Magnetoresistance Study of Skyrmionic MTJs

Report submitted in partial fulfillment
of
MTP Phase 1

of

MTech

by

Chaitanya Tejaswi
214102408

Under the guidance of

Dr. Tanmay Dutta

DEPARTMENT OF ELECTRONICS & ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

Department of Department of Electronics & Electrical Engineering
Indian Institute of Technology, Guwahati
India - 781039

CERTIFICATE

This is to certify that we have examined the thesis entitled **Magnetoresistance Study of Skyrmionic MTJs**, submitted by **Chaitanya Tejaswi**(Roll Number: *214102408*) a postgraduate student of **Department of Department of Electronics & Electrical Engineering** in partial fulfillment for the award of degree of MTech. We hereby accord our approval of it as a study carried out and presented in a manner required for its acceptance in partial fulfillment for the Post Graduate Degree for which it has been submitted. The thesis has fulfilled all the requirements as per the regulations of the Institute and has reached the standard needed for submission.

Dr. Tanmay Dutta

Department of Department of Electronics & Electrical Engineering
Indian Institute of Technology, Guwahati

Place: Guwahati
Date:

ACKNOWLEDGEMENTS

I express my sincere gratitude to my parents, my supervisor, Dr. Tanmay Dutta, and my friends for their continual support during my undertaking of this project work.

Chaitanya Tejaswi
IIT Guwahati
Date:

ABSTRACT

The electronics industry is always on the lookout for novel technologies for designing faster, denser, energy-efficient, robust computational devices. An active field of research is "Spintronics" - where we exploit the spin-properties of magnetic materials (alongwith the traditional charge) to create memory & computational blocks.

One such device is the Magnetic Tunnel Junction (MTJ) - a trilayer composed of 2 ferromagnetic thinfilms separated by an insulating metal oxide (such as Al_2O_3 or MgO). The spin-polarization of one of these layers ("free layer") can be controlled by sending a charge current through the other layer ("reference layer"). The spin-polarization (parallel or antiparallel) can be determined by measuring the potential developed across the free layer - yielding a resistance term - called the "magnetoresistance". The materials are chosen such that there is a large difference is measured magnetoresistance values for the parallel & antiparallel states. This way, a two state logic can be implemented using MTJs.

While the inherent magnetization of the material can be used as a state variable, topologically stable micro-magnetic textures such as "skyrmions", are being actively pursued for this in order to create miniaturized computational devices apart from just memories - as their presence affects the magnetization of the free layer.

Spintronics thus aims at studying magnetic structures (such as an MTJ) that can contain & manipulate the spin & charge of a particle-like object (such as a skyrmion) to realise computational blocks at-par with existing semiconductor CMOS technologies.

The aim of this study is to model & simulate MTJs (using micromagnetic calculations & spice), and implement digital logic blocks using the same.

Keywords: Spintronics, MTJ, Skyrmions, Micromagnetics, SPICE, Digital Logic.

Contents

List of Figures

Chapter 1

Introduction

While macroscopic forms of magnetization have been in use for a long time (eg. reading/writing to magnetic memories using external fields), practical applications of microscopic magnetization is a relatively new field.
The major-most breakthrough was in the 1980s when experiments were done at nanoscale to control charge transport through magnetization. This has been aided with advancements in nanoscale fabrication techniques (such as sputtering, molecular beam epitaxy). This has grown into the field of **Spintronics**, which deals with manipulating the spin angular-momentum of an electron to realise alternative microelectronic devices.

A skyrmion is one of the smallest observed topologically-stable magnetic textures ($\approx 14 - 250nm$). It's a point-like region of reversed magnetization - surrounded by a vortex of axially-symmetric twist of magnetization. **?** and **?** formulated the behaviour interaction of skyrmions in magnetic crystals and magnetic thinfilms respectively. These were extensively imaged in the works of **?**, **?**, and **?**. Their size & property of being manipulated by low-energy fields and electric currents makes them as ideal candidate for representing information.

Despite their stability, given their extremely small size, it isn't possible to manipulate them in in their native environment - crystal lattice or thinfilm. A pragmatic way of using them would be to contain them in a magnetic structure such an a Magnetic Tunnel Junction (MTJ) - a two-terminal device whose magnetoresistance may be varied (and easily detected by means of a terminal voltage) by sheer presence/absence of these skyrmions - thus representing a two-state logic. In addition, an ensemble of skyrmions may be used to vary the magnetoresistance of the MTJ in several quantized levels - yielding a multi-state

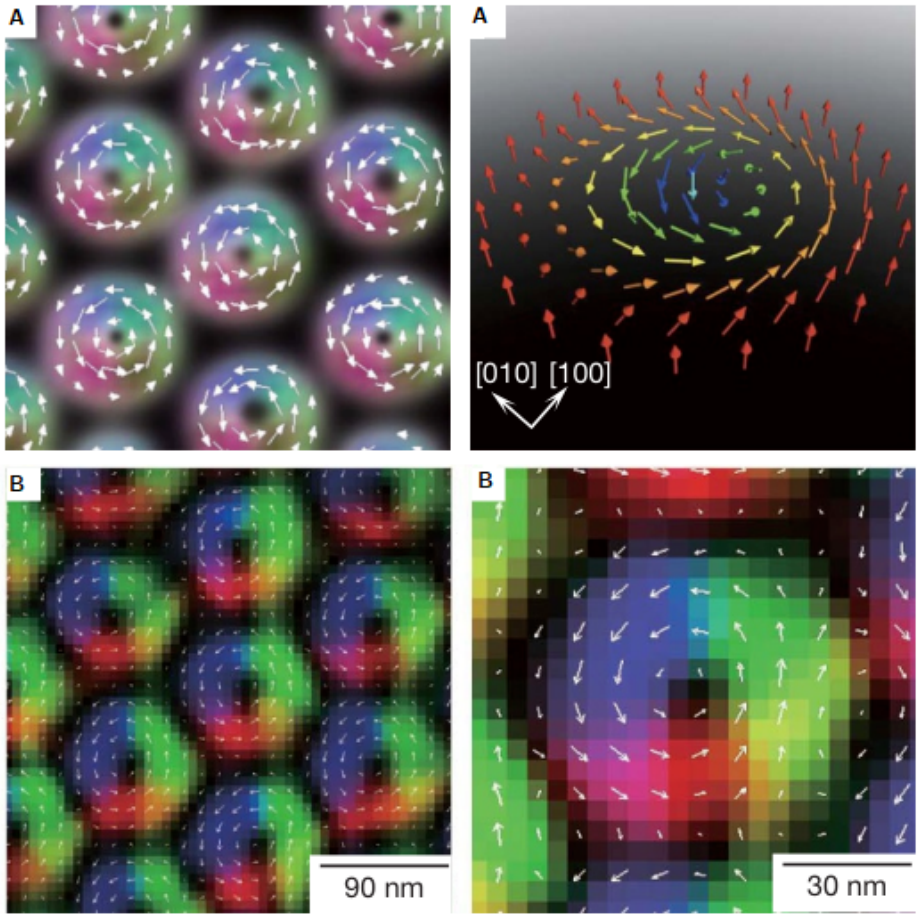


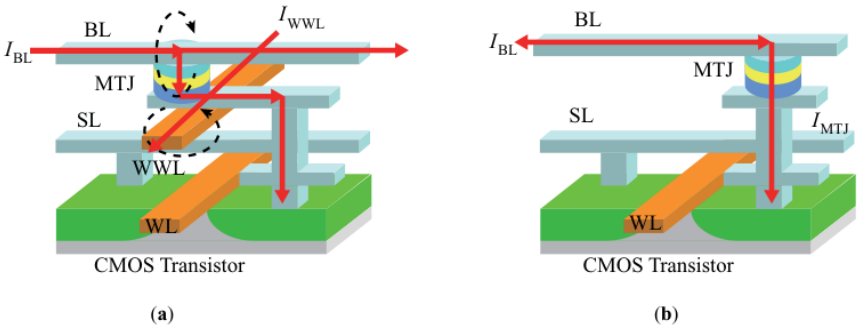
Figure 1.1: A: MicroMagnetic simulation of skyrmions
B: Experimentally-detected skyrmions in $Fe_{0.5}Co_{0.5}Si$ **?**

logic. The MTJ may itself be controlled by a wide variety of switching methods - field-induced, thermally assisted, spin torque transfer, voltage-controlled **?** - as desired by the use-case.

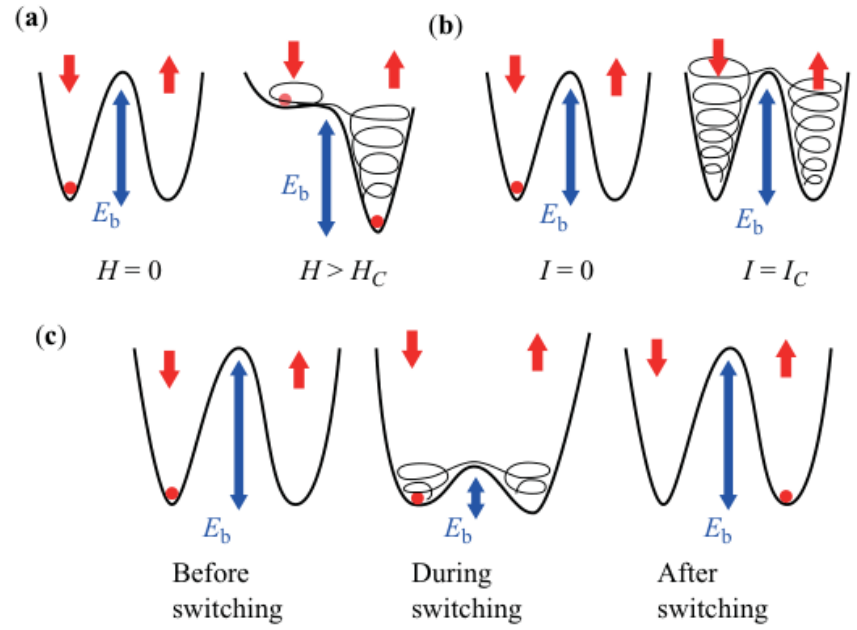
1.1 Giant Magnetoresistance (GMR)

Magnetoresistance is the change in the resistance offered by a magnetic sample in presence of an external magnetic field.

If we imagine two ferromagnets separated by a non-magnetic metal spacer, we observe two conditions: either their magnetizations are in parallel or antiparallel configuration.
In parallel configuration, when electrons are injected through the first ferromagnetic layer, they align along its magnetization, and flow through the second ferromagnet without much



Field-induced switching and spin transfer torque switching.



Schematic drawings of different switching methods: (a) field-induced switching; (b) STT switching; (c) voltage-controlled switching with a variable barrier height.

Figure 1.2: Various Switching methods for an MTJ **?**

scattering. The overall device resistance is low.

In antiparallel configuration, when electrons are injected through the first ferromagnetic layer, they align along its magnetization, but have a hard time penetrating through the second ferromagnet. The overall device resistance is high.

This is the GMR effect. Carrier transport is due to "diffusion".

GMR was first observed in spin-valves - trilayered stucture consisting of two ferromagnetic

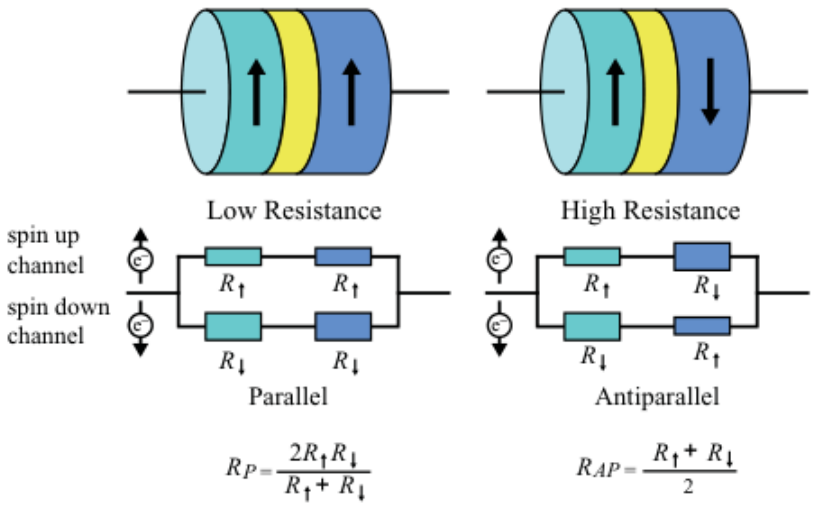


Figure 1.3: GMR: 2-Channel Model

layers - one of which is pinned using an antiferromagnetic layer while the magnetization of the other layer is free. Magnetization of free layer can be easily reversed using a small field. Spin valves have found applications in HDD read-heads. Albert Fert observed $\approx 50\%$ magnetoresistance in Fe/Cr crystal lattices grown by molecular beam epitaxy (MBE) which they coined GMR.

This can be explained by the "two-channel model" (mott1936), which reasons that electrons passing through a trilayer film will maintain their spin orientations provided the film is thin. Thus, we can think of charge transport in terms of separated spin-up and spin-down channels (neglecting spin-flips), and quantify resistances for parallel and anti-parallel states.

1.2 Tunneling Magnetoresistance (TMR)

A similar but larger variation in magnetoresistance (upto 600%) with thinfilms consisting of two ferromagnets separated by a thin insulating "barrier" layer ($\approx 1nm$) has been termed TMR. Carrier transport occurs due to "tunneling".

The resulting structure is termed "Magnetic Tunnel Junction (MTJ)", and the "barrier" layer is usually Al_2O_3 or MgO, although other materials have also been used.

As shown in figure, an MTJ has reference & free layers separated by a barrier layer. In addition, an antiferromagnetic layer (AFM) (eg. PtMn, IrMn) is needed to fix the magne-

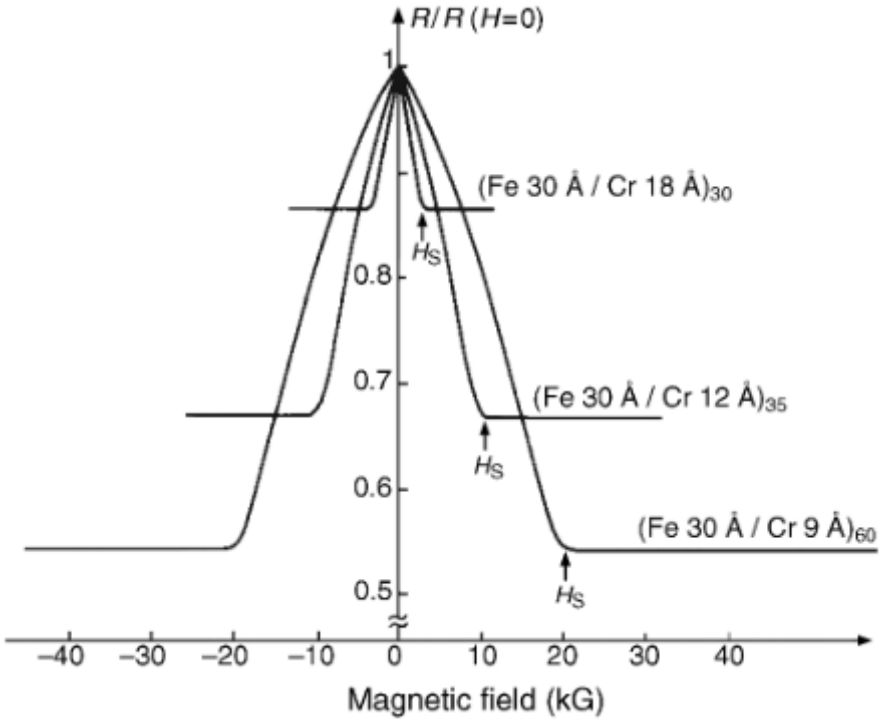


Figure 1.4: GMR: Albert Fert's GMR experiments in Fe/Cr lattices at 4.2K

tization of the reference layer.

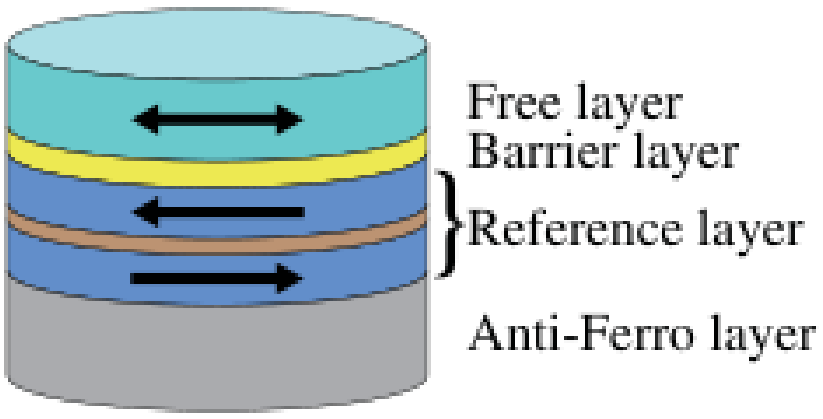


Figure 1.5: TMR: MTJ structure

The reason for high magnetoresistances in MTJs is attributed to the use of crystalline

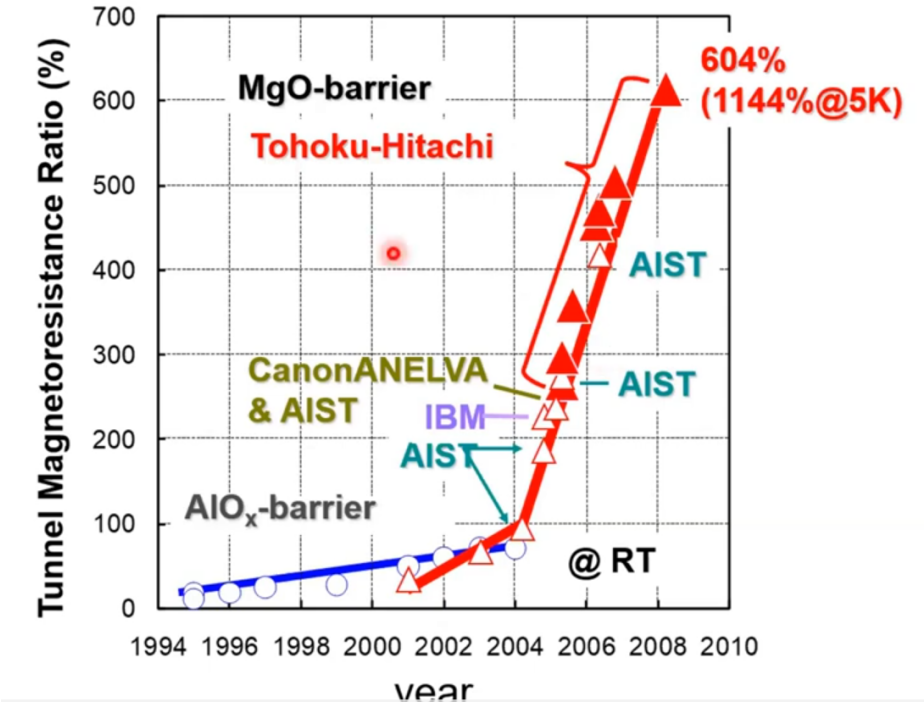


Figure 1.6: TMR: Magnetoresistances achieved using Al_2O_3 and MgO barrier layers

barrier layers (such as MgO) instead of amorphous ones (such as Al_2O_3) which allows selective tunneling of certain electron states, which end up maintaining their spin-polarization as they reach the free layer.

1.3 Spin Transfer Torque (STT)

STT is the application of a spin-polarized current to control the magnetization of a ferromagnet. This was observed in 2000, in Co/Cu/Co nanostructures, but the switching currents were too high for practical use ($10^8 A$). STT was applied to MgO barrier based MTJs by Diao in 2005, where the switching currents were lowered to $3 \times 10^6 A$ - an order of 10^2. STT is applicable to GMR & TMR based structures - ie, spinvalves & MTJs.

STT is attributed to 4s-3d exchange interaction - free carrier 4s electrons interacting with fixed 3d electrons of the free layer ferromagnet (eg. Co), resulting in the 4s electrons aligning their spins parallel to that of the free layer.

By conservation of angular momentum, the energy lost by 4s electrons in this process is transferred to the free layer, which experiences a torque, forcing it to slowly align its spin

polarization parallel to the incoming 4s electrons - ie, parallel to that of the reference layer.

1.4 In-Plane & Out-of-Plane MTJs

In in-plane devices (eg. iMTJ), magnetization is in the plane of layer, while in out-of-plane or perpendicular-plane devices (eg. pMTJ), magnetization is in the plane perpendicular to that of the layer.

Since our goal with MTJs is to create devices that can be downscaled & controlled using a lower spin-polarized current, pMTJs have (particularly CoFeB/MgO/CoFeB) gained significant interest despite expensive fabrication.

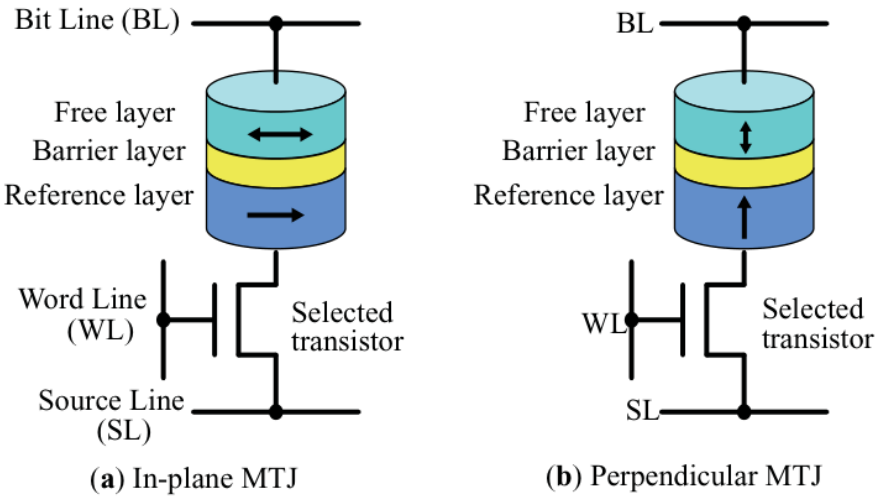


Figure 1.7: MTJ: iMTJ/pMTJ blocks from a memory array with read/write lines

In-Plane MTJs

$$H_K = 2\frac{4\pi M_S t(\mathrm{AR}-1)}{w\mathrm{AR}}$$

$$\Delta_{IP} = \frac{\pi^2 (M_S t)^2 w(\mathrm{AR}-1)}{k_B T}$$

$$J_{c0} = \frac{1}{\eta}\frac{2\alpha e}{\hbar}(M_S t)\left(\frac{1}{2}(4\pi M_{\mathrm{eff}}) + H_K\right)$$

$$I_{c0} = \left[\frac{4ek_B T}{\hbar}\right]\frac{\alpha}{\eta}\Delta\left(1+\frac{4\pi M_{\mathrm{eff}}}{2H_K}\right)$$

thermal stability factor Δ

$$\Delta = \frac{E_b}{k_B T} = \frac{H_K M_S V}{2k_B T}$$

Perpendicular MTJs

$$H_K = \begin{cases} \dfrac{2K_u^{\mathrm{bulk}}}{M_S} - 4\pi M_S & (\mathrm{bulk}) \\ \dfrac{2\sigma}{M_S t} - 4\pi M_S & (\mathrm{interface}) \end{cases}$$

$$\Delta = \begin{cases} \left(K_u^{\mathrm{bulk}} t - 2\pi M_S^2 t\right)\dfrac{\pi A R w^2}{4k_B T} & (\mathrm{bulk}) \\ \left(\sigma - 2\pi M_S^2 t\right)\dfrac{\pi A R w^2}{4k_B T} & (\mathrm{interface}) \end{cases}$$

$$J_{c0} = \frac{1}{\eta}\frac{2\alpha e}{\hbar}(M_S t)H_K$$

$$I_{c0} = \left[\frac{4ek_B T}{\hbar}\right]\frac{\alpha}{\eta}\Delta$$

Figure 1.8: MTJ: Comparison of iMTJ/pMTJ blocks

Here:

I_{c0} = crictical switching current
J_{c0} = crictical switching current density
δ = thermal stability
H_K = anisotropy field

Chapter 2

MicroMagnetism

2.1 Magnetic Moment

The magnetic moment of an electron $\vec{\mu}$ is associated with its orbital angular momentum $\vec{l}$:
$\vec{\mu}_l = -\frac{e}{2m_e}\vec{l}$
The magnetic moment of an electron $\vec{\mu}$ is associated with its spin angular momentum $\vec{s}$:
$\vec{\mu}_s = -\frac{e}{2m_e}\vec{s}$
For hydrogen atom:
$\|\vec{l}\| = \sqrt{l(l+1)}\hbar$, and $l_z = m_l\hbar, m_l \in [-l, l]$
$\|\vec{s}\| = \sqrt{s(s+1)}\hbar$
The quantity $\mu_B = 5.78 \times 10^{-5} eV/T$ is termed Bohr Magneton, and it quantifies magnetic moments of various magnetic materials.
The net angular momentum is represented by $\vec{J} = \vec{L} + \vec{S}$. (representative of "spin-orbit coupling").
Above momentums are also written as : $\mu_{spin} = -\frac{e}{m_e}\frac{\hbar}{2}$, $\mu_{orbital} = -\frac{e}{2m_e}\hbar$:
to denoted the fact that electrons, being fermions, are quantized into $(2n+1)\hbar$ orbital & $n\hbar$ spin quantum levels.
For atoms & materials, we use the notation $\vec{m}$ instead of μ.
The **magnetization ($\vec{M}$)** of a material is the sum of all magnetic moments taken over the entire volume.
For example, the magnetization of a diamagnet is expressed by: (Langevin Theory)
$\vec{M} = -\frac{\mu_0 e^2}{6m_e}\vec{H}Z\frac{N_0\rho}{A}r^2$.

2.2 Magnetic Materials

Based on the kind of magnetic moment a material exhibits, it may be classified as a diamagnet, paramagnet, ferromagnet, ferrimagnet, and antiferromagnet. Each of these is quantified by a term called **Magnetic Susceptibility** (χ) which is the magnetization response to an applied magnetic field.

$\chi = \frac{\vec{M}}{\vec{H}}$

As shown, materials with $\chi \geq 1$ are ferromagnets, while diamagnets exhibit $\chi \approx -10^{-4}$. $\chi = -1$ is the characteristic of an ideal superconductor, which opposes and cancels out the applied field.

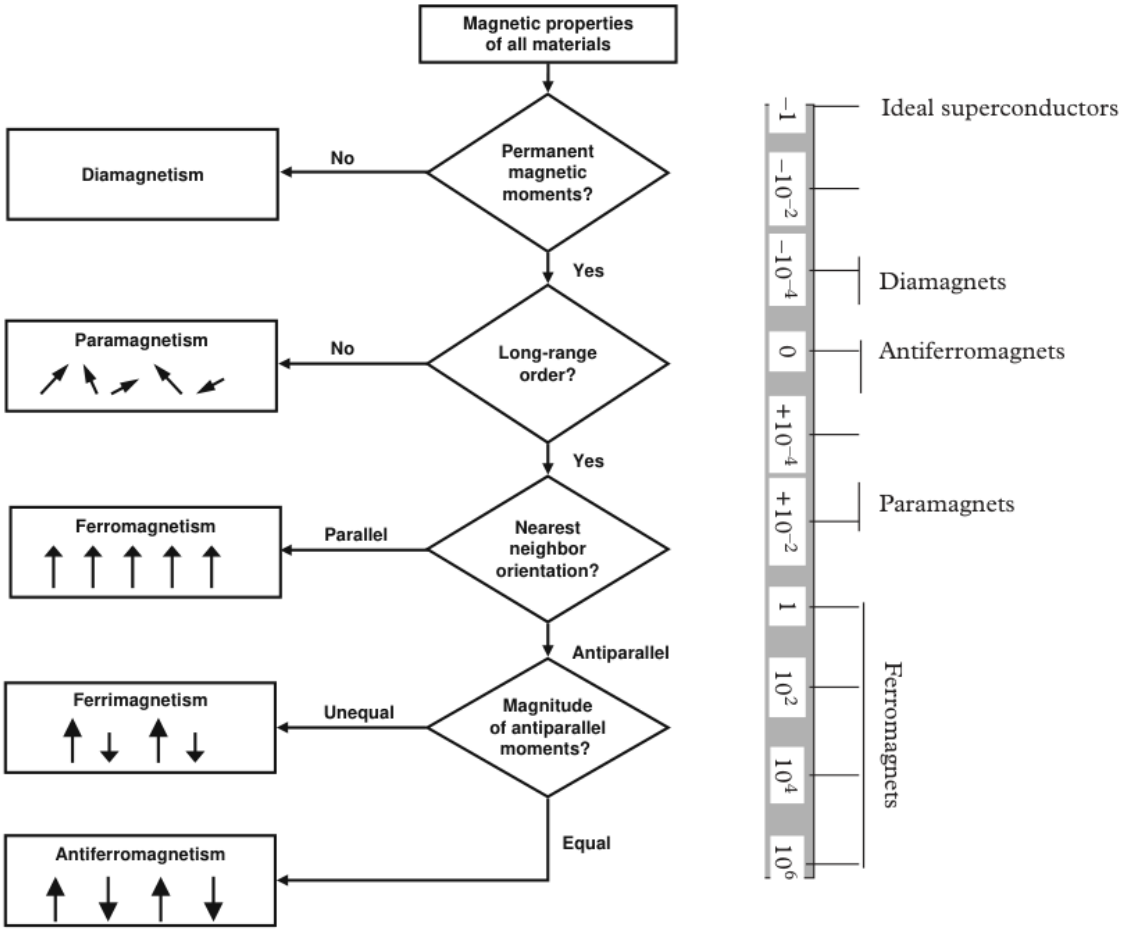


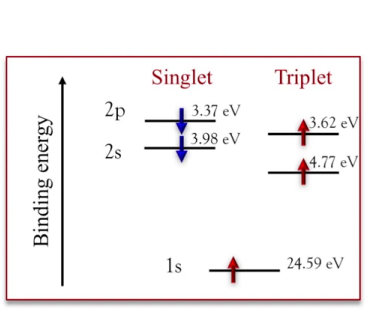
Figure 2.1: Types of Magnets

2.3 MicroMagnetic Energies

Based on interactions between electrons and associated magnetic fields, there are several interactions (and associated energies) operating at micro and nano-meter scales.

2.3.1 Fundamental Energies

1. Heisenberg Interaction (aka. Dipole Interaction or Coulombic Repulsion):
   This describes the interaction between two spins. It consists of two terms - "direct" and "exchange" interaction.
   For example, in the original 2-electron model for He, the total energy associated with interaction of two electrons is given by:
   $E = \frac{\vec{p_1}^2}{2m} + \frac{\vec{p_2}^2}{2m} - \frac{2e^2}{4\pi\epsilon_0 r_1} - \frac{2e^2}{4\pi\epsilon_0 r_2} + \frac{e^2}{4\pi\epsilon_0 r_{12}}$
   Here, terms 1-2 represent kinetic energy, terms 3-4 represent coulombic attraction (nucleus), term 5 represents coulombic repulsion (electron). Terms 1-4 thus represent the individual contributions ("direct"), while term 5 represents interaction ("exchange"). This is the term of our interest!
   If the consider the **groundstate** of He atom, we'll have two electrons aligned antiparallel ($\uparrow\downarrow$) in the 1s state.
   In an **excitedstate**, 1 electron is in the 1s state, while the other occupies either of 2s or 2p states. Depending on antiparallel or parallel alignment, we have two states - termed the "singlet" & "triplet" states respectively.
   Figure represents this situation alongwith energy expressions in terms of electron wavefunctions.



Energy of the excited states

$$\langle\psi_{ET}|\hat{H}_0|\psi_{ET}\rangle = \langle\psi_{ES}|\hat{H}_0|\psi_{ES}\rangle = E_{1s} + E_{nlm}$$

$$\uparrow\downarrow \quad \langle\psi_{ES}|\frac{e^2}{4\pi\epsilon_0 r_{12}}|\psi_{ES}\rangle = I + J$$

$$\uparrow\uparrow \quad \langle\psi_{ET}|\frac{e^2}{4\pi\epsilon_0 r_{12}}|\psi_{ET}\rangle = I - J$$

$$I = \frac{e^2}{4\pi\epsilon_0}\iint |\phi_{1s}(r_1)|^2 \frac{1}{r_{12}} |\phi_{nlm}(r_2)|^2 dr_1 dr_2$$

$$J = \frac{e^2}{4\pi\epsilon_0}\iint \phi_{1s}(r_1)\phi_{nlm}(r_2)\frac{1}{r_{12}}\phi^*_{1s}(r_2)\phi^*_{nlm}(r_1) dr_1 dr_2$$

Figure 2.2: Parallel & Antiparallel spin configurations of two interacting electrons.
I is the "direct interaction intergral", J is the "exchange interaction integral".
For $J > 0$, $\uparrow\uparrow$ represents lower energy, as validated by experimental values on the left.

2. Zeeman energy:
   the magnetization of a material tries to align itself parallel to the applied external field.

$$E_{zeeman} = -\vec{M}.\vec{B} = -\mu_0 M_s \vec{m}.\vec{H}$$

   where: $\vec{M}$ is the magnetization, $\vec{m}$ is the magnetization normalized to the saturation value - M_s, and $\vec{H}$ is the applied field.

3. Demagnetization:
   It's the magnetic field generated by the magnetization of a material (represented by $\vec{H_D}$). It's named so as it acts in a manner that reduces the overall magnetic moment.

$$E_{demagnetization} = -\frac{\mu_0}{2}\int \vec{m}\vec{H_D}dV$$

4. Spin-Orbit Coupling:
   An electron moving in an electric field experiences a magnetic field: $\vec{B} = \frac{1}{2c^2}\vec{v} \times \nabla V$. In turn, this field couples to the electron's magnetic moment: $E = -\vec{\mu}.\vec{B}$ This is the spin-orbit coupling. The associated field is given by:

$$\vec{H}_{soc} = -\frac{g\mu}{2mc^2}\vec{S}.(\nabla V \times \vec{p})$$

2.3.2 Exchange Energies

Ferromagnetic metals such as Fe, Co, Ni have a fixed 3d electron cloud, and sparsely populated 4s, p orbitals. Based on the kind of interaction, there is an overlap between interacting electrons giving rise to exchange energies.
For example, A typical ferromagnetic energy expression is:

$$E_{FM} = A\vec{m}\nabla^2\vec{m} = A(\nabla\vec{m})^2$$

where: A is the exchange constant (J/m).
The tendency is to align all magnetic moments parallel to each other.

1. Direct Exchange: In 3d metals, there is delocalization of electrons in empty/half-filled/fully-filled 3d shells.

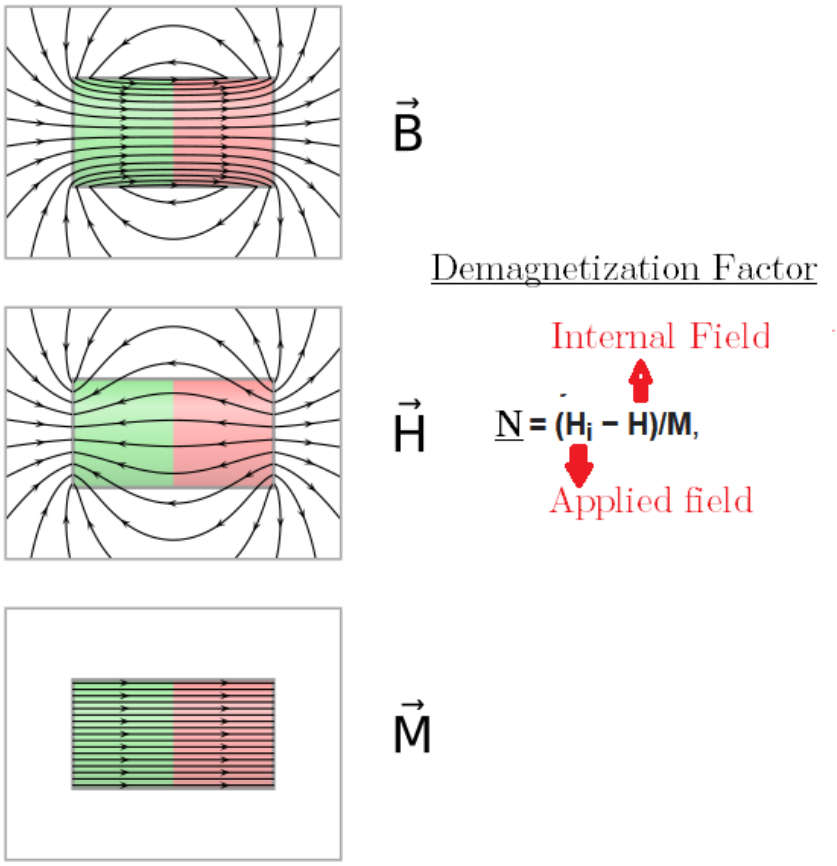


Figure 2.3: Demagnetization Field

2. s-d Exchange: As stated above, overlapping of electron wave functions of 3d-4s electrons, which gives rise to spin-polarization (for example, in case of STT-spinvalves & mtjs).

3. RKKY Exchange: Alike the 3d-4s interaction, here, localized electrons in 4f shells interact with 5d/6s conduction band electrons. This is a non-uniform, periodic, oscillatory form of spin polarization. It was originally observed when a single magnetic impurity was introduced into a 3d structure.

4. Double Exchange: This is observed in 3d ions with both localised and delocalised d electrons.

5. Super Exchange: In insulators such as metallic-oxides, there is little 3d-3d overlap. Instead, there's hybridization-based overlapping between 3d and oxygen's 2p orbitals - resulting in a stable 3d-2p-3d structure.

6. AntiSymmetric Exchange: In materials lacking symmetry, there's a weak coupling of spins leading to low energy magnetic structures. In materials lacking inversion-symmetry, the Dzyaloshinskii-Moriya Interaction (DMI) exchange results in perpendicular orientation of magnetic moments. At nanoscale, this results in formation of non-collinear magnetic structures such as skyrmions.

$$E_{DMI} = \pm\vec{D}(\vec{S_i} \times \vec{S_j})$$

where: $\vec{D}$ is a material constant, S is the spin angular momentum.

DMI is of typical interest to this study as it explains the formation of Bloch & Neel type skyrmions (bulk DMI & interficial DMI respectively). Figure explains this in some detail.

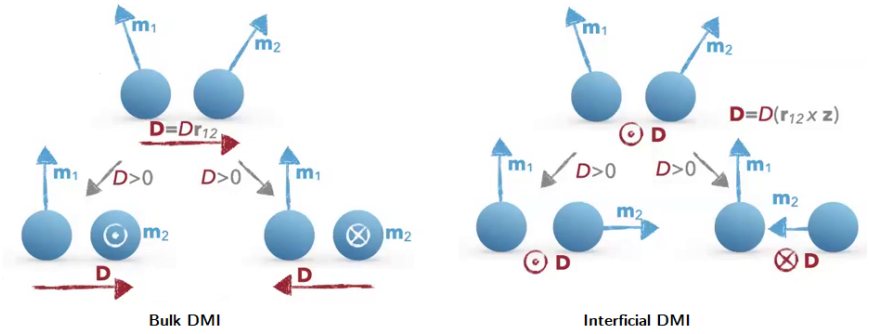


Figure 2.4: Bulk & Interficial DMI

2.3.3 Anisotropy Energies

Anisotropy is preferential magnetic behaviour depending on how the applied field is aligned with the material. The most common form of anisotropy is observed in magnetic crystalls in which it's easy to spin-polarize atoms along an axis or plane. Usually, there's two easy directions to magnetize the material, which are at 180° to each other. The line joining these is the "Easy Axis". Similary for "Hard Axis" - usually perpendicular to the "Easy Axis". A common way to quantify anisotropy is the Anisotropy Energy E_A, which is the difference in magnetostatic energy associated with easy/hard directions.

1. Shape Anisotropy: When a particle is not perfectly spherical, the demagnetizing field will not be equal for all directions, creating one or more easy axes.

Commonly we have **Uniaxial Easy-Axis** or **Uniaxial Easy-Plane** anisotropy.

$$E_A = -K(\vec{m}.\vec{u})^2$$
$$= -K(\cos^2\theta) = -K(1-\sin^2\theta) = -K + K\sin^2\theta$$

where: K is the anisotropy constant (J/m^3), $\vec{u}$ is the unit vector pointing along the easy axis/plane.

2. Crystalline Anisotropy: Depending on crystal symmetry, some axes are easy to magnetize.
   For example, **cubic anisotropy**: $E_A = -K(m_1^2m_2^2 + m_2^2m_3^2 + m_3^2m_2^2)$[1]

3. Induced Anisotropy: When we apply stress/annealing to create an Easy axis, and alter magnetic behaviour (depends on annealing procedure, stress factor)

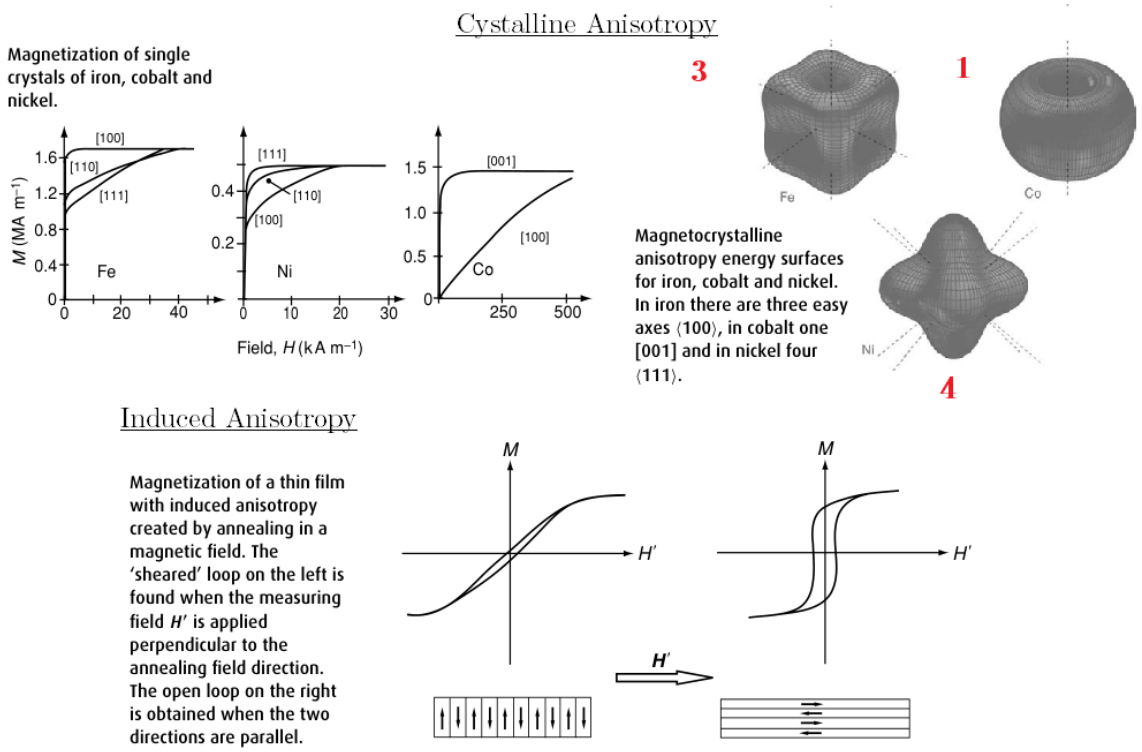


Figure 2.5: Examples of Crystalline Anisotropy

Shape Anisotropy

$E_a = \frac{1}{2}\mu_0 V M_s^2[\frac{1}{2}(1-\mathcal{N}) - \mathcal{N}]$ *Ellipsoid*

$= \frac{1}{4}\mu_0 M_s^2(1-3\mathcal{N}).$

demagnetization factor.
eg. 1/3 for sphere

Crystalline Anisotropy

$E_a = K_1 \sin^2\theta + K_2 \sin^4\theta + K_3 \sin^6\theta + K_3' \sin^6\theta \sin 6\phi,$ *Hexagonal*:

$E_a = K_1 \sin^2\theta + K_2 \sin^4\theta + K_2' \sin^4\theta \cos 4\phi + K_3 \sin^6\theta + K_3' \sin^6\theta \sin 4\phi,$ *Tetragonal*:

$E_a = K_{1c}(\alpha_1^2\alpha_2^2 + a_2^2\alpha_3^2 + \alpha_3^2\alpha_1^2) + K_{2c}(\alpha_1^2\alpha_2^2\alpha_3^2),$ *Cubic*:

direction cosines

Induced Anisotropy

$E_a = ?$ *Annealed*

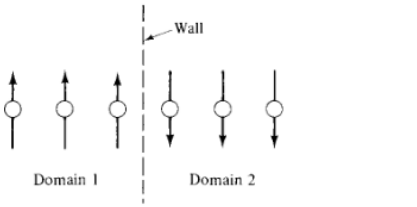
$E_a = \frac{3}{2}\sigma\lambda_s$ *Stressed*

magnetostriction

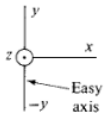
Figure 2.6: Energies associated with Anisotropy

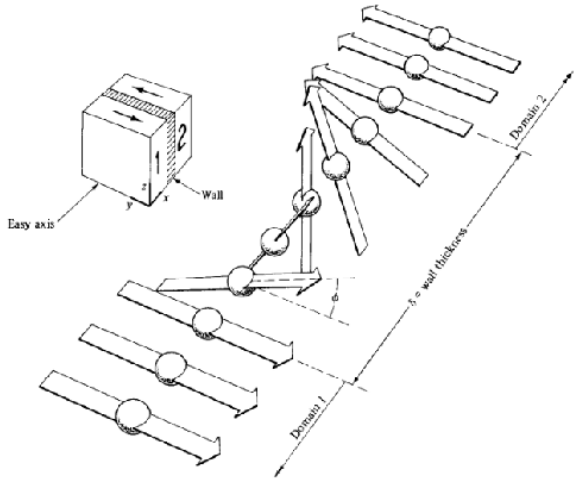
2.4 Domains & Domain Walls

Domains are regions where the magnetization is directed in a specific, uniform direction. Domains exist in order to minimize dipolar coupling and stray field (ie, magnetostatic fields). Domain walls are interfaces between regions in which the spontaneous magnetization has different directions. At or within the wall the magnetization must change direction, from one easy crystallographic direction to another.



