Computational micromagnetics with Commics

Most recent ASC Reports

32/2018  A. Gerstenmayer and A. Jüngel
Comparison of a finite-element and finite-volume scheