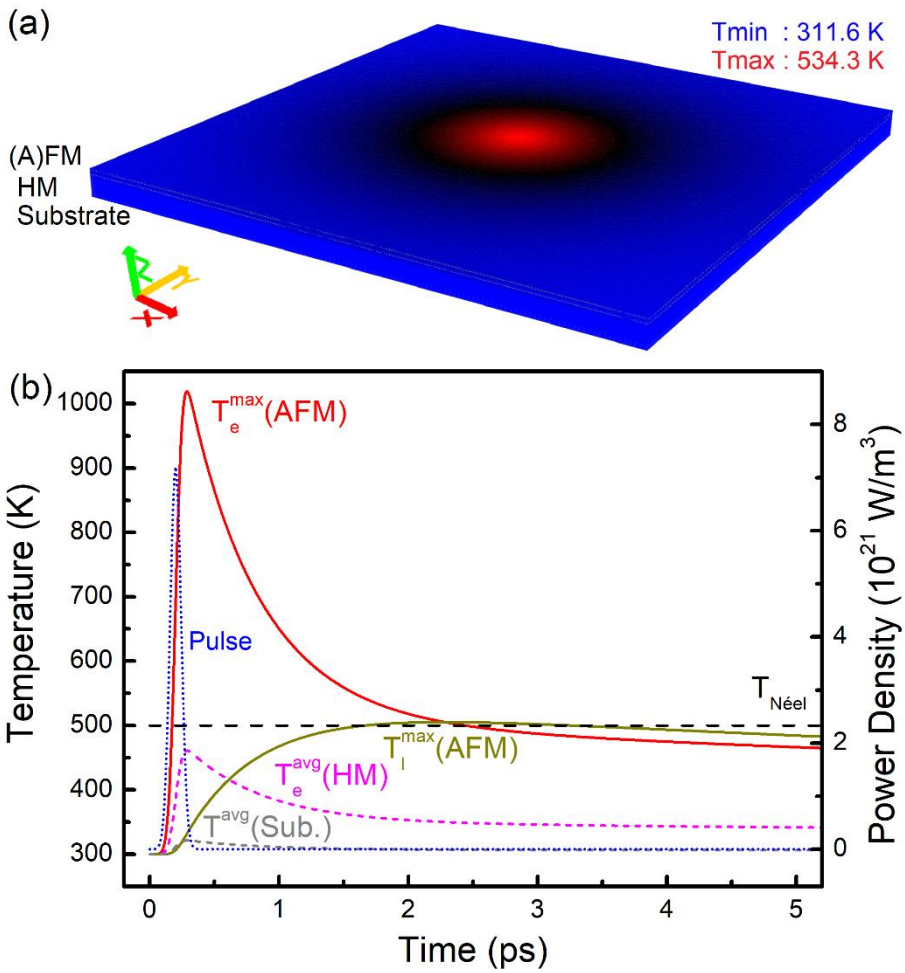
Exercise 28.1

In this exercise you will simulate the effect of a single laser pulse on a Co/Pt/SiO_2 trilayer, similar to that used in S. Lepadatu (2020) arXiv:2005.13238, when the electron temperature rises above the Curie temperature, and observe creation of Néel skyrmions, plotting the variation in topological charge magnitude as a function of time. The structure to simulate is shown in Figure 28.1. The topological charge is computed as:

$$Q = \frac{1}{4\pi}\int_A \mathbf{m}.\left(\frac{\partial \mathbf{m}}{\partial x}\times\frac{\partial \mathbf{m}}{\partial y}\right)dxdy$$

The topological charge takes on unit values for skyrmions, and may be computed in Boris using the **dp_topocharge** command.

Figure 28.1 – Trilayer structure similar to that used for Exercise 28.1, consisting of Co (2 nm) / Pt (8 nm) / SiO_2 (40 nm), showing (a) temperature during a Gaussian profile laser pulse, and (b) typical ultrafast laser pulse and temperature time dependence in the three layers: Co layer maximum electron and lattice temperatures, Pt layer average electron temperature and SiO_2 average temperature. Reproduced from S. Lepadatu (2020) arXiv:2005.13238.

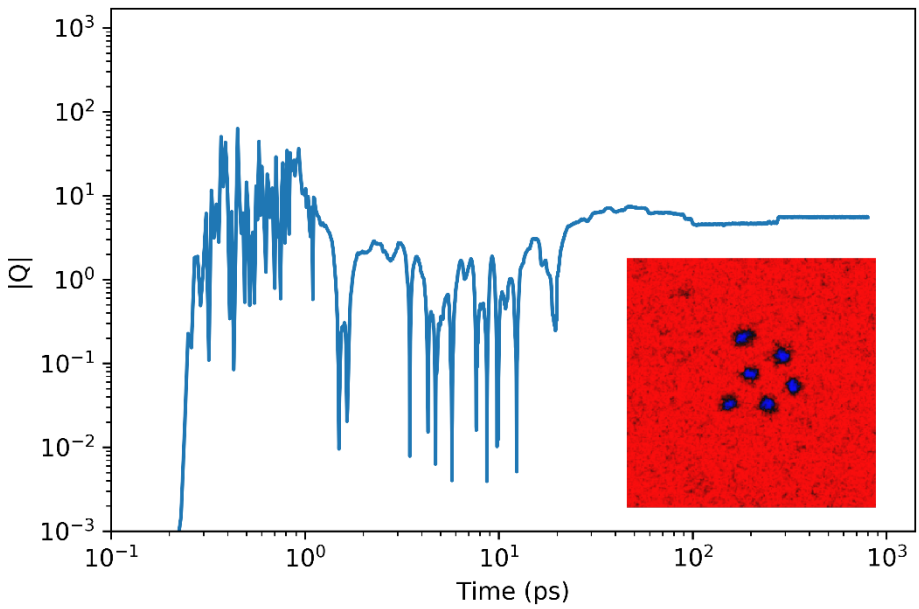


See the ufsky_creation.py script in the Examples/Tutorial 0 folder. This script sets up the simulation for this exercise from scratch. Study the script to understand what all the commands are used for (if needed look up the command help in the console or manual). Run this script several times and build a probability distribution of number of skyrmions created in each run. Is the computed probability distribution described by a Poisson counting distribution? What is the mean number of skyrmions created?

Hint: to obtain a reasonable probability distribution you will need to run the script at least 50 times – this skyrmion counting process can be automated, and will require up to 1h or 2h of simulation time depending on your workstation.

Further hint: the computed Q value with dp_topocharge will not be an integer for two reasons: i) stochasticity, ii) cellsize could be too large. However, rounding the Q value will result in a correct integer Q value. The other possibility is to turn off the stochasticity at the end of the simulation and allow the magnetisation to quickly relax before obtaining the Q value with dp_topocharge. With an in-plane cellsize of 1 nm this will result in a value very close to an integer (e.g. -3.98 for 4 skyrmions etc.).

Figure 28.2 – Topological charge magnitude as a function of time, computed for a run of Exercise 28.1. The resulting skyrmion state after 800 ps is shown in the inset, showing 6 skyrmions created (Q ≅ -6).



Tutorial 29 – Magneto-Optical Effect

With circularly polarised pulses the polarisation can be clockwise or anti-clockwise. Due to the circular polarisation of the laser pulse a strong perpendicular magneto-optical field is present, given by $H_{MO} = \sigma^{\pm} H_{MO}^{0} f_{MO}(\mathbf{r},t)\hat{\mathbf{z}}$. Here f_{MO} gives the spatial and temporal dependence of the laser pulse, H_{MO} (A/m) gives the strength of the magneto-optical field, and $\sigma^{\pm} = \pm 1$ its helicity. Thus for the Gaussian laser pulse from the previous Tutorial, the spatial and temporal dependence of the magneto-optical field is given by:

$$f_{MO}(\mathbf{r},t) = \exp\left(\frac{-|\mathbf{r}-\mathbf{r}_0|}{d^2/4\ln(2)}\right)\exp\left(\frac{-(t-t_0)^2}{t_R^{\,2}/4\ln(2)}\right)$$

In Boris you can enable the magneto-optical effect with the *moptical* module (**addmodule** *meshname moptical*).

The H_{MO} parameter appears as a material parameter (see **params**), named *Hmo*. Thus you can set the strength of the magneto-optical effect by setting the parameter value. You can also set the f_{MO} dependence as above by setting a material parameter variation for *Hmo* (**paramsvar**). Thus for the above example you need to set the spatial variation using a text equation set to:

*exp(-sqrt((x/Lx - 0.5)^2 + (y/Ly - 0.5)^2) / ((d0/Lx)^2/(4*ln(2)))) * exp(-(t-2*tau)^2/(tau^2/(4*ln(2))))*

Exercise 29.1

Based on the simulation file from Exercise 28.1, study the effect of a train of circularly polarised pulses with positive or negative helicities on skyrmion creation. Use a reduced laser power density which does not raise the temperature above the Curie temperature (set Q0 = 9e20 W/m^3). Write a Python script to apply 20 laser pulses with a 6 ps repetition period and strength of 40 MA/m. Compare the results after 30 pulses for the two helicities.

Tutorial 30 – Spin-Wave Dispersion

Similar to the method of simulating frequency-swept FMR (see tutorial on FMR first), we can simulate spin-wave dispersion by applying a sinc pulse which has not only a time dependence, but also a spatial dependence. In the xy plane the excitation field has the form:

$$H(t) = H_e sinc(k_c(x - x_0))sinc(k_c(y - y_0))sinc(2\pi f_c(t - t_0))$$

As before f_c is the frequency cut-off (Hz) with t_0 the temporal sinc pulse centre. To excite spin-waves with non-zero wave-vector we also need a spatial dependence for the sinc pulse. Here k_c is the wave-vector cut-off (rad/m), and (x_0, y_0) is the spatial sinc pulse centre. When obtaining the spin-wave dispersion, we need to consider the direction of the wave-vector, **k**. Instead of sampling the average magnetisation, we need to obtain a magentisation profile along a given direction. The direction in which we sample is the wave-vector direction **k**. You can obtain a magentisation profile using the **dp_getexactprofile** command. Also we need to sample the magnetisaiton at fixed steps, determined by the Nyquist criterion (as for temporal samping):

$$\Delta_S = \frac{\pi}{k_C} \quad (m)$$

We only need to sample one component of the magnetisation, and the spin wave dispersion is obtained by performing a 2D Fourier transform on the spatial-temporal 2D data, transforming it to wave-vector-frequency 2D space.

Consider a spin-wave waveguide as an elongated magnetic track. There are three possible configurations, determined by the direction of the bias field.

1) Bias field along length – this is called the backward volume configuration.
2) Bias field along thickness – this is called the forward volume configuration.
3) Bias field along width – this is called the surface spin wave configuration.

In all three cases the excitation and analysed magnetisation component need to be perpendicular to the bias field; for simplicity the analysed magnetisation component can be chosen to be along the excitation direction. When simulating the spin-wave dispersion it is important to choose the cell-size correctly, as the computed spin-wave dispersion will be inaccurate at larger wave-vector values if the cellsize is not small enough.

Exercise 30.1

Read through the reference IEEE Trans. Mag. 49, 524 (2013). Simulate the spin-wave dispersion as described in this reference for the three spin-wave configurations described above. Use periodic boundary conditions along the length only.

You should make the following modifications to the proposed problem:

1) Cellsize along length should be 1 nm; the specified 2 nm cellsize is too large and results in a large discrepancy between computations and analytical formulas at larger k values.
2) You should simulate for a total time of $2\times t_0$ as explained in the tutorial on FMR. The specified 5 ns simulation time for t_0 of 50 ps results in a very noisy spectrum. For this exercise it is suggested you use a t_0 value of 200 ps, although 100 ps is still fine. A value of 50 ps results in a coarse frequency spectrum.

3) The spatial samping interval should be 4 nm, not 2 nm, due to the Nyquist criterion. A 2 nm spatial sampling interval results in wasted wave-vector spectrum (try it!).

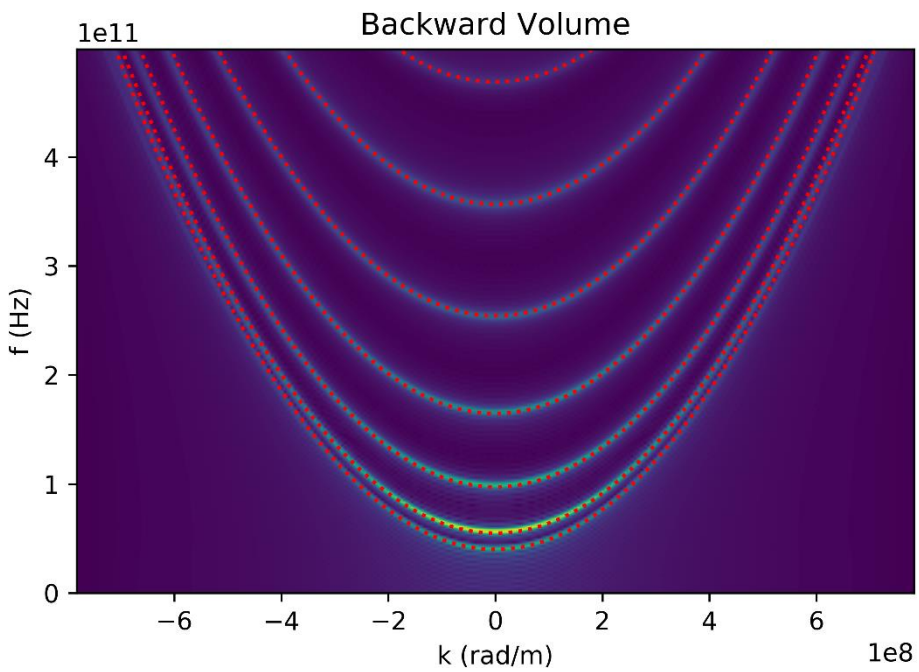
*Hint: use an Hequation stage set to 'H0, He * sinc(kc*(x-Lx/2))*sinc(kc*(y-Ly/2))*sinc(2*PI*fc*(t-t0)), 0' for the backward volume configuration.*

When analysing the spin-wave dispersion you can use the following formula:

$$w = w_n + w_M \frac{2A}{\mu_0 M_S^2} k^2 \quad (rad/s)$$

Here w_n is the resonance frequency (rad/s) for the nth spin-wave mode at k = 0, and may be extracted from the computed spectrum at k = 0, or alternatively you can use an analytical formula to predict it (not given here). The value w_M is given as γM_S, where $\gamma = \mu_0|\gamma_e|$ (2.212761569×10^5 mA/s).

Figure 30.1 – Spin-wave dispersion computed in Exercise 30.1 together with analytical predictions (even spin-wave modes excited n = 0, 2, 4, 6, 8, 10, 12). A damping value of 0.01 was used.



Tutorial 31 – Two-Sublattice Model

In a micromagnetics formulation we can study antiferromagnetic, ferrimagnetic, as well as binary ferromagnetic alloys using a two-sublattice model, where we consider magnetic orders of two sub-lattices A, B, and couple them using inter-lattice exchange stiffness terms. The two-sublattice stochastic LLB equation (sLLB) is given below:

$$\frac{\partial \mathbf{M}_i}{\partial t} = -\tilde{\gamma}_i \mathbf{M}_i \times \mathbf{H}_{eff,i} - \tilde{\gamma}_i \frac{\tilde{\alpha}_{\perp,i}}{M_i} \mathbf{M}_i \times \left(\mathbf{M}_i \times \left(\mathbf{H}_{eff,i} + \mathbf{H}_{th,i}\right)\right)$$
$$+ \gamma_i \frac{\tilde{\alpha}_{\parallel,i}}{M_i} \left(\mathbf{M}_i . \mathbf{H}_{\parallel,i}\right) \mathbf{M}_i + \boldsymbol{\eta}_{th,i} \quad (i = A, B)$$

The reduced gyromagnetic ratio is given by $\tilde{\gamma}_i = \gamma_i / \left(1 + \alpha_{\perp,i}^2\right)$, and the reduced transverse and longitudinal damping parameters by $\tilde{\alpha}_{\perp(\parallel),i} = \alpha_{\perp(\parallel),i} / m_i$, where $m_i(T) = M_i(T) / M_{S,i}^0$, with $M_{S,i}^0$ denoting the zero-temperature saturation magnetisation, and $M_i \equiv |\mathbf{M}_i|$. The damping parameters are continuous at T_N – the phase transition temperature – and given by:

$$\alpha_{\perp,i} = \alpha_i \left(1 - \frac{T}{3\left(\tau_i + \tau_{ij} m_{e,j} / m_{e,i}\right) \tilde{T}_N}\right), \quad T < T_N$$
$$\alpha_{\parallel,i} = \alpha_i \left(\frac{2T}{3\left(\tau_i + \tau_{ij} m_{e,j} / m_{e,i}\right) \tilde{T}_N}\right), \quad T < T_N$$
$$\alpha_{\perp,i} = \alpha_{\parallel,i} = \frac{2T}{3T_N}, \quad T \geq T_N$$

We denote $\tilde{T}_N$ the re-normalized transition temperature, given by:

$$\tilde{T}_N = \frac{2T_N}{\tau_A + \tau_B + \sqrt{\left(\tau_A - \tau_B\right)^2 + 4\tau_{AB}\tau_{BA}}}$$

The micromagnetic parameters τ_i and $\tau_{ij} \in [0, 1]$, are coupling parameters between exchange integrals and the phase transition temperature, such that $\tau_A + \tau_B = 1$ and $|J| = 3\tau k_B T_N$. Here J is the exchange integral for intra-lattice ($i = A,B$) and inter-lattice

($i,j = A,B$, $i \neq j$) coupling respectively. For a simple antiferromagnet we have $\tau_A = \tau_B = \tau_{AB} = \tau_{BA} = 0.5$. The normalised equilibrium magnetisation functions $m_{e,i}$ are obtained from the Curie-Weiss law as:

$$m_{e,i} = B\left[\left(m_{e,i}\tau_i + m_{e,j}\tau_{ij}\right)3\tilde{T}_N / T + \mu_i \mu_0 H_{ext} / k_B T\right],$$

where $B(x) = \coth(x) - 1/x$, and μ_i is the atomic magnetic moment. The magnetisation length is not constant, and can differ from the equilibrium magnetisation length, giving rise to a longitudinal relaxation field which includes both intra-lattice and inter-lattice contributions:

$$\mathbf{H}_{\|,i} = \left\{\frac{1}{2\mu_0\tilde{\chi}_{\|,i}}\left(1 - \frac{m_i^2}{m_{e,i}^2}\right) + \frac{3\tau_{ij}k_B T_N}{2\mu_0\mu_i}\left[\frac{\tilde{\chi}_{\|,j}}{\tilde{\chi}_{\|,i}}\left(1 - \frac{m_i^2}{m_{e,i}^2}\right) - \frac{m_{e,j}}{m_{e,i}}\left(\hat{\mathbf{m}}_i.\hat{\mathbf{m}}_j\right)\left(1 - \frac{m_j^2}{m_{e,j}^2}\right)\right]\right\}\mathbf{m}_i, \quad T < T_N$$

$$\mathbf{H}_{\|,i} = -\left\{\frac{1}{\mu_0\tilde{\chi}_{\|,i}} + \frac{3\tau_{ij}k_B T_N}{\mu_0\mu_i}\left[\frac{\tilde{\chi}_{\|,j}}{\tilde{\chi}_{\|,i}} - \frac{m_{e,j}}{m_{e,i}}\left(\hat{\mathbf{m}}_i.\hat{\mathbf{m}}_j\right)\right]\right\}\mathbf{m}_i, \quad T > T_N$$

Here $\hat{\mathbf{m}}_i = \mathbf{m}_i / m_i$, and the relative longitudinal susceptibility is $\tilde{\chi}_{\|,i} = \chi_{\|,i} / \mu_0 M_{S,i}^0$, where:

$$k_B T\tilde{\chi}_{\|,i} = \frac{\mu_i B_i'\left(1 - 3\tau_j\tilde{T}_N B_j' / T\right) + \mu_j 3\tau_{ij}\tilde{T}_N B_i' B_j' / T}{\left(1 - 3\tau_i\tilde{T}_N B_i' / T\right)\left(1 - 3\tau_j\tilde{T}_N B_j' / T\right) - \tau_{ij}\tau_{ji}B_i' B_j'\left(3\tilde{T}_N / T\right)^2}$$

and $B_i' \equiv B'_{m_{e,i}}\left[\left(m_{e,i}\tau_i + m_{e,j}\tau_{ij}\right)3\tilde{T}_N / T\right]$.

The direct exchange term includes the usual intra-lattice contribution, as well as homogeneous and non-homogeneous inter-lattice contributions, and is given by:

$$\mathbf{H}_{ex,i} = \frac{2A_i}{\mu_0 M_{e,i}^2}\nabla^2\mathbf{M}_i + \frac{4A_{h,i}}{\mu_0 M_{e,i}M_{e,j}}\mathbf{M}_j + \frac{A_{nh,i}}{\mu_0 M_{e,i}M_{e,j}}\nabla^2\mathbf{M}_j$$

The intra-lattice exchange stiffness A_i has the temperature dependence $A_i = A_i^0 m_{e,i}^2$, whilst the inter-lattice exchange stiffnesses have the temperature dependences $A_{h(nh),i} = A_{h(nh),i}^0 m_{e,i}m_{e,j}$.

Finally, the terms $\mathbf{H}_{th,i}$ and $\boldsymbol{\eta}_{th,i}$ are stochastic quantities with zero spatial, vector components, and inter-lattice correlations, and whose components follow Gaussian distributions with zero mean and standard deviations given respectively by:

$$H_{th,i}^{std.} = \frac{1}{\alpha_{\perp,i}} \sqrt{\frac{2k_B T\left(\alpha_{\perp,i} - \alpha_{\parallel,i}\right)}{\gamma_i \mu_0 M_{S,i}^0 V \Delta t}}$$

$$\eta_{th,i}^{std.} = \sqrt{\frac{2k_B T \alpha_{\parallel,i} \gamma_i M_{S,i}^0}{\mu_0 V \Delta t}}$$

Here V is the stochastic computational cellsize volume, and Δt is the stochastic time-step.

In Boris a mesh with the two-sublattice model may be added using the command:

addafmesh *meshname rectangle*

Default temperature dependences may be generated as for a ferromagnetic mesh with the command (same command as for a ferromagnetic mesh, hence the naming Curie):

curietemperature *value (meshname)*

Control of two-sublattice model meshes is exactly the same as for a ferromagnetic mesh, but the list of parameters and available modules is different (see modules and params). For a description of how the modules handle the two-sublattice model meshes see the chapter on Modules in the manual.

Some important parameters you need to control are:

1) Micromagnetic τ coupling factors. Set these using the tau command as:

   tau *tau11 tau22 tau12 tau21*

2) Atomic moments μ_A, μ_B. Set these using the atomicmoment command as:

atomicmoment *mu1 mu2 meshname*

Here *mu1* and *mu2* are the atomic moment values in units of the Bohr magneton, and meshname must be the name of an antiferromagnetic mesh. Note this command is also used to set the atomic moment for the LLB equation for a ferromagnetic mesh, which only takes a single *mu* value – this is the default behaviour, so giving the meshname parameter is important here.

3) Most magnetic parameters now have 2-sublattice values, so you can control them separately if needed. The two inter-lattice exchange stiffness coupling terms, *Ah* and *Anh*, only appear in two-sublattice model meshes.

When using the sLLB equation, by default the stochastic field generation time-step is the same as the time-step using for the equation evaluation. You may need to control this separately (set it to a larger value), and this is possible using the **setdtstoch** command – see the output of the **stochastic** command for an interactive display.

You can also set the stochastic cellsize value to be different than the magnetic cellsize value using the **scellsize** command.

Exercise 31.1 (Advanced)

The two-sublattice sLLB equation produces a distribution of magnetisation lengths on the two sub-lattices, m_A, m_B, which is expected to obey the following bi-variate Boltzmann probability distribution:

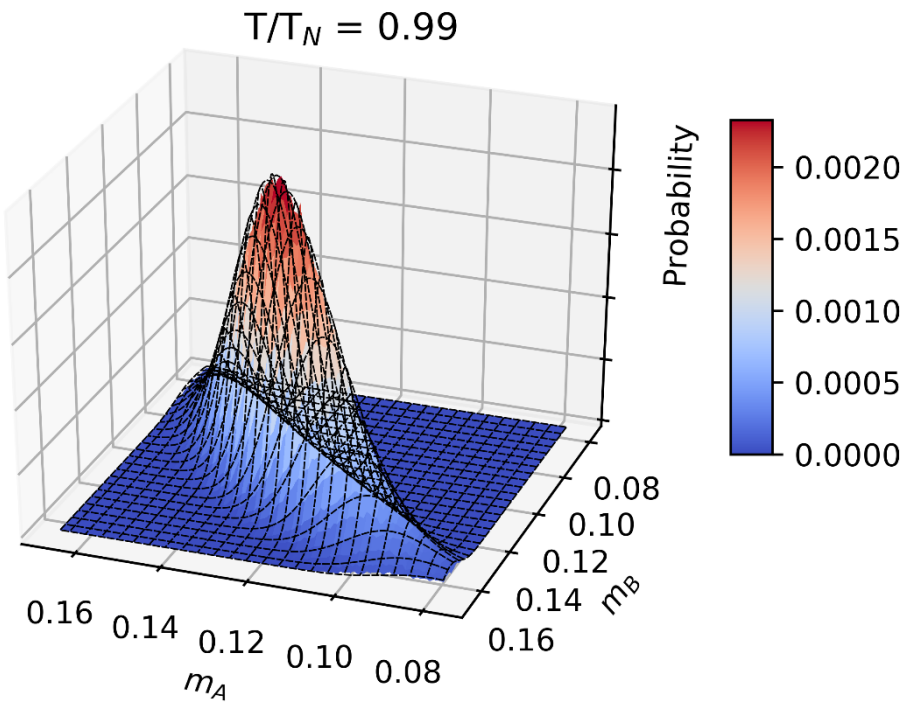
$$P_i(m_A, m_B) \propto m_i^2 \exp\left\{ -\frac{M_{S,i}^0 V}{4\mu_i m_{e,i} k_B T} \left[\frac{\left(m_i^2 - m_{e,i}^2\right)^2}{m_{e,i}} \frac{\left(\mu_i + 3\tau_{ij} k_B T_N \tilde{\chi}_{\parallel,j}\right)}{2\tilde{\chi}_{\parallel,i}} + \frac{\left(m_j^2 - m_{e,j}^2\right)}{m_{e,j}} 3\tau_{ij} k_B T_N m_i^2 \right] \right\}$$

$$(i, j = A, B, i \neq j)$$

Write a general-purpose Python script which tests the above equation against computed bi-variate probability distributions for any possible combination of T, T_N, τ_i, τ_{ij}, μ_i, and $M^0_{S,i}$ parameters. Test it for particular values (e.g. antiferromagnetic case).

Hint: there is a useful command built into Boris which computes a histogram for the two-sublattice model, namely ***dp_histogram2*** *– see help for this command (there's also a* ***dp_histogram*** *command which works for ferromagnetic meshes). See Figure 31.1 for a typical output you should obtain from your script.*

Figure 31.1 – Computed two-sublattice normalised magnetisation length probability distribution at $T/T_N = 0.99$ (colored surface) for an antiferromagnet in Exercise 31.1, compared with the bi-variate Boltzmann probability distribution prediction (wire-frame).



Exercise 31.2

Using the default antiferromagnetic mesh (**addafmesh**) in Boris, verify the Kittel formula for antiferromagnetic resonance is reproduced (see F. Keefer and C. Kittel, Phys. Rev. 85, 329 (1952)):

$$\frac{w}{\gamma} = H_0 + \sqrt{H_A(2H_E + H_A)}$$

Here H_A is the uniaxial anisotropy field along the bias field, $H_A = 2K_1/\mu_0 M_S$, where M_S is the magnetisation length on a sub-lattice, and H_E is the Weiss exchange field, given by $H_E = 4A_h/\mu_0 M_S$.

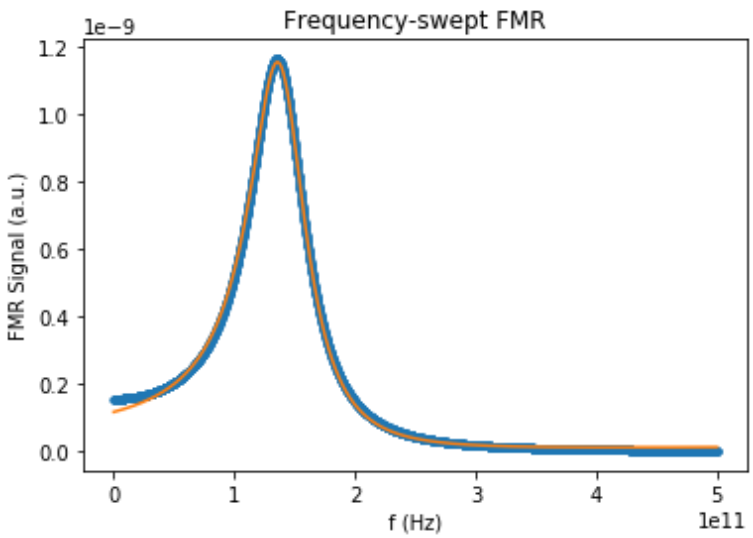
Hint: use the LLG equation, and adapt a previous exercise on frequency-swept FMR to simulate a frequency-swept FMR peak and obtain the resonance frequency. You should either use H_0 set to zero, or small bias field values as the above formula becomes inaccurate at large bias field values.

For the above exercise you should fit a Lorentz peak function with both symmetric and asymmetric components for a more accurate result:

$$f(x) = y_0 + S\frac{w + A(x - x_0)}{4(x - x_0)^2 + w^2}$$

Such a fitting procedure has already been built into Boris and can be accessed using the dp_**fitlorentz2** command.

Figure 31.2 – Antiferromagnetic resonance peak with fitted symmetric and asymmetric Lorentz peak function, verifying Kittel's formula in Exercise 31.2.



Tutorial 32 – Exchange Bias

The exchange bias field on a ferromagnetic layer from an antiferromagnet is given as:

$$\mathbf{H}_{ex} = \frac{J}{\mu_0 M_S t_F} \mathbf{m}_j$$

Here M_S and t_F are the saturation magnetisation and thickness of the ferromagnetic layer, and $\mathbf{m}_j$ is the exchange bias field direction from the antiferromagnet. This effect may be modelled in Boris using the *surfexchange* module, since the bilinear surface exchange field is given as:

$$\mathbf{H}_i = \frac{J_1}{\mu_0 M_S t_F} \mathbf{m}_j$$

Here $\mathbf{m}_j$ is the magnetisation direction on sub-lattice A of an interfacing antiferromagnetic mesh. Thus in order to model exchange bias in Boris you need an antiferromagnetic mesh (**addafmesh**), and a ferromagnetic mesh (**addmesh**) in contact with it, and they both need the *surfexchange* module enabled, remembering it is the top mesh (in order of z axis direction) which sets the *J1* value.

Exercise 32.1

Simulate the exchange bias effect in a Fe 2nm thin film using a generic antiferromagnetic material, 10 nm thick (use the **addafmesh** command to create a default antiferromagnetic mesh). Enable in-plane uniaxial anisotropy for the antiferromagnetic material (x axis), and set the antiferromagnetic sub-lattice A magnetisation direction to result in a bias effect towards the +ve side.

You can add the Fe material from the materials database, and you should enable cubic anisotropy for it. You can use periodic boundary conditions in the xy plane, simulating an area of 320 × 320 nm^2. Set the J_1 surface exchange constant to 0.2 mJ/m^2.

Tutorial 33 – Magneto-Elastic Effect

The magneto-elastic effect may be simulated in Boris using the *melastic* module. The magneto-elastic effect can be included for a cubic crystal using a strain tensor. The strain tensor is given as:

$$\mathbf{S} = \begin{pmatrix} \varepsilon_{xx} & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{xy} & \varepsilon_{yy} & \varepsilon_{yz} \\ \varepsilon_{xz} & \varepsilon_{yz} & \varepsilon_{zz} \end{pmatrix}.$$

Here we define the diagonal strain vector as $\mathbf{S}_d = (\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{zz})$, and off-diagonal strain vector as $\mathbf{S}_{od} = (\varepsilon_{yz}, \varepsilon_{xz}, \varepsilon_{xy})$. The strain tensor can have a spatial dependence, and currently needs to either be loaded from ovf2 files (strain computed with an external package), or alternatively a displacement vector field can be loaded (using ovf2 files, computed externally), and the strain tensor computed as:

$$\mathbf{S}_d = \left(\frac{\partial u_x}{\partial x}, \frac{\partial u_y}{\partial y}, \frac{\partial u_z}{\partial z} \right)$$

$$\mathbf{S}_{od} = \frac{1}{2}\left(\frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y}, \frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x}, \frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right)$$

In the simplest case a uniform stress may be applied which results in a constant strain with zero off-diagonal terms. In a future version an elastostatics solver as, well as a dynamical elastic solver will be included.

From the strain tensor, for a cubic crystal with orthogonal axes $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$, and magneto-elastic constants B1, B2, we have the following diagonal and off-diagonal energy density terms:

$$\varepsilon_{mel,d} = B_1\left[(\mathbf{m}.\mathbf{e}_1)^2(\mathbf{S}_d.\mathbf{e}_1) + (\mathbf{m}.\mathbf{e}_2)^2(\mathbf{S}_d.\mathbf{e}_2) + (\mathbf{m}.\mathbf{e}_3)^2(\mathbf{S}_d.\mathbf{e}_3)\right]$$

$$\varepsilon_{mel,od} = 2B_2\left[(\mathbf{m}.\mathbf{e}_1)(\mathbf{m}.\mathbf{e}_2)(\mathbf{S}_{od}.\mathbf{e}_3) + (\mathbf{m}.\mathbf{e}_1)(\mathbf{m}.\mathbf{e}_3)(\mathbf{S}_{od}.\mathbf{e}_2) + (\mathbf{m}.\mathbf{e}_2)(\mathbf{m}.\mathbf{e}_3)(\mathbf{S}_d.\mathbf{e}_1)\right]$$

To set a uniform stress use the command (similar to the **setfield** command):

setstress *magnitude polar azimuthal (meshname)*

A uniform stress may also be set using the *Sunif* stage. When applying a uniform stress, **T**, the strain tensor is generated based on the material Young's modules and Poisson ratio as:

$$\mathbf{S} = \frac{1}{Y_m}\begin{pmatrix} 1 & -\nu & -\nu \\ -\nu & 1 & -\nu \\ -\nu & -\nu & 1 \end{pmatrix}\mathbf{T}$$

Young's moduls and Poisson ratio are available as material parameters as *Ym* and *Pr* respectively. The magneto-elastic constants are available as material parameters as *MEc* (2-component parameter for B1 and B2 respectively). The orthogonal axes $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$ are set by the magento-crystalline anisotropy axes (ea1 is **e1**, ea2 is **e2** and $\mathbf{e}_3 = \mathbf{e}_1 \times \mathbf{e}_2$.

When applying a uniform stress the mesh origin (0, 0, 0) is a fixed point, and no shear strain or physical displacement is allowed. This means a positive stress value along the x axis results in elongation, whilst a negative stress value along the x axis results in compression.

You can run computations with a non-uniform strain, but in the current version this must be computed externally. There are two ways of setting a non-uniform strain. The simplest method is to compute the displacement vector map **u** externally and save it into a ovf2 file. This can then be loaded into the currently focused mesh using the command (must have *melastic* module enabled):

loadovf2disp *filename*

After the displacement map is loaded the strain tensor used for computations is obtained using the equations above.

You can also load the strain tensor directly, but this requires saving the diagonal and off-diagonal components in two separate ovf2 files in vector data format. The diagonal file will then contain vector data with strain components xx, yy, zz, and the off-diagonal file will contain vector data with strain components yz, xz, xy (in this order). The ovf2 files can then be loaded as:

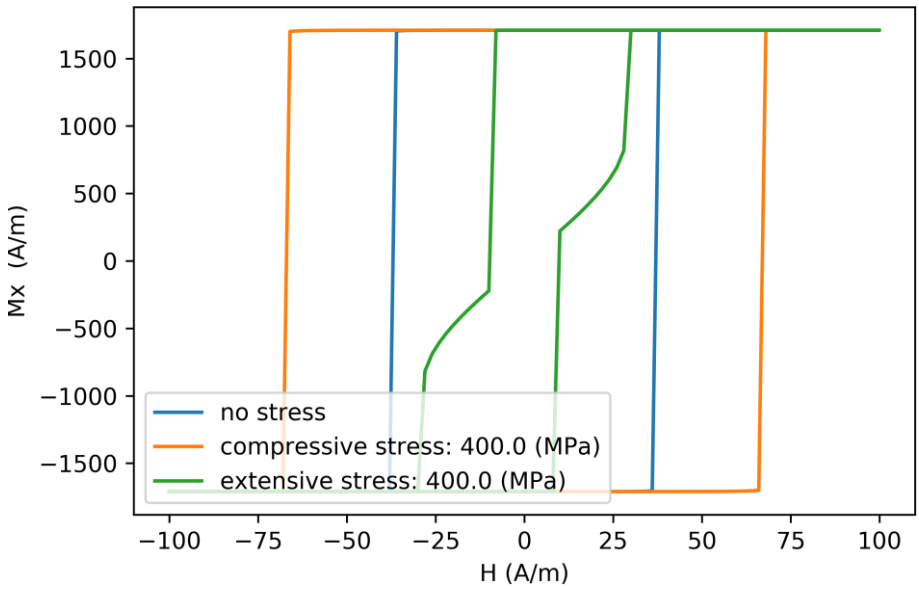
loadovf2strain *filename_diag filename_odiag*

With the *melastic* module enabled, there is a separate cellsize for the strain tensor, controlled using the **mcellsize** command, and this should be set before loading the externally computed strain or displacement.

Exercise 33.1

Simulate hysteresis loops for a 5 nm thick Fe thin film (found in materials database) with cubic anisotropy and *melastic* module enabled, along the x axis. Simulate three cases: i) no strain, ii) compressive stress along the x axis of 100 MPa, iii) extensive stress along the x axis of 100 MPa. Explain the differences between the 3 curves.

Figure 33.1 – Hysteresis loops computed in Exercise 33.1 for a Fe thin film, with and without mechanical stress.



Tutorial 34 – Atomistic Modelling

This is a placeholder.

The current version has a simple cubic atomistic mesh (**addameshcubic**), which implements a number of atomistic modules (Heisenberg exchange, DM and iDM exchange, uniaxial and cubic anisotropies, Zeeman, dipole-dipole interaction, as well as demagnetising fields obtained by computing magnetisation from atomic moments, LLG and stochastic LLG; the heat and moptical modules are also enabled). Some initial testing has already been done (e.g. computation of Curie temperature), but this area of the software is set to be significantly expanded in the next version release (bcc, fcc, hcp atomistic meshes, as well as full integration with micromagnetic meshes (multi-layered demagnetisation and surface exchange) for multi-scale computations; for this reason this area of the code has not been documented yet.

User-Defined Text Equations

Arbitrary text equations may be supplied by the user using fundamental functions, a number of special functions, and mathematical operators as detailed below. These equations are evaluated efficiently at run-time every iteration. Text equations use a number of reserved variables depending on the context, including *x*, *y*, *z* to define spatial variation, *t* to define temporal dependence, *T* to define temperature dependence, and also use a number of reserved constants as listed below. Both scalar and vector text equations may be supplied as appropriate. The contexts in which text equations may be used are:

1. Setting stage values

The available stage types are: i) *Hequation* (set external field using a vector text equation), ii) *Vequation* (set electrode potential drop using a scalar text equation), iii) *Iequation* (set ground electrode current using a scalar text equation), iv) *Tequation* (set mesh base temperature using a scalar text equation – applicable when heat module disabled), v) *Qequation* (set heat source in heat equation using a scalar equation).

Reserved variables:
x, *y*, *z* (spatial coordinates in meters, relative to mesh where used), *t* (stage time in seconds).

Defined constants:
Lx, *Ly*, *Lz* (mesh dimensions in meters), *Tb* (mesh base temperature), *Ss* (stage step).

Example: *0, 0, He * sinc(kc*(x-Lx/2))*sinc(kc*(y-Ly/2))*sinc(2*PI*fc*(t-t0)),*
The above example is used with *Hequation* (vector equation), and applies a spatial and temporal sinc pulse at the centre of the mesh with time centre at t0, for the z field component. The constants *He*, *kc*, *fc*, *t0* are defined by the user – see below.

2. Parameter temperature dependence

Any material parameter for which a temperature dependence is allowed (see output of **paramstemp** command), may be assigned a temperature dependence defined using a text equation. To set this either use the interactive console, or use the following command:

setparamtempequation *meshname paramname text_equation*

Reserved variables:

T (temperature, either mesh base temperature if *heat* module not enabled, or the local temperature if *heat* module is enabled – in the latter case the temperature can be non-uniform and the parameter value is evaluated according to the local computational cell temperature).

Defined constants:

Tb (mesh base temperature), *Tc* (set mesh Curie (or Néel) temperature).

Example: *me(T/Tc)^2*

The above example applies a squared Curie-Weiss scaling relation (me), for temperatures ranging from 0 to *Tc*.

3. Parameter spatial variation

Any material parameter for which a spatial variation dependence is allowed (see output of **paramsvar** command), may be assigned a spatial and temporal dependence defined using a text equation. To set this either use the interactive console, or use the following command:

setparamvar *meshname paramname equation text_equation*

Note in the above command the field "equation" specifies the type of generator used to generate a spatial variation and must be inputted literally as above.

Reserved variables:
x, *y*, *z* (spatial coordinates in meters, relative to mesh where used), *t* (stage time in seconds).

Defined constants:
Lx, *Ly*, *Lz* (mesh dimensions in meters).

Example: *exp(-(x-Lx/2)^2/Sx) * exp(-(y-Ly/2)^2/Sy) / (exp(0)^2)*
The above example applies a Gaussian spatial variation scaling in the xy plane at the centre of the mesh.

Functions

Allowed functions in text equations are given below.

Function Name	Description
sin	Fundamental sin function.
sinc	sinc(x) = sin(x) / x
cos	Fundamental cos function.
tan	tan(x) = sin(x) / cos(x)
sinh	sinh(x) = [exp(x) - exp(-x)] / 2
cosh	cosh(x) = [exp(x) + exp(-x)] / 2
tanh	tanh(x) = sinh(x) / cosh(x)
sqrt	Square root.
exp	Fundamental exp function.
asin	Inverse sin function.
acos	Inverse cos function.
atan	Inverse tan function.
asinh	Inverse hyperbolic sin function.
acosh	Inverse hyperbolic cos function.
atanh	Inverse hyperbolic tan function.
ln	Natural logarithm.
log	Logarithm base 10.
abs	Modulus function.
step	Step function: step(x) = 0 for x < 0, = 1 for x>= 0.
swav	Square wave function with period 2π, s.t. swav(0+) = +1, swav(π+) = -1.
twav	Triangular wave function with period 2π, s.t. twav(0) = +1, twav(π) = -1.
me	Normalised Curie-Weiss law.
chi	Normalised relative longitudinal susceptibility.
me1	Normalised Curie-Weiss law for sub-lattice A.
me2	Normalised Curie-Weiss law for sub-lattice B.
chi1	Normalised relative longitudinal susceptibility for sub-lattice A.
chi2	Normalised relative longitudinal susceptibility for sub-lattice B.
alpha1	Transverse damping temperature dependence.
alpha2	Longitudinal damping temperature dependence.

Any nested combination of functions is allowed.

Special equation expanders

1. *sum*

This is the sum function, with format given as: *sum(i;low;high;func(<i>))*.
Thus *func(i)* is summed for *i* ranging from integers *low* to *high* inclusive; note the format *i* must appear in *func*, namely *<i>*. The variable named *i* can be changed to a different string literal but must not clash with any equation variables, or reserved names. For example the expression:

*sum(j;0;10;sin(2*PI*<j>*x))*, evaluates to: $\sum_{j=0}^{10} \sin(2\pi jx)$

Nested *sum* functions are also allowed, and may be combined with any of the reserved function names in the table above. Note that using excessive limits (e.g. integer range over 1000) in the sum function will result in very slow evaluation times and should be avoided, especially if the evaluation is to be performed in every computational mesh cell.

Allowed mathematical operators

Operator Symbol (in precedence order)	**Description**
^	Exponentiation operator.
/	Division operator.
*	Multiplication operator.
-	Subtraction operator.
+	Addition operator.

All functions, variables, and constants must be linked by an operator, assumed multiplication is not allowed, e.g. *2*sin(x)* is a valid equation, *2sin(x)* is not a valid equation in the current program version.

Bracketing

Only round brackets are allowed, (,), which must appear in equal numbers.

Numerical constants

Numbers may be specified in floating point or scientific format.

Reserved constants

A number of pre-defined numerical constants are available. Since the names in the table below are reserved, they must not clash with any user defined constants.

Symbol	Description
PI	3.14159265358979323846264338 33
mu0	Free space permeability: 4*PI*10^{-7} (N/A^2)
muB	Bohr magneton: 9.27400968e-24 (Am^2)
ec	Electron charge: 1.60217662e-19 (C)
hbar	Reduced Planck constant: 1.054571817e-34 (m^2kg/s)
kB	Boltzmann constant: 1.3806488e-23 (m^2kg/s^2K)
gamma	Gyromagnetic ratio modulus: 2.212761569e5 (m/As)

User defined constants

Any number of user defined constants, named using alphanumeric strings, may be given, with the restriction they must not clash with any of the reserved names in the tables above, or equation variables. Equation constants may be given as:

equationconstants *name value*

Other related commands are **delequationconstant**, and **clearequationconstants**.

Vector and scalar equations

Scalar equation have a single component which must conform to the above rules. Vector equations with 3 components are allowed where appropriate, and must be specified using comma separators as: *component1*, *component2*, *component3*. Here the three components are scalar equations.

Working with OVF2 Files

Data may be loaded and saved using the OVF2 file format introduced in OOMMF. This allows setting magnetic shapes programmatically, as well as importing and exporting data to other software which support this file format.

OVF2 files may also be used to save mesh data numerically, not only for magnetisation, but for any mesh quantity which may be displayed on screen, including material parameter spatial variation. The latter may also be set programmatically using OVF2 files, via the ovf2 spatial variation generator (see below).

OVF2 files allow scalar or vector formats, and both are handled in Boris. Data may be saved in natural text format (*text*), single precision binary (*bin4*), or double precision binary (*bin8*).

<u>Commands which handle OVF2 files</u>

loadovf2mag *(renormalize_value) (directory/)filename*

Load an OOMMF-style OVF 2.0 file containing magnetisation data, into the currently focused mesh (which must be ferromagnetic), mapping the data to the current mesh dimensions. By default the loaded data will not be renormalized: renormalize_value = 0. If a value is specified for renormalize_value, the loaded data will be renormalized to it (e.g. this would be an Ms value).

saveovf2mag *(n) (data_type) (directory\)filename*

Save an OOMMF-style OVF 2.0 file containing magnetisation data from the currently focused mesh (which must be ferromagnetic). You can normalize the data to Ms0 value by specifying the n flag (e.g. saveovf2mag n filename) - by default the data is not normalized. You can specify the data type as data_type = bin4 (single precision 4

bytes per float), data_type = bin8 (double precision 8 bytes per float), or data_type = text. By default bin8 is used.

saveovf2 *(data_type) (directory/)filename*

Save an OOMMF-style OVF 2.0 file containing data from the currently focused mesh. You can specify the data type as data_type = bin4 (single precision 4 bytes per float), data_type = bin8 (double precision 8 bytes per float), or data_type = text. By default bin8 is used.

setparamvar *meshname paramname ovf2 filename*

Set the named parameter, *paramname*, spatial dependence for the named mesh, *meshname*, using an ovf2 file with *filename*, located in current working directory. The type of data in the OVF2 file must match the expected format of the parameter (scalar or vector). Once you've loaded the ovf2 file you can display the set spatial variation by selecting the *ParamVar* option under **display**, and clicking on the required parameter under **paramsvar**.

The provided NetSocks.py module used for Python scripts, provides a convenient method to handle OVF2 files: Write_OVF2, which allows writing a Python list into an OVF2 file. An example of programmatically setting a magnetic shape using an OVF2 file generated in a Python script is given in Tutorial 0.

saveovf2param *(data_type) (meshname) paramname (directory/)filename*

Save an OOMMF-style OVF 2.0 file containing the named parameter spatial variation data from the named mesh (currently focused mesh if not specified). You can specify the data type as data_type = bin4 (single precision 4 bytes per float), data_type = bin8 (double precision 8 bytes per float), or data_type = text. By default bin8 is used.

Materials Database

Material definitions can be saved in a materials database. This includes base material parameter values. The default database is called BorisMDB.txt. You can see this using the command:

materialsdatabase

The default materials database can be updated from a shared database stored on a server. The shared database can be seen at: https://boris-spintronics.uk/online-materials-database. To update your local BorisMDB database using the latest material parameter definitions, use the command:

updatemdb

You can also switch to an alternative custom database using the **materialsdatabase** command. To add a new computational mesh with given material parameters you can use the **addmaterial** command. The type of material will determine the type of computational mesh generated. For example the *ferromagnetic* type will generate a computational mesh with LLG/LLB solvers enabled. The *conductor* mesh type will generate a computational mesh with only the transport and heat solvers enabled, while the *insulator* mesh will only have the heat solver enabled (e.g. a substrate material).

Users can also send in entries to be added to the centrally stored database. This can be done using the **requestmdbsync** command. Before sending entries, you must properly format the material entry. The procedure is described as follows.

1. Add a new entry to your local BorisMDB file.

   Suppose you want to enter a new ferromagnetic material. First create a ferromagnetic mesh in Boris (**addmesh**), and set as many parameter values as possible. Next, add the entry to your local BorisMDB file using:

addmdbentry *meshname (materialname)*

Here meshname is the name of the mesh as it appears in Boris (the one you created using **addmesh**), and *materialname* is the name of the material, or entry, you want to create, if different.

2. Edit the material description fields in BorisMDB.txt

There are 5 description fields for the entry:

Name, Formula, Type, Description, Contributor

Name is the name of your new material entry, which should not already be in the shared online database. This should already be filled.

Formula is the symbolic formula for the material. Make sure to fill this.

Type is already filled for you, and is the type of computational mesh for which the material applies.

Description should have a very brief description for the material entry, with any useful information. Make sure to fill this.

Contributor is the name of the entry contributor; leave as N/A if you don't want to specify this.

There is another field called *State*. This specifies if the entry was taken from the online database (SHARED) or if it's a new user-created entry (LOCAL). You don't need to change this.

3. Set references for the parameters

   After each parameter there is a column called DOI. This must hold a DOI reference for where the material parameter value was taken from, or derived. You can leave it as N/A only if not applicable, but in most cases should be properly referenced.

4. Unspecified material parameters

   If you cannot reasonably give an entry for a material parameter value then override it as “N/A”.

5. Send in the entry

   After properly formatting the entry, you can upload it to a holding database using the **requestmdbsync** command:

   requestmdbsync *materialname (email)*

   Materialname is the name of the material you’ve just created. If you specify an email address you will receive feedback about whether the entry was added to the shared database or not – the entry will be verified for validity before being added to the online materials database. Once entered there, it will be visible at https://boris-spintronics.uk/online-materials-database, and other users can update their databases with it.

Differential Equations

This section outlines the magnetization dynamics equations solved as selected using the **ode** command. For descriptions of parameters used see the Material Parameters section. To set an equation to solve and evaluation method use the **setode** command as **setode** *equation evaluation*, e.g. **setode** *LLG-STT RK4*. In a Python script you can set this as *ns.setode('equation', 'evaluation')*. For fixed time step evaluation methods you can set the time step as **setdt** *value*. For a list of available equations and evaluation methods see below.

A number of evaluation methods are available for the magnetization dynamics equations. These are fixed step methods *Euler* (1st order), trapezoidal Euler (*TEuler* – 2nd order) and Runge-Kuta (*RK4* - 4th order). Adaptive time-step methods are the adaptive Heun (*AHeun* – 2nd order), the multi-step Adams-Bashforth-Moulton (*ABM* – 2nd order), Runge-Kutta-Bogacki-Shampine (*RK23* – 3rd order with embedded 2nd order error estimator), Runge-Kutta-Fehlberg (*RKF45* – 4th order with embedded 5th order error estimator), Runge-Kutta-Cash-Karp (*RKCK45* – 4th order with embedded 5th order error estimator), and Runge-Kutta-Dormand-Prince (*RKDP54* – 5th order with embedded 4th order error estimator). For static problems a steepest descent solver is available, *SDesc*, using Barzilai-Borwein stepsize selection formulas.

<u>Landau-Lifshitz-Gilbert (*LLG*)</u>

The normalised LLG equation in implicit form is given by:

$$\frac{\partial \mathbf{m}}{\partial t} = -\gamma \mathbf{m} \times \mathbf{H} + \alpha \mathbf{m} \times \frac{\partial \mathbf{m}}{\partial t}$$

Here $\gamma = \mu_0 g_{rel} |\gamma_e|$, where $\gamma_e = -g\mu_B / \hbar$ is the electron gyromagnetic ratio and g_{rel} is a relative g-factor (g_{rel} = 1 by default giving γ = 2.212761569×10^5 m/As).

In explicit form the normalised LLG equation is given by:

$$\frac{\partial \mathbf{m}}{\partial t} = -\frac{\gamma}{1+\alpha^2}\mathbf{m}\times\mathbf{H} - \frac{\alpha\gamma}{1+\alpha^2}\mathbf{m}\times\mathbf{m}\times\mathbf{H}$$

Two-Sublattice Landau-Lifshitz-Gilbert (*LLG*)

In antiferromagnetic (or ferrimagnetic, or binary ferromagnetic alloys) meshes a two-sublattice model is used, where the two sub-lattices are labelled A, B, and each have an LLG equation set. Coupling between the equations is done through the effective fields, in particular the exchange field, but also the demagnetizing field. Each sub-lattice has its own set of simulation parameters. In implicit and normalised form this is given by:

$$\frac{\partial \mathbf{m}_i}{\partial t} = -\gamma_i \mathbf{m}_i \times \mathbf{H}_i + \alpha_i \mathbf{m}_i \times \frac{\partial \mathbf{m}_i}{\partial t}, \quad (i = A, B)$$

Landau-Lifshitz-Gilbert with Spin-Transfer Torques (*LLG-STT*)

The LLG equation can be complemented by Zhang-Li spin-transfer torques. In implicit form this becomes:

$$\frac{\partial \mathbf{m}}{\partial t} = -\gamma\mathbf{m}\times\mathbf{H} + \alpha\mathbf{m}\times\frac{\partial \mathbf{m}}{\partial t} + (\mathbf{u}.\nabla)\mathbf{m} - \beta\mathbf{m}\times(\mathbf{u}.\nabla)\mathbf{m}$$

The spin-drift velocity **u** is given by:

$$\mathbf{u} = \mathbf{J}\frac{Pg\mu_B}{2eM_s}\frac{1}{1+\beta^2}$$

The LLG-STT equation in explicit form is given by:

$$\frac{\partial \mathbf{m}}{\partial t} = -\frac{\gamma}{1+\alpha^2}\mathbf{m}\times\mathbf{H} - \frac{\alpha\gamma}{1+\alpha^2}\mathbf{m}\times(\mathbf{m}\times\mathbf{H}) +$$
$$\frac{1}{1+\alpha^2}\left[(1+\alpha\beta)(\mathbf{u}.\nabla)\mathbf{m} - (\beta-\alpha)\mathbf{m}\times(\mathbf{u}.\nabla)\mathbf{m} - \alpha(\beta-\alpha)(\mathbf{m}.(\mathbf{u}.\nabla)\mathbf{m})\mathbf{m}\right]$$

Two-Sublattice Landau-Lifshitz-Gilbert with Spin-Transfer Torques (*LLG-STT*)

In implicit and normalised form this is given by:

$$\frac{\partial \mathbf{m}_i}{\partial t} = -\gamma_i \mathbf{m}_i \times \mathbf{H}_i + \alpha_i \mathbf{m}_i \times \frac{\partial \mathbf{m}_i}{\partial t} + (\mathbf{u}_i.\nabla)\mathbf{m}_i - \beta \mathbf{m}_i \times (\mathbf{u}_i.\nabla)\mathbf{m}_i, \quad (i = A, B)$$

Here:

$$\mathbf{u}_i = \mathbf{J}\frac{Pg\mu_B}{2eM_{s,i}}\frac{1}{1+\beta^2}, \quad (i = A, B)$$

Landau-Lifshitz-Bloch (*LLB*)

For non-zero temperature simulations the LLB equation should be used and in implicit form is given by (un-normalised):

$$\frac{\partial \mathbf{M}}{\partial t} = -\gamma \mathbf{M}\times\mathbf{H} + \frac{\tilde{\alpha}_\perp}{|\mathbf{M}|}\mathbf{M}\times\frac{\partial \mathbf{M}}{\partial t} + \frac{\gamma\tilde{\alpha}_\parallel}{|\mathbf{M}|}(\mathbf{M}.\mathbf{H})\mathbf{M}$$

Here for T < T_C (T_C is the Curie temperature) $\alpha_\perp = \alpha(1 - T/3T_C)$, $\alpha_\parallel = \alpha 2T/3T_C$ and $\tilde{\alpha}_\perp = \alpha_\perp / m$, $\tilde{\alpha}_\parallel = \alpha_\parallel / m$, where *m* is the magnetization length normalised to its zero temperature value, i.e. $m = |\mathbf{M}| / M_S^0$. For T > T_C $\alpha_\perp = \alpha_\parallel = 2T/3T_C$.

The effective field **H** must be complemented by a longitudinal susceptibility field given by (which due to vector cross products only affects the longitudinal torque term):

$$\mathbf{H}_l = \begin{cases} \left(1 - \dfrac{m^2}{m_e^2}\right)\dfrac{\mathbf{m}}{2\mu_0 \tilde{\chi}_{\parallel}}, & T \le T_C \\ -\dfrac{\mathbf{m}}{\mu_0 \tilde{\chi}_{\parallel}} & , \quad T > T_C \end{cases}$$

The field and temperature-dependent equilibrium magnetization, m_e, is given by:

$$m_e(T) = B\left[m_e \frac{3T_C}{T} + \frac{\mu\mu_0 H_{ext}}{k_B T} \right],$$

where B(x) = coth(x) – 1/x is the Langevin function, μ is the atomic moment.

The relative longitudinal susceptibility is given by (units 1/T), where $\chi_{\parallel}$ is the longitudinal susceptibility (unitless):

$$\tilde{\chi}_{\parallel}(T) = \frac{\mu}{k_B T} \frac{B'(x)}{1 - B'(x)(3T_C / T)} = \chi_{\parallel}(T) / \mu_0 M_S^0, \quad with \;\; x = m_e 3T_C / T$$

Further, we have the temperature dependences $M_S(T) = M_S^0 m_e(T)$, and $A(T) = A_0 m_e^2(T)$ for the exchange stiffness.

In explicit form the LLB equation becomes:

$$\frac{\partial \mathbf{M}}{\partial t} = -\frac{\gamma}{1+\alpha_\perp^2}\mathbf{M} \times \mathbf{H} - \frac{\tilde{\alpha}_\perp \gamma}{1+\alpha_\perp^2}\frac{1}{|\mathbf{M}|}\mathbf{M} \times (\mathbf{M} \times \mathbf{H}) + \frac{\gamma \tilde{\alpha}_{\parallel}}{|\mathbf{M}|}(\mathbf{M}.\mathbf{H})\mathbf{M}$$

Two-Sublattice Landau-Lifshitz-Bloch (*LLB*)

In explicit and un-normalised form this is given by:

$$\frac{\partial \mathbf{M}_i}{\partial t} = -\tilde{\gamma}_i \mathbf{M}_i \times \mathbf{H}_{eff,i} - \tilde{\gamma}_i \frac{\tilde{\alpha}_{\perp,i}}{M_i} \mathbf{M}_i \times \left(\mathbf{M}_i \times \mathbf{H}_{eff,i}\right) + \gamma_i \frac{\tilde{\alpha}_{\parallel,i}}{M_i} \left(\mathbf{M}_i . \mathbf{H}_{\parallel,i}\right) \mathbf{M}_i, \quad (i = A, B)$$

Here $\tilde{\gamma}_i = \gamma_i / \left(1 + \alpha_{\perp,i}^2\right)$ and $M_i \equiv |\mathbf{M}_i|$. As before $\tilde{\alpha}_{\perp(\parallel),i} = \alpha_{\perp(\parallel),i} / m_i$, where $m_i(T) = M_i(T) / M_{S,i}^0$ with $M_{S,i}^0$ denoting the zero-temperature saturation magnetisation. The damping parameters are continuous at T_N – the phase transition temperature – and given by:

$$\alpha_{\perp,i} = \alpha_i \left(1 - \frac{T}{3\left(\tau_i + \tau_{ij} m_{e,j} / m_{e,i}\right) \tilde{T}_N}\right), \quad T < T_N$$

$$\alpha_{\parallel,i} = \alpha_i \left(\frac{2T}{3\left(\tau_i + \tau_{ij} m_{e,j} / m_{e,i}\right) \tilde{T}_N}\right) \quad , \quad T < T_N$$

$$\alpha_{\perp,i} = \alpha_{\parallel,i} = \frac{2T}{3T_N} \quad , \quad T \geq T_N$$

Here we denote $\tilde{T}_N$ the re-normalized transition temperature, given by:

$$\tilde{T}_N = \frac{2T_N}{\tau_A + \tau_B + \sqrt{\left(\tau_A - \tau_B\right)^2 + 4\tau_{AB}\tau_{BA}}}$$

The micromagnetic parameters τ_i and $\tau_{ij} \in [0, 1]$, are coupling parameters between exchange constants and the phase transition temperature, such that $\tau_A + \tau_B = 1$ and $|J| = 3\tau k_B T_N$. Here J is the exchange constant for intra-lattice ($i = A,B$) and inter-lattice ($i,j = A,B$, $i \neq j$) coupling respectively.

The normalised equilibrium magnetisation functions $m_{e,i}$ are obtained from the Curie-Weiss law as:

$$m_{e,i} = B\left[\left(m_{e,i}\tau_i + m_{e,j}\tau_{ij}\right)3\tilde{T}_N / T + \mu_i \mu_0 H_{ext} / k_B T\right]$$

The longitudinal relaxation field includes both intra-lattice and inter-lattice contributions as:

$$\mathbf{H}_{\|,i} = \left\{\frac{1}{2\mu_0 \tilde{\chi}_{\|,i}}\left(1 - \frac{m_i^2}{m_{e,i}^2}\right) + \frac{3\tau_{ij} k_B T_N}{2\mu_0 \mu_i}\left[\frac{\tilde{\chi}_{\|,j}}{\tilde{\chi}_{\|,i}}\left(1 - \frac{m_i^2}{m_{e,i}^2}\right) - \frac{m_{e,j}}{m_{e,i}}\left(\hat{\mathbf{m}}_i.\hat{\mathbf{m}}_j\right)\left(1 - \frac{m_j^2}{m_{e,j}^2}\right)\right]\right\}\mathbf{m}_i, \quad T < T_N$$

$$\mathbf{H}_{\|,i} = -\left\{\frac{1}{\mu_0 \tilde{\chi}_{\|,i}} + \frac{3\tau_{ij} k_B T_N}{\mu_0 \mu_i}\left[\frac{\tilde{\chi}_{\|,j}}{\tilde{\chi}_{\|,i}} - \frac{m_{e,j}}{m_{e,i}}\left(\hat{\mathbf{m}}_i.\hat{\mathbf{m}}_j\right)\right]\right\}\mathbf{m}_i \quad , \quad T > T_N$$

Here $\hat{\mathbf{m}}_i = \mathbf{m}_i / m_i$, and the relative longitudinal susceptibility is $\tilde{\chi}_{\|,i} = \chi_{\|,i} / \mu_0 M_{S,i}^0$, where:

$$k_B T \tilde{\chi}_{\|,i} = \frac{\mu_i B_i'\left(1 - 3\tau_j \tilde{T}_N B_j' / T\right) + \mu_j 3\tau_{ij} \tilde{T}_N B_i' B_j' / T}{\left(1 - 3\tau_i \tilde{T}_N B_i' / T\right)\left(1 - 3\tau_j \tilde{T}_N B_j' / T\right) - \tau_{ij}\tau_{ji} B_i' B_j' \left(3\tilde{T}_N / T\right)^2},$$

and $B_i' \equiv B'_{m_{e,i}}\left[\left(m_{e,i}\tau_i + m_{e,j}\tau_{ij}\right)3\tilde{T}_N / T\right]$.