Hypothetical 180° domain wall of zero thickness.





Structure of a 180° domain wall.

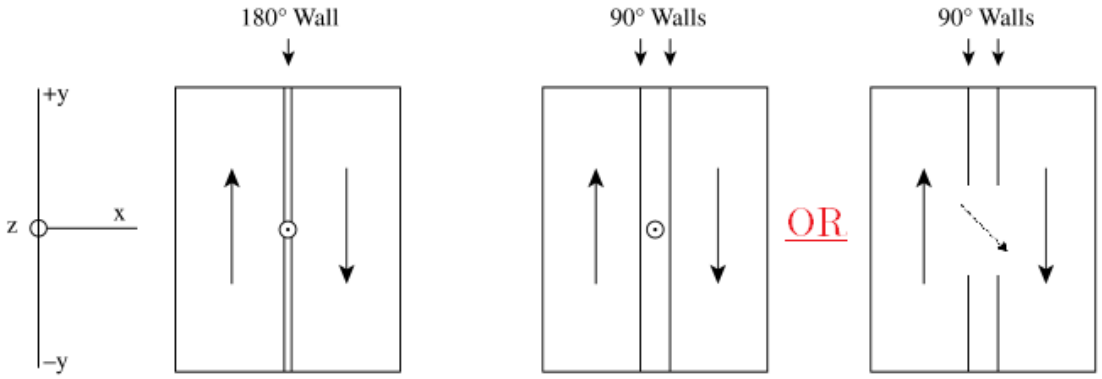
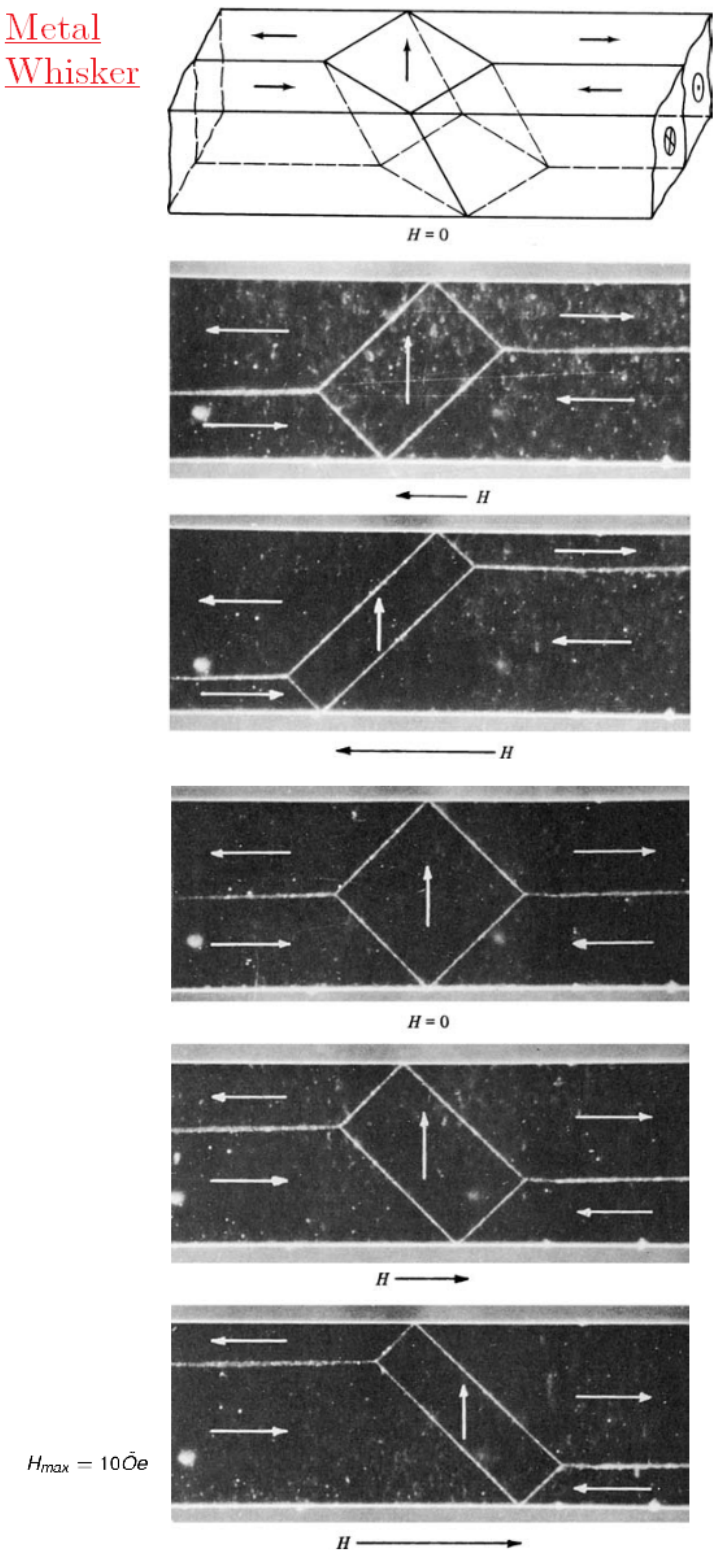


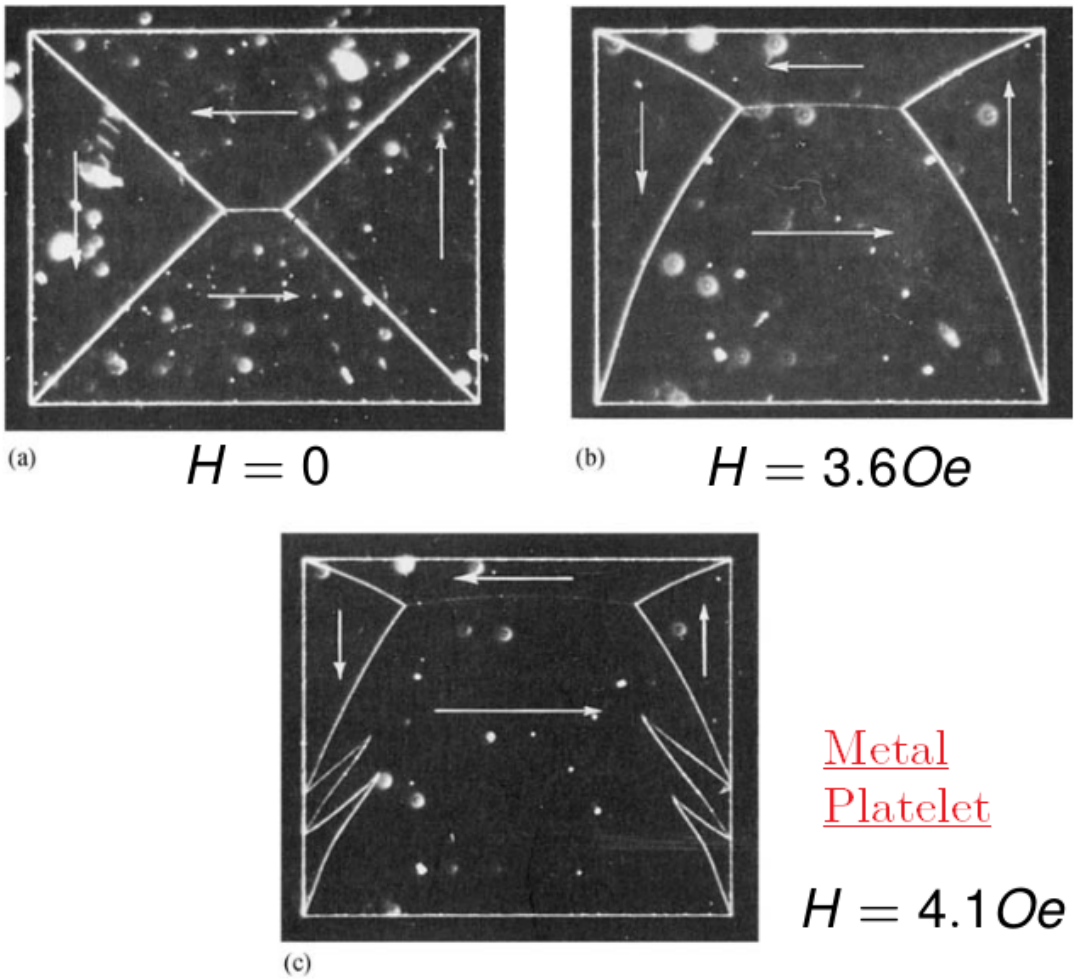
Figure 2.7: Ideal 1D Domain Walls

Domain walls usually exist in a state of equlibrium. This is visible int the following images of single crystals **?** (observed using Bitter method):



Reversible wall motion in an iron whisker. Bitter patterns under dark-field illumination.

Figure 2.8: Domain Walls in single crystal "whiskers"



Wall motion in a Ni–Co alloy platelet 165 μm across. Bitter patterns, dark field illumination.

Figure 2.9: Domain Walls in single crystal "platelets"

Domains aren't always boxed regions. Even in 2D they could be a swirl of magnetization, as is visible in the following figure which shows the magnetization is simulated for width comparable to the exchange length λ.

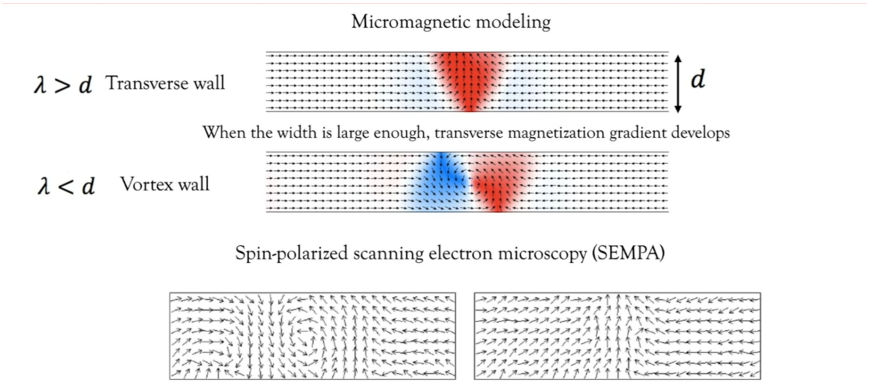


Figure 2.10: Competing domainwalls in 2D FeGe thinfilms **?**

2.5 MicroMagnetic Simulation & Visualization of Energies

Here I have simulated micromagnetic energies in Ubermag **?** - a popular micromagnetic simulation framework. The code is attached in the Appendix 1.

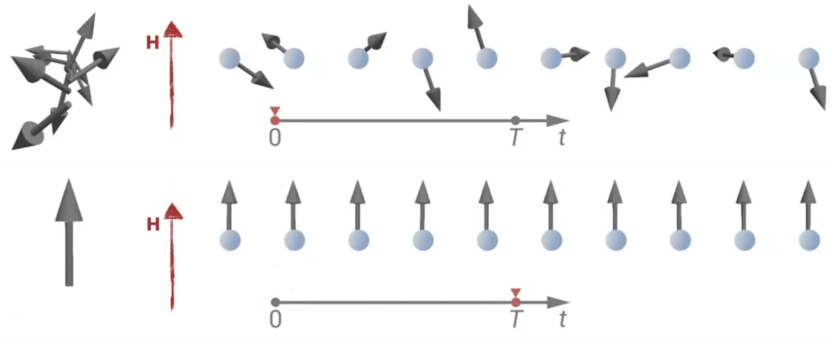


Figure 2.11: Simulation result for Zeeman Energy: initial & final states

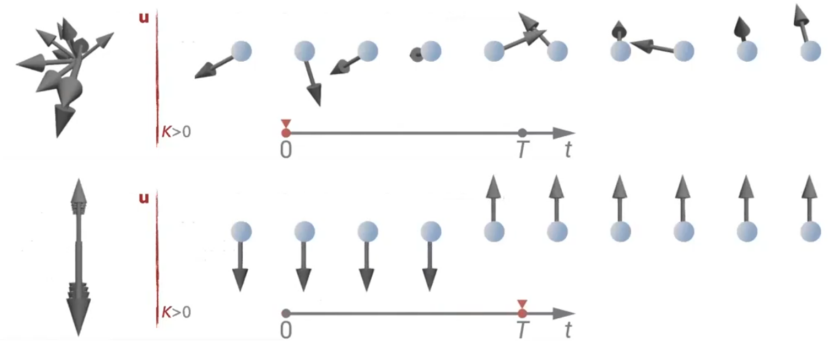


Figure 2.12: Simulation result for Easy-Axis Anisotropy Energy: initial & final states

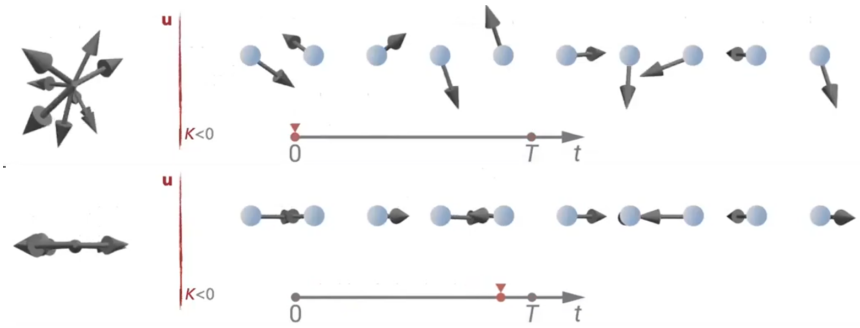


Figure 2.13: Simulation result for Easy-Plane Anisotropy Energy: initial & final states

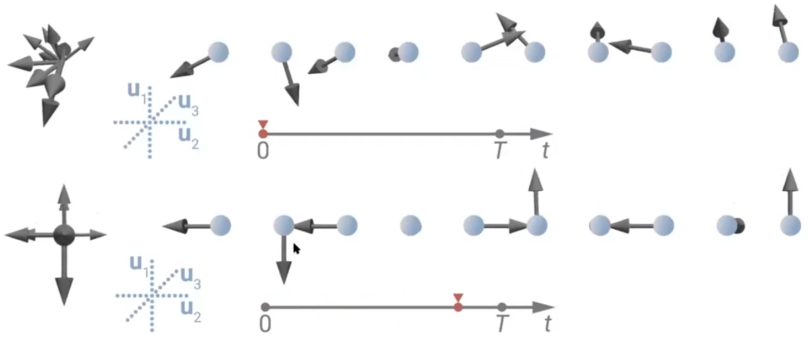


Figure 2.14: Simulation result for Cubic Anisotropy Energy: initial & final states

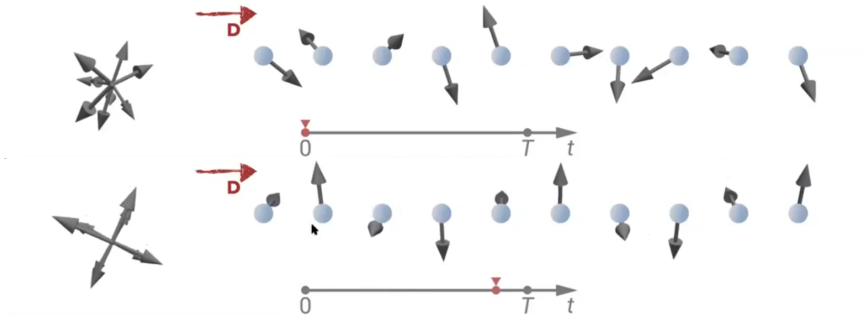


Figure 2.15: Simulation result for Atomic DMI Energy: initial & final states

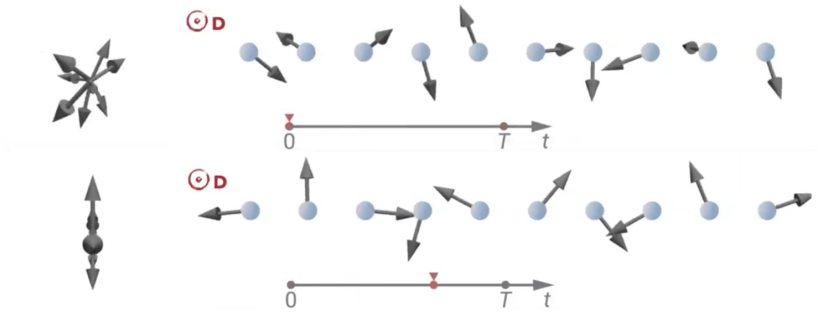


Figure 2.16: Simulation result for Interficial DMI Energy: initial & final states

2.6 Simulation of MicroMagnetic Phenomena

All the above expressions for E represent the Energy Density. To get the energy for each interaction, we integrate it over the entire volume: $E = \int E(m)dV$

1. To model a system, we first have to decide what all energies are at play. Then, we calculate the magnetization of the initial state $\vec{m}$ using the various energy expressions. For this, we formulate the **effective field, ($\vec{H}_{eff}$)** in terms of magnetization:

$$\vec{H_{eff}} = \frac{-1}{\mu_0 M_s}\frac{dE}{d\vec{m}}$$

2. The main aim of simulation is to study how $\vec{m}$ is varying in space/time. We ask ourselves the question - "How does the magnetization change in order to minimize the system's energy?"
   For that, we determine the **system dynamics**:
   There are two aspects usually considered - **precession** & **damping**:
   precession measures how $\vec{m}$ precesses around the effective field $\vec{H_{eff}}$, while **damping** measures how $\vec{m}$ aligns itself parallel to $\vec{H_{eff}}$.
   $\frac{d\vec{m}}{dt} = -\gamma_0 \vec{m} \times \vec{H_{eff}}$
   $\frac{d\vec{m}}{dt} = \alpha(\vec{m} \times \frac{d\vec{m}}{dt})$
   where: γ, α are the gyromagnetic ratio & damping coefficient.

   Combining these two, we get the **LLG Equation**:
   $\frac{d\vec{m}}{dt} = -\gamma_0 \vec{m} \times \vec{H_{eff}} + \alpha(\vec{m} \times \frac{d\vec{m}}{dt})$
   Separating variables:
   $\frac{d\vec{m}}{dt} = \frac{-\gamma_0}{1+\alpha^2}\vec{m} \times \vec{H}_{eff} - \frac{-\gamma_0\alpha}{1+\alpha^2}\vec{m} \times (\vec{m} \times \vec{H_{eff}})$

3. We calculate the initial state using this expression, then we use $\frac{d\vec{m}}{dt}\Delta\vec{m}$ to obtain the next set of states using a finite element approach. This iteration converges when $\vec{m} \times \vec{H} \to 0$ indicating system to be at minimum energy ("relaxed state").

This code for this simulation is given in Appendix 2, 3.

1

November 14, 2022

1 MicroMagnetic Simulation: Visualising Energies

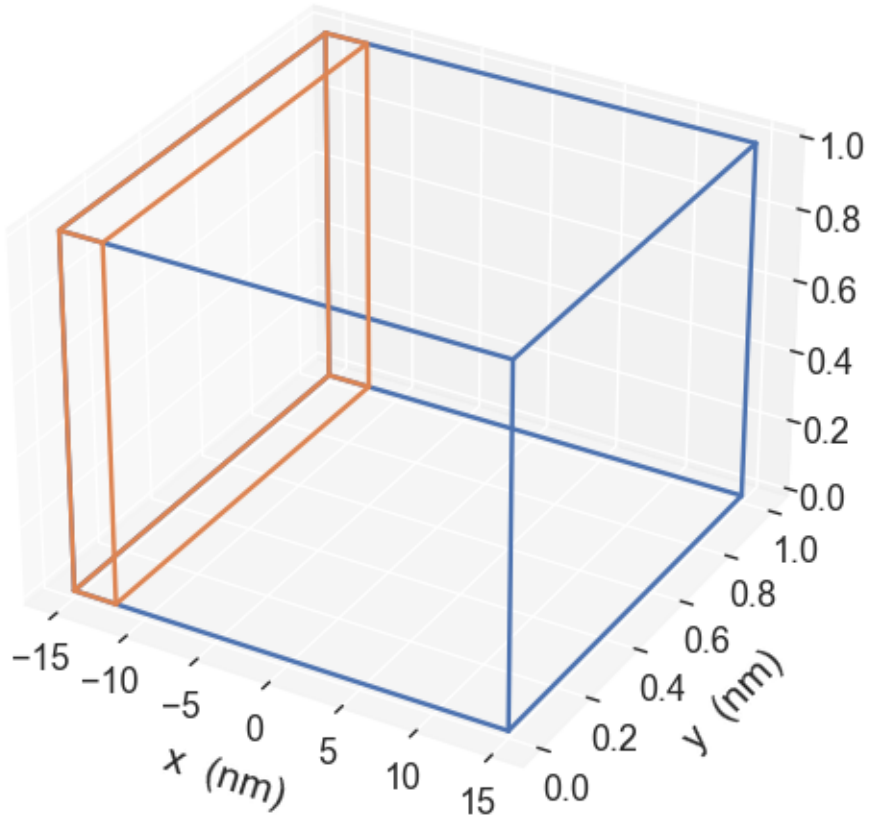
Here, we simulate/visualise energies associated with magnetic phenomena.

A **system** is simulated by considering **energies (energy densitites)**, **dynamics**, and an **initial magnetic configuration**.