Landau-Lifshitz-Bloch with Spin-Transfer Torques (*LLB-STT*)

In implicit form we have the LLB-STT equation as:

$$\frac{\partial \mathbf{M}}{\partial t} = -\gamma \mathbf{M} \times \mathbf{H} + \frac{\tilde{\alpha}_{\perp}}{|\mathbf{M}|} \mathbf{M} \times \frac{\partial \mathbf{M}}{\partial t} + \frac{\gamma \tilde{\alpha}_{\parallel}}{|\mathbf{M}|} (\mathbf{M}.\mathbf{H}) \mathbf{M} + (\mathbf{u}.\nabla) \mathbf{M} - \frac{\beta}{|\mathbf{M}|} \mathbf{M} \times (\mathbf{u}.\nabla) \mathbf{M}$$

In this case the spin-drift velocity is given by:

$$\mathbf{u} = \mathbf{J} \frac{Pg\mu_B}{2eM_S} \frac{1}{1+\beta^2}$$

In explicit form the LLB-STT equation becomes:

$$\frac{\partial \mathbf{M}}{\partial t} = -\frac{\gamma}{1+\alpha_{\perp}^2} \mathbf{M} \times \mathbf{H} - \frac{\tilde{\alpha}_{\perp}\gamma}{1+\alpha_{\perp}^2} \frac{1}{|\mathbf{M}|} \mathbf{M} \times (\mathbf{M} \times \mathbf{H}) + \frac{\gamma \tilde{\alpha}_{\parallel}}{|\mathbf{M}|} (\mathbf{M}.\mathbf{H}) \mathbf{M} +$$
$$\frac{1}{(1+\alpha_{\perp}^2)} \left[(1+\tilde{\alpha}_{\perp}\beta)(\mathbf{u}.\nabla)\mathbf{M} - \frac{(\beta - \tilde{\alpha}_{\perp})}{|\mathbf{M}|} \mathbf{M} \times (\mathbf{u}.\nabla)\mathbf{M} - \frac{\tilde{\alpha}_{\perp}(\beta - \tilde{\alpha}_{\perp})}{|\mathbf{M}|^2} (\mathbf{M}.(\mathbf{u}.\nabla)\mathbf{M})\mathbf{M} \right]$$

Two-Sublattice Landau-Lifshitz-Bloch with Spin-Transfer Torques (*LLB-STT*)

In explicit and un-normalised form this is given by:

$$\frac{\partial \mathbf{M}_i}{\partial t} = -\frac{\gamma_i}{1+\alpha_{\perp,i}^2} \mathbf{M}_i \times \mathbf{H}_i - \frac{\tilde{\alpha}_{\perp,i}\gamma_{\mathrm{i}}}{1+\alpha_{\perp,i}^2} \frac{1}{|\mathbf{M}_i|} \mathbf{M}_i \times (\mathbf{M}_i \times \mathbf{H}_i) + \frac{\gamma_{\mathrm{i}} \tilde{\alpha}_{\parallel,\mathrm{i}}}{|\mathbf{M}_i|} (\mathbf{M}_i.\mathbf{H}_i) \mathbf{M}_i +$$
$$\frac{1}{(1+\alpha_{\perp,i}^2)} \left[(1+\tilde{\alpha}_{\perp,i}\beta_{\mathrm{i}})(\mathbf{u}_i.\nabla)\mathbf{M}_i - \frac{(\beta - \tilde{\alpha}_{\perp,i})}{|\mathbf{M}_i|} \mathbf{M}_i \times (\mathbf{u}_i.\nabla)\mathbf{M}_i - \frac{\tilde{\alpha}_{\perp,i}(\beta - \tilde{\alpha}_{\perp,i})}{|\mathbf{M}_i|^2} (\mathbf{M}_i.(\mathbf{u}_i.\nabla)\mathbf{M}_i)\mathbf{M}_i \right]$$

Here:

$$\mathbf{u}_i = \mathbf{J} \frac{Pg\mu_B}{2eM_{S,i}} \frac{1}{1+\beta^2}$$

Stochastic Landau-Lifshitz-Gilbert (s*LLG*)

The explicit and normalised sLLG equation is given as:

$$\frac{\partial \mathbf{m}}{\partial t} = -\frac{\gamma}{1+\alpha^2}\mathbf{m}\times\left(\mathbf{H}+\mathbf{H}_{thermal}\right) - \frac{\alpha\gamma}{1+\alpha^2}\mathbf{m}\times\mathbf{m}\times\left(\mathbf{H}+\mathbf{H}_{thermal}\right)$$

Each vector component of the thermal field follows a Gaussian distribution with zero mean and standard deviation given by:

$$H_\sigma = \sqrt{\frac{2\alpha k_B T}{\gamma\mu_0 M_s^0 V\Delta t}}$$

V is the volume of the stochastic computational cell, and Δt is the time step used to update the stochastic field. The stochastic field has zero spatial and vector component correlations.

Two-Sublattice Stochastic Landau-Lifshitz-Gilbert (s*LLG*)

In explicit and normalised form this is given by:

$$\frac{\partial \mathbf{m}_i}{\partial t} = -\frac{\gamma_i}{1+\alpha_i^2}\mathbf{m}_i\times\left(\mathbf{H}_i+\mathbf{H}_{th,i}\right) - \frac{\alpha_i\gamma_i}{1+\alpha_i^2}\mathbf{m}_i\times\mathbf{m}_i\times\left(\mathbf{H}_i+\mathbf{H}_{th,i}\right), \quad (i=A,B)$$

As with the sLLG equation the thermal field standard deviation is given by:

$$H_{th,i}^{std.} = \sqrt{\frac{2\alpha_i k_B T}{\gamma_i\mu_0 M_{S,i}^0 V\Delta t}}$$

Stochastic Landau-Lifshitz-Bloch (s*LLB*)

For the stochastic LLB equation we have both a thermal field and thermal torque, and is given by:

$$\frac{\partial \mathbf{M}}{\partial t} = -\frac{\gamma}{1+\tilde{\alpha}_{\perp}^{2}}\mathbf{M}\times\mathbf{H} - \frac{\tilde{\alpha}_{\perp}\gamma}{1+\tilde{\alpha}_{\perp}^{2}}\frac{1}{|\mathbf{M}|}\mathbf{M}\times\left(\mathbf{M}\times\left(\mathbf{H}+\mathbf{H}_{thermal}\right)\right) + \frac{\gamma\tilde{\alpha}_{\parallel}}{|\mathbf{M}|}\left(\mathbf{M}.\mathbf{H}\right)\mathbf{M} + \boldsymbol{\eta}_{thermal}$$

The components of the thermal field and torque follow Gaussian distributions with no correlations, zero mean and standard deviation given respectively by:

$$H_{\sigma} = \frac{1}{\alpha_{\perp}}\sqrt{\frac{2k_B T(\alpha_{\perp}-\alpha_{\parallel})}{\gamma\mu_0 M_s^0 V\Delta t}}$$

$$\eta_{\sigma} = \sqrt{\frac{2k_B T\alpha_{\parallel}\gamma M_s^0}{\mu_0 V\Delta t}}$$

Two-Sublattice Stochastic Landau-Lifshitz-Bloch (s*LLB*)

This is given by:

$$\frac{\partial \mathbf{M}_i}{\partial t} = -\tilde{\gamma}_i\mathbf{M}_i\times\mathbf{H}_{eff,i} - \tilde{\gamma}_i\frac{\tilde{\alpha}_{\perp,i}}{M_i}\mathbf{M}_i\times\left(\mathbf{M}_i\times\left(\mathbf{H}_{eff,i}+\mathbf{H}_{th,i}\right)\right) + \gamma_i\frac{\tilde{\alpha}_{\parallel,i}}{M_i}\left(\mathbf{M}_i.\mathbf{H}_{\parallel,i}\right)\mathbf{M}_i + \boldsymbol{\eta}_{th,i} \quad (i = A, B)$$

As with sLLB we have (i = A, B):

$$H_{th,i}^{std.} = \frac{1}{\alpha_{\perp,i}}\sqrt{\frac{2k_B T\left(\alpha_{\perp,i}-\alpha_{\parallel,i}\right)}{\gamma_i\mu_0 M_{S,i}^0 V\Delta t}}$$

$$\eta_{th,i}^{std.} = \sqrt{\frac{2k_B T\alpha_{\parallel,i}\gamma_i M_{S,i}^0}{\mu_0 V\Delta t}}$$

Stochastic Landau-Lifshitz-Gilbert with Spin-Transfer Torques (s*LLG-STT*)

This is similar to the LLG-STT equation, but also has the thermal field from the sLLG equation added.

Two-Sublattice Stochastic Landau-Lifshitz-Gilbert with Spin-Transfer Torques (s*LLG-STT*)

This is similar to the two-sublattice LLG-STT equation, but also has the thermal field from the sLLG equation added.

Stochastic Landau-Lifshitz-Bloch with Spin-Transfer Torques (s*LLB-STT*)

This is similar to the LLB-STT equation, but also has the thermal field from the sLLB equation added to the damping torque term, as well as the additional thermal torque term.

Two-Sublattice Stochastic Landau-Lifshitz-Bloch with Spin-Transfer Torques (s*LLB-STT*)

This is similar to the two-sublattice LLB-STT equation, but also has the thermal field from the two-sublattice sLLB equation added to the damping torque term, as well as the additional thermal torque term.

Equations with Spin Accumulation

The LLG, LLB, sLLG, and sLLB equations also appear in the forms LLG-SA, LLB-SA, sLLG-SA, and sLLB-SA. When using these equations the spin transport solver is enabled and a spin accumulation **S** is calculated. This gives rise to bulk and interfacial torques which are added to the respective equation. For example for the LLG equation we obtain the LLG-SA equation as:

$$\frac{\partial \mathbf{M}}{\partial t} = -\gamma \mathbf{M} \times \mathbf{H} + \frac{\alpha}{|\mathbf{M}|} \mathbf{M} \times \frac{\partial \mathbf{M}}{\partial t} + \mathbf{T}_S$$

The bulk spin-accumulation torque is given by:

$$\mathbf{T}_\mathrm{S} = -\frac{D_e}{\lambda_J^2} \mathbf{m} \times \mathbf{S} - \frac{D_e}{\lambda_\varphi^2} \mathbf{m} \times (\mathbf{m} \times \mathbf{S})$$

This is included as an additional effective field in the explicit forms of the equations:

$$\mathbf{H}_\mathrm{S} = \frac{D_e}{\gamma |\mathbf{M}|} \left(\frac{\mathbf{S}}{\lambda_J^2} + \frac{\mathbf{m} \times \mathbf{S}}{\lambda_\varphi^2} \right)$$

Note there are no SA version for the STT equations. This is because the Zhang-Li STTs result from the bulk $\mathbf{T}_\mathrm{S}$ torque as a special case (see e.g. S. Lepadatu, Scientific Reports 7, 12937 (2017)).

Interfacial spin-accumulation torques are also present when N/F interfaces are used:

$$\mathbf{T}_S^{interface} = \frac{g\mu_B}{ed_h} \left[\mathrm{Re}\{G^{\uparrow\downarrow}\} \mathbf{m} \times (\mathbf{m} \times \Delta \mathbf{V}_S) + \mathrm{Im}\{G^{\uparrow\downarrow}\} \mathbf{m} \times \Delta \mathbf{V}_S \right],$$

where $\Delta\mathbf{V}_\mathrm{S} = \mathbf{V}_\mathrm{S,F} - \mathbf{V}_\mathrm{S,N}$ and $\mathbf{V}_S = (D_e / \sigma)(e / \mu_B)\mathbf{S}$.

This is included as an additional effective field in the explicit forms of the equations:

$$\mathbf{H}_{\mathrm{S}} = \frac{-1}{\gamma\,|\mathbf{M}|}\frac{g\mu_B}{ed_h}\left(\mathrm{Re}\left\{G^{\uparrow\downarrow}\right\}\mathbf{m}\times\Delta\mathbf{V}_S + \mathrm{Im}\left\{G^{\uparrow\downarrow}\right\}\Delta\mathbf{V}_S\right)$$

Currently antiferromagnetic meshes do not have a drift-diffusion model included, so the SA versions are not active with two-sublattice equations. This will be included in a future version.

<u>Static Landau-Lifshitz-Gilbert (*LLGStatic*)</u>

This equation is used for static problems, i.e. where only the relaxed magnetization state is required. It is the explicit LLG equation without the precession term, and with the damping factor set to 1. This is given by:

$$\frac{\partial\mathbf{m}}{\partial t} = -\frac{\gamma}{2}\mathbf{m}\times\mathbf{m}\times\mathbf{H}$$

Modules

To add a module to simulations use the **addmodule** command as **addmodule** *meshname modulename*, e.g. **addmodule** *permalloy iDMexchange*. In Python scripts you can use *ns.addmodule('meshname', 'modulename')*. To add a supermesh module you need to use the *meshname* as *supermesh*, e.g. **addmodule** *supermesh sdemag*. To remove a module from simulations use the **delmodule** command, with the same parameters as **addmodule**. For a list of available module names see below. To see a list of available modules in the console use the **modules** command.

Modules typically correspond to an additive field in the total effective field **H** appearing in the equations shown in the Differential Equations section:

$$\mathbf{H} = \mathbf{H}_{eff} = \mathbf{H}_1 + \mathbf{H}_2 + \ldots$$

Most modules also have an energy density term associated with their effective field contributions, available as an output data parameter.

All contributions are evaluated on a cell-centered uniform finite difference mesh, with all differential operators evaluated to second order accuracy.

<u>aniuni</u> – Uniaxial Magneto-Crystalline Anisotropy

Effective field contribution:

$$\mathbf{H} = \frac{2K_1}{\mu_0 M_S}(\mathbf{m}.\mathbf{e}_A)\mathbf{e}_A + \frac{4K_2}{\mu_0 M_S}[1-(\mathbf{m}.\mathbf{e}_A)^2](\mathbf{m}.\mathbf{e}_A)\mathbf{e}_A$$

Energy density term (output data parameter: *e_anis*):

$$\varepsilon = K_1\left[1-(\mathbf{m}.\mathbf{e}_A)^2\right] + K_2\left[1-(\mathbf{m}.\mathbf{e}_A)^2\right]^2$$

For two-sublattice models the same expression is used, but K1, K2, and M_S can have different values on the two sublattices.

<u>anicubi</u> – Cubic Magneto-Crystalline Anisotropy

Effective field contribution:

$$\mathbf{H} = -\frac{2K_1}{\mu_0 M_S}[\mathbf{e}_1\alpha(\beta^2+\gamma^2)+\mathbf{e}_2\beta(\alpha^2+\gamma^2)+\mathbf{e}_3\gamma(\alpha^2+\beta^2)] \\ -\frac{2K_2}{\mu_0 M_S}[\mathbf{e}_1\alpha\beta^2\gamma^2+\mathbf{e}_2\alpha^2\beta\gamma^2+\mathbf{e}_3\alpha^2\beta^2\gamma]$$

Here α = **m**.**e**$_1$, β = **m**.**e**$_2$, and γ = **m**.**e**$_3$, where **e**$_3$ = **e**$_1$ × **e**$_2$.

Energy density term (output data parameter: *e_anis*):

$$\varepsilon = K_1[\alpha^2\beta^2+\alpha^2\gamma^2+\beta^2\gamma^2]+K_2\alpha^2\beta^2\gamma^2$$

For two-sublattice models the same expression is used, but K1, K2, and M_S can have different values on the two sublattices.

<u>demag_N</u> – Stoner-Wohlfarth Magnetostatic Interaction

Effective field contribution:

$$H_i = -N_i M_i \quad (i = x, y, z)$$

Here $N_z = 1 - N_x - N_y$.

Energy density term (output data parameter: *e_demag*):

$$\varepsilon = -\frac{\mu_0}{2}\mathbf{M}.\mathbf{H}$$

For two-sublattice models the same expression is used, but applied to the average magnetisation value.

<u>demag</u> - Magnetostatic Interaction

Effective field contribution:

$$\mathbf{H}(\mathbf{r}_0) = -\int_{\mathbf{r}\in V} \mathbf{N}(\mathbf{r}-\mathbf{r}_0)\mathbf{M}(\mathbf{r})d\mathbf{r}$$

Here **N** is a rank-2 tensor with the following symmetry:

$$\mathbf{N} = \begin{pmatrix} N_{xx} & N_{xy} & N_{xz} \\ N_{xy} & N_{yy} & N_{yz} \\ N_{xz} & N_{yz} & N_{zz} \end{pmatrix}$$

N is computed using the formulas in A.J. Newell et al., “A Generalization of the Demagnetizing Tensor for Nonuniform Magnetization” J. Geophys. Res. **98**, 9551 (1993).

Energy density term (output data parameter: *e_demag*):

$$\varepsilon = -\frac{\mu_0}{2}\mathbf{M}.\mathbf{H}$$

The convolution function is evaluated using the convolution theorem, i.e. both **N** and **M** are transformed using an FFT algorithm, multiplied in the transform space, then **H** is obtained using the inverse FFT; **M** is zero-padded before computing the FFT.

For two-sublattice models the same expression is used, but applied to the average magnetisation value.

DMExchange – Dzyaloshinskii-Moriya Bulk Exchange Interaction

Effective field contribution:

$$\mathbf{H} = -\frac{2D}{\mu_0 M_S^2} \nabla \times \mathbf{M}$$

Non-homogeneous Neumann boundary conditions are used to evaluate the curl operator (single lattice only):

$$\frac{\partial \mathbf{M}}{\partial \mathbf{n}} = \frac{D}{2A} \mathbf{n} \times \mathbf{M}$$

The DM exchange field adds to the direct exchange field.

Energy density term (output data parameter: *e_exch*):

$$\varepsilon = -\frac{\mu_0}{2} \mathbf{M}.\mathbf{H}$$

For two-sublattice models the same expression is used, but D and M_S can have different values on the two sublattices.

exchange – Direct Exchange Interaction

Effective field contribution:

$$\mathbf{H} = \frac{2A}{\mu_0 M_S^2} \nabla^2 \mathbf{M}$$

Homogeneous Neumann boundary conditions are used to evaluate the Laplacian operator.

Energy density term (output data parameter: *e_exch*):

$$\varepsilon = -\frac{\mu_0}{2}\mathbf{M}.\mathbf{H}$$

This is equivalent to:

$$\varepsilon = A\left[\left(\frac{\partial \mathbf{m}}{\partial x}\right)^2 + \left(\frac{\partial \mathbf{m}}{\partial y}\right)^2 + \left(\frac{\partial \mathbf{m}}{\partial z}\right)^2\right]$$

For two-sublattice models the same expression is used, but A and M_S can have different values on the two sublattices. Moreover for two-sublattices there are additional inter-lattice exchange interactions, which include both homogeneous and non-homogeneous contributions. In this case the full exchange field is given by:

$$\mathbf{H}_{ex,i} = \frac{2A_i}{\mu_0 M_{e,i}^2}\nabla^2\mathbf{M}_i + \frac{4A_{h,i}}{\mu_0 M_{e,i} M_{e,j}}\mathbf{M}_j + \frac{A_{nh,i}}{\mu_0 M_{e,i} M_{e,j}}\nabla^2\mathbf{M}_j, \quad (i, j = A, B,\ i \neq j)$$

<u>heat</u> – Heat Equation Solver

The heat equation in the 1-temperature model with Joule heating and any other additional heat sources (S) is given by:

$$C\rho\frac{\partial T(\mathbf{r},t)}{\partial t} = \nabla.K\nabla T(\mathbf{r},t) + \frac{\mathbf{J}^2}{\sigma} + S$$

Robin boundary conditions are used to evaluate the differential operators.

The heat equation is evaluated using the simple forward-time centered-space method. The heat equation time-step required is normally comparable to the magnetization equation time-step thus a more time-efficient method (e.g. Crank-Nicolson) is not normally required.

In the two-temperature model, the heat equation is given as:

$$C_e \rho \frac{\partial T_e(\mathbf{r},t)}{\partial t} = \nabla . K \nabla T_e(\mathbf{r},t) - G_e(T_e - T_l) + S$$
$$C_l \rho \frac{\partial T_l(\mathbf{r},t)}{\partial t} = G_e(T_e - T_l)$$

Here C_e and C_l are the electron and lattice specific heat capacities, ρ is the mass density, K is the thermal conductivity, and G_e is the electron-lattice coupling constant, typically of the order 10^{18} W/m^{3}K.

<u>iDMExchange</u> – Dzyaloshinskii-Moriya Interfacial Exchange Interaction

Effective field contribution for thin film in xy plane:

$$\mathbf{H} = -\frac{2D}{\mu_0 M_S^2}\left(\frac{\partial M_z}{\partial x}, \frac{\partial M_z}{\partial y}, -\frac{\partial M_x}{\partial x} - \frac{\partial M_y}{\partial y}\right)$$

Non-homogeneous Neumann boundary conditions are used to evaluate the differential operators:

$$\frac{\partial \mathbf{M}}{\partial \mathbf{n}} = \frac{D}{2A}(\hat{\mathbf{z}} \times \mathbf{n}) \times \mathbf{M}$$

The iDM exchange field adds to the direct exchange field.

Energy density term (output data parameter: *e_exch*):

$$\varepsilon = -\frac{\mu_0}{2}\mathbf{M}.\mathbf{H}$$

For two-sublattice models the same expression is used, but D and M_S can have different values on the two sublattices. Also the boundary conditions for differential

operators are modified due to the non-homogeneous inter-lattice exchange coupling term, and given by:

$$\frac{\partial \mathbf{M}_i}{\partial \mathbf{n}} = \frac{D_i}{2A_i(1-c_i^2)}\left[(\hat{\mathbf{z}}\times\mathbf{n})\times\left(\mathbf{M}_i - c_i\mathbf{M}_j\right)\right], \quad (i, j = A, B, \ i \neq j),$$

where $c_i = A_{nh} / 2A_i$.

<u>melastic</u> – Magneto-elastic effect

The magneto-elastic effect can be included for a cubic crystal using a strain tensor. The strain tensor is given as:

$$\mathbf{S} = \begin{pmatrix} \varepsilon_{xx} & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{xy} & \varepsilon_{yy} & \varepsilon_{yz} \\ \varepsilon_{xz} & \varepsilon_{yz} & \varepsilon_{zz} \end{pmatrix}.$$

Here we define the diagonal strain vector as $\mathbf{S}_d = (\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{zz})$, and off-diagonal strain vector as $\mathbf{S}_{od} = (\varepsilon_{yz}, \varepsilon_{xz}, \varepsilon_{xy})$. The strain tensor can have a spatial dependence, and currently needs to either be loaded from ovf2 files (strain computed with an external package), or alternatively a displacement vector field can be loaded (using ovf2 files, computed externally), and the strain tensor computed as:

$$\mathbf{S}_d = \left(\frac{\partial u_x}{\partial x}, \frac{\partial u_y}{\partial y}, \frac{\partial u_z}{\partial z}\right)$$

$$\mathbf{S}_{od} = \frac{1}{2}\left(\frac{\partial u_y}{\partial z} + \frac{\partial u_z}{\partial y}, \frac{\partial u_x}{\partial z} + \frac{\partial u_z}{\partial x}, \frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}\right)$$

In the simplest case a uniform stress may be applied which results in a constant strain with zero off-diagonal terms. In a future version an elastostatics solver as, well as a dynamical elastic solver will be included.

From the strain tensor, for a cubic crystal with orthogonal axes $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$, and magneto-elastic constants B1, B2, we have the following diagonal and off-diagonal energy density terms:

$$\varepsilon_{mel,d} = B_1\left[(\mathbf{m}.\mathbf{e}_1)^2(\mathbf{S}_d.\mathbf{e}_1) + (\mathbf{m}.\mathbf{e}_2)^2(\mathbf{S}_d.\mathbf{e}_2) + (\mathbf{m}.\mathbf{e}_3)^2(\mathbf{S}_d.\mathbf{e}_3)\right]$$

$$\varepsilon_{mel,od} = 2B_2\left[(\mathbf{m}.\mathbf{e}_1)(\mathbf{m}.\mathbf{e}_2)(\mathbf{S}_{od}.\mathbf{e}_3) + (\mathbf{m}.\mathbf{e}_1)(\mathbf{m}.\mathbf{e}_3)(\mathbf{S}_{od}.\mathbf{e}_2) + (\mathbf{m}.\mathbf{e}_2)(\mathbf{m}.\mathbf{e}_3)(\mathbf{S}_d.\mathbf{e}_1)\right]$$

The effective field can be computed using the usual formula: $\mathbf{H}_{mel} = -\frac{1}{\mu_0 M_S}\frac{\partial \varepsilon_{mel}}{\partial \mathbf{m}}$.

Currently this module is not enabled for two-sublattice models.

<u>moptical</u> – Magneto-optical effect

This module applies a z-axis field as given by the *Hmo* parameter (A/m) as:

$$H_{MO} = \sigma^{\pm} H_{MO}^0 f_{MO}(\mathbf{r},t)\hat{\mathbf{z}}$$

Here $\sigma^{\pm} H_{MO}^0$ is set using the *Hmo* parameter, and f_{MO} is its spatial (and temporal) variation.

For two-sublattice models the same field is applied to both sublattices.

<u>Oersted</u> – Oersted Field

Effective field contribution:

$$\mathbf{H}(\mathbf{r}_0) = \int_{\mathbf{r}\in V} \mathbf{K}(\mathbf{r}-\mathbf{r}_0)\mathbf{J}_C(\mathbf{r})d\mathbf{r}$$

Here **K** is a rank-2 tensor with the following symmetry:

$$\mathbf{K} = \begin{pmatrix} 0 & K_{xy} & K_{xz} \\ -K_{xy} & 0 & K_{yz} \\ -K_{xz} & -K_{yz} & 0 \end{pmatrix}$$

K is computed using the formulas in B. Krüger, “Current-Driven Magnetization Dynamics: Analytical Modeling and Numerical Simulation”, PhD Dissertation, University of Hamburg (2011) – Appendix D, page 118.

For two-sublattice models the Oersted field is applied equally to both sublattices.

<u>roughness</u> – Roughness Field and Staircase Magnetostatic Corrections

Effective field contribution computed on the coarse mesh (i.e. the actual mesh discretisation used at run-time with N_V number of discretisation cells):

$$\mathbf{H}(\mathbf{r}_0) = -\left[\sum_{\mathbf{r}\in V} \mathbf{N}(\mathbf{r}-\mathbf{r}_0) G(\mathbf{r},\mathbf{r}_0)\right] \mathbf{M}(\mathbf{r}_0) \quad (\mathbf{r}_0 \in V)$$

Here **N** is the demagnetizing tensor computed on the fine mesh with N_{Vr} number of discretisation cells, and:

$$G(\mathbf{r},\mathbf{r}_0) = \begin{cases} \dfrac{N_V}{N_{Vr}} - 1 & \mathbf{r} \wedge \mathbf{r}_0 \in V_R \\ -1 & \mathbf{r} \vee \mathbf{r}_0 \in V - V_R \end{cases}$$

V is the smooth body without roughness and V_R is the mesh with roughness, and we require $V_R \subseteq V$. If the coarse cellsize has dimensions (h_x, h_y, h_z), the fine cellsize must have dimensions (h_x / m_x, h_y / m_y, h_z / m_z), where the *m* factors are integers. The function $\sum_{\mathbf{r}\in V} \mathbf{N}(\mathbf{r}-\mathbf{r}_0) G(\mathbf{r},\mathbf{r}_0)$ is computed at initialisation on the finely discretised mesh then averaged up to the coarse mesh (each coarse cellsize value is obtained as an average of its contained fine cellsize values).

Energy density term (output data parameter: *e_rough*):

$$\varepsilon = -\frac{\mu_0}{2}\mathbf{M}.\mathbf{H}$$

Details can be found in: S. Lepadatu, “Effective field model of roughness in magnetic nano-structures” J. Appl. Phys. **118**, 243908 (2015).

For two-sublattice models the roughness field is computed using the average magnetization, and applied equally to both sublattices.

<u>sdemag</u> – Supermesh Magnetostatic Interaction

I. Supermesh demagnetization (**multiconvolution** 0).

The same formulas as for the *Demag* module are used when computing demagnetizing fields on the uniformly discretised super-mesh. The ferromagnetic super-mesh may have a cellsize which differs from that of the individual ferromagnetic meshes. In this case a weighted average smoother is used to transfer magnetization to the super-mesh and demagnetizing field values back from the super-mesh.

Consider a discrete distribution of magnetization values **M** at points $V = \{\mathbf{r}_i;\ i \in P\}$. Let **h** be the cellsize of the input mesh, with the set of cells $\{c_i;\ i \in P\}$ centered around the points $\mathbf{r}_i$. To obtain the magnetization value at a point $\mathbf{r}'$ in a cell c with dimensions $\mathbf{h}'$ we introduce the definitions $d_i = |\mathbf{r}' - \mathbf{r}_i|$, $d_V = |\mathbf{h}' + \mathbf{h}|/2$, and $\tilde{d}_i = d_V - d_i$. The weighted average is given as:

$$\mathbf{M}(\mathbf{r}') = \sum_{i \in P} w_i \mathbf{M}(\mathbf{r}_i)$$

where

$$w_i = \frac{\tilde{d}_i \delta_i}{\tilde{d}_T}$$

$$\delta_i = \begin{cases} 1, & c_i \cap c \neq \varnothing \\ 0, & otherwise \end{cases}$$

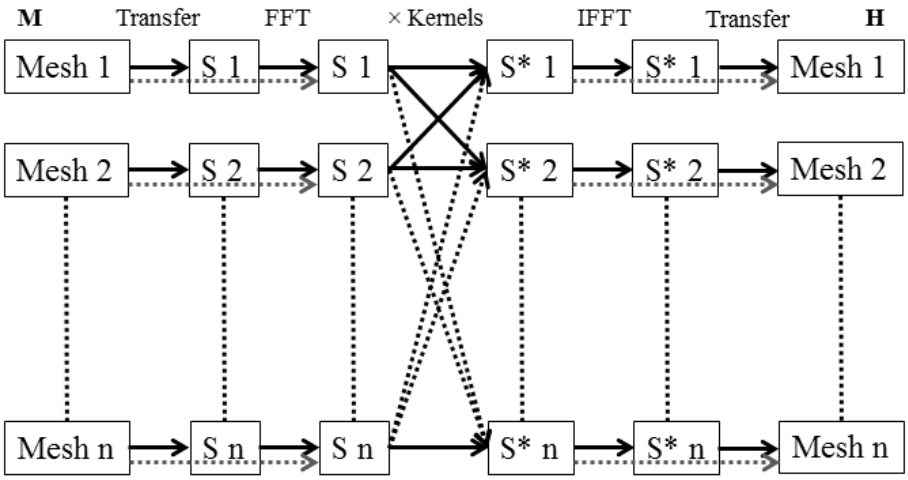
$$\tilde{d}_T = \sum_{i \in P} \tilde{d}_i \delta_i$$

II. Multi-Layered convolution (**multiconvolution** 1)

A generalisation of the single layer convolution algorithm is used here. We can write the convolution sum as:

$$\mathbf{H}(\mathbf{r}'_{kl}) = -\sum_{\substack{i=1,\ldots,n \\ \mathbf{r}_{ij} \in V_i}} \mathbf{N}(\mathbf{r}'_{kl} - \mathbf{r}_{ij}, \mathbf{h}_k, \mathbf{h}_i)\mathbf{M}(\mathbf{r}_{ij}), \quad k = 1,\ldots,n; \quad \mathbf{r}'_{kl} \in V_k$$

In the demagnetizing tensor for the equation above we explicitly specify the cellsize, **h**, of the two computational meshes the tensor relates. Since we have *n* terms of the form appearing in the single layer convolution sum, we can again apply the convolution theorem. This time for each output mesh (**H**) we have *n* input meshes (**M**), together with *n* kernels. Thus to calculate the outputs in all *n* meshes we require a total of n^2 sets of kernel multiplications in the transform space. This is illustrated in the figure below.



Multi-Layered convolution algorithm for *n* computational meshes. The magnetization input of each mesh is transformed separately using a FFT algorithm, either directly (dotted line), or by first transferring to a scratch space with a common

discretisation cellsize, using a weighted average smoother (solid lines). In the transform space the inputs are multiplied with pre-computed kernels for a total of n^2 sets of point-by-point multiplications. Finally the output demagnetizing fields are obtained using an inverse FFT algorithm, which are set directly in the output meshes (dotted line), or transferred using a weighted average smoother if the discretisation cellsizes differ (solid lines).

<u>SOTfield</u> – Spin-Orbit Torque Field

The spin-orbit torque is given by:

$$\mathbf{T}_{SOT} = \theta_{SHA,eff} \frac{\mu_B}{e} \frac{|J_c|}{d_h} (\mathbf{m} \times (\mathbf{m} \times \mathbf{p}) + r_G \mathbf{m} \times \mathbf{p})$$

Here **p** = **z** × $\mathbf{e}_{\mathbf{J}_c}$. This results in an effective field in the magnetization dynamics equation given by:

$$\mathbf{H}_{SOT} = -\frac{1}{\gamma M_S} \theta_{SHA,eff} \frac{\mu_B}{e} \frac{|J_c|}{d_h} (\mathbf{m} \times \mathbf{p} + r_G \mathbf{p})$$

For two-sublattice models the same expression is used, but γ and M_S can have different values on the two sublattices.

<u>strayfield</u> – Stray field from magnetic dipoles

If $\mathbf{M}_d$ is the magnetization of a uniformly magnetized prism with dimensions **d**, then the magnetic field (stray field) at a distance **r** from the centre of the prism is given by:

$$\mathbf{H} = -\mathbf{N}_d(\mathbf{r}, \mathbf{d})\mathbf{M}_d$$

Here $\mathbf{N}_d$ is a rank-2 tensor with the following symmetry:

$$\mathbf{N}_d = \begin{pmatrix} N_{xx} & N_{xy} & N_{xz} \\ N_{xy} & N_{yy} & N_{yz} \\ N_{xz} & N_{yz} & N_{zz} \end{pmatrix}$$

$\mathbf{N}_d$ is computed using the formulas in A. Andreev et al., "Universal Method for the Calculation of Magnetic Microelectronic Components: the Saturated Ferromagnetic Rectangular Prism and the Rectangular Coil." ICSE2000 Proceedings, Nov. 2000, 187. Note these formulas are equivalent to the Newell formulas used to compute the demagnetizing tensor.

For two-sublattice models the field is applied equally to both sublattices.

surfexchange – Surface Exchange Interaction

Let $\mathbf{m}_i$ and $\mathbf{m}_j$ be the normalised magnetization values of two cells, *i* and *j*, which are surface exchange coupled across a gap between two ferromagnetic meshes. Let Δ be the thickness of the ferromagnetic layer for which the surface exchange field is computed. The surface exchange field at cell *i*, from cell *j*, is given as:

$$\mathbf{H}_i = \frac{J_1}{\mu_0 M_S \Delta}\mathbf{m}_j + \frac{2J_2}{\mu_0 M_S \Delta}(\mathbf{m}_i.\mathbf{m}_j)\mathbf{m}_j$$

The surface exchange field is applied for all cells along the z direction if a 3D simulation mesh is used (surface exchange field applicable for thin films).

Energy density term (output data parameter: *e_surfexch*):

$$\varepsilon = -\frac{J_1}{\Delta}\mathbf{m}_i.\mathbf{m}_j - \frac{J_2}{\Delta}(\mathbf{m}_i.\mathbf{m}_j)^2$$

The surfexchange module also allows coupling between an antiferromagnetic mesh and a ferromagnetic mesh, with the resultant coupling being the exchange bias field. In this case the same formulas above are used, but the coupling is only done to/from sub-lattice A.

<u>transport</u> – Charge and Spin-Transport Solver

<u>Charge Transport</u>

When solving only for the charge current density, a Poisson-type equation for V is solved as:

$$\nabla^2 V = -\frac{(\nabla V).(\nabla \sigma)}{\sigma}$$

For *V* Dirichlet boundary conditions are used at boundaries containing a fixed potential electrode, otherwise Neumann boundary conditions are used. The conductivity may have an AMR contribution (AMR given as a percentage value) calculated as:

$$\sigma = \frac{\sigma_0}{1 + (AMR/100)\mathbf{d}^2},$$

where

$$\mathbf{d} = \frac{\mathbf{J}_C . \mathbf{M}}{|\mathbf{J}_C||\mathbf{M}|}.$$

From *V* the charge current density is obtained as $\mathbf{J}_c = -\sigma\nabla V$, and is used for Joule heating computations and to obtain spin torques (Zhang-Li STT and SOT).

The Poisson equation is evaluated using the successive over-relaxation (SOR) algorithm with black-red ordering for parallelization.

<u>Charge and Spin Transport</u>

Charge and spin current densities are given as (see S. Lepadatu, “Unified treatment of spin torques using a coupled magnetization dynamics and three-dimensional spin current solver” Scientific Reports 7, 12937 (2017) for details):

$$\mathbf{J}_C = \sigma\mathbf{E} + \beta_D D_e \frac{e}{\mu_B}(\nabla\mathbf{S})\mathbf{m} + \theta_{SHA} D_e \frac{e}{\mu_B}\nabla\times\mathbf{S} + P\sigma\frac{\hbar}{2e}\mathbf{E}^{\sigma} - P\frac{\sigma^2\hbar}{e^2 n}\mathbf{E}\times\mathbf{B}^{\sigma}$$

$$\mathbf{J}_S = -\frac{\mu_B}{e}P\sigma\mathbf{E}\otimes\mathbf{m} - D_e\nabla\mathbf{S} + \theta_{SHA}\frac{\mu_B}{e}\boldsymbol{\varepsilon}\sigma\mathbf{E} + \frac{\hbar\mu_B\sigma}{2e^2}\sum_i \mathbf{e}_i\otimes(\dot{\mathbf{m}}\times\partial_i\mathbf{m}) + \frac{\hbar\mu_B\sigma^2}{e^3 n}(\mathbf{z}\times\mathbf{E})\otimes(\partial_x\mathbf{m}\times\partial_y\mathbf{m})$$

Where:

$$E_i^{\sigma} = (\dot{\mathbf{m}}\times\partial_i\mathbf{m}).\mathbf{m}$$

$$\mathbf{B}^{\sigma} = \mathbf{z}(\partial_x\mathbf{m}\times\partial_y\mathbf{m}).\mathbf{m}$$

Here $\mathbf{E}^{\sigma}$ and $\mathbf{B}^{\sigma}$ are the directions of the emergent electric field due to charge pumping, and emergent magnetic field due to topological Hall effect respectively.

With the full spin transport solver enabled both *V* and ***S*** are computed using Poisson-type equations as:

$$\nabla^2 V = -\frac{(\nabla V).(\nabla\sigma)}{\sigma} + \frac{\beta_D D_e}{\sigma}\frac{e}{\mu_B}\nabla.(\nabla\mathbf{S})\mathbf{m}$$
$$+\frac{P\hbar}{2e}\left[\partial_x\dot{\mathbf{m}}\times\partial_x\mathbf{m} + \partial_y\dot{\mathbf{m}}\times\partial_y\mathbf{m} + \dot{\mathbf{m}}\times(\partial_x^2\mathbf{m} + \partial_y^2\mathbf{m})\right].\mathbf{m}$$
$$-\frac{P\sigma\hbar}{e^2 n}\left[\mathbf{x}(\partial_{xy}^2\mathbf{m}\times\partial_y\mathbf{m} + \partial_x\mathbf{m}\times\partial_y^2\mathbf{m}).\mathbf{m} - y(\partial_x^2\mathbf{m}\times\partial_y\mathbf{m} + \partial_x\mathbf{m}\times\partial_{xy}^2\mathbf{m}).\mathbf{m}\right].\nabla V$$

and

$$\nabla^2\mathbf{S} = -\frac{\mu_B}{e}\frac{P\sigma}{D_e}(\mathbf{E}.\nabla)\mathbf{m} + \frac{\mu_B}{e}\frac{P\sigma}{D_e}(\nabla^2 V)\mathbf{m} + \frac{\theta_{SHA}}{D_e}\frac{\mu_B}{e}\sigma\nabla.(\boldsymbol{\varepsilon}\mathbf{E})$$
$$+\frac{\hbar\mu_B\sigma}{2e^2 D_e}\left[\partial_x\dot{\mathbf{m}}\times\partial_x\mathbf{m} + \partial_y\dot{\mathbf{m}}\times\partial_y\mathbf{m} + \dot{\mathbf{m}}\times(\partial_x^2\mathbf{m} + \partial_y^2\mathbf{m})\right]$$
$$+\frac{\hbar\mu_B\sigma^2}{e^3 n D_e}\left[E_x(\partial_{xy}^2\mathbf{m}\times\partial_y\mathbf{m} + \partial_x\mathbf{m}\times\partial_y^2\mathbf{m}) - E_y(\partial_x^2\mathbf{m}\times\partial_y\mathbf{m} + \partial_x\mathbf{m}\times\partial_{xy}^2\mathbf{m})\right]$$
$$+\frac{\mathbf{S}}{\lambda_{sf}^2} + \frac{\mathbf{S}\times\mathbf{m}}{\lambda_J^2} + \frac{\mathbf{m}\times(\mathbf{S}\times\mathbf{m})}{\lambda_{\varphi}^2}$$

For boundaries containing an electrode with a fixed potential, differential operators applied to V use a Dirichlet boundary condition. For other external boundaries the following non-homogeneous Neumann boundary conditions are used:

$$\nabla V.\mathbf{n} = \theta_{\mathrm{SHA}} \frac{D_e}{\sigma} \frac{e}{\mu_{\mathrm{B}}} (\nabla \times \mathbf{S}).\mathbf{n}$$

$$(\nabla \mathbf{S}).\mathbf{n} = \theta_{\mathrm{SHA}} \frac{\sigma}{D_e} \frac{\mu_{\mathrm{B}}}{e} (\boldsymbol{\varepsilon}\mathbf{E}).\mathbf{n}$$

At N/F composite media boundaries the following conditions are applied:

$$\mathbf{J}_C.\mathbf{n}\big|_N = \mathbf{J}_C.\mathbf{n}\big|_F = -\left(G^{\uparrow} + G^{\downarrow}\right)\Delta V + \left(G^{\uparrow} - G^{\downarrow}\right)\Delta \mathbf{V}_S.\mathbf{m}$$

$$\mathbf{J}_S.\mathbf{n}\big|_N - \mathbf{J}_S.\mathbf{n}\big|_F = \frac{2\mu_B}{e}\left[\mathrm{Re}\left\{G^{\uparrow\downarrow}\right\}\mathbf{m} \times (\mathbf{m} \times \Delta \mathbf{V}_S) + \mathrm{Im}\left\{G^{\uparrow\downarrow}\right\}\mathbf{m} \times \Delta \mathbf{V}_S\right]$$

$$\mathbf{J}_S.\mathbf{n}\big|_F = \frac{\mu_B}{e}\left[-\left(G^{\uparrow} + G^{\downarrow}\right)(\Delta \mathbf{V}_S.\mathbf{m})\mathbf{m} + \left(G^{\uparrow} - G^{\downarrow}\right)\Delta V \mathbf{m}\right]$$

Spin pumping is included on the N side of the above equations as:

$$\mathbf{J}_S^{pump} = \frac{\mu_B \hbar}{e^2}\left[\mathrm{Re}\left\{G^{\uparrow\downarrow}\right\}\mathbf{m} \times \frac{\partial \mathbf{m}}{\partial t} + \mathrm{Im}\left\{G^{\uparrow\downarrow}\right\}\frac{\partial \mathbf{m}}{\partial t}\right]$$

At N/F interfaces, interfacial spin torques are obtained as (h_F is the discretisation cellsize of the F layer in the direction normal to the composite media boundary) :

$$\mathbf{T}_S = \frac{g\mu_B}{e h_F}\left[\mathrm{Re}\left\{G^{\uparrow\downarrow}\right\}\mathbf{m} \times (\mathbf{m} \times \Delta \mathbf{V}_S) + \mathrm{Im}\left\{G^{\uparrow\downarrow}\right\}\mathbf{m} \times \Delta \mathbf{V}_S\right]$$

From **S**, bulk spin torques are obtained as:

$$\mathbf{T}_S = -\frac{D_e}{\lambda_J^2}\mathbf{m} \times \mathbf{S} - \frac{D_e}{\lambda_\varphi^2}\mathbf{m} \times (\mathbf{m} \times \mathbf{S})$$

Both Poisson equations are evaluated using the SOR algorithm with black-red ordering for parallelization. Note, whilst the SOR algorithm is robust in evaluating the spin transport equations in arbitrary multi-layers with composite media boundary conditions, it does suffer from slow convergence in particular for lower target solver errors. This algorithm is due to be replaced with a more efficient method in the next version. Currently the alternating direction implicit method with parallelized Thomas algorithm, as well as a FFT-based Poisson solver are being evaluated. Another possibility is a bi-conjugate gradient method.