- Zeeman Energy
- Easy-Axis Anisotropy
- Easy-Plane Anisotropy
- Cubic Anisotropy
- Bulk DMI
- Interfacial DMI
- Exchange Energy
- Exchange Zeeman Energy
- Exchange Anisotropy
- Exchange DMI

[1]:
```
import os
import k3d
import random
import discretisedfield as df
import micromagneticmodel as mm
import micromagneticdata as md
import oommfc as mc
docker_runner = mc.oommf.DockerOOMMFRunner(image='oommf/oommf')

# Regions: 30nm x 1nm x 1nm
p1, p2 = (-15e-9, 0, 0), (15e-9, 1e-9, 1e-9)
n = (10, 1, 1)
region = df.Region(p1=p1, p2=p2)
mesh = df.Mesh(region=region, n=n)
mesh.plane('z').mpl()

# Magnetization
Ms = 1e6
```

```
random.seed(2)
def value_random(point):
    m = [random.random()*2 - 1 for i in range(3)]
    return m
```



1.1 Zeeman Energy

[2]:
```
# System
H = (0, 0, 1e6)
system = mm.System(name='zeeman')
system.energy = mm.Zeeman(H=H)
system.dynamics = mm.Damping(alpha=0.5)  # No precession for faster make␣
↪animations
system.m = df.Field(mesh, dim=3, value=value_random, norm=Ms)

driver = mc.TimeDriver()
driver.drive(system, t=70e-12, n=250)  # 10s video n=10*25=250
```

```
# Plot a sequence of samples
data  = md.Data(system.name)
drive = data[-1]
# drive.ovf2vtk()

plot = k3d.plot()
@df.interact(nstep = drive.slider())
def systemplot(nstep):
    drive[nstep].plane('z').mpl()
    drive[nstep].plane('z').orientation.k3d.vector(plot=plot,␣
 ↪interactive_field=system.m, head_size=2)
plot.display()
```

```
Running OOMMF (DockerOOMMFRunner)[2022/11/14 12:43]... (6.9 s)
```

```
interactive(children=(IntSlider(value=0, description='step', max=249), Output()),␣
 ↪_dom_classes=('widget-intera...
```

```
Output()
```

1.2 Easy-Axis Anisotropy

[3]:
```
u = (0, 0, 1)
K = 7e5

system = mm.System(name='easyaxis_anisotropy')
system.energy = mm.UniaxialAnisotropy(K=K, u=u)
system.dynamics = mm.Damping(alpha=0.5)  # No precession to make animations␣
 ↪faster
system.m = df.Field(mesh, dim=3, value=value_random, norm=Ms)

driver = mc.TimeDriver()
drive = data[-1]

plot = k3d.plot()
@df.interact(nstep = drive.slider())
def systemplot(nstep):
    drive[nstep].plane('z').mpl()
    drive[nstep].plane('z').orientation.k3d.vector(plot=plot,␣
 ↪interactive_field=system.m, head_size=2)
plot.display()
```

```
interactive(children=(IntSlider(value=0, description='step', max=249), Output()),␣
 ↪_dom_classes=('widget-intera...
```

```
Output()
```

1.3 Easy-Plane Anisotropy

[4]:
```
u = (0, 0, 1)
K = -5e5

system = mm.System(name='easyplane_anisotropy')
system.energy = mm.UniaxialAnisotropy(K=K, u=u)
system.dynamics = mm.Damping(alpha=0.5)  # No precession to make animations␣
 ↪faster
system.m = df.Field(mesh, dim=3, value=value_random, norm=Ms)

driver = mc.TimeDriver()
drive = data[-1]

plot = k3d.plot()
@df.interact(nstep = drive.slider())
def systemplot(nstep):
    drive[nstep].plane('z').mpl()
    drive[nstep].plane('z').orientation.k3d.vector(plot=plot,␣
 ↪interactive_field=system.m, head_size=2)
plot.display()
```

```
interactive(children=(IntSlider(value=0, description='step', max=249), Output()),␣
 ↪_dom_classes=('widget-intera...
```

```
Output()
```

1.4 Cubic Anisotropy

[5]:
```
u1,u2 = (1, 0, 0), (0, 1, 0)
K = 7e5

system = mm.System(name='cubic_anisotropy')
system.energy = mm.CubicAnisotropy(K=K, u1=u1, u2=u2)
system.dynamics = mm.Damping(alpha=0.5)  # No precession to make animations␣
 ↪faster
system.m = df.Field(mesh, dim=3, value=value_random, norm=Ms)

driver = mc.TimeDriver()
drive = data[-1]

plot = k3d.plot()
@df.interact(nstep = drive.slider())
def systemplot(nstep):
    drive[nstep].plane('z').mpl()
    drive[nstep].plane('z').orientation.k3d.vector(plot=plot,␣
 ↪interactive_field=system.m, head_size=2)
plot.display()
```

```
interactive(children=(IntSlider(value=0, description='step', max=249), Output()),␣
↪_dom_classes=('widget-intera...
```

```
Output()
```

1.5 Bulk DMI

[6]:
```
D = 2e-2

system = mm.System(name='dmi_bulk')
system.energy = mm.DMI(D=D, crystalclass='T')
system.dynamics = mm.Damping(alpha=0.5)  # No precession to make animations␣
↪faster
system.m = df.Field(mesh, dim=3, value=value_random, norm=Ms)

driver = mc.TimeDriver()
drive = data[-1]

plot = k3d.plot()
@df.interact(nstep = drive.slider())
def systemplot(nstep):
    drive[nstep].plane('z').mpl()
    drive[nstep].plane('z').orientation.k3d.vector(plot=plot,␣
↪interactive_field=system.m, head_size=2)
plot.display()
```

```
interactive(children=(IntSlider(value=0, description='step', max=249), Output()),␣
↪_dom_classes=('widget-intera...
```

```
Output()
```

1.6 Interfacial DMI

[7]:
```
D = 1e-2

system = mm.System(name='dmi_interfacial')
system.energy = mm.DMI(D=D, crystalclass='Cnv')
system.dynamics = mm.Damping(alpha=0.5)  # No precession to make animations␣
↪faster
system.m = df.Field(mesh, dim=3, value=value_random, norm=Ms)

driver = mc.TimeDriver()
drive = data[-1]

plot = k3d.plot()
@df.interact(nstep = drive.slider())
def systemplot(nstep):
    drive[nstep].plane('z').mpl()
```

```
    drive[nstep].plane('z').orientation.k3d.vector(plot=plot,␣
 ↪interactive_field=system.m, head_size=2)
plot.display()
```

```
interactive(children=(IntSlider(value=0, description='step', max=249), Output()),␣
 ↪_dom_classes=('widget-intera...
```

```
Output()
```

1.7 Exchange Energy

[8]:
```
A = 5e-11
H = (5e4, 0, 5e4)  # just to speed up simulations

system = mm.System(name='exchange')
system.energy = mm.Exchange(A=A) + mm.Zeeman(H=H)

driver = mc.TimeDriver()
driver.drive(system, t=70e-12, n=250)  # 10s video n=10*25=250

# Plot a sequence of samples
data  = md.Data(system.name)
drive = data[-1]
# drive.ovf2vtk()

plot = k3d.plot()
@df.interact(nstep = drive.slider())
def systemplot(nstep):
    drive[nstep].plane('z').mpl()
    drive[nstep].plane('z').orientation.k3d.vector(plot=plot,␣
 ↪interactive_field=system.m, head_size=2)
plot.display()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In [8], line 8
      5 system.energy = mm.Exchange(A=A) + mm.Zeeman(H=H)
      7 driver = mc.TimeDriver()
----> 8 driver.drive(system, t=70e-12, n=250)  # 10s video n=10*25=250
     10 # Plot a sequence of samples
     11 data  = md.Data(system.name)

File ~\AppData\Roaming\Python\Python311\site-packages\oommfc\drivers\driver.py:
 ↪200, in Driver.drive(self, system, dirname, append, fixed_subregions, compute,
 ↪output_step, n_threads, runner, ovf_format, verbose, **kwargs)
    197 jsonfilename = 'info.json'
    199 # Generate and save mif file.
--> 200 mif = oc.scripts.system_script(system, ovf_format=ovf_format)
```

```
    201 mif += oc.scripts.driver_script(self, system,
    202                                   fixed_subregions=fixed_subregions,
    203                                   output_step=output_step,
    204                                   compute=compute, **kwargs)
    205 with open(miffilename, 'w') as miffile:

File ~\AppData\Roaming\Python\Python311\site-packages\oommfc\scripts\system.py:23␣
 ↪in system_script(system, ovf_format, **kwargs)
     20 mif += '}\n\n'
     22 # Mesh and energy scripts.
---> 23 mif += oc.scripts.mesh_script(system.m.mesh)
     24 mif += oc.scripts.energy_script(system)
     26 # Magnetisation script.

AttributeError: 'NoneType' object has no attribute 'mesh'
```

1.8 Exchange + Zeeman

[ ]:
```
A = 1e-10
H = (0, 0, 1e6)

system = mm.System(name='exchange_zeeman')
system.energy = mm.Exchange(A=A) + mm.Zeeman(H=H)

driver = mc.TimeDriver()
driver.drive(system, t=70e-12, n=250)  # 10s video n=10*25=250

# Plot a sequence of samples
data  = md.Data(system.name)
drive = data[-1]
# drive.ovf2vtk()

plot = k3d.plot()
@df.interact(nstep = drive.slider())
def systemplot(nstep):
    drive[nstep].plane('z').mpl()
    drive[nstep].plane('z').orientation.k3d.vector(plot=plot,␣
 ↪interactive_field=system.m, head_size=2)
plot.display()
```

1.9 Exchange + Anisotropy

[ ]:
```
A = 8e-12
K = 3e5
u = (0, 0, 1)
```

```
def _m(point):
    x, y, z = point

    if x < 0:
        return (0, 0.1, -1)
    else:
        return (0, 0.1, 1)

system = mm.System(name='exchange_anisotropy')
system.energy = mm.Exchange(A=A) + mm.UniaxialAnisotropy(K=K, u=u)
system.dynamics = mm.Damping(alpha=0.5)  # No precession to make animations␣
 ↪faster
system.m = df.Field(mesh, dim=3, value=_m, norm=Ms)

driver = mc.TimeDriver()
driver.drive(system, t=70e-12, n=250)  # 10s video n=10*25=250

# Plot a sequence of samples
data  = md.Data(system.name)
drive = data[-1]

plot = k3d.plot()
@df.interact(nstep = drive.slider())
def systemplot(nstep):
    drive[nstep].plane('z').mpl()
    drive[nstep].plane('z').orientation.k3d.vector(plot=plot,␣
 ↪interactive_field=system.m, head_size=2)
plot.display()
```

1.10 Exchange + DMI

[ ]:
```
D = 2e-3
A = 4.77e-12

system = mm.System(name='exchange_anisotropy')
system.energy = mm.Exchange(A=A) + mm.UniaxialAnisotropy(K=K, u=u)
system.dynamics = mm.Damping(alpha=0.5)  # No precession to make animations␣
 ↪faster
system.m = df.Field(mesh, dim=3, value=_m, norm=Ms)

driver = mc.TimeDriver()
driver.drive(system, t=70e-12, n=250)  # 10s video n=10*25=250

# Plot a sequence of samples
data  = md.Data(system.name)
drive = data[-1]
```

```
plot = k3d.plot()
@df.interact(nstep = drive.slider())
def systemplot(nstep):
    drive[nstep].plane('z').mpl()
    drive[nstep].plane('z').orientation.k3d.vector(plot=plot,␣
 ↪interactive_field=system.m, head_size=2)
plot.display()
```

2

November 9, 2022

1 MicroMagnetic Simulations: System Simulation

Here, we simulate/visualise/analyse a simple system.

The mesh can be visualised by `system.m.mesh.region.k3d()`.

A **system** is simulated by considering **energies (energy densitites)**, **dynamics**, and an **initial magnetic configuration**.

[1]:
```
import micromagneticmodel as mm
import discretisedfield as df

system = mm.System()

# Energies: zeeman, anisotropy, demagnetisation
A = 1e-12
H = (5e6, 0, 0)
system.energy   = (mm.Zeeman(H=H)
                 + mm.Exchange(A=A)
                 + mm.Demag())
# Dynamics: precession/damping
gamma0 = 2.211e5
alpha  = 0.2
system.dynamics = mm.Precession(gamma0=gamma0) + mm.Damping(alpha=alpha)
# Initial Conditions:
L = 50e-9
Ms = 8e6
region = df.Region(p1=(0,0,0), p2=(L,L,L))
mesh = df.Mesh(region=region, n=(10,10,10))
system.m        = df.Field(mesh,dim=3,value=(0,1,0), norm=Ms)
```

Now, we visualise the system's behaviour.

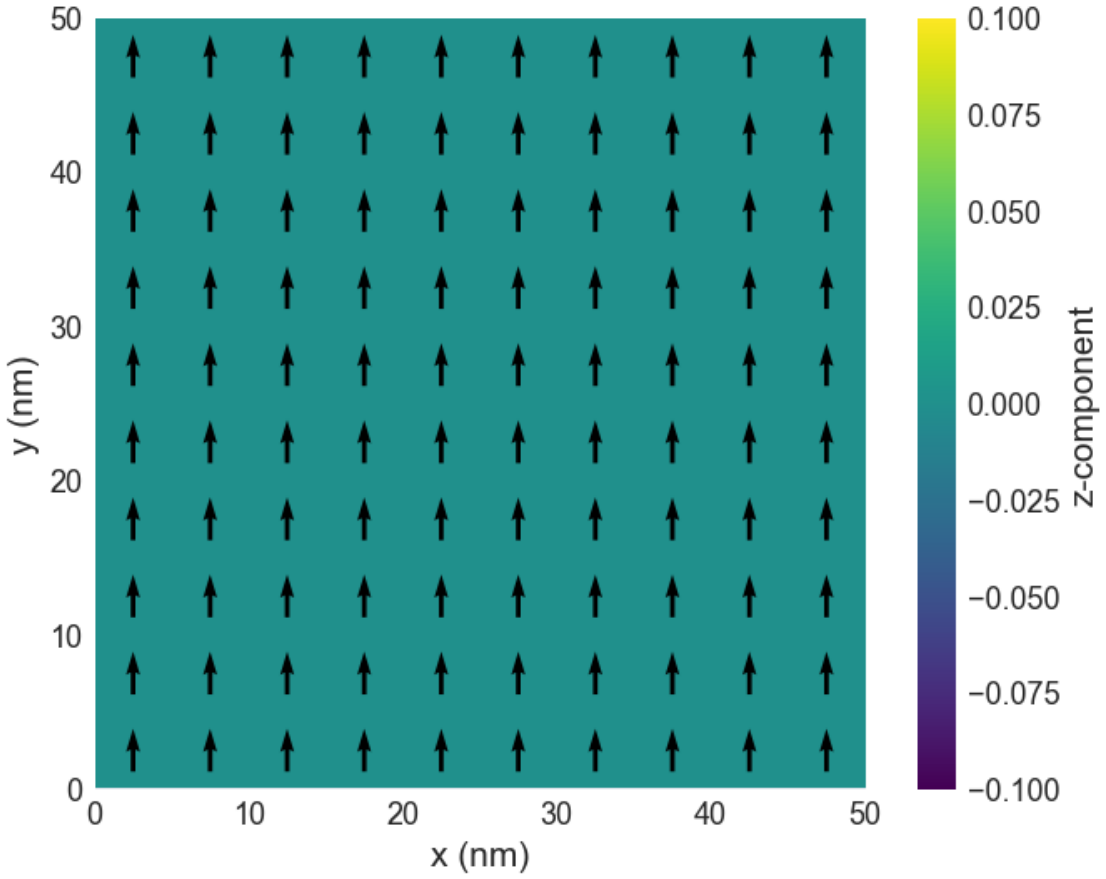
[2]:
```
system.energy
```

[2]:
$$-\mu_0 M_\mathrm{s}\mathbf{m}\cdot\mathbf{H} - A\mathbf{m}\cdot\nabla^2\mathbf{m} - \tfrac{1}{2}\mu_0 M_\mathrm{s}\mathbf{m}\cdot\mathbf{H}_\mathrm{d}$$

[3]:
```
system.dynamics
```

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

[4]:
```
# Initial State
system.m.plane('z').mpl()
```



[5]:
```
system.m.plane('z').k3d.vector(head_size=10)
```

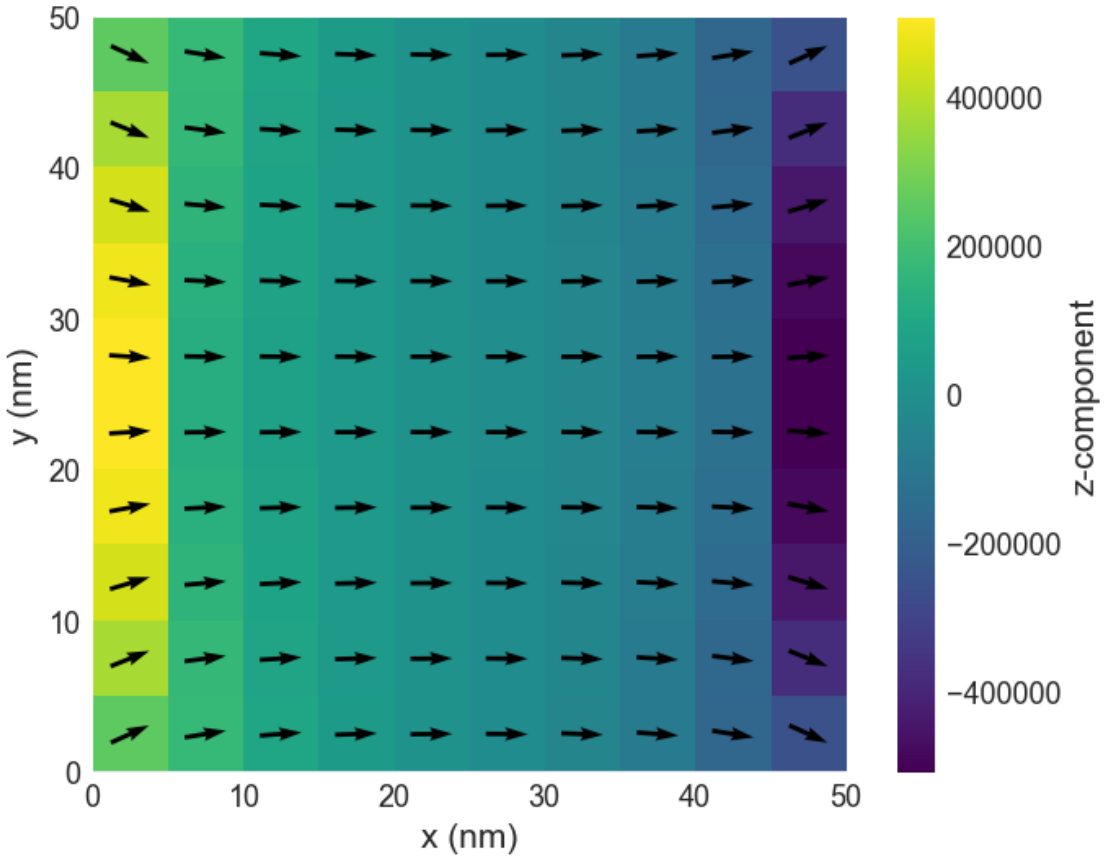
```
Output()
```

Now, we make micro-magnetic calculations.

[6]:
```
import oommfc as mc
docker_runner = mc.oommf.DockerOOMMFRunner(image='oommf/oommf')

driver = mc.MinDriver()
driver.drive(system, runner=docker_runner)
```

```
Running OOMMF (DockerOOMMFRunner)[2022/11/09 21:51]... (1.9 s)
```

[7]:
```
# Final State
system.m.plane('z').mpl()
```



[8]:
```
system.m.plane('z').k3d.vector(head_size=10)
```

```
Output()
```

[9]:
```
system.table.data.T
```

[9]:

	0
max_mxHxm	9.956389e-02
E	-4.724965e-15
delta_E	-7.888609e-31
bracket_count	4.400000e+01
line_min_count	1.900000e+01
conjugate_cycle_count	1.700000e+01
cycle_count	3.700000e+01
cycle_sub_count	2.000000e+01
energy_calc_count	6.400000e+01
E_zeeman	-6.173354e-15

```
E_exchange                   8.037441e-20
max_spin_ang_exchange        1.528564e+01
stage_max_spin_ang_exchange  1.560823e+01
run_max_spin_ang_exchange    1.560823e+01
E_demag                      1.448308e-15
iteration                    6.100000e+01
stage_iteration              6.100000e+01
stage                        0.000000e+00
mx                           9.825198e-01
my                          -9.169040e-10
mz                           4.540198e-18
```

3

November 10, 2022

1 MicroMagnetic Simulation: Specifying System Geometries

Various geometries (such as cuboidal, spherical, vortex) can be approximated for system simulation. Everything is implemented as an approximation of cuboidal **cells**.

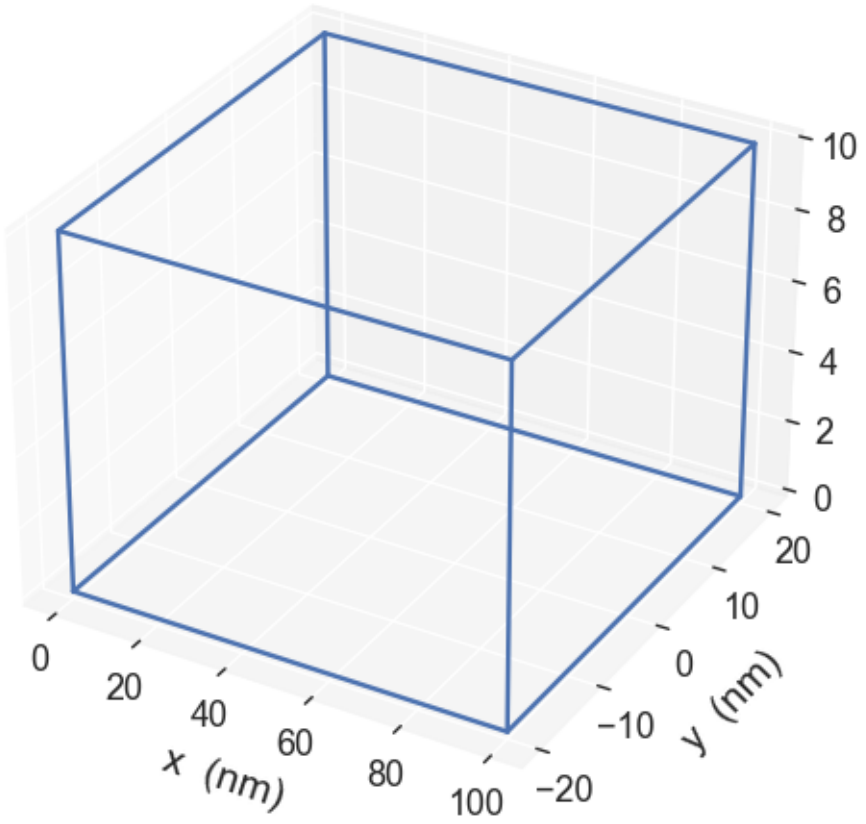
Region, Mesh, Field (magnetization)

- **region** defines a physical cuboid (using 2 diagonal points), **mesh** defines a discrete version of the **region**. A **field** (such as $\vec{M}, \vec{m}, \vec{H}$) may be defined over this **mesh**. Additionally, the **field** may be normalized using a saturation value (such as M_s), or by defining a norm that makes use of this saturation value. Once a **field** is defined, we can operate on it (average, derivative, div, curl, ...).
- A **region** may be decomposed into **subregions** (such as Fe, Co, Ni, ...). We can define separate fields over each and simulate the system as a whole.

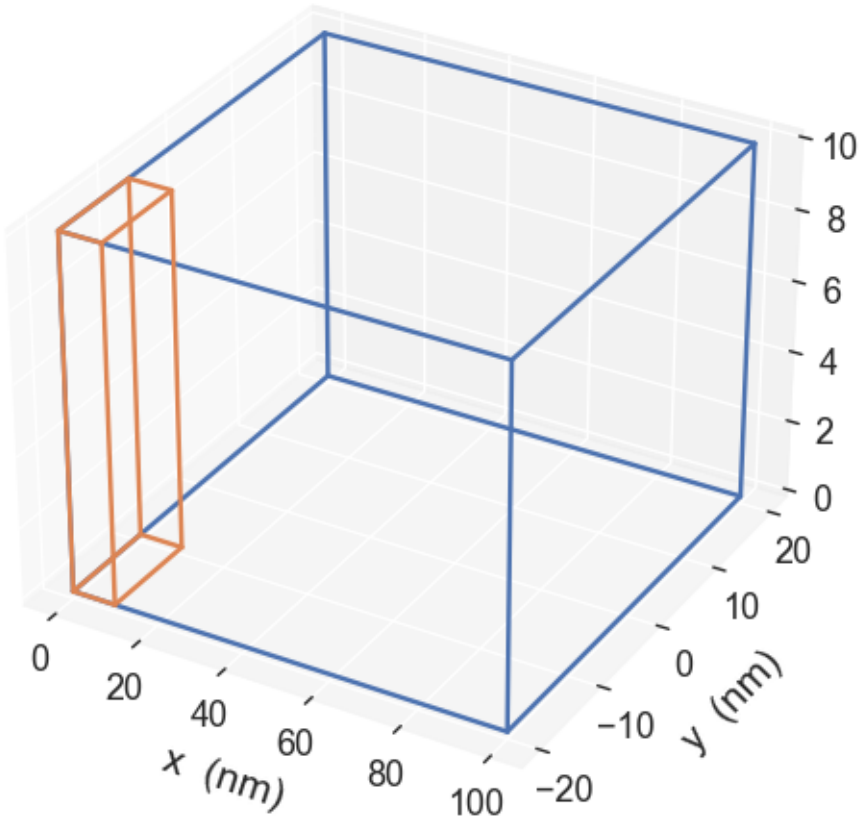
[1]:
```
import discretisedfield as df
import micromagneticmodel as mm


# Region (defined using diagonal points)
p1,p2 = (0,-20e-9, 0), (100e-9, 20e-9, 10e-9)
region = df.Region(p1=p1, p2=p2)

# Mesh
cell = (10e-9, 10e-9, 10e-9)
mesh = df.Mesh(region=region, cell=cell)

# Field - const vector & spatially-varying vector fields
def _field(point):
    x,y,z = point
    if y > 0:
        return (1,0,0)
    else:
        return (-1,1,0)

Ms = 8e6
```

```
m1= df.Field(mesh, dim=3, value=(1,0,0), norm=Ms)
m2= df.Field(mesh, dim=3, value=_field, norm=Ms)
```

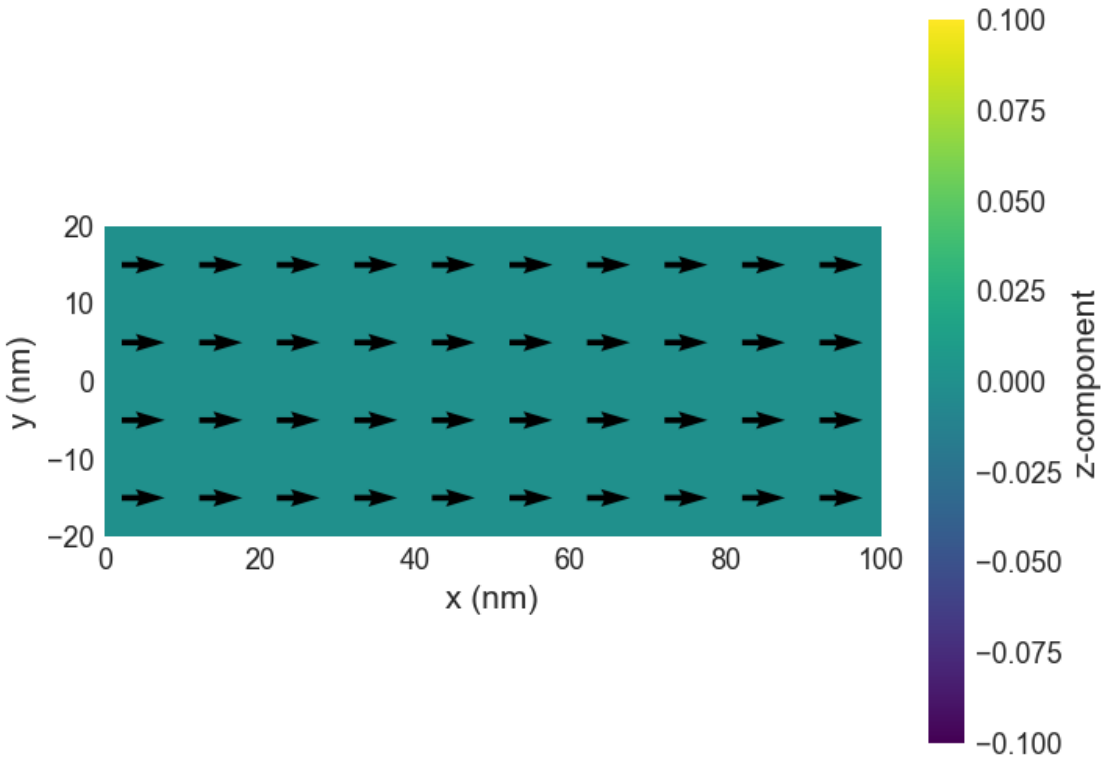
[2]:
```
region.mpl()
```

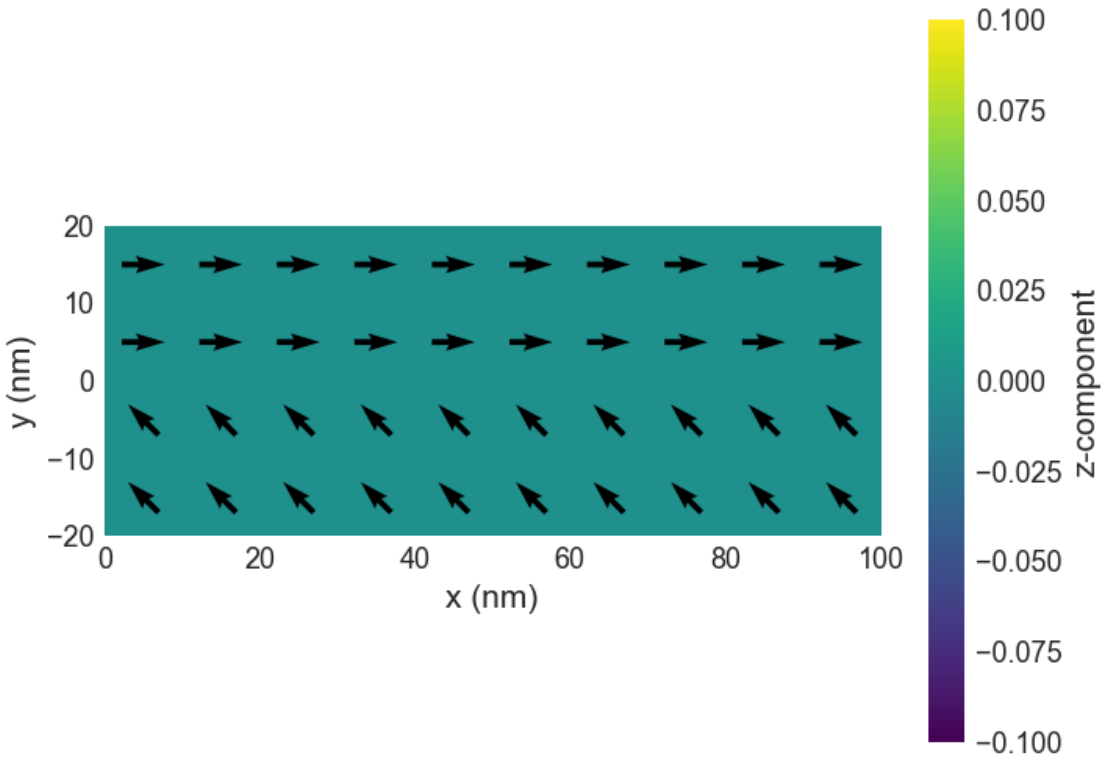


[3]:
```
mesh.mpl()
```



[4]:
```
# Fixed & Variable fields
m1.plane('z').mpl()
m2.plane('z').mpl()
```


20
10
0
−10
−20
y (nm)
0
20
40
60
80
100
x (nm)
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
z-component


20
10
0
−10
−20
y (nm)
0
20
40
60
80
100
x (nm)
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
z-component

Subregions

- **subregions** are a handy way to specify different materials/geometries. Fixed or Variable (**spatial/time varying**) fields may be specified for each of these regions.

We simulate a $20nm$ **region** of two $10nm$ **subregions**, with a cell-size of $1nm$, with fixed & variable external fields.

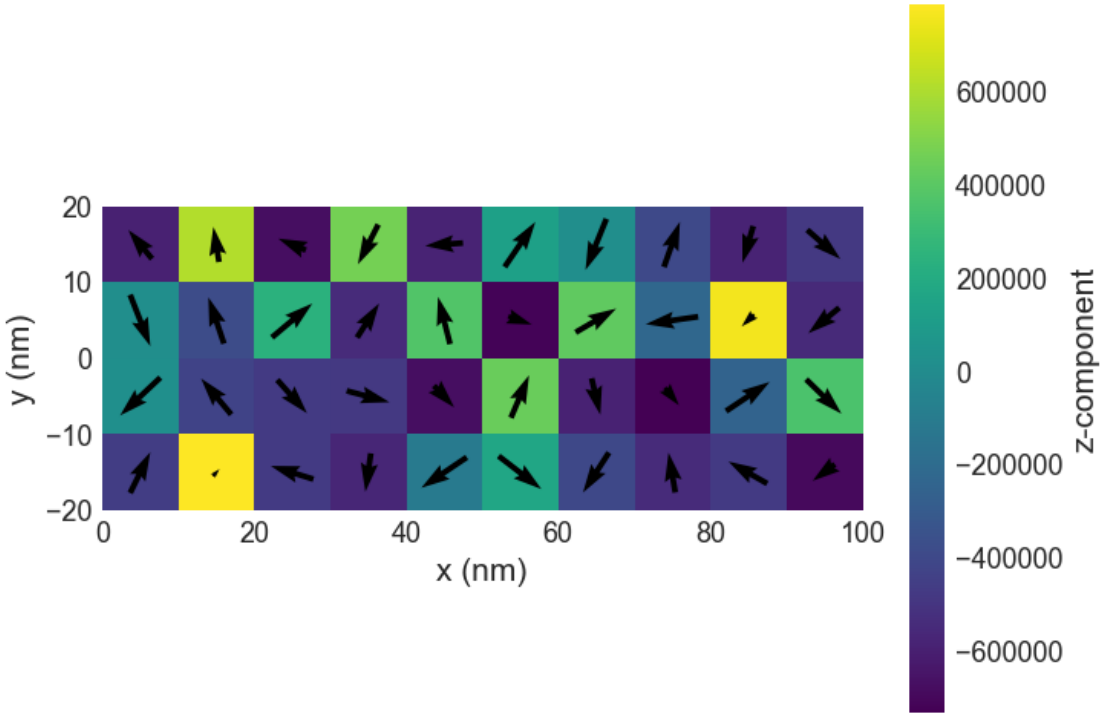
The variable field is defined using: $\mathbf{H}(x, y, z) = (c^2x, 0, c)$ (where $c = 10^9$ and the entire field is normalised with $H = 10^6\,\mathrm{Am}^{-1}$)

The magnetisation is normalized, and is defined as a variable field itself.

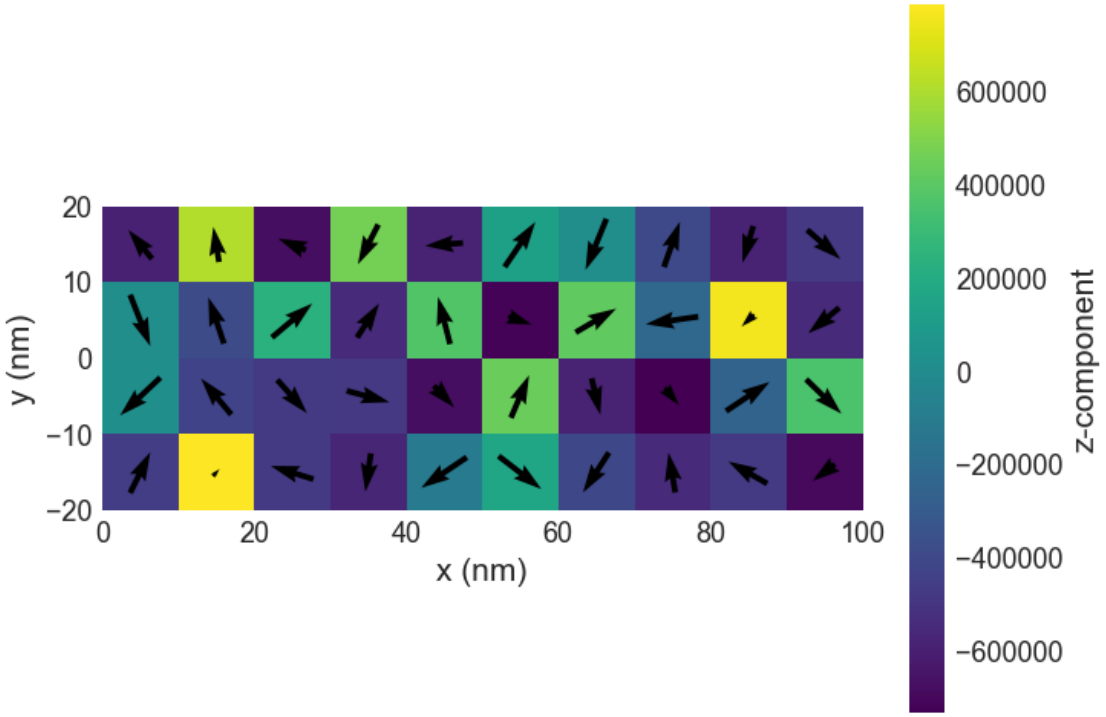
We will visualise the system's initial state.

[5]:
```
# Geometries
p1,p2 = (-10e-9, 0, 0), (10e-9, 1e-9, 1e-9)
cell = (1e-9, 1e-9, 1e-9)
subregions = {'Fe': df.Region(p1=(-10e-9, 0, 0), p2=(0, 1e-9, 1e-9)),
              'Co': df.Region(p1=(0, 0, 0), p2=(10e-9, 1e-9, 1e-9))}

# Magnetisation
import random
Ms = 8e5
def _m(pos):
    return [2*random.random()-1 for i in range(3)]

# System
system = mm.System()
system.m = df.Field(mesh, dim=3, value=_m, norm=Ms)
region = df.Region(p1=p1, p2=p2)
mesh = df.Mesh(region=region, cell=cell, subregions=subregions)
```

[6]:
```
# Fixed field
H = {'Fe': (1e6, 0, 0), 'Co': (0, 0, -1e6)}
system.energy = mm.Zeeman(H=H)

# Initial State
system.m.plane('z').mpl()
```



[7]:
```
# Variable field
def _H(point):
    x, y, z = point
    c = 1e9
    return (c*c*x, 0, c)
H = df.Field(mesh, dim=3, value=_H, norm=1e6)
system.energy = mm.Zeeman(H=H)

# Initial State
system.m.plane('z').mpl()
```



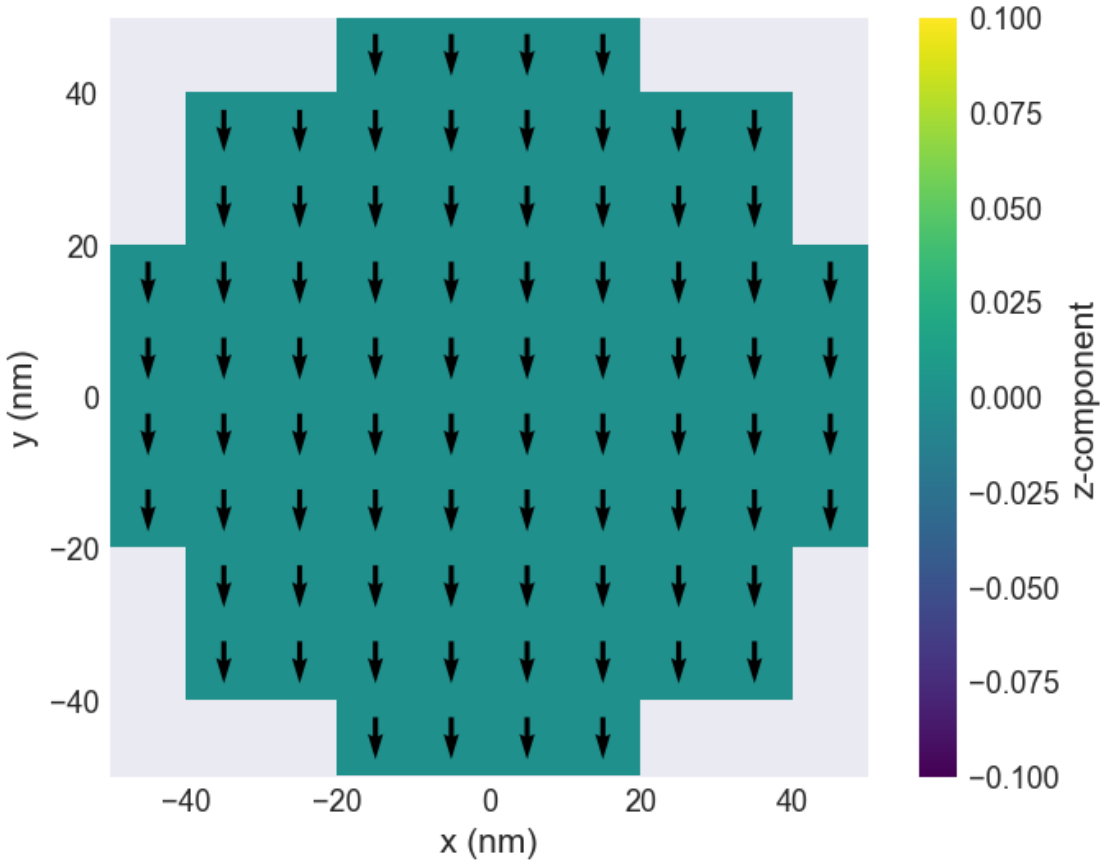
Saturation Magnetisation (M_s)

Different geometries can be simulated by specifying a norm (ie, M_s).