Zeeman – Applied Magnetic Field

Effective field contribution:

$$\mathbf{H} = \mathbf{H}_{ext}$$

Energy density term (output data parameter: *e_zee*):

$$\varepsilon = -\mu_0 \mathbf{M}.\mathbf{H}$$

For two-sublattice models the field is applied equally to both sublattices.

Material Parameters

To set a material parameter value use the **setparam** command as **setparam** *meshname paramname value*, e.g. **setparam** *permalloy A* 1e-11, or even **setparam** *permalloy A* 10pJ/m. In Python scripts you can set a parameter value as *ns.setparam('meshname', 'paramname', value)*. For list of available parameter names see below. To see a list of available parameters in the console use the **params** command.

Format: **paramname**: *name in equations* (units) Description.

A: A (J/m)
Exchange stiffness.

A_AFM: A_i (J/m)
Exchange stiffness.

Ah: $A_{h,i}$ (J/m^3)
Homogeneous inter-lattice exchange coupling per lattice constant.

Anh: $A_{nh,i}$ (J/m^3)
Non-homogeneous inter-lattice exchange stiffness.

amr: *AMR* (%)
Anisotropic magneto-resistance as a percentage of base resistance.

beta: β (unitless)
Spin-transfer torque non-adiabaticity parameter.

betaD: β_D (unitless)
Diffusion spin polarisation.

cHA: *cHA* (unitless)
Applied field spatial variation parameter, which multiplies the applied field value.

cpump_eff: cpump_eff (unitless)
Charge-pumping efficiency.

cT: *cT* (unitless)
Set temperature spatial variation parameter, which multiplies the set temperature value. To enable spatial variation of temperature you need to have the *heat* module enabled. If you want the set temperature to remain constant you need to disable the heat equation by using **setheatdt** 0.

D: *D* (J/m^2)
Dzyaloshinskii-Moriya exchange constant.

D_AFM: D_i (J/m^2)
Dzyaloshinskii-Moriya exchange constant.

damping: α (unitless)
Gilbert magnetization damping.

damping_AFM: α_i (unitless)
Gilbert magnetization damping.

De: *De* (m^2/s)
Electron diffusion constant.

density: ρ (kg/m^3)
Mass density.

ea1: $\boldsymbol{e}_{a1}$ (unit vector)
Uniaxial magneto-crystalline anisotropy symmetry axis, or first cubic magneto-crystalline anisotropy symmetry axis.

ea2: $\boldsymbol{e}_{a2}$ (unit vector)
Second cubic magneto-crystalline anisotropy symmetry axis ($\boldsymbol{e}_{a3} = \boldsymbol{e}_{a1} \times \boldsymbol{e}_{a2}$).

elC: σ (S/m)
Base electrical conductivity.
flSOT: r_G (unitless)
Field-like spin orbit torque coefficient.

G_e: G_e (W/m^{3}K)
Electron-lattice coupling constant (two temperature model).

Gi: $G^{\uparrow}$, $G^{\downarrow}$ (S/m^2)
Interface spin-dependent conductivity (for majority and minority carriers). The top contacting mesh sets the interface value, thus *Gi* is available in both magnetic and non-magnetic meshes.

Gmix: $G^{\uparrow\downarrow} = Re\{G\} + i\, Im\{G\}$ (S/m^2)
interface spin-mixing conductivity (real and imaginary parts). The top contacting mesh sets the interface value, thus *Gmix* is available in both magnetic and non-magnetic meshes.

grel: g_{rel} (unitless)
Relative electron gyromagnetic ratio.

grel_AFM: $g_{rel,i}$ (unitless)
Relative electron gyromagnetic ratio.

Hmo: Hmo (A/m)
Magneto-optical field strength.

iSHA: θ_{SHA} (unitless)
Spin Hall angle used for the inverse spin Hall effect.

J1: *J1* (J/m^2)
Bilinear surface exchange coupling. For coupled meshes it is the top mesh that sets the J values.

J2: *J2* (J/m^2)
Biquadratic surface exchange coupling. For coupled meshes it is the top mesh that sets the J values.

K1: *K1* (J/m^3)
Magneto-crystalline anisotropy energy.

K1_AFM: $K1_i$ (J/m^3)
Magneto-crystalline anisotropy energy.

K2: *K2* (J/m^3)
Magneto-crystalline anisotropy energy, higher order.

K2_AFM: $K2_i$ (J/m^3)
Magneto-crystalline anisotropy energy, higher order.

l_J: λ_J (m)
Spin exchange rotation length.

l_phi: λ_φ (m)
Spin dephasing length.

l_sf: λ_{sf} (m)
Spin-flip length.

MEc: B_1, B_2 (J/m^3)
Magneto-elastic coefficients.

Ms: *Ms* (A/m)
Saturation magnetization.

Ms_AFM: Ms_i (A/m)
Saturation magnetization.
n: *n* (m^{-3})
Conduction electrons density.

Nx, **Ny**: *Nxy* (unitless)
In-plane demagnetizing factors (used by demag_N module)

P: *P* or β_σ (unitless)
Charge current spin polarization.

Pr: ν (unitless)
Poisson's ratio.

pump_eff: η_{pump} (unitless)
Spin pumping efficiency.

Q: *Q* (W/m^3)
Heat source added to the heat equation. Can be non-uniform by setting a spatial variation.

SHA: θ_{SHA} (unitless)
Spin Hall angle used for the spin Hall effect.

shc: *C* (J/kgK).
Specific heat capacity (total – one temperature model, lattice – two temperature model).

shc_e: *C* (J/kgK).
Electronic specific heat capacity (two temperature model).

susrel: $\chi_{\parallel}$ (As^2/kg)
Longitudinal (parallel) susceptibility divided by $\mu_0 Ms$.

susrel_AFM: $\chi_{\parallel,i}$ (As^2/kg)
Longitudinal (parallel) susceptibility divided by $\mu_0 Ms$.

tau_ii: τ_{ii} (unitless)
Coupling between exchange constants and phase transition temperature: two-sublattice model, intra-lattice.

tau_ij: τ_{ij} (unitless)
Coupling between exchange constants and phase transition temperature: two-sublattice model, inter-lattice.

the_eff: the_eff (unitless)
Topological Hall effect efficiency.

thermK: K (W/mK)
Thermal conductivity.

ts_eff: η_{ts} (unitless)
Spin accumulation torque efficiency in the bulk.

tsi_eff: η_{tsi} (unitless)
Spin accumulation torque efficiency at interfaces.

Ym: Y (Pa)
Young's modulus.

Commands – Essential

Essential commands only. These commands are used most often, or unlock a large part of the functionality through interactive objects console output. For descriptions see Commands – All section or type them in the console.

addconductor	**modules**
addinsulator	**multiconvolution**
addmesh	**ode**
center	**params**
chdir	**pbc**
computefields	**reset**
cuda	**run**
data	**savesim**
default	**setangle**
display	**setfield**
loadsim	**stages**
mesh	**stop**

Commands – Important

Important commands for more advanced users, but might not be used as often as the essential commands. For descriptions see Commands – All section or type them in the console.

ambient	**savemeshimage**
curietemperature	**setcurrent**
dwall	**setdefaultelectrodes**
electrodes	**setdt**
dp_getprofile	**setheatdt**
iterupdate	**setpotential**
paramstemp	**skyrmion**
paramsvar	**showdata**
preparemovingmesh	**temperature**
resetmesh	**tsolverconfig**

Commands – Useful

Other useful commands. For descriptions see Commands – All section or type them in the console.

adddipole	**loadmaskfile**
addelectrode	**makevideo**
clearelectrodes	**refineroughness**
clearroughness	**roughenmesh**
copymeshdata	**scalemeshrects**
copyparams	**setparamtemparray**
dp_averagemeshrect	**surfroughenjagged**

Commands – All (Alphabetical)

2dmulticonvolution

USAGE : **2dmulticonvolution** status

Switch to multi-layered convolution and force it to 2D layering in each mesh (2), or 2D convolution for each mesh (1), or allow 3D (0).

addafmesh

USAGE : **addafmesh** name rectangle

Add antiferromagnetic mesh with given name and rectangle (m). The rectangle can be specified as: sx sy sz ex ey ez for the start and end points in Cartesian coordinates, or as: ex ey ez with the start point as the origin.

addameshcubic

USAGE : **addameshcubic** name rectangle

Add an atomistic mesh with simple cubic structure, with given name and rectangle (m). The rectangle can be specified as: sx sy sz ex ey ez for the start and end points in Cartesian coordinates, or as: ex ey ez with the start point as the origin.

addconductor

USAGE : **addconductor** name rectangle

Add a normal metal mesh with given name and rectangle (m). The rectangle can be specified as: sx sy sz ex ey ez for the start and end points in Cartesian coordinates, or as: ex ey ez with the start point as the origin.

adddata

USAGE : **adddata** dataname (meshname, (rectangle))

Add dataname to list of output data. If applicable specify meshname and rectangle (m) in mesh. If not specified and required, active mesh is used with entire mesh rectangle.

adddiamagnet

USAGE : **adddiamagnet** name rectangle

Add a diamagnetic mesh with given name and rectangle (m). The rectangle can be specified as: sx sy sz ex ey ez for the start and end points in Cartesian coordinates, or as: ex ey ez with the start point as the origin.

adddipole

USAGE : **adddipole** name rectangle

Add a rectangular dipole with given name and rectangle (m). The rectangle can be specified as: sx sy sz ex ey ez for the start and end points in Cartesian coordinates, or as: ex ey ez with the start point as the origin.

addelectrode

USAGE : **addelectrode** electrode_rect

Add an electrode in given rectangle (m).

addinsulator

USAGE : **addinsulator** name rectangle

Add an insulator mesh with given name and rectangle (m). The rectangle can be specified as: sx sy sz ex ey ez for the start and end points in Cartesian coordinates, or as: ex ey ez with the start point as the origin.

addmaterial

USAGE : **addmaterial** name rectangle

Add a new mesh with material parameters loaded from the materials database. The name is the material name as found in the mdb file (see materialsdatabase command); this also determines the type of mesh to create, as well as the created mesh name. The rectangle (m) can be specified as: sx sy sz ex ey ez for the start and end points in Cartesian coordinates, or as: ex ey ez with the start point as the origin.
Script return values: meshname - return name of mesh just added (can differ from the material name).

addmdbentry

USAGE : **addmdbentry** meshname (materialname)

Add new entry in the local materials database from parameters in the given mesh. The name of the new entry is set to materialname if specified, else set to meshname. For a complete entry you should then edit the mdb file manually with all the appropriate fields shown there.

addmesh

USAGE : **addmesh** name rectangle

Add a ferromagnetic mesh with given name and rectangle (m). The rectangle can be specified as: sx sy sz ex ey ez for the start and end points in Cartesian coordinates, or as: ex ey ez with the start point as the origin.

addmodule

USAGE : **addmodule** meshname handle

Add module with given handle to named mesh.

addpinneddata

USAGE : **addpinneddata** dataname (meshname, (rectangle))

Add new entry in data box (at the end) with given dataname and meshname if applicable. A rectangle may also be specified if applicable, however this will not be shown in the data box.

addrect

USAGE : **addrect** rectangle (meshname)

Fill rectangle (m) within given mesh (active mesh if name not given). The rectangle coordinates are relative to specified mesh.

addstage

USAGE : **addstage** stagetype (meshname)

Add a generic stage type to the simulation schedule with name stagetype, specifying a meshname if needed (if not specified and required, active mesh is used).

ambient

USAGE : **ambient** ambient_temperature (meshname)

Set mesh ambient temperature (all meshes if meshname not given) for Robin boundary conditions : flux normal = alpha * (T_boundary - T_ambient).
Script return values: ambient_temperature - ambient temperature for mesh in focus.

astepctrl

USAGE : **astepctrl** err_fail err_high err_low dT_incr dT_min dT_max

Set parameters for adaptive time step control: err_fail - repeat step above this, err_high - decrease dT abnove this, err_low - increase dT below this, dT_incr - increase dT using fixed multiplier, dT_min, dT_max - dT bounds.

atomicmoment

USAGE : **atomicmoment** ub_multiple (meshname)

Set atomic moment as a multiple of Bohr magnetons (all applicable meshes if meshname not given) for given mesh. This affects the temperature dependence of 'me' (see curietemperature command). A non-zero value will result in me(T) being dependent on the applied field.
Script return values: ub_multiple - atomic moment multiple of Bohr magneton for mesh in focus.

averagemeshrect

USAGE : **averagemeshrect** (rectangle)

Calculate the average value depending on currently displayed quantities. The rectangle is specified in relative coordinates to the currently focused mesh; if not specified average the entire focused mesh.
Script return values: value

benchtime

USAGE : **benchtime**

Show the last simulation duration time in ms, between start and stop; used for performance becnhmarking.
Script return values: value

blochpreparemovingmesh

USAGE : **blochpreparemovingmesh** (meshname)

Setup the named mesh (or active mesh) for moving Bloch domain wall simulations: 1) set movingmesh trigger, 2) set domain wall structure, 3) set dipoles left and right to remove end magnetic charges, 4) enable strayfield module.

cellsize

USAGE : **cellsize** value

Change cellsize of mesh in focus (m). The cellsize can be specified as: hx hy hz, or as: hxyz
Script return values: cellsize - return cellsize of mesh in focus.

center

USAGE : **center**

Center mesh view and scale to fit window size.

chdir

USAGE : **chdir** directory

Change working directory.
Script return values: directory

checkupdates

USAGE : **checkupdates**

Connect to boris-spintronics.uk to check if updates to program or materials database are available.

clearelectrodes

USAGE : **clearelectrodes**

Delete all currently set electrodes.

clearequationconstants

USAGE : **clearequationconstants**

Clear all user-defined constants for text equations.

clearmovingmesh

USAGE : **clearmovingmesh**

Clear moving mesh settings made by a prepare command.

clearparamstemp

USAGE : **clearparamstemp** (meshname, (paramname))

Clear material parameter temperature dependence in given mesh. If meshname not given clear temperature dependences in all meshes for all parameters. If paramname not given clear all parameters temperature dependences in named mesh.

clearparamsvar

USAGE : **clearparamsvar** (meshname)

Clear all material parameters spatial dependence in given mesh. If meshname not given clear spatial dependence in all meshes.

clearparamvar

USAGE : **clearparamvar** meshname paramname

Clear parameter spatial dependence in given mesh.

clearroughness

USAGE : **clearroughness** (meshname)

Clear roughness by setting the fine shape same as the coarse M shape.

clearscreen

USAGE : **clearscreen**

Clear all console text.

computefields

USAGE : **computefields**

Run simulation from current state for a single iteration without advancing the simulation time.

copymeshdata

USAGE : **copymeshdata** meshname_from meshname_to (...)

Copy all primary mesh data (e.g. magnetisation values and shape) from first mesh to all other meshes given - all meshes must be of same type.

copyparams

USAGE : **copyparams** meshname_from meshname_to (...)

Copy all mesh parameters from first mesh to all other meshes given - all meshes must be of same type.

coupletodipoles

USAGE : **coupletodipoles** status

Set/unset coupling to dipoles : if ferromagnetic meshes touch a dipole mesh then interface magnetic cells are exchange coupled to the dipole magnetisation direction.

cuda

USAGE : **cuda** status

Switch CUDA GPU computations on/off.
Script return values: status

curietemperature

USAGE : **curietemperature** curie_temperature (meshname)

Set Curie temperature (all ferromagnetic meshes if meshname not given) for ferromagnetic mesh. This will set default temperature dependencies as: Ms = Ms0*me, A = Ah*me^2, D = D0*me^2, K = K0*me^3 (K1 and K2), damping = damping0*(1-T/3Tc) T < Tc, damping = damping0*2T/3Tc T >= Tc, susrel = dme/d(mu0Hext). Setting the Curie temperature to zero will disable temperature dependence for these parameters.
Script return values: curie_temperature - Curie temperature for mesh in focus.

data

USAGE : **data**

Shows list of currently set output data and available data.
Script return values: number of set output data fields

default

USAGE : **default**

Reset program to default state.

deldata

USAGE : **deldata** index

Delete data from list of output data at index number. If index number is -1 then delete all data fields, leaving just a default time data field - there must always be at least 1 output data field.

delelectrode

USAGE : **delelectrode** index

Delete electrode with given index.

delequationconstant

USAGE : **delequationconstant** name

Delete named user constant used in text equations.

delmdbentry

USAGE : **delmdbentry** materialname

Delete entry in the local materials database (see materialsdatabase for current selection).

delmesh

USAGE : **delmesh** name

Delete mesh with given name.

delmodule

USAGE : **delmodule** meshname handle

Delete module with given handle from named mesh.

delpinneddata

USAGE : **delpinneddata** index

Delete entry in data box at given index (index in order of appearance in data box from 0 up).

delrect

USAGE : **delrect** rectangle (meshname)

Void rectangle (m) within given mesh (active mesh if name not given). The rectangle coordinates are relative to specified mesh.

delstage

USAGE : **delstage** index

Delete stage from simulation schedule at index number. If index number is -1 then delete all stages, leaving just a default Relax stage - there must always be at least 1 stage set.

designateground

USAGE : **designateground** electrode_index

Change ground designation for electrode with given index.

display

USAGE : **display** name (meshname)

Change quantity to display for given mesh (active mesh if name not given).

displaybackground

USAGE : **displaybackground** name (meshname)

Change background quantity to display for given mesh (active mesh if name not given).

displaythresholds

USAGE : **displaythresholds** minimum maximum

Set thresholds for foreground mesh display : magnitude values outside this range are not rendered. If both set to 0 then thresholds are ignored.

displaythresholdtrigger

USAGE : **displaythresholdtrigger** trigtype

For vector quantities, set component to trigger thresholds on. trigtype = 1 (x component), trigtype = 2 (y component), trigtype = 3 (z component), trigtype = 5 (magnitude only)

displaytransparency

USAGE : **displaytransparency** foreground background

Set alpha transparency for display. Values range from 0 (fully transparent) to 1 (opaque). This is applicable in dual display mode when we have a background and foreground for the same mesh.

dmcellsize

USAGE : **dmcellsize** value

Change demagnetizing field macrocell size of mesh in focus, for atomistic meshes (m). The cellsize can be specified as: hx hy hz, or as: hxyz
Script return values: cellsize - return demagnetizing field macrocell size.

dp_add

USAGE : **dp_add** dp_source value (dp_dest)

Add value to dp array and place it in destination (or at same position if destination not specified).

dp_adddp

USAGE : **dp_adddp** dp_x1 dp_x2 dp_dest

Add dp arrays : dp_dest = dp_x1 + dp_x2

dp_append

USAGE : **dp_append** dp_original dp_new

Append data from dp_new to the end of dp_original.

dp_calcexchange

USAGE : **dp_calcexchange**

Calculate spatial dependence of exchange energy density for the focused mesh (must be magnetic and have an exchange module enabled). Output available in Cust_S.

dp_calcsot

USAGE : **dp_calcsot** hm_mesh fm_mesh

For the given heavy metal and ferromagnetic meshes calculate the expected effective spin Hall angle and field-like torque coefficient according to analytical equations (see manual).
Script return values: SHAeff, flST.

dp_calctopochargedensity

USAGE : **dp_calctopochargedensity**

Calculate topological charge density spatial dependence for the focused mesh (must be magnetic). Output available in Cust_S.

dp_cartesiantopolar

USAGE : **dp_cartesiantopolar** dp_in_x dp_in_y (dp_out_r dp_out_theta)

Convert from Cartesian coordinates (x,y) to polar (r, theta).

dp_clear

USAGE : **dp_clear** indexes...

Clear dp arrays with specified indexes.

dp_clearall

USAGE : **dp_clearall**

Clear all dp arrays.

dp_coercivity

USAGE : **dp_coercivity** dp_index_x dp_index_y

Obtain coercivity from x-y data: find first crossings of x axis in the two possible directions, with uncertainty obtained from step size.
Script return values: Hc_up Hc_up_err- Hc_up_err+ Hc_dn Hc_dn_err- Hc_dn_err+.

dp_completehysteresis

USAGE : **dp_completehysteresis** dp_index_x dp_index_y

For a hysteresis loop with only one branch continue it by constructing the other direction branch (invert both x and y data and add it in continuation) - use only with hysteresis loops which are expected to be symmetric.

dp_countskyrmions

USAGE : **dp_countskyrmions** (x y radius)

Calculate the number of skyrmions for focused mesh (must be magnetic), optionally in the given circle with radius and centered at x y (relative values). Use Qmag = Integral(|m.(dm/dx x dm/dy)| dxdy) / 4PI.
Script return values: Q - the calculated topological charge.

dp_crossingsfrequency

USAGE : **dp_crossingsfrequency** dp_in_x dp_in_y dp_level dp_freq_up dp_freq_dn (steps)

From input x-y data build a histogram of average frequency the x-y data crosses a given line (up and down, separated). The line varies between minimum and maximum of y data in given number of steps (100 by default). Output the line values in dp_level with corresponding crossings frequencies in dp_freq_up and dp_freq_dn.

dp_crossingshistogram

USAGE : **dp_crossingshistogram** dp_in_x dp_in_y dp_level dp_counts (steps)

From input x-y data build a histogram of number of times x-y data crosses a given line (up or down). The line varies between minimum and maximum of y data in given number of steps (100 by default). Output the line values in dp_level with corresponding number of crossings in dp_counts.

dp_div

USAGE : **dp_div** dp_source value (dp_dest)

Divide dp array by value and place it in destination (or at same position if destination not specified).

dp_divdp

USAGE : **dp_divdp** dp_x1 dp_x2 dp_dest

Divide dp arrays : dp_dest = dp_x1 / dp_x2

dp_dotprod

USAGE : **dp_dotprod** dp_vector ux uy uz dp_out

Take dot product of (ux, uy, uz) with vectors in dp arrays dp_vector, dp_vector + 1, dp_vector + 2 and place result in dp_out.

dp_dotproddp

USAGE : **dp_dotproddp** dp_x1 dp_x2

Take dot product of dp arrays : value = dp_x1.dp_x2

dp_dumptdep

USAGE : **dp_dumptdep** meshname paramname max_temperature dp_index

Get temperature dependence of named parameter from named mesh up to max_temperature, at dp_index - temperature scaling values obtained.

dp_erase

USAGE : **dp_erase** dp_index start_index length

From dp_index array erase a number of points - length - starting at start_index.

dp_extract

USAGE : **dp_extract** dp_in dp_out start_index (length)

From dp_in array extract a number of points - length - starting at start_index, and place them in dp_out.

dp_fitadiabatic

USAGE : **dp_fitadiabatic** (abs_err Rsq T_ratio (stencil))

Fit the computed self-consistent spin torque (see below) using Zhang-Li STT with fitting parameters P and beta (non-adiabaticity) using a given square in-plane stencil (default size 3) in order to extract the spatial variation of P. Cut-off values for absolute fitting error (default 0.1), Rsq measure (default 0.9), and normalized torque magnitude (default 0.1) can be set - value of zero disables cutoff. The focused mesh must be ferromagnetic, have the transport module set with spin solver enabled, and we also require Jc and either Ts or Tsi to have been computed. The fitting is done on Ts, Tsi, or on their sum depending if they've been enabled or not. Output available in Cust_S.

dp_fitlorentz

USAGE : **dp_fitlorentz** dp_x dp_y

Fit Lorentz peak function to x y data : f(x) = y0 + S dH / (4(x-H0)^2 + dH^2).
Script return values: S, H0, dH, y0, std_S, std_H0, std_dH, std_y0.

dp_fitlorentz2

USAGE : **dp_fitlorentz2** dp_x dp_y

Fit Lorentz peak function with both symmetric and asymmetric parts to x y data : f(x) = y0 + S (dH + A * (x - H0)) / (4(x-H0)^2 + dH^2).
Script return values: S, A, H0, dH, y0, std_S, std_A, std_H0, std_dH, std_y0.

dp_fitnonadiabatic

USAGE : **dp_fitnonadiabatic** (abs_err Rsq T_ratio (stencil))

Fit the computed self-consistent spin torque (see below) using Zhang-Li STT with fitting parameters P and beta (non-adiabaticity) using a given square in-plane stencil (default size 3) in order to extract the spatial variation of beta. Cut-off values for absolute fitting error (default 0.1), Rsq measure (default 0.9), and normalized torque magnitude (default 0.1) can be set - value of zero disables cutoff. The focused mesh must be ferromagnetic, have the transport module set with spin solver enabled, and we also require Jc and either Ts or Tsi to have been computed. The fitting is done on Ts, Tsi, or on their sum depending if they've been enabled or not. Output available in Cust_S.

dp_fitskyrmion

USAGE : **dp_fitskyrmion** dp_x dp_y

Fit skyrmion z component to obtain radius and center position : Mz(x) = Ms * cos(2*arctan(sinh(R/w)/sinh((x-x0)/w))).
Script return values: R, x0, Ms, w, std_R, std_x0, std_Ms, std_w.

dp_fitsot

USAGE : **dp_fitsot** hm_mesh (rectangle)

Fit the computed self-consistent interfacial spin torque using SOT with fitting parameters SHAeff and flST (field-like torque coefficient). hm_mesh specifies the heavy metal mesh from which to obtain the current density. The fitting is done inside the specified rectangle for the focused mesh, with the rectangle specified using relative coordinates as sx sy sz ex ey ez (entire mesh if not specified). The focused mesh must be ferromagnetic, have the transport module set with spin solver enabled, and we also require Jc and Tsi to have been computed.
Script return values: SHAeff, flST, std_SHAeff, std_flST, Rsq.

dp_fitsotstt

USAGE : **dp_fitsotstt** hm_mesh (rectangle)

Fit the computed self-consistent spin torque (see below) using Zhang-Li STT with fitting parameters P and beta (non-adiabaticity), and simultaneously also using SOT with fitting parameters SHAeff and flST (field-like torque coefficient). hm_mesh specifies the heavy metal mesh from which to obtain the current density for SOT. The fitting is done inside the specified rectangle for the focused mesh, with

the rectangle specified using relative coordinates as sx sy sz ex ey ez (entire mesh if not specified). The focused mesh must be ferromagnetic, have the transport module set with spin solver enabled, and we also require Jc and either Ts or Tsi to have been computed to have been computed. The fitting is done on Ts, Tsi, or on their sum depending if they've been enabled or not.
Script return values: SHAeff, flST, P, beta, std_SHAeff, std_flST, std_P, std_beta, Rsq.

dp_fitstt

USAGE : **dp_fitstt** (rectangle)

Fit the computed self-consistent spin torque (see below) using Zhang-Li STT with fitting parameters P and beta (non-adiabaticity). The fitting is done inside the specified rectangle for the focused mesh, with the rectangle specified using relative coordinates as sx sy sz ex ey ez (entire mesh if not specified). The focused mesh must be ferromagnetic, have the transport module set with spin solver enabled, and we also require Jc and either Ts or Tsi to have been computed. The fitting is done on Ts, Tsi, or on their sum depending if they've been enabled or not.
Script return values: P, beta, std_P, std_beta, Rsq.

dp_get

USAGE : **dp_get** dp_arr index

Show value in dp_arr at given index - the index must be within the dp_arr size.
Script return values: value

dp_getampli

USAGE : **dp_getampli** dp_source pointsPeriod

Obtain maximum amplitude obtained every pointsPeriod points.
Script return values: amplitude.

dp_getexactprofile

USAGE : **dp_getexactprofile** start end step dp_index

Extract profile of physical quantity displayed on screen, directly from the mesh so using the exact mesh resolution not the displayed resolution, along the line specified with given start and end cartesian absolute coordinates (m), and with the given step size (m). If stencil specified - as x y z (m) - then obtain profile values using weighted averaging with stencil centered on profile point. Place profile

in given dp arrays: 4 consecutive dp arrays are used, first for distance along line, the next 3 for physical quantity so allow space for these starting at dp_index.

dp_getpath

USAGE : **dp_getpath** dp_index_in dp_index_out

Extract profile of physical quantity displayed on screen, directly from stored mesh data thus independent of display resolution, along the path specified in Cartesian absolute coordinates (m) through dp arrays at dp_index_in, dp_index_in + 1, dp_index_in + 2 (x, y, z coordinates resp.). Place extracted profile in given dp arrays dp_index_out, dp_index_out + 1, dp_index_out + 2 (x, y, z components for vector data).

dp_getprofile

USAGE : **dp_getprofile** start end dp_index

Extract profile of physical quantity displayed on screen, at the current display resolution, along the line specified with given start and end cartesian absolute coordinates (m). Place profile in given dp arrays: 4 consecutive dp arrays are used, first for distance along line, the next 3 for physical quantity so allow space for these starting at dp_index.

dp_histogram

USAGE : **dp_histogram** dp_x dp_y (bin min max)

Calculate a histogram with given bin, minimum and maximum values, from the magnetisation magnitude of the focused mesh (must be magnetic). Save histogram in dp arrays at dp_x, dp_y. If histogram parameters not given use a bin with 100 steps between minimum and maximum magnetisation magnitude.

dp_histogram2

USAGE : **dp_histogram2** dp_x dp_y (bin min max M2 deltaM2)

Calculate a histogram for a 2-sublattice mesh with given bin, minimum and maximum values for sub-lattice A, if the corresponding magnetisation magnitude in sub-lattice B equals M2 within the given deltaM2. Save histogram in dp arrays at dp_x, dp_y. If histogram parameters not given use a bin with 100 steps between minimum and maximum magnetisation magnitude, with M2 set to MeB and deltaM2 set 0.01*MeB respectively.

dp_linreg

USAGE : **dp_linreg** dp_index_x dp_index_y (dp_index_z dp_index_out)

Fit using linear regression to obtain gradient and intercept with their uncertainties. If dp_index_z is specified multiple linear regressions are performed on adjacent data points with same z value; output in 5 dp arrays starting at dp_index_out as: z g g_err c c_err.
Script return values: g g_err c c_err.

dp_load

USAGE : **dp_load** (directory\)filename file_indexes... dp_indexes...

Load data columns from filename into dp arrays. file_indexes are the column indexes in filename (.txt termination by default), dp_indexes are used for the dp arrays; count from 0. If directory not specified, the default one is used.

dp_mean

USAGE : **dp_mean dp**_index

Obtain mean value with standard deviation.
Script return values: mean stdev.

dp_minmax

USAGE : **dp_minmax** dp_index

Obtain absolute minimum and maximum values, together with their index position.
Script return values: min_value min_index max_value max_index.

dp_monotonic

USAGE : **dp_monotonic** dp_in_x dp_in_y dp_out_x dp_out_y

From input x-y data extract monotonic sequence and place it in output x y arrays.

dp_mul

USAGE : **dp_mul** dp_source value (dp_dest)

Multiply dp array with value and place it in destination (or at same position if destination not specified).

dp_muldp

USAGE : **dp_muldp** dp_x1 dp_x2 dp_dest

Multiply dp arrays : dp_dest = dp_x1 * dp_x2

dp_newfile

USAGE : **dp_newfile** (directory/)filename

Make new file, erasing any existing file with given name. If directory not specified, the default one is used.

dp_peaksfrequency

USAGE : **dp_peaksfrequency** dp_in_x dp_in_y dp_level dp_freq (steps)

From input x-y data build a histogram of average frequency of peaks in the x-y data in bands given by the number of steps. The bands vary between minimum and maximum of y data in given number of steps (100 by default). Output the line values in dp_level with corresponding peak frequencies in dp_freq.

dp_rarefy

USAGE : **dp_rarefy** dp_in dp_out (skip)

Pick elements from dp_in using the skip value (1 by default) and set them in dp_out; e.g. with skip = 2 every 3rd data point is picked. The default skip = 1 picks every other point.

dp_remanence

USAGE : **dp_remanence** dp_index_x dp_index_y

Obtain remanence from x-y data: find values at zero field in the two possible directions.
Script return values: Mr_up Mr_dn.

dp_removeoffset

USAGE : **dp_removeoffset** dp_index (dp_index_out)

Subtract the first point (the offset) from all the points in dp_index. If dp_index_out not specified then processed data overwrites dp_index.

dp_replacerepeats

USAGE : **dp_replacerepeats** dp_index (dp_index_out)

Replace repeated points from data in dp_index using linear interpolation: if two adjacent sets of repeated points found, replace repeats between the mid-points of the sets. If dp_index_out not specified then processed data overwrites dp_index.

dp_save

USAGE : **dp_save** (directory\)filename dp_indexes...

Save specified dp arrays in filename (.txt termination by default). If directory not specified, the default one is used. dp_indexes are used for the dp arrays; count from 0.

dp_saveappend

USAGE : **dp_saveappend** (directory\)filename dp_indexes...

Save specified dp arrays in filename (.txt termination by default) by appending at the end. If directory not specified, the default one is used. dp_indexes are used for the dp arrays; count from 0.

dp_saveappendasrow

USAGE : **dp_saveappendasrow** (directory\)filename dp_index

Save specified dp array in filename (.txt termination by default) as a single row with tab-spaced values, appending to end of file. If directory not specified, the default one is used.

dp_saveasrow

USAGE : **dp_saveasrow** (directory\)filename dp_index

Save specified dp array in filename (.txt termination by default) as a single row with tab-spaced values, appending to end of file. If directory not specified, the default one is used.

dp_sequence

USAGE : **dp_sequence** dp_index start_value increment points

Generate a sequence of data points in dp_index from start_value using increment.

dp_set

USAGE : **dp_set** dp_arr index value

Set value in dp_arr at given index - the index must be within the dp_arr size.

dp_showsizes

USAGE : **dp_showsizes** (dp_arr)

List sizes of all non-empty dp arrays, unless a specific dp_arr index is specified, in which case only show the size of dp_arr.
Script return values: dp_arr size if specified

dp_smooth

USAGE : **dp_smooth** dp_in dp_out window_size

Smooth data in dp_in using nearest-neighbor averaging with given window size, and place result in dp_out (must be different).
dp_sub

USAGE : **dp_sub** dp_source value (dp_dest)

Subtract value from dp array and place it in destination (or at same position if destination not specified).

dp_subdp

USAGE : **dp_subdp** dp_x1 dp_x2 dp_dest

Subtract dp arrays : dp_dest = dp_x1 - dp_x2

dp_topocharge

USAGE : **dp_topocharge** (x y radius)

Calculate the topological charge for focused mesh (must be magnetic), optionally in the given circle with radius and centered at x y (relative values). Q = Integral(m.(dm/dx x dm/dy) dxdy) / 4PI.
Script return values: Q - the calculated topological charge.

dwall

USAGE : **dwall** longitudinal transverse width position (meshname)

Create an idealised domain wall (tanh profile for longitudinal component, 1/cosh profile for transverse component) along the x-axis direction in the given mesh (active mesh if name not specified). For longitudinal and transverse specify the components of magnetisation as x, -x, y, -y, z, -z, i.e. specify using these string literals. For width and position use metric units.

ecellsize

USAGE : **ecellsize** value

Change cellsize of mesh in focus for electrical conduction (m). The cellsize can be specified as: hx hy hz, or as: hxyz
Script return values: cellsize - return electrical conduction cellsize of mesh in focus.

editdata

USAGE : **editdata** index dataname (meshname, (rectangle))

Edit entry in list of output data at given index in list. If applicable specify meshname and rectangle (m) in mesh. If not specified and required, active mesh is used with entire mesh rectangle.

editdatasave

USAGE : **editdatasave** index savetype (savevalue)

Edit data saving condition in simulation schedule. Use index < 0 to set condition for all stages.

editstage

USAGE : **editstage** index stagetype (meshname)

Edit stage type from simulation schedule at index number.

editstagestop

USAGE : **editstagestop** index stoptype (stopvalue)

Edit stage/step stopping condition in simulation schedule. Use index < 0 to set condition for all stages.

editstagevalue

USAGE : **editstagevalue** index value

Edit stage setting value in simulation schedule. The value type depends on the stage type.

electrodes

USAGE : **electrodes**

Show currently configured electrodes.

equationconstants

USAGE : **equationconstants** name value

Create or edit user constant to be used in text equations.

errorlog

USAGE : **errorlog** status

Set error log status.

escellsize

USAGE : **escellsize** value

Change cellsize for electric super-mesh (m). The cellsize can be specified as: hx hy hz, or as: hxyz
Script return values: cellsize - return cellsize for electric super-mesh.

evalspeedup

USAGE : **evalspeedup** status

!!!Experimental!!!Status levels: 0 (no speedup), 1 (accurate), 2 (aggressive), 3 (extreme).

exchangecoupledmeshes

USAGE : **exchangecoupledmeshes** status (meshname)

Set/unset direct exchange coupling to neighboring meshes : if neighboring ferromagnetic meshes touch the named mesh (set for focused mesh if meshname not given) then interface magnetic cells are direct exchange coupled to them.
Script return values: status

excludemulticonvdemag

USAGE : **excludemulticonvdemag** status meshname

Set exclusion status (0 or 1) of named mesh from multi-layered demag convolution.
Script return values: status

flusherrorlog

USAGE : **flusherrorlog**

Clear error log.

fmscellsize

USAGE : **fmscellsize** value

Change cellsize for ferromagnetic super-mesh (m). The cellsize can be specified as: hx hy hz, or as: hxyz
Script return values: cellsize - return cellsize for ferromagnetic super-mesh.

generate2dgrains

USAGE : **generate2dgrains** spacing (seed)

Generate 2D Voronoi cells in the xy plane at given average spacing. The seed is used for the pseudo-random number generator, 1 by default.

generate3dgrains

USAGE : **generate3dgrains** spacing (seed)

Generate 3D Voronoi cells at given average spacing. The seed is used for the pseudo-random number generator, 1 by default.

getvalue

USAGE : **getvalue** abspos

Get data value at abspos (absolute position in Cartesian coordinates) depending on currently displayed quantities.
Script return values: value

imagecropping

USAGE : **imagecropping** left bottom right top

Set cropping of saved mesh images using normalized left, bottom, right, top values: 0, 0 point is left, bottom of mesh window and 1, 1 is right, top of mesh window.

individualshape

USAGE : **individualmaskshape** status

When changing the shape inside a mesh, e.g. through a mask file, set this flag to true so the shape is applied only to the primary displayed physical quantity. If set to false then all relevant physical quantities are shaped.
Script return values: status

insulatingside

USAGE : **insulatingside** side_literal status (meshname)

Set temperature insulation (Neumann boundary condition) for named mesh side (active mesh if not given). side_literal : x, -x, y, -y, z, -z.
Script return values: status_x status_-x status_y status_-y status_z status_-z - insulating sides status for mesh in focus.

invertmag

USAGE : **invertmag** (components) (meshname)

Invert magnetisation direction. If mesh name not specified, the active mesh is used. You can choose to invert just one or two components instead of the entire vector: specify components as x y z, e.g. invertmag x

isrunning

USAGE : **isrunning**

Checks if the simulation is running and sends state value to the calling script.
Script return values: state - return simulation running state.

iterupdate

USAGE : **iterupdate** iterations

Update mesh display every given number of iterations during a simulation.
Script return values: iterations - return number of iterations for display update.

linkdtspeedup

USAGE : **linkdtspeedup** flag

Links speedup time-step to ODE time-step if set, else speedup time-step is independently controlled. Applicable in extreme mode only.

linkdtstochastic

USAGE : **linkdtstochastic** flag

Links stochastic time-step to ODE time-step if set, else stochastic time-step is independently controlled.

linkstochastic

USAGE : **linkstochastic** flag (meshname)

Links stochastic cellsize to magnetic cellsize if flag set to 1 for given mesh, else stochastic cellsize is independently controlled. If meshname not given set for all meshes.

loadmaskfile

USAGE : **loadmaskfile** (z_depth) (directory\)filename

Apply .png mask file to magnetization in active mesh (i.e. transfer shape from .png file to mesh - white means empty cells). If image is in grayscale then void cells up to given depth top down (z_depth > 0) or down up (z_depth < 0). If z-depth = 0 then void top down up to all z cells.

loadovf2disp

USAGE : **loadovf2disp** (directory\)filename

Load an OOMMF-style OVF 2.0 file containing mechanical displacement data, into the currently focused mesh (which must be ferromagnetic and have the melastic module enabled), mapping the data to the current mesh dimensions. From the mechanical displacement the strain tensor is calculated.

loadovf2mag

USAGE : **loadovf2mag** (renormalize_value) (directory\)filename

Load an OOMMF-style OVF 2.0 file containing magnetisation data, into the currently focused mesh (which must be ferromagnetic), mapping the data to the current mesh dimensions. By default the loaded data will not be renormalized: renormalize_value = 0. If a value is specified for renormalize_value, the loaded data will be renormalized to it (e.g. this would be an Ms value).

loadovf2mesh

USAGE : **loadovf2mesh** (renormalize_value) (directory\)filename

Load an OOMMF-style OVF 2.0 file containing 3-component vector data. This will create a new permalloy ferromagnetic mesh with dimensions and magnetization data obtained from the OVF 2.0 file. By default the loaded data will not be renormalized: renormalize_value = 0. If a value is specified for renormalize_value, the loaded data will be renormalized to it (e.g. this would be an Ms value).

loadovf2strain

USAGE : **loadovf2strain** (directory\)filename_diag filename_odiag

Load an OOMMF-style OVF 2.0 file containing strain tensor data, into the currently focused mesh (which must be ferromagnetic and have the melastic module enabled), mapping the data to the current mesh dimensions. The symmetric strain tensor is applicable for a cubic crystal, and has 3 diagonal component (specified in filename_diag with vector data as xx, yy, zz), and 3 off-diagonal components (specified in filename_odiag with vector data as yz, xz, xy).

loadsim

USAGE : **loadsim** (directory\)filename

Load simulation with given name.

makevideo

USAGE : **makevideo** (directory\)filebase fps quality

Make a video from .png files sharing the common filebase name. Make video at given fps and quality (0 to 5 worst to best).

manual

USAGE : **manual**

Opens Boris manual for current version.

matcurietemperature

USAGE : **matcurietemperature** curie_temperature (meshname)

Set indicative material Curie temperature for ferromagnetic mesh (focused ferromagnetic mesh if meshname not given). This is not used in calculations, but serves as an indicative value - set the actual Tc value with the curietemperature command.
Script return values: curie_temperature - Indicative material Curie temperature for mesh in focus.

materialsdatabase

USAGE : **materialsdatabase** (mdbname)

Switch materials database in use. This setting is not saved by savesim, so using loadsim doesn't affect this setting; default mdb set on program start.

mcellsize

USAGE : **mcellsize** value

Change cellsize of mesh in focus for mechanical solver (m). The cellsize can be specified as: hx hy hz, or as: hxyz
Script return values: cellsize - return mechanical cellsize of mesh in focus.

memory

USAGE : **memory**

Show CPU and GPU-addressable memory information (total and free).

mesh

USAGE : **mesh**

Display information for all meshes.

meshfocus

USAGE : **meshfocus** meshname

Change mesh focus to given mesh name.
Script return values: meshname - return name of mesh in focus.

meshfocus2

USAGE : **meshfocus2** meshname

Change mesh focus to given mesh name but do not change camera orientation.
Script return values: meshname - return name of mesh in focus.