- Spherical (radius, $r = 50nm$)
- Disc (thickness: $t = 50nm$, diameter $d = 120nm$)
- Custom
- Vortex

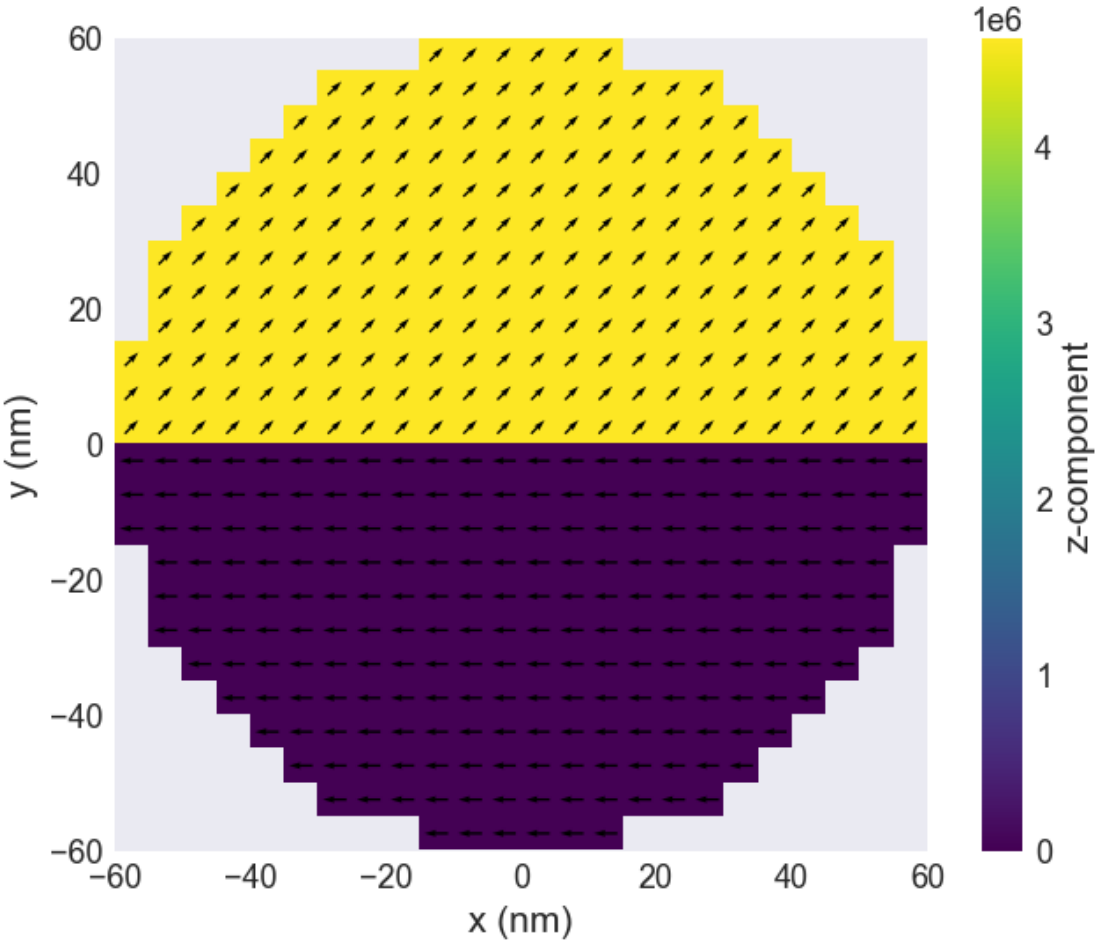
[8]:
```
# Spherical Sample

# Region, Mesh
Ms = 8e6
r,n = 50e-9, (10,10,10)
region = df.Region(p1=(-r,-r,-r), p2=(r,r,r))
mesh   = df.Mesh(region=region, n=n)

# Norm using Ms
def _norm(point):
    x,y,z = point
    if x**2 + y**2 + z**2 < r**2:
        return Ms
    else:
        return 0
```

```
m = df.Field(mesh, dim=3, value=(0,-1,0), norm=_norm)
m.plane('z').mpl()
```



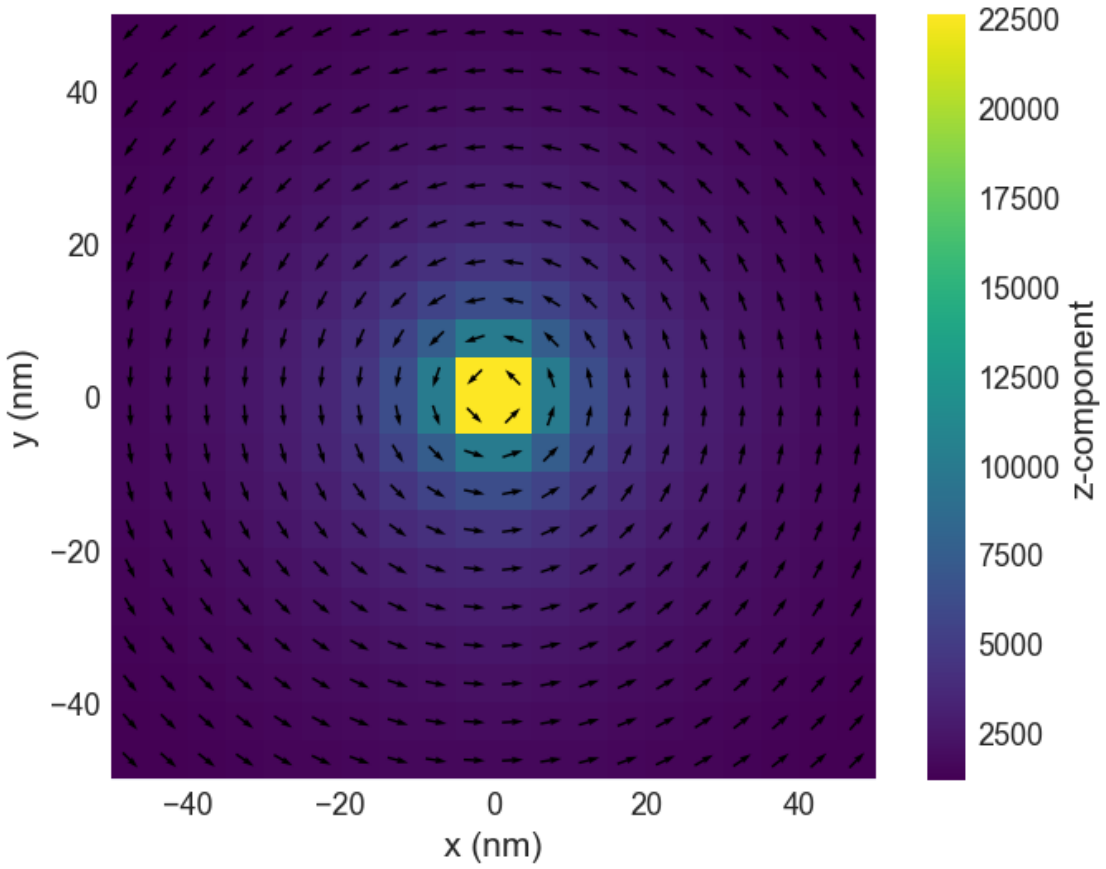
[9]:
```
# Disk Sample

# Region, Mesh
Ms = 8e6
t,d = 10e-9, 120e-9
cell = (5e-9, 5e-9, 5e-9)
region = df.Region(p1=(-d/2, -d/2, -t/2), p2=(d/2, d/2, t/2))
mesh   = df.Mesh(region=region, cell=cell)

# Field - const vector & spatially-varying vector fields
def _norm(point):
    x,y,z = point
    if (x**2 + y**2) ** 0.5 < d/2:
        return Ms
    else:
        return 0
```

```
# Normalized magnetisation (m) as a variable vector field
def _m(point):
    x,y,z = point
    if y <= 0:
        return (-1,0,0)
    else:
        return (1,1,1)

m= df.Field(mesh, dim=3, value=_m, norm=_norm)
m.plane('z').mpl()
```



[10]:
```
# Vortex field

# Geometry
L = 100e-9  # sample edge length (m)
thickness = 5e-9  # sample thickness (m)
```

```
# Material parameters
Ms = 8e5  # saturation magnetisation (A/m)
A = 13e-12  # exchange energy constant (J/m)

# Dynamics (LLG equation) parameters
gamma0 = mm.consts.gamma0  # gyromagnetic ratio (m/As)
alpha = 0.2  # Gilbert damping

system = mm.System(name='vortex_dynamics')
system.energy = mm.Exchange(A=A) + mm.Demag()
system.dynamics = mm.Precession(gamma0=gamma0) + mm.Damping(alpha=alpha)

# initial magnetisation state
def _m(point):
    x, y, z = point
    c = 1e9  # (1/m)
    return (-c*y, c*x, 0.1)

# Sample's centre is placed at origin
region = df.Region(p1=(-L/2, -L/2, -thickness/2), p2=(L/2, L/2, thickness/2))
mesh = df.Mesh(region=region, cell=(5e-9, 5e-9, 5e-9))
system.m = df.Field(mesh, dim=3, value=_m, norm=Ms)

# Initial State
system.m.plane('z').mpl()
```



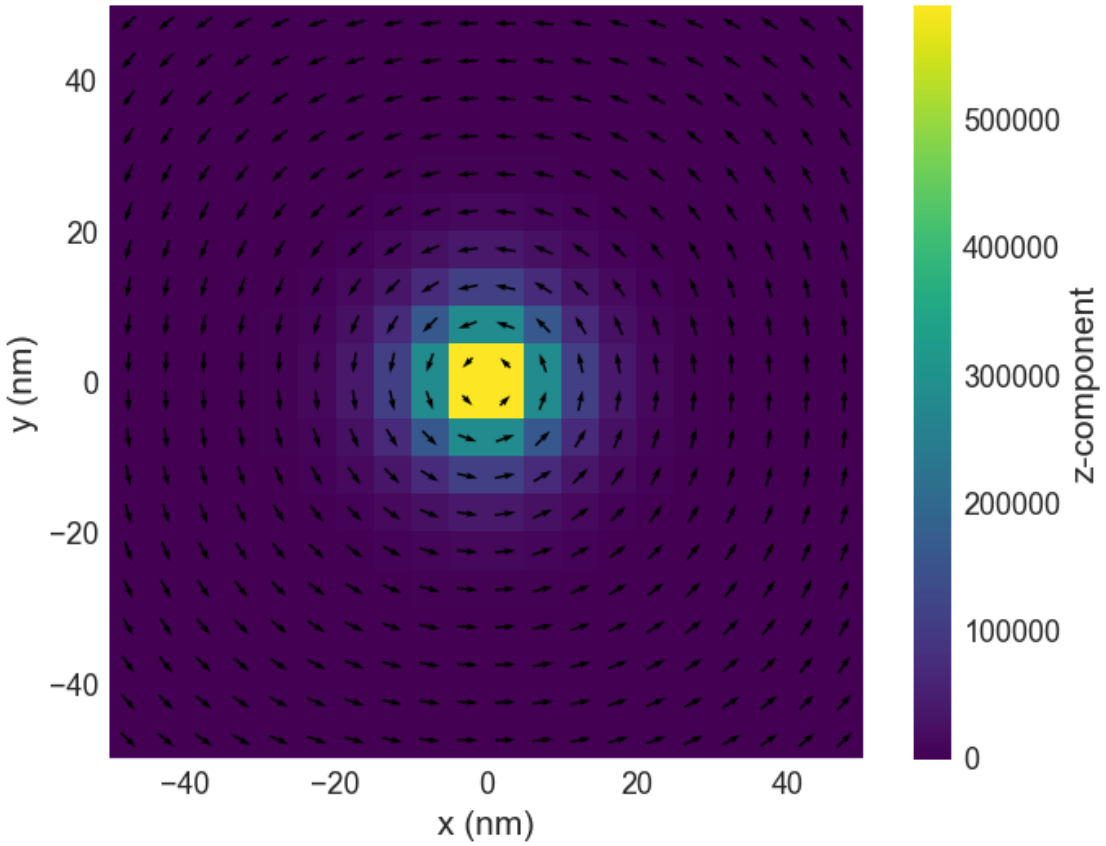
Simulating a system geometry

We'll simulate the Vortex field after specifying the intial conditions.

Given the initial system state, we will: 1. Relax it (move core to centre) by minimising system energy 2. Apply an external magnetic field H_x (Zeeman field along $+X$) 3. Remove the field, and save dynamics for $t = 5ns, samples = 500$ 4. Visualise dynamics saved in previous step

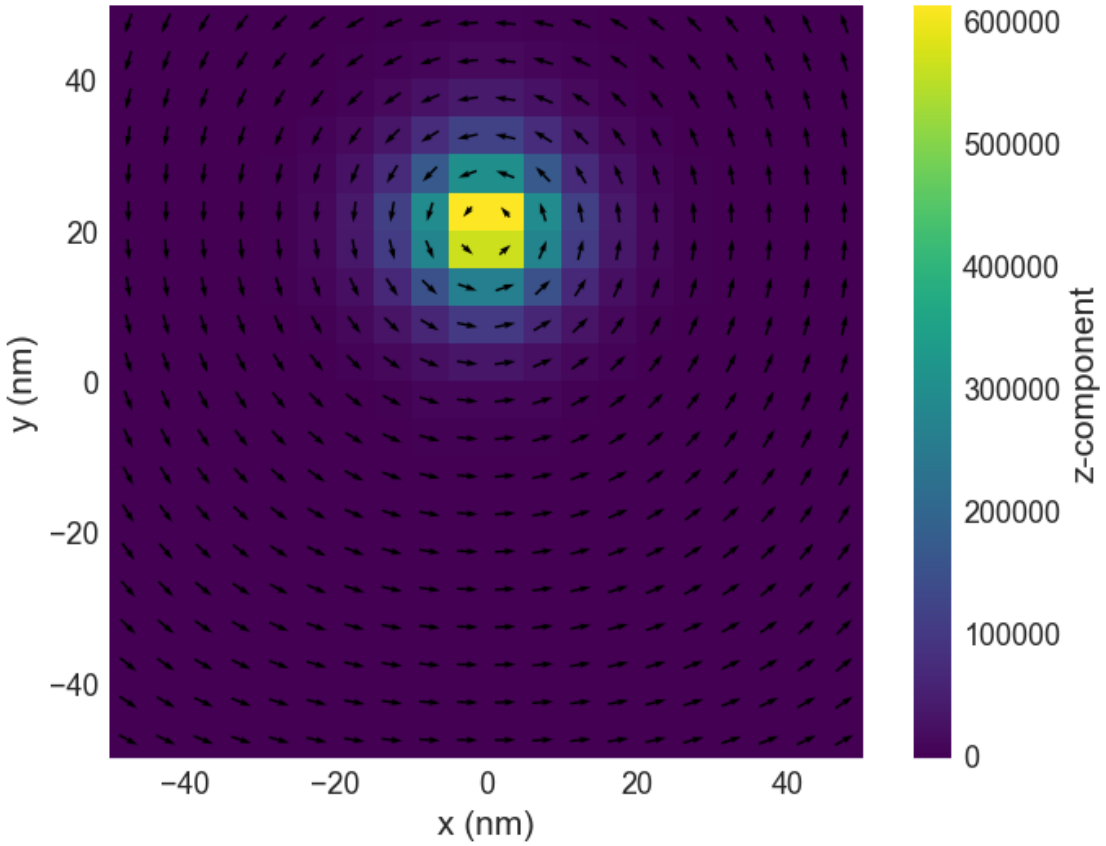
[11]:
```
import oommfc as mc
docker_runner = mc.oommf.DockerOOMMFRunner(image='oommf/oommf')

# 1. Relax State
driver1 = mc.MinDriver()
driver1.drive(system)
system.m.plane('z').mpl()
```

```
Running OOMMF (DockerOOMMFRunner)[2022/11/10 02:55]... (1.6 s)
```



[12]:
```
# 2. Apply External Field
H = (1e4, 0, 0)
system.energy += mm.Zeeman(H=H)
driver1.drive(system)
system.m.plane('z').mpl()
```

```
Running OOMMF (DockerOOMMFRunner)[2022/11/10 02:55]... (1.8 s)
```



[14]:
```
%%timeit -r1 -n1
# 3. Remove field; Observe system behaviour for 5ns
system.energy.zeeman.H = (0,0,0)
driver2 = mc.TimeDriver()
driver2.drive(system, t=5e-9, n=500, runner=docker_runner)
```

```
Running OOMMF (DockerOOMMFRunner)[2022/11/10 02:58]... (19.7 s)
19.7 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
```

[15]:
```
# 4. Plot a sequence of 500 samples
import micromagneticdata as md

data  = md.Data(system.name)
drive = data[-1]

@df.interact(nstep = drive.slider())
#@df.interact(nstep = drive.slider(continuous_update=False))
def systemplot(nstep):
    drive[nstep].plane('z').mpl()
```

```
interactive(children=(IntSlider(value=0, description='step', max=499), Output()),␣
↪_dom_classes=('widget-intera...
```

[16]:
```
data.info
```

[16]:

	drive_number	date	time	driver	t	n
0	0	2022-11-09	21:16:14	MinDriver	NaN	NaN
1	1	2022-11-09	21:16:16	MinDriver	NaN	NaN
2	2	2022-11-09	21:16:19	TimeDriver	5.000000e-09	500.0
3	3	2022-11-09	21:23:57	MinDriver	NaN	NaN
4	4	2022-11-09	21:23:59	MinDriver	NaN	NaN
5	5	2022-11-09	21:24:04	MinDriver	NaN	NaN
6	6	2022-11-09	21:26:29	MinDriver	NaN	NaN
7	7	2022-11-09	21:26:31	MinDriver	NaN	NaN
8	8	2022-11-09	21:26:34	TimeDriver	5.000000e-09	500.0
9	9	2022-11-09	21:52:01	MinDriver	NaN	NaN
10	10	2022-11-09	21:52:03	MinDriver	NaN	NaN
11	11	2022-11-09	21:52:05	TimeDriver	5.000000e-09	500.0
12	12	2022-11-10	02:55:42	MinDriver	NaN	NaN
13	13	2022-11-10	02:55:44	MinDriver	NaN	NaN
14	14	2022-11-10	02:55:46	TimeDriver	5.000000e-09	500.0
15	15	2022-11-10	02:55:59	TimeDriver	5.000000e-09	500.0
16	16	2022-11-10	02:56:12	TimeDriver	5.000000e-09	500.0
17	17	2022-11-10	02:56:27	TimeDriver	5.000000e-09	500.0
18	18	2022-11-10	02:56:46	TimeDriver	5.000000e-09	500.0
19	19	2022-11-10	02:58:32	TimeDriver	5.000000e-09	500.0

4

November 10, 2022

1 MicroMagnetic Simulation: Skyrmion

Here, we simulate/visualise/analyse a single Neel Skyrmion in a disk with periodic boundary conditions.

The region is a $100nm$ wide disk of thickness $t = 10nm$. Boundary Condition: Periodic in X & Y.

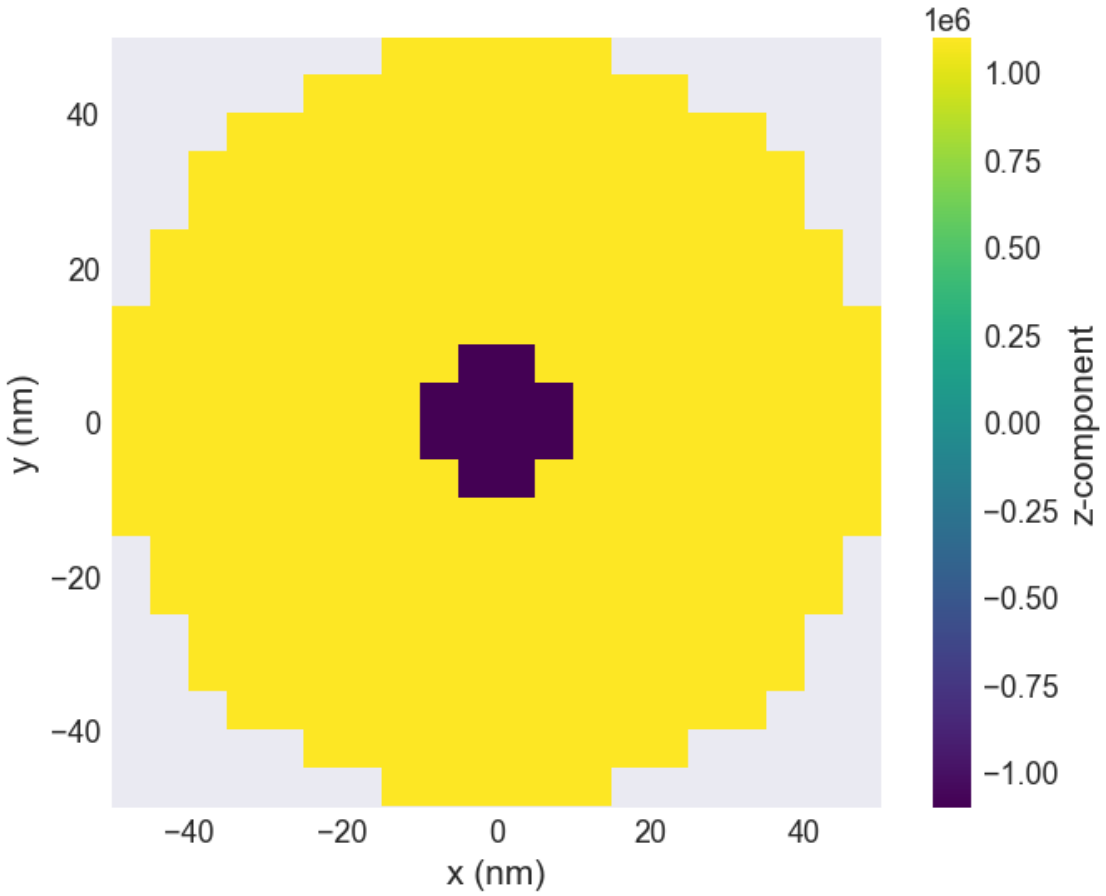
To create a skyrmion, we set magnetisation $(0, 0, -1)$ in a small cylindrical element, and $(0, 0, 1)$ otherwise.

For disk geometry, we define the norm: $\sqrt{x^2 + y^2} < \frac{r}{2}$.

[1]:
```
import micromagneticmodel as mm
import discretisedfield as df
import oommfc as mc
docker_runner = mc.oommf.DockerOOMMFRunner(image='oommf/oommf')

# Geometry
p1,p2 = (-50e-9,-50e-9,0), (50e-9,50e-9,10e-9)
cell = (5e-9, 5e-9, 5e-9)
region = df.Region(p1=p1, p2=p2)
mesh = df.Mesh(region=region, cell=cell, bc='xy') # Periodic wrt x

system = mm.System(name='skyrmion1')

# Energies: zeeman, exchange, anisotropy, dmi (interficial)
H = (0,0,1e5)
A = 1.6e-11
K, u = 0.2e6, (0,0,1)
D = 4e-3

system.energy   = (mm.Zeeman(H=H)
                 + mm.Exchange(A=A)
                 + mm.UniaxialAnisotropy(K=K, u=u)
                 + mm.DMI(D=D, crystalclass='Cnv_z'))

# Initial Conditions:
```

```
Ms = 1.1e6
def _Ms(point):
    x, y, z = point
    if (x**2 + y**2)**0.5 < 50e-9:
        return Ms
    else:
        return 0

def _m(point):
    x,y,z = point
    if (x**2 + y**2)**0.5 < 10e-9:
        return (0,0,-1)
    else:
        return (0,0,1)

# Initial State
system.m        = df.Field(mesh,dim=3,value=_m, norm=_Ms)
system.m.plane('z').mpl()
```

```
C:\Users\me\AppData\Roaming\Python\Python311\site-
packages\matplotlib\quiver.py:658: RuntimeWarning: divide by zero encountered in
double_scalars
  length = a * (widthu_per_lenu / (self.scale * self.width))
C:\Users\me\AppData\Roaming\Python\Python311\site-
packages\matplotlib\quiver.py:658: RuntimeWarning: invalid value encountered in
multiply
  length = a * (widthu_per_lenu / (self.scale * self.width))
```



[2]:
```
system.energy
```

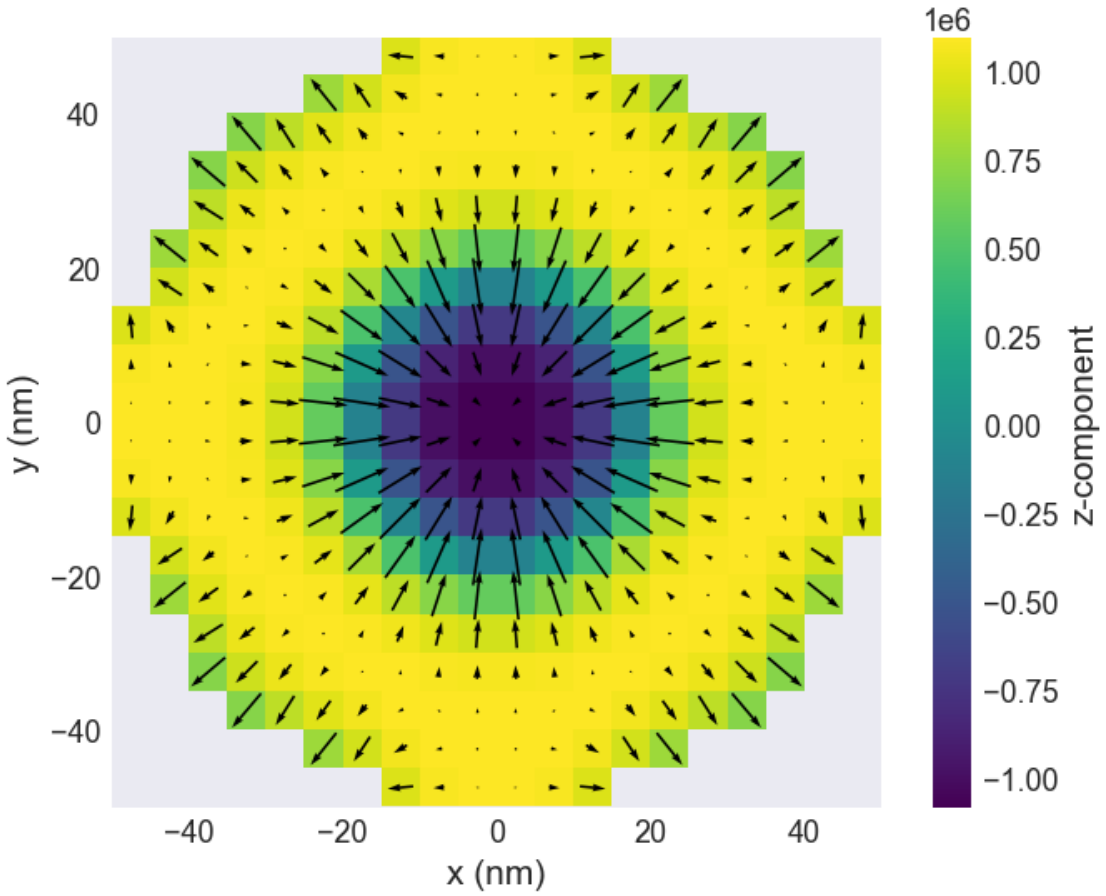
[2]: $-\mu_0 M_\mathrm{s}\mathbf{m}\cdot\mathbf{H} - A\mathbf{m}\cdot\nabla^2\mathbf{m} - K(\mathbf{m}\cdot\mathbf{u})^2 + D(\mathbf{m}\cdot\nabla m_z - m_z\nabla\cdot\mathbf{m})$