meshrect

USAGE : **meshrect** rectangle

Change rectangle of mesh in focus (m). The rectangle can be specified as: sx sy sz ex ey ez for the start and end points in Cartesian coordinates, or as: ex ey ez with the start point as the origin.
Script return values: rectangle - return rectangle of mesh in focus.

mirrormag

USAGE : **mirrormag** axis (meshname)

Mirror magnetisation in a given axis, specified as x, y, z, e.g. mirrormag x. If mesh name not specified, the active mesh is used

modules

USAGE : **modules**

Show interactive list of available and currently set modules.

movingmesh

USAGE : **movingmesh** status_or_meshname

Set/unset trigger for movingmesh algorithm. If status_or_meshname = 0 then turn off, if status_or_meshname = 1 then turn on with trigger set on first ferromagnetic mesh, else status_or_meshname should specify the mesh name to use as trigger.

movingmeshasym

USAGE : **movingmeshasym** status

Change symmetry type for moving mesh algorithm: 1 for antisymmetric (domain walls), 0 for symmetric (skyrmions).
Script return values: status

movingmeshthresh

USAGE : **movingmeshthresh** value

Set threshold used to trigger a mesh shift for moving mesh algorithm - normalised value between 0 and 1.
Script return values: threshold

multiconvolution

USAGE : **multiconvolution** status

Switch between multi-layered convolution (true) and supermesh convolution (false).

ncommon

USAGE : **ncommon** sizes

Switch to multi-layered convolution and force it to user-defined discretisation, specifying sizes as nx ny nz.

ncommonstatus

USAGE : **ncommonstatus** status

Switch to multi-layered convolution and force it to user-defined discretisation (status = true), or default discretisation (status = false).

neelpreparemovingmesh

USAGE : **neelpreparemovingmesh** (meshname)

Setup the named mesh (or active mesh) for moving Neel domain wall simulations: 1) set movingmesh trigger, 2) set domain wall structure, 3) set dipoles left and right to remove end magnetic charges, 4) enable strayfield module.

ode

USAGE : **ode**

Show interactive list of available and currently set ODEs and evaluation methods.

params

USAGE : **params** (meshname)

List all material parameters. If meshname not given use the active mesh.

paramstemp

USAGE : **paramstemp** (meshname)

List all material parameters temperature dependence. If meshname not given use the active mesh.

paramsvar

USAGE : **paramsvar** (meshname)

List all material parameters spatial variation. If meshname not given use the active mesh.

pbc

USAGE : **pbc** meshname flag images

Set periodic boundary conditions for magnetization in given mesh (must be ferromagnetic). Flags specify types of perodic boundary conditions: x, y, or z; images specify the number of mesh images to use either side for the given direction when calculating the demagnetising kernel - a value of zero disables pbc. e.g. pbc x 10 sets x periodic boundary conditions with 10 images either side for the focused mesh; pbc x 0 clears pbc for the x axis.

preparemovingmesh

USAGE : **preparemovingmesh** (meshname)

Setup the named mesh (or active mesh) for moving transverse (or vortex) domain wall simulations: 1) set movingmesh trigger, 2) set domain wall structure, 3) set dipoles left and right to remove end magnetic charges, 4) enable strayfield module.

random

USAGE : **random** (meshname)

Set random magnetisation distribution in mesh. If mesh name not specified, set for focused mesh.

refineroughness

USAGE : **refineroughness** value (meshname)

Set roughness refinement cellsize divider in given mesh, i.e. cellsize used for roughness initialization is the ferromagnetic cellsize divided by value (3 components, so divide component by component).
Script return values: value - roughness refinement.

refreshmdb

USAGE : **refreshmdb**

Reload the local materials database (see materialsdatabase for current selection). This is useful if you modify the values in the materials database file externally.

refreshscreen

USAGE : **refreshscreen**

Refreshes entire screen.

renamemesh

USAGE : **renamemesh** (old_name) new_name

Rename mesh. If old_name not specified then the mesh in focus is renamed.

requestmdbsync

USAGE : **requestmdbsync** materialname (email)

Request the given entry in the local materials database is added to the online shared materials database. This must be a completed entry - see manual for instructions. The entry will be checked before being made available to all users through the online materials database. If you want to receive an update about the status of this request include an email address.

reset

USAGE : **reset**

Reset simulation state to the starting state.

resetmesh

USAGE : **resetmesh** (meshname)

Reset to constant magnetization in given mesh (active mesh if name not given).

robinalpha

USAGE : **robinalpha** robin_alpha (meshname)

Set alpha coefficient (all meshes if meshname not given) for Robin boundary conditions : flux normal = alpha * (T_boundary - T_ambient).

Script return values: robin_alpha - Robin alpha value for mesh in focus.

roughenmesh

USAGE : **roughenmesh** depth (axis, (seed))

Roughen active mesh to given depth (m) along a named axis (use axis = x, y, or z as literal, z by default). The seed is used for the pseudo-random number generator, 1 by default.

run

USAGE : **run**

Run simulation from current state.

savecomment

USAGE : **savecomment** (directory\)filename comment

Save comment in given file by appending to it.

savedatafile

USAGE : **savedatafile** (directory\)filename

Change output data file (and working directory if specified).
Script return values: filename

savedataflag

USAGE : **savedataflag** status

Set data saving flag status.
Script return values: status

saveimagefile

USAGE : **saveimagefile** (directory\)filename

Change image file base (and working directory if specified).
Script return values: filename

saveimageflag

USAGE : **saveimageflag** status

Set image saving flag status.
Script return values: status

savemeshimage

USAGE : **savemeshimage** ((directory\)filename)

Save currently displayed mesh image to given file (as .png). If directory not specified then default directory is used. If filename not specified then default image save file name is used.

saveovf2

USAGE : **saveovf2** (data_type) (directory\)filename

Save an OOMMF-style OVF 2.0 file containing data from the currently focused mesh. You can specify the data type as data_type = bin4 (single precision 4 bytes per float), data_type = bin8 (double precision 8 bytes per float), or data_type = text. By default bin8 is used.

saveovf2mag

USAGE : **saveovf2mag** (n) (data_type) (directory\)filename

Save an OOMMF-style OVF 2.0 file containing magnetisation data from the currently focused mesh (which must be ferromagnetic). You can normalize the data to Ms0 value by specifying the n flag (e.g. saveovf2mag n filename) - by default the data is not normalized. You can specify the data type as data_type = bin4 (single precision 4 bytes per float), data_type = bin8 (double precision 8 bytes per float), or data_type = text. By default bin8 is used.

saveovf2param

USAGE : **saveovf2param** (data_type) (meshname) paramname (directory\)filename

Save an OOMMF-style OVF 2.0 file containing the named parameter spatial variation data from the named mesh (currently focused mesh if not specified). You can specify the data type as data_type = bin4 (single precision 4 bytes per float), data_type = bin8 (double precision 8 bytes per float), or data_type = text. By default bin8 is used.

savesim

USAGE : **savesim** (directory\)filename

Save simulation with given name. If no name given, the last saved/loaded file name will be used.

scalemeshrects

USAGE : **scalemeshrects** status

When changing a mesh rectangle scale and shift all other mesh rectangles in proportion if status set.
Script return values: status

scellsize

USAGE : **scellsize** value

Change cellsize of mesh in focus for stochastic properties (m). The cellsize can be specified as: hx hy hz, or as: hxyz
Script return values: cellsize - return stochastic properties cellsize of mesh in focus.

scriptserver

USAGE : **scriptserver** status

Enable or disable the script communication server. When enabled the program will listen for commands received using network sockets on port 1542.

setafmesh

USAGE : **setafmesh** name rectangle

Set a single antiferromagnetic mesh (deleting all other meshes) with given name and rectangle (m). The rectangle can be specified as: sx sy sz ex ey ez for the start and end points in Cartesian coordinates, or as: ex ey ez with the start point as the origin.

setameshcubic

USAGE : **setameshcubic** name rectangle

Set a single atomistic mesh (deleting all other meshes) with simple cubic structure, with given name and rectangle (m). The rectangle can be specified as: sx sy sz ex ey ez for the start and end points in Cartesian coordinates, or as: ex ey ez with the start point as the origin.

setangle

USAGE : **setangle** polar azimuthal (meshname)

Set magnetisation angle in mesh uniformly using polar coordinates. If mesh name not specified, this is set for all ferromagnetic meshes.

setatomode

USAGE : **setatomode** equation evaluation

Set differential equation to solve in atomistic meshes, and method used to solve it (same method is applied to micromagnetic and atomistic meshes).

setcurrent

USAGE : **setcurrent** current

Set a constant current source with given value. The potential will be adjusted to keep this constant current.
Script return values: current

setdata

USAGE : **setdata** dataname (meshname, (rectangle))

Delete all currently set output data and set dataname to list of output data. If applicable specify meshname and rectangle (m) in mesh. If not specified and required, active mesh is used with entire mesh rectangle.

setdefaultelectrodes

USAGE : **setdefaultelectrodes**

Set electrodes at the x-axis ends of the given mesh, both set at 0V. Set the left-side electrode as the ground. Delete all other electrodes.

setdisplayedparamsvar

USAGE : **setdisplayedparamsvar** meshname paramname

Set param to display for given mesh when ParamVar display is enabled (to show spatial variation if any).

setdt

USAGE : **setdt** value

Set differential equation time-step (only applicable to fixed time-step methods).
Script return values: dT

setdtspeedup

USAGE : **setdtspeedup** value

Set time step for evaluation speedup, to be used in when in extreme mode.
Script return values: dTspeedup

setdtstoch

USAGE : **setdtstoch** value

Set time step for stochastic field generation.
Script return values: dTstoch

setelectrodepotential

USAGE : **setelectrodepotential** electrode_index potential

Set potential on electrode with given index.
Script return values: potential

setelectroderect

USAGE : **setelectroderect** electrode_index electrode_rect

Edit rectangle (m) for electrode with given index.

setfield

USAGE : **setfield** magnitude polar azimuthal (meshname)

Set uniform magnetic field (A/m) using polar coordinates. If mesh name not specified, this is set for all magnetic meshes - must have Zeeman module added.
Script return values: <Ha_x, Ha_y, Ha_z> - applied field in Cartesian coordinates for mesh in focus.

setheatdt

USAGE : **setheatdt** value

Set heat equation solver time step.
Script return values: value - heat equation time step.

setmaterial

USAGE : **setmaterial** name

Set a single mesh with material parameters loaded from the materials database (deleting all other meshes). The name is the material name as found in the mdb file (see materialsdatabase command); this also determines the type of mesh to create, as well as the created mesh name. The rectangle (m) can be specified as: sx sy sz ex ey ez for the start and end points in Cartesian coordinates, or as: ex ey ez with the start point as the origin.
Script return values: Script return values: meshname - return name of mesh just added (can differ from the material name).

setmesh

USAGE : **setmesh** name rectangle

Set a single ferromagnetic mesh (deleting all other meshes) with given name and rectangle (m). The rectangle can be specified as: sx sy sz ex ey ez for the start and end points in Cartesian coordinates, or as: ex ey ez with the start point as the origin.

setode

USAGE : **setode** equation evaluation

Set differential equation to solve in both micromagnetic and atomistic meshes, and method used to solve it (same method is applied to micromagnetic and atomistic meshes).

setodeeval

USAGE : **setodeeval** evaluation

Set differential equation method used to solve it (same method is applied to micromagnetic and atomistic meshes).

setparam

USAGE : **setparam** meshname paramname (value)

Set the named parameter to given value.
Script return values: value - return value of named parameter in named mesh.

setparamtemparray

USAGE : **setparamtemparray** meshname paramname filename

Set the named parameter temperature dependence using an array in the given mesh. This must contain temperature values and scaling coefficients. Load directly from a file (tab spaced).

setparamtempequation

USAGE : **setparamtempequation** meshname paramname text_equation

Set the named parameter temperature dependence equation for the named mesh.

setparamvar

USAGE : **setparamvar** meshname paramname generatorname (arguments...)

Set the named parameter spatial dependence for the named mesh using the given generator (including any required arguments for the generator - if not given, default values are used).

setpotential

USAGE : **setpotential** potential

Set a symmetric potential drop : -potential/2 for ground electrode, +potential/2 on all other electrodes.
Script return values: potential

setrect

USAGE : **setrect** polar azimuthal rectangle (meshname)

Set magnetisation angle in given rectangle of mesh (relative coordinates) uniformly using polar coordinates. If mesh name not specified, the active mesh is used.

setsordamping

USAGE : **setsordamping** damping_v damping_s

Set fixed damping values for SOR algorithm used to solve the Poisson equation for V (electrical potential) and S (spin accumulation) respectively.
Script return values: damping_v damping_s

setstage

USAGE : **setstage** stagetype (meshname)

Delete all currently set stages, and set a new generic stage type to the simulation schedule with name stagetype, specifying a meshname if needed (if not specified and required, active mesh is used).

setstress

USAGE : **setstress** magnitude polar azimuthal (meshname)

Set uniform mechanical stress (Pa) using polar coordinates. If mesh name not specified, this is set for all magnetic meshes - must have MElastic module added.
Script return values: <Tsig_x, Tsig_y, Tsig_z> - applied mechanical stress in Cartesian coordinates for mesh in focus.

showa

USAGE : **showa**

Show predicted exchange stiffness (J/m) value for current mesh in focus (must be atomistic), using formula A = J*n/2a, where n is the number of atomic moments per unit cell, and a is the atomic cell size.
Script return values: A

showdata

USAGE : **showdata** dataname (meshname, (rectangle))

Show value(s) for dataname. If applicable specify meshname and rectangle (m) in mesh. If not specified and required, active mesh is used with entire mesh rectangle.
Script return values: varies

showk

USAGE : **showk**

Show predicted uniaxial anisotropy (J/m^3) constant value for current mesh in focus (must be atomistic), using formula K = k*n/a^3, where n is the number of atomic moments per unit cell, and a is the atomic cell size.
Script return values: A

showlengths

USAGE : **showlengths**

Calculate a number of critical lengths for the focused mesh (must be ferromagnetic) to inform magnetisation cellsize selection. lex = sqrt(2 A / mu0 Ms^2) : exchange length, l_Bloch = sqrt(A / K1) : Bloch wall width, l_sky = PI D / 4 K1 : Neel skyrmion wall width.

showmcells

USAGE : **showmcells**

Show number of discretisation cells for magnetisation for focused mesh (must be ferromagnetic).
Script return values: n

showms

USAGE : **showms**

Show predicted saturation magnetisation (A/m) value for current mesh in focus (must be atomistic), using formula Ms = mu_s*n/a^3, where n is the number of atomic moments per unit cell, and a is the atomic cell size.
Script return values: Ms

showtc

USAGE : **showtc**

Show predicted Tc value (K) for current mesh in focus (must be atomistic), using formula Tc = J*e*z/3kB, where e is the spin-wave correction factor, and z is the coordination number.
Script return values: Tc

skyrmion

USAGE : **skyrmion** core chirality diameter position (meshname)

Create an idealised Neel-type skyrmion with given diameter and centre position in the x-y plane (2 relative coordinates needed only) of the given mesh (active mesh if name not specified). Core specifies the skyrmion core direction: -1 for down, 1 for up. Chirality specifies the radial direction rotation: 1 for towards core, -1 away from core. For diameter and position use metric units.

skyrmionbloch

USAGE : **skyrmionbloch** core chirality diameter position (meshname)

Create an idealised Bloch-type skyrmion with given diameter and centre position in the x-y plane (2 relative coordinates needed only) of the given mesh (active mesh if name not specified). Core specifies the skyrmion core direction: -1 for down, 1 for up. Chirality specifies the radial direction rotation: 1 for clockwise, -1 for anti-clockwise. For diameter and position use metric units.

skyrmionpreparemovingmesh

USAGE : skyrmionpreparemovingmesh (meshname)

Setup the named mesh (or active mesh) for moving skyrmion simulations: 1) set movingmesh trigger, 2) set domain wall structure, 3) set dipoles left and right to remove end magnetic charges, 4) enable strayfield module.

ssolverconfig

USAGE : **ssolverconfig** s_convergence_error (s_iters_timeout)

Set spin-transport solver convergence error and iterations for timeout (if given, else use default).
Script return values: s_convergence_error s_iters_timeout

stages

USAGE : **stages**

Shows list of currently set simulation stages and available stage types.
Script return values: number of set stages

startupscriptserver

USAGE : **startupscriptserver** status

Set startup script server flag.

startupupdatecheck

USAGE : **startupupdatecheck** status

Set startup update check flag.

statictransportsolver

USAGE : **statictransportsolver** status

If static transport solver is set, the transport solver is only iterated at the end of a stage or step. You should set a high iterations timeout if using this mode.
Script return values: status

stochastic

USAGE : **stochastic**

Shows stochasticity settings : stochastic cellsize for each mesh and related settings.

stop

USAGE : **stop**

Stop simulation without resetting it.

surfroughenjagged

USAGE : **surfroughenjagged** depth spacing (seed, (sides))

Roughen active mesh surfaces using a jagged pattern to given depth (m) and peak spacing (m). Roughen both sides by default, unless sides is specified as -z or z (string literal). The seed is used for the pseudo-random number generator, 1 by default.

tau

USAGE : **tau** tau_11 tau_22 (tau_12 tau_21) (meshname)

Set ratio of exchange parameters to critical temperature (Neel) (all antiferromagnetic meshes if meshname not given) for antiferromagnetic mesh. tau_11 and tau_22 are the intra-lattice contributions, tau_12 and tau_21 are the inter-lattice contributions.
Script return values: tau_11 tau_22 tau_12 tau_21

tcellsize

USAGE : **tcellsize** value

Change cellsize of mesh in focus for thermal conduction (m). The cellsize can be specified as: hx hy hz, or as: hxyz
Script return values: cellsize - return thermal conduction cellsize of mesh in focus.

temperature

USAGE : **temperature** value (meshname)

Set mesh base temperature (all meshes if meshname not given) and reset temperature. Also set ambient temperature if Heat module added. If the base temperature setting has a spatial dependence specified through cT, this command will take it into account but only if the Heat module is added. If you want the temperature to remain fixed you can still have the Heat module enabled but disable the heat equation by setting the heat dT to zero (setheatdt 0).
Script return values: value - temperature value for mesh in focus.

tmodel

USAGE : **tmodel** num_temperatures (meshname)

Set temperature model (determined by number of temperatures) in given meshname (focused mesh if meshname not given). Note insulating meshes only allow a 1-temperature model.

tsolverconfig

USAGE : **tsolverconfig** convergence_error (iters_timeout)

Set transport solver convergence error and iterations for timeout (if given, else use default).
Script return values: convergence_error iters_timeout

updatemdb

USAGE : **updatemdb**

Switch to, and update the local materials database from the online shared materials database.

updatescreen

USAGE : **updatescreen**

Updates all displayed values on screen and also refreshes.

vecrep

USAGE : vecrep meshname vecreptype

Set representation type for vectorial quantities in named mesh (or supermesh). vecreptype = 0 (full), vecreptype = 1 (x component), vecreptype = 2 (y component), vecreptype = 3 (z component), vecreptype = 4 (direction only), vecreptype = 5 (magnitude only).

vortex

USAGE : **vortex** longitudinal rotation core (rectangle) (meshname)

Create a vortex domain wall with settings: longitudinal (-1: tail-to-tail, 1: head-to-head), rotation (-1: clockwise, 1: counter-clockwise), core (-1: down, 1: up). The vortex may be set in the given rectangle (entire mesh if not given), in the given mesh (focused mesh if not given).

Selected Publications using Boris

1. M.M. Vopson, M. Belusky, and S. Lepadatu “Diamagnetic coupling for magnetic tuning in nano-thin films” Applied Physics Letters **116**, 252402 (2020)

2. S. Lepadatu, “Efficient computation of demagnetizing fields for magnetic multilayers using multi-layered convolution” Journal of Applied Physics **126**, 103903 (2019)

3. M. Belusky, S. Lepadatu, J. Naylor, M.M. Vopson, “Study of roughness effect in Fe and Co thin films prepared by plasma magnetron sputtering” Physica B **574**, 411666 (2019)

4. S. Lepadatu, “Effect of inter-layer spin diffusion on skyrmion motion in magnetic multilayers” Scientific Reports **9**, 9592 (2019)

5. M. Belusky, S. Lepadatu, J. Naylor, M.M. Vopson, “Evidence of substrate roughness surface induced magnetic anisotropy in Ni80Fe20 flexible thin films” J. Magn. Magn. Mater. **478**, 77 (2019)

6. S. Lepadatu, “Unified treatment of spin torques using a coupled magnetization dynamics and three-dimensional spin current solver” Scientific Reports **7**, 12937 (2017)

7. M.M. Vopson, J. Naylor, T. Saengow, E.G.Rogers, S. Lepadatu, Y.K. Fetisov, "Development of flexible Ni80Fe20 magnetic nano-thin films" Physica B **525**, 12 (2017)

8. S. Lepadatu, M.M. Vopson, “Heat assisted multiferroic solid-state memory” Materials 10, **991** (2017)

9. S. Lepadatu, H. Saarikoski, R. Beacham, M.J.B. Romero, T.A. Moore, G. Burnell, S. Sugimoto, D. Yesudas, M.C. Wheeler, J. Miguel, S.S. Dhesi, D. McGrouther, S. McVitie, G. Tatara, and C.H. Marrows, “Very low critical current density for motion of

coupled domain walls in synthetic ferrimagnet nanowires" Scientific Reports **7**, 1640 (2017)

10. S. Lepadatu, "Interaction of Magnetization and Heat Dynamics for Pulsed Domain Wall Movement with Joule Heating" Journal of Applied Physics **120**, 163908 (2016)

11. S. Lepadatu, "Effective field model of roughness in magnetic nano-structures" Journal of Applied Physics **118**, 243908 (2015)

12. M.M. Vopson, S. Lepadatu, "Solving the electrical control of magnetic coercive field paradox" Appl. Phys. Lett. **105**, 122901 (2014)