[3]:
```
system.dynamics
```

[3]: 0

[4]:
```
# Relaxed State
driver = mc.MinDriver()
driver.drive(system, runner=docker_runner)
system.m.plane('z').mpl()
```

```
Running OOMMF (DockerOOMMFRunner)[2022/11/10 04:33]... (1.6 s)
```

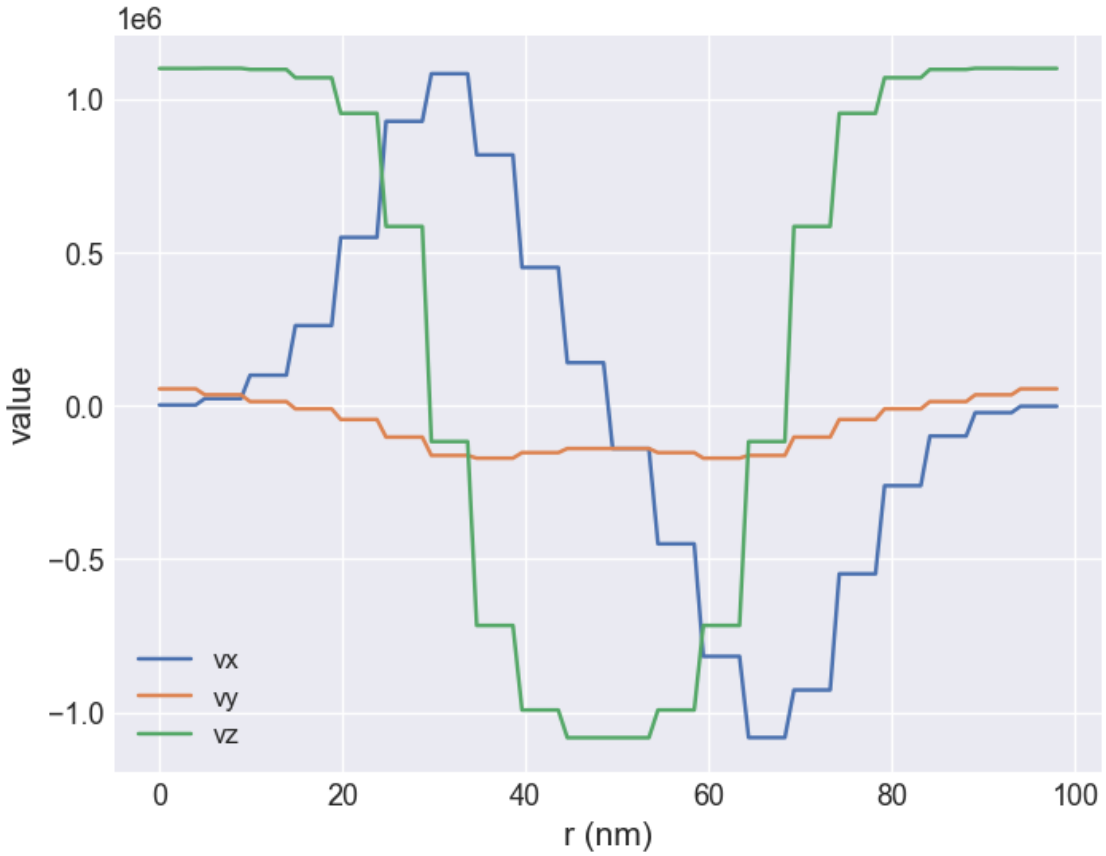


We can visualise **magnetisation** $(\vec{m}_x, \vec{m}_y, \vec{m}_z)$ along an axis passing through the centre of the cylinder.

This helps us understand how magnetisation changes for an element passing through this field (to model behaviour for a passing current).

As visible, there's a spin-flip for X, Z components.

[7]:
```
p1, p2 = (-49e-9, 0, 0), (49e-9, 0, 0)
system.m.line(p1=p1, p2=p2, n=100).mpl()
```


1e6
1.0
0.5
0.0
−0.5
−1.0
value
vx
vy
vz
0
20
40
60
80
100
r (nm)

5

November 14, 2022

1 MicroMagnetic Simulation: External Fields

Constant, Spatially-Variable, Time-Variable Fields

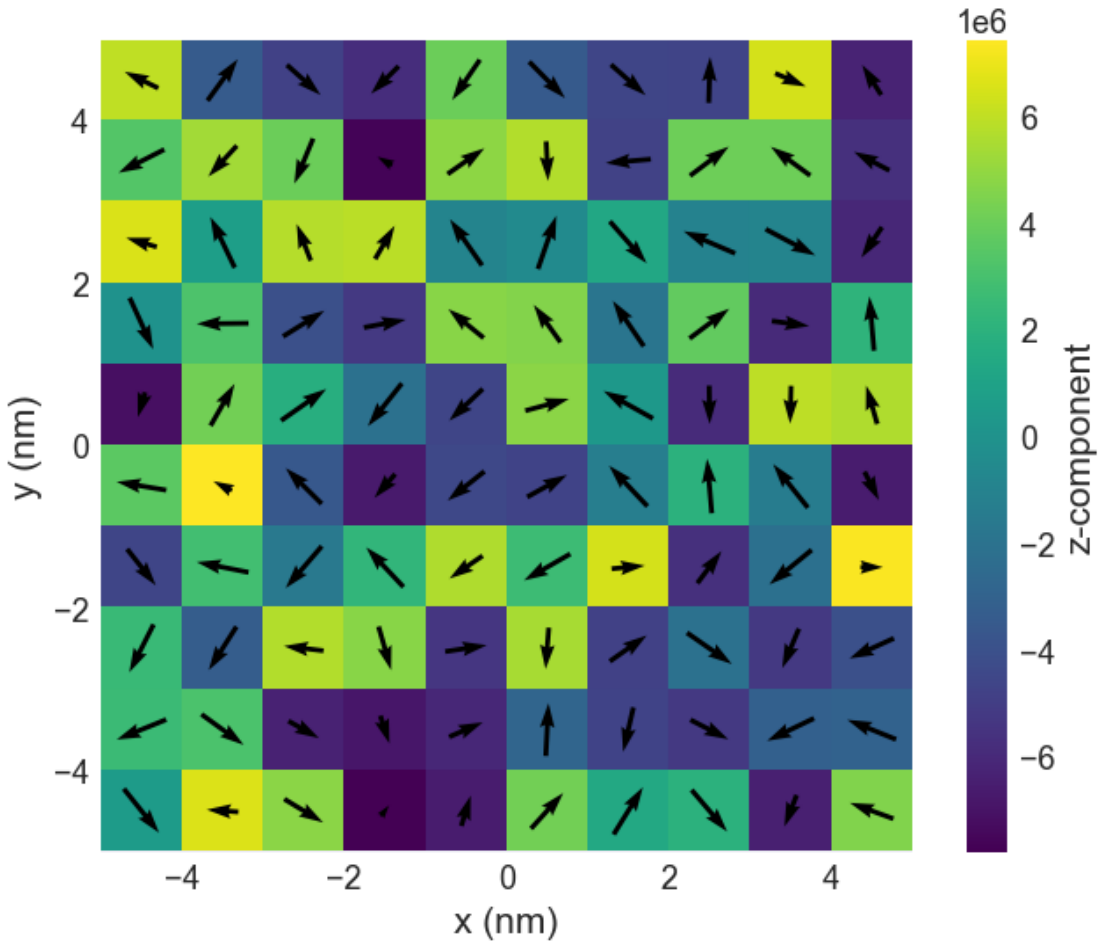
- A constant field, $\vec{H} = \hat{i} + \hat{j} + \hat{k}$ can be specified by: `H = (1,1,1)  m=df.Field(mesh, dim=3, value=H, norm=Ms)`
- A spatially-varying field, $\vec{H}$ can be specified by a function: "' def H(point): x,y,z = point if y > 0: return (1,0,0) else: return (-1,1,0)

  m=df.Field(mesh, dim=3, value=H, norm=Ms) "'
- A time-varying field, $\vec{H} = \vec{H_0} sin(2\pi f(t - t_0))$ can be specified by defining the amplitude (vector) $\vec{H_0}$ separately, and the sinusoid, in the Zeeman energy term. `H = (1,1,1)  ...  system.energy = mm.Zeeman(H=H, wave='sin', f=1e9, t0=1e-9)  # + ... other energies`
- Multiple fields can be specified by specifying individual Zeeman terms

[1]:
```
import discretisedfield as df
import micromagneticmodel as mm
import random
random.seed(1)


# Region
p1,p2 = (-5e-9,-5e-9,-5e-9), (5e-9,5e-9,5e-9)
region = df.Region(p1=p1, p2=p2)
cell = (1e-9, 1e-9, 1e-9)
mesh = df.Mesh(region=region, cell=cell)

# Fields - H1,H2,H3
H1 = (1,1,1)
H1 = df.Field(mesh, dim=3, value=H1)
def H2(point):
    x,y,z = point
    if y > 0:
        return (1,0,0)
```

```
    else:
        return (-1,1,0)
H2 = df.Field(mesh, dim=3, value=H2)
H3 = H1

# Energies: zeeman only
system = mm.System(name='multifield')
system.energy = (mm.Zeeman(H=H1, name='zeeman1')
                +mm.Zeeman(H=H2, name='zeeman2')
                +mm.Zeeman(H=H3, wave='sin', f=1e9, t0=1e-9, name='zeeman3'))
# Dynamics: precession/damping
gamma0, alpha = 2.211e5, 0.2
system.dynamics = mm.Precession(gamma0=gamma0) + mm.Damping(alpha=alpha)

# Magnetization
Ms = 8e6
def _m(point):
    # defines magnetization in range [-1,1] for x,y,z directions
    return [2*random.random()-1 for i in range(3)]
system.m = df.Field(mesh,dim=3,value=_m, norm=Ms)
```

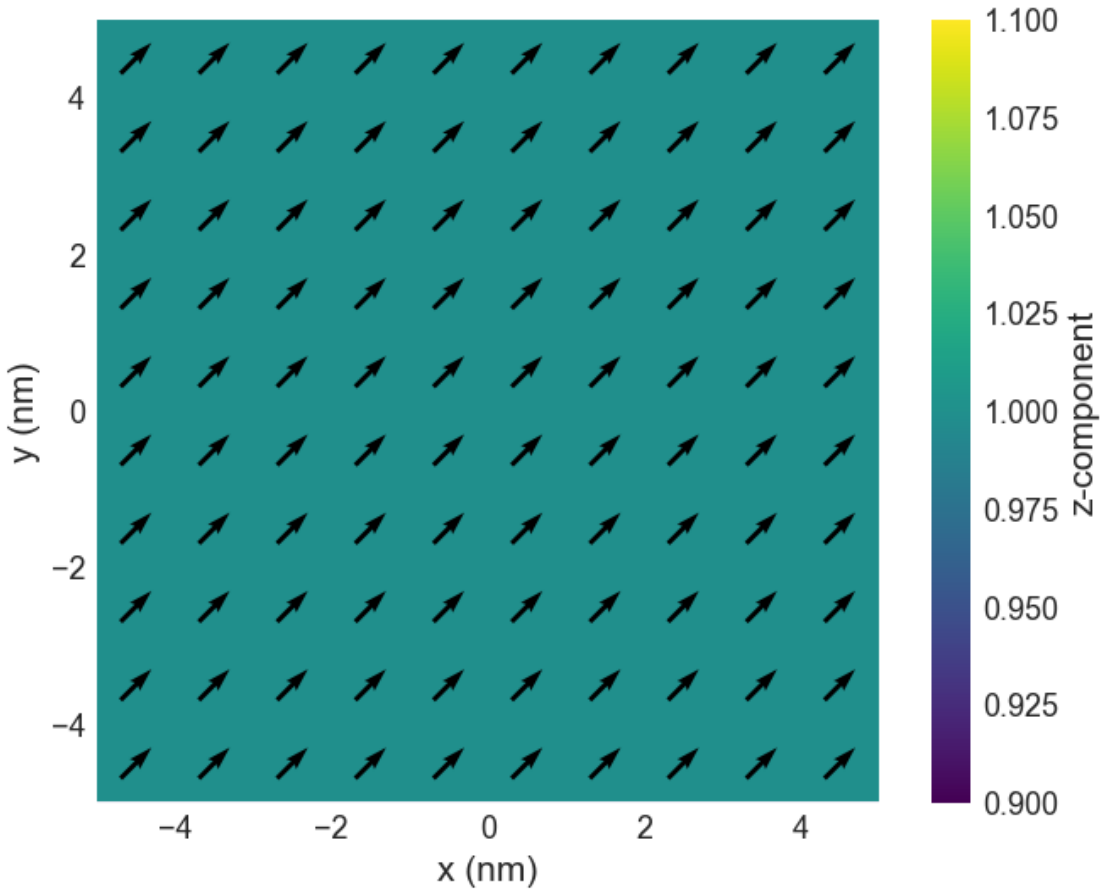
[2]:
```
system.m.plane('z').mpl()
```

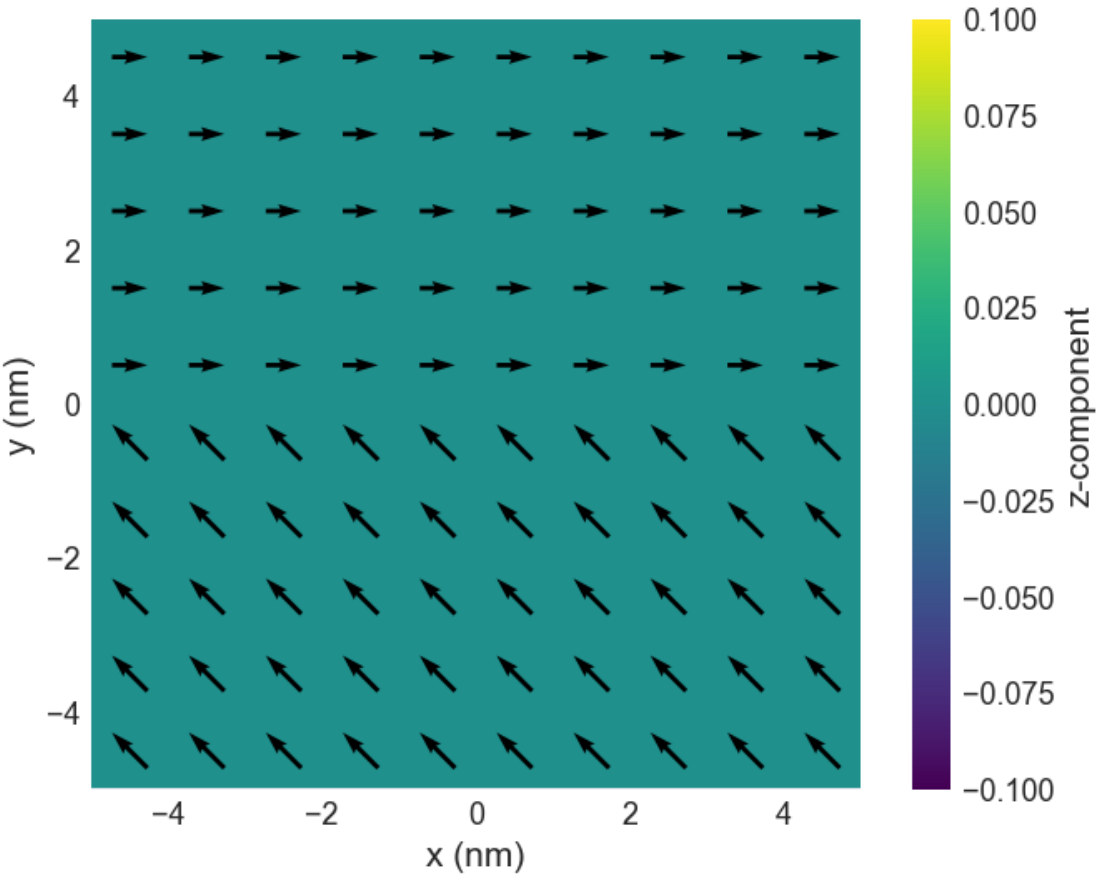


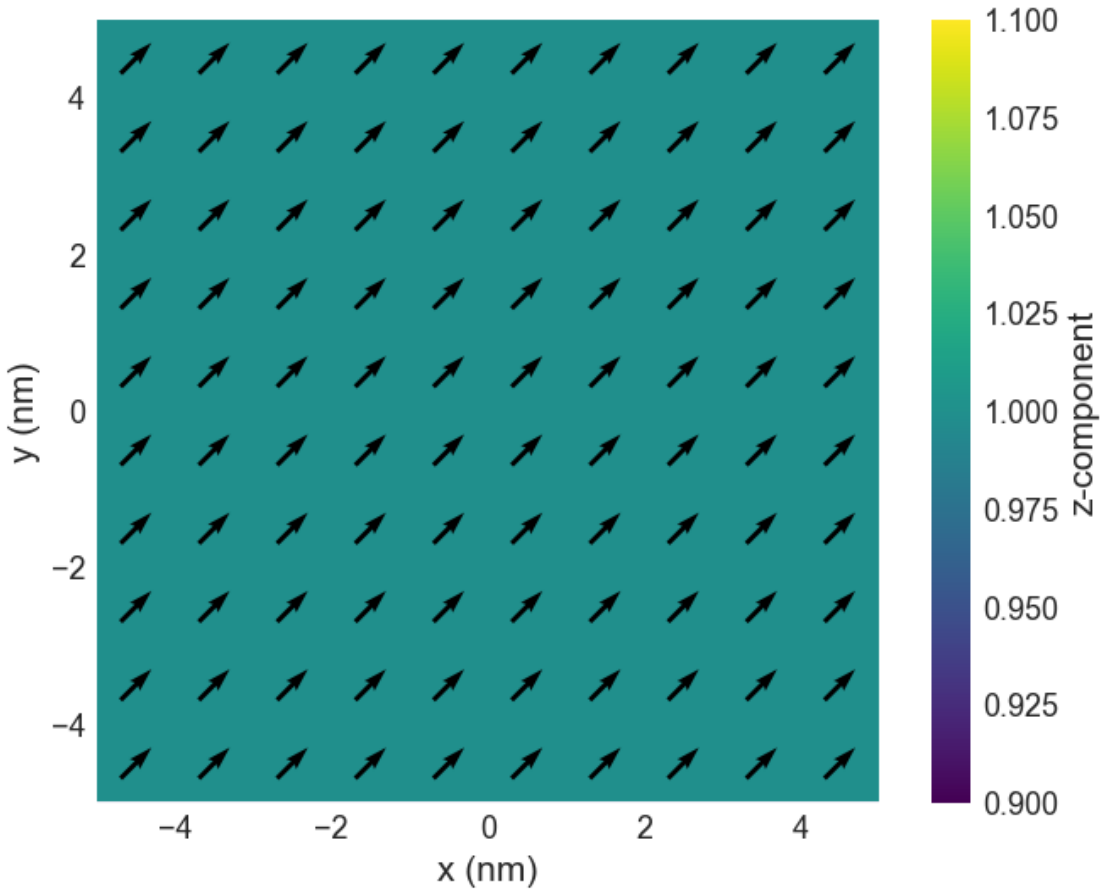
[3]:
```
system.m.plane('z').k3d.vector(head_size=5)
```

```
Output()
```

[4]:
```
H1.plane('z').mpl()
H2.plane('z').mpl()
H3.plane('z').mpl()
```


4
2
0
−2
−4
y (nm)
−4
−2
0
2
4
x (nm)
1.100
1.075
1.050
1.025
1.000
0.975
0.950
0.925
0.900
z-component


y (nm)
4
2
0
−2
−4
−4
−2
0
2
4
x (nm)
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
z-component


y (nm)
4
2
0
−2
−4
−4
−2
0
2
4
x (nm)
1.100
1.075
1.050
1.025
1.000
0.975
0.950
0.925
0.900
z-component

Chapter 3

Circuit-Level Simulation of Spintronic Devices

3.1 Introduction

The need for circuit-level modeling of spintronic phenomena arose with the experimental observations of GMR effect (circa 1988) when magnetoresistance (MR) values of $\approx 80\%$ were observed, first at low temperatures ($4.2K$), then, at room temperature ($300K$).

In the **G-matrix** approach, we specify a 4×4 transconductance matrix that relates 4×1 currents to 4×1 voltages (out of 4, 1 is associated with charge, while 3 are associated with spin (along X, Y, Z directions)).

3.2 SPICE for Circuit-Level Simulation

SPICE is a modified nodal analysis (MNA) solver. **?**
A circuit is formulated in terms of nodal equations (ie, $GV = I$, in matrix notation). The nodes are related to various branches using VI relations (eg. $V_R = RI$, $V_L = L\frac{dI}{dt}$, $V_C = V_0 + \frac{1}{C}\int Idt \leftarrow$ for R, L, C respectively).
The "modified" part has to do with the fact that nodal equations cannot resolve **voltage sources**, as these have infinite conductances. To fix this, the nodal equations for nodes V_x & V_y of a voltage source are replaced by a single KCL equation consisting of currents moving in/out of both the nodes, and a separate equation $V_x = V_y + V_{xy} \leftarrow$ representing the voltage source.
The choice of G is key, because two disconnected nodes have $G = 0$; and practical circuits have several nodes, yet each node is connected to $\approx 3 - 4$ nodes at best.

In Matrix terminology, we'll be dealing with **sparse conductance matrices** - which can be solved using **sparse matrix solvers**.
This explains why Spintronic modules are modelled in terms of **G-matrices**.

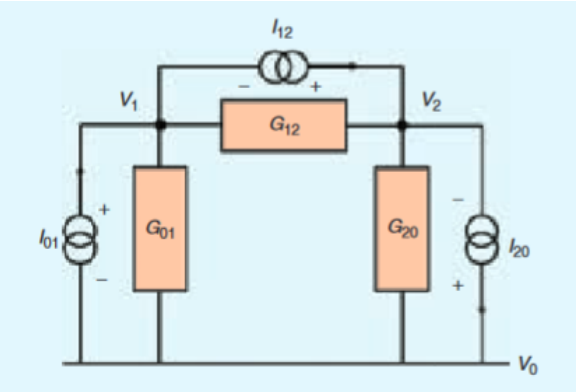


Figure 3.1: π-network with 3 Ohmic conductances & 3 known current sources

Another concept is **device stamp**. Formulating the circuit in fig1, we have:

$$
\begin{aligned}
G_{01}(V_1 - V_0) - G_{20}(V_0 - V_2) + I_{20} - I_{01} = 0 \\
G_{12}(V_2 - V_1) - G_{01}(V_1 - V_0) + I_{01} - I_{12} = 0 \\
G_{20}(V_0 - V_2) - G_{12}(V_2 - V_1) + I_{12} - I_{20} = 0
\end{aligned}
$$

$$
\begin{bmatrix} (G_{01}+G_{20}) & -G_{01} & -G_{20} \\ -G_{01} & (G_{12}+G_{01}) & -G_{12} \\ -G_{20} & -G_{12} & (G_{20}+G_{12}) \end{bmatrix} \begin{bmatrix} V_0 \\ V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} I_{20}-I_{01} \\ I_{01}-I_{12} \\ I_{12}-I_{20} \end{bmatrix}
$$

$$
\begin{bmatrix} (G_{01}+G_{20}) & -G_{01} & -G_{20} \\ -G_{01} & (G_{12}+G_{01}) & -G_{12} \\ -G_{20} & -G_{12} & (G_{20}+G_{12}) \end{bmatrix} \begin{bmatrix} V_0 \\ V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} I_{20}-I_{01} \\ I_{01}-I_{12} \\ I_{12}-I_{20} \end{bmatrix}
$$

Figure 3.2: **device stamps** of G_{01}, G_{20}, G_{12}

We observe that the conductance G of every element stamps itself in the form: $\begin{bmatrix} G_{xy} & -G_{xy} \\ -G_{xy} & G_{xy} \end{bmatrix}$ where G_{xy} is the conductance between nodes x, y.

Although here it's obtained for 2-terminal devices, the same holds for 3-4 terminal devices as well. We therefore, have n **device stamps** for a circuit containing n elements. This allows us to represent each element using its **device stamp** in the expression for G. In matrix form, each element can be represented using: $g(u_i - u_j)(u_i - u_j)^T$
where u_i is a unit row-vector. This allows quick calculation & validation of solutions of the nodal equation $GV = I$, obtained using $V = G^{-1}I$.

3.3 SPICE for Non-Local SpinValve (NLSV) Simulation

NLSVs are commonly used for testing spin-transport models. As shown, a charge current is injected into the **injector ferromagnet (FM_1)**, which gets spin-polarised, and diffuses into the **non-magnetic channel (NM)**, and is detected by the **detector ferromagnet (FM_2)** as a charge voltage. The consequent **non-local resistance** R_{NL} when positive indicates $FM_{1,2}$ to be parallel, and antiparallel when negative. ($R_{NL} > 0 \Rightarrow FM_{1,2} \uparrow\uparrow$; $R_{NL} < 0 \Rightarrow FM_{1,2} \uparrow\downarrow$)

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \\ I_5 \\ I_6 \\ I_7 \end{bmatrix} = [G]_{7\times 7} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \end{bmatrix}$$

where $G_{7\times 7}$ is given by: **?**

$$\begin{bmatrix} G_{FM1}+G'_{FM1} & -G_{FM1} & 0 & 0 & 0 & 0 & 0 \\ -G_{FM1} & G_{FM1}+G'_{FM1}+G_{TB1} & 0 & 0 & -G_{TB1} & 0 & 0 \\ 0 & 0 & G_{FM2}+G'_{FM2} & -G_{FM2} & 0 & 0 & 0 \\ 0 & 0 & -G_{FM2} & G_{FM2}+G'_{FM2}+G_{TB2} & 0 & -G_{TB2} & 0 \\ 0 & -G_{TB1} & 0 & 0 & G_{NM1}+G'_{NM1}+G_{NM2}+G'_{NM2}+G_{TB1} & -GNM2 & 0 \\ 0 & 0 & 0 & -G_{TB2} & -G_{NM2} & G_{NM2}+G'_{NM2}+G_{NM3}+G'_{NM3}+G_{TB2} & -G_{NM3} \\ 0 & 0 & 0 & 0 & 0 & -G_{NM3} & G_{NM3}+G'_{NM3} \end{bmatrix}$$

where $G, G^{'}$ represent series & shunt conductance matrices for the layers. FM, NM, TB represent ferromagnetic, non-magnetic & tunnel-barrier layers. TB layer doesn't have shunt component as its thickness is much smaller than spin-diffusion length.
R_{NL} is analytically derived in **?**. A SPICE solver approach, gives this result: (the code is given in the Appendix)

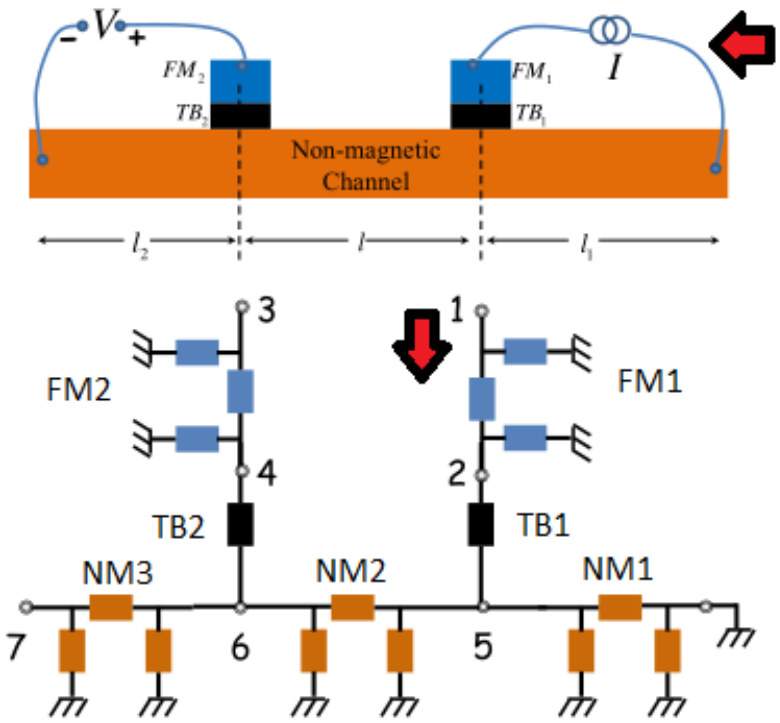


Figure 3.3: Circuit-level model for NLSV. $R_{NL} = \frac{V_{3c}-V_{7c}}{I_{1c}}$
?

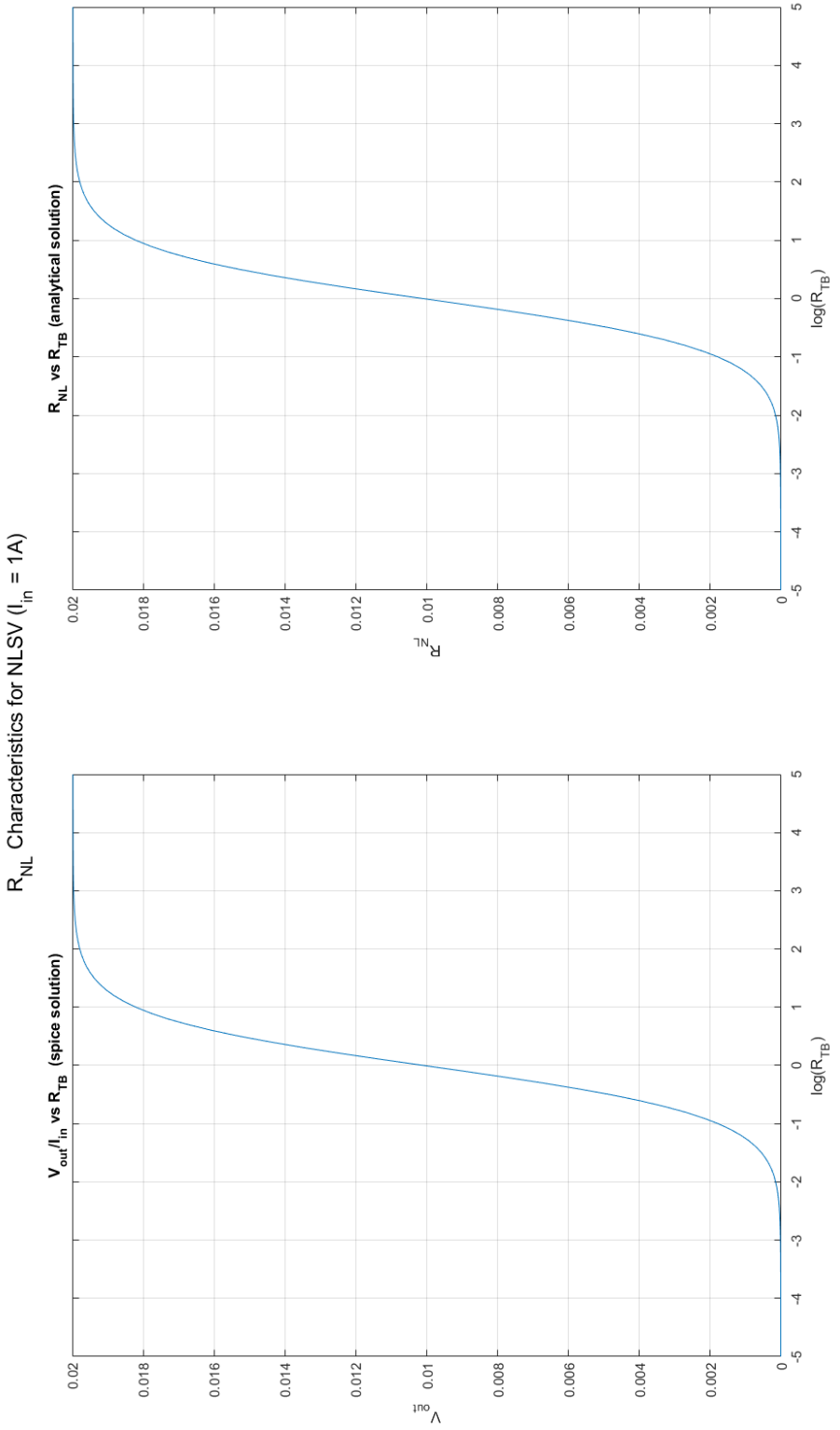


Figure 3.4: Circuit-level model for NLSV: R_{NL} v/s R_{TB}

6

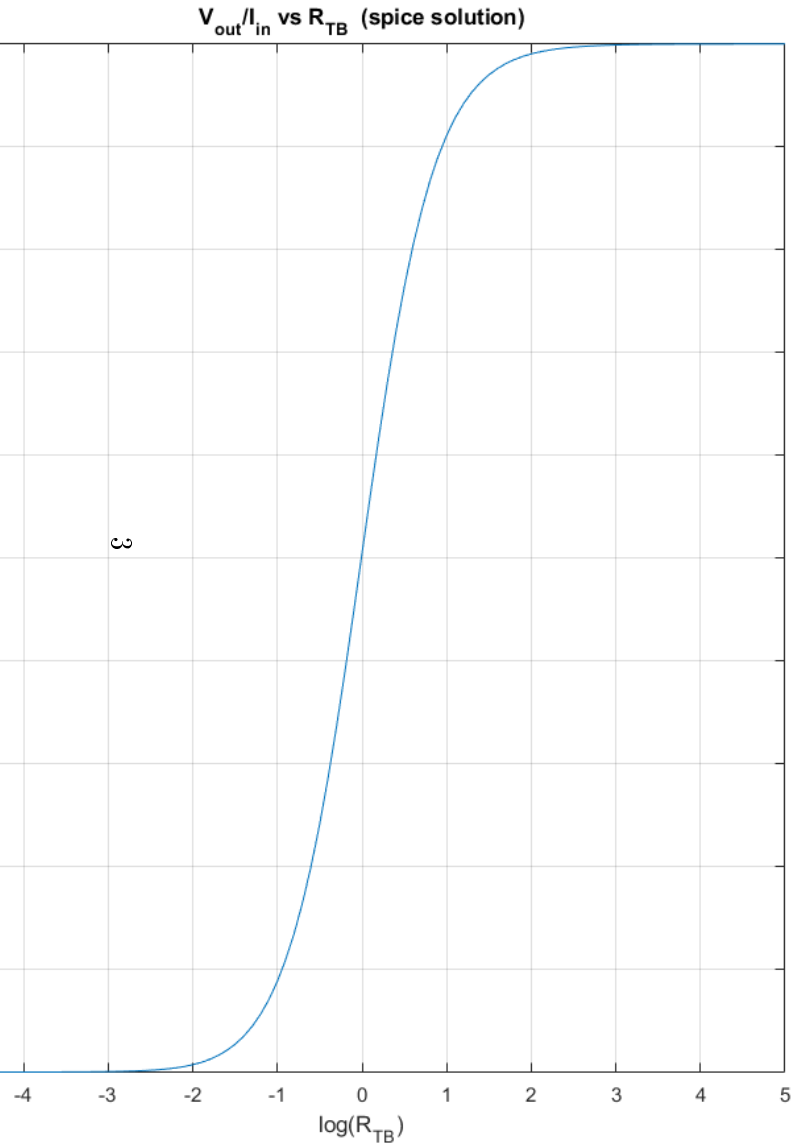
November 14, 2022

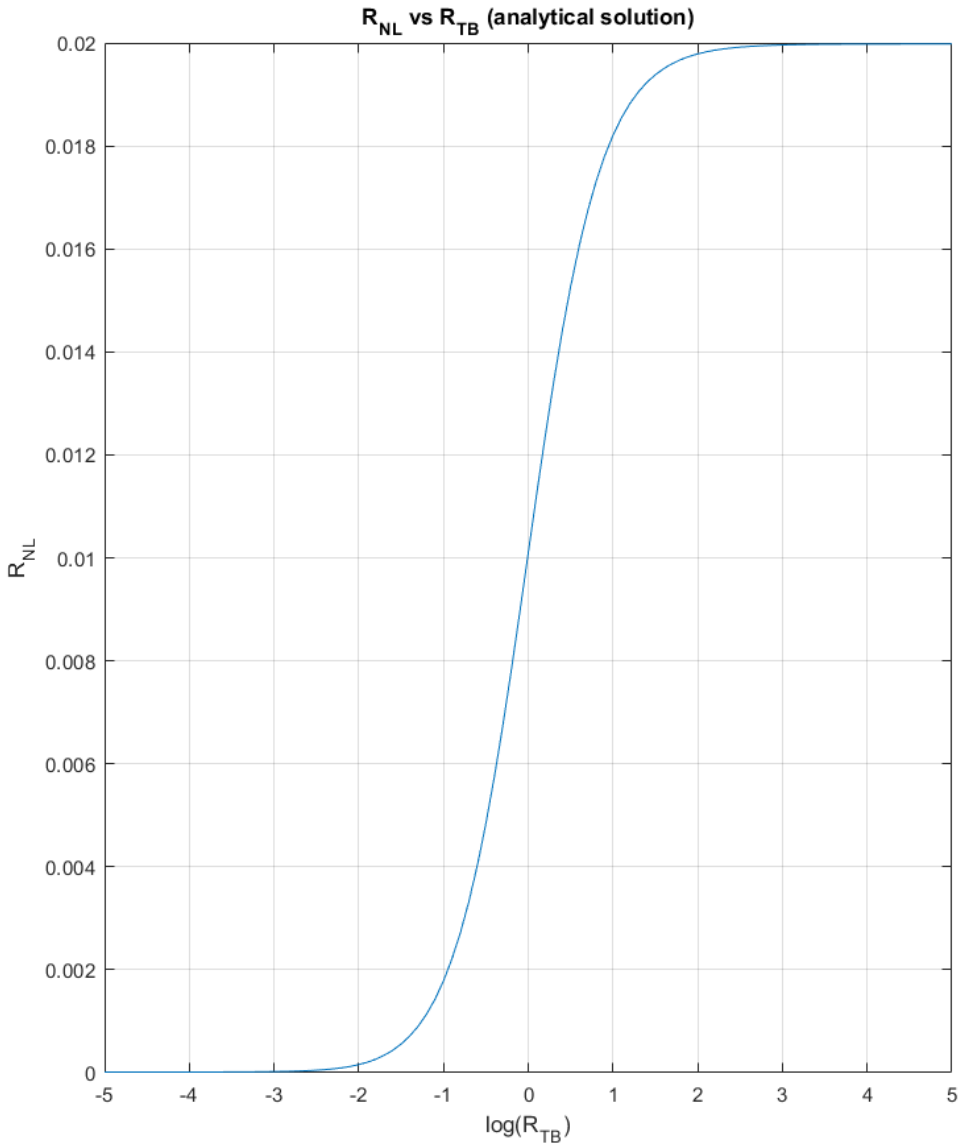
1 SPICE Simulations: Magnetoresistance of a SpinValve

[ ]:
```
clc; clear all; close all;

%Constants (all MKS, except energy which is in eV)
q=1.6e-19;Z=zeros(2,2);

% Parameters
% note: R => rho*lambda_sf/A; L => L/lambda_sf
ii=0;
for X=-5:0.1:5
   ii=ii+1; RT(ii)=X; RT1=10^X; RT2=10^X;% Tunnel Resistance
   PT1=0.2; PT2=0.2;% Polarization of Tunnel Contacts
   RF=1e-2; LF1=100; LF2=100; PF1=0.05; PF2=0.05;% Ferromagnetic contacts
   RN=1; LN1=100; LN2=1e-3; LN3=100;% Nonmagnetic Channel

  % FM layers
  G_FM1 = ((1/RF/LF1)*[1 PF1;PF1 PF1*PF1])+(((1-PF1*PF1)/RF)*[0 0; 0 ...
            csch(LF1)]);
  G_FM2 = ((1/RF/LF2)*[1 PF2;PF2 PF2*PF2])+(((1-PF2*PF2)/RF)*[0 0; 0 ...
            csch(LF2)]);
  G0_FM1 = ((1-PF1*PF1)/RF)*[0 0;0 coth(LF1)-csch(LF1)];
  G0_FM2 = ((1-PF2*PF2)/RF)*[0 0;0 coth(LF2)-csch(LF2)];

  % NM channel
  G_NM1 = (1/RN/LN1)*[1 0;0 LN1*csch(LN1)];
  G_NM2 = (1/RN/LN2)*[1 0;0 LN2*csch(LN2)];
  G_NM3 = (1/RN/LN3)*[1 0;0 LN3*csch(LN3)];
  G0_NM1 = (1/RN)*[0 0;0 coth(LN1)-csch(LN1)];
  G0_NM2 = (1/RN)*[0 0;0 coth(LN2)-csch(LN2)];
  G0_NM3 = (1/RN)*[0 0;0 coth(LN3)-csch(LN3)];

  % Tunnel Barrier
  G_TB1=(1/RT1)*[1 PT1;PT1 1];
  G_TB2=(1/RT2)*[1 PT2;PT2 1];
```

```
  % Conductance matrix from KCL
  G = [G0_FM1+G_FM1 -G_FM1 Z Z Z Z Z;
    -G_FM1 G0_FM1+G_FM1+G_TB1 Z Z -G_TB1 Z Z;
    Z Z G0_FM2+G_FM2 -G_FM2 Z Z Z;
    Z Z -G_FM2 G0_FM2+G_FM2+G_TB2 Z -G_TB2 Z;
    Z -G_TB1 Z Z G_NM1+G0_NM1+G0_NM2+G_NM2+G_TB1 -G_NM2 Z;
    Z Z Z -G_TB2 -G_NM2 G_NM3+G0_NM2+G0_NM3+G_NM2+G_TB2 -G_NM3;
    Z Z Z Z Z -G_NM3 G_NM3+G0_NM3];

  % SPICE Solution
  C = [1; PF1; zeros(12,1)];% Terminal currents
  V=G\C; V=reshape(V,2,7);% Terminal voltages
  Vout(ii)=V(1,3)-V(1,7);% Output voltage

  % Analytical Solution
  RF1=RF; RF2=RF;
  Numer = 2*RN*exp(-LN2)*(PT1*RT1/RN/(1-PT1^2) + PF1*RF1/RN/(1-PF1^2))...
    *(PT2*RT2/RN/(1-PT2^2) + PF2*RF2/RN/(1-PF2^2));
  denom = (1+ 2*RT1/RN/(1-PT1^2) + 2*RF1/RN/(1-PF1^2))...
    *(1+ 2*RT2/RN/(1-PT2^2) + 2*RF2/RN/(1-PF2^2)) - exp(-LN2);
  Rnon_local(ii)=Numer/denom;
end

figure;
subplot(1,2,1); plot(RT, Vout/C(1)); ylabel(['V_{out}']);␣
 ↪xlabel(['log(R_{TB})']); title(['V_{out}/I_{in} vs R_{TB}  (spice solution)']);
 ↪ grid on;
subplot(1,2,2); plot(RT, Rnon_local); ylabel(['R_{NL}']);␣
 ↪xlabel(['log(R_{TB})']); title(['R_{NL} vs R_{TB} (analytical solution)']);␣
 ↪grid on;
sgtitle(['R_{NL} Characteristics for NLSV (I_{in} = 1A)']);
```

R_{NL} Characteristics for NLSV (I_{in} = 1A)





Chapter 4

Conclusion & Future Work

As a part of this study, I have gain insight into the various magnetic phenomena that operate at the nanometer scale. I have performed analytical & spice simulations for a spinvalve, as discussed in **?**, and modelled micromagnetic simulations for skyrmions.
The next steps for my work include:

1. SPICE simulation of MTJ module as described in **?** & **?**.
2. SPICE simulation of digital logic using MTJs based on work on spinvalves in **?**.
3. Micromagnetic simulations of skyrmion dynamics inside an MTJ to determine magnetic properties suitable for circuit-level simulation.
4. Circuit-level simulation of MTJs containing skyrmions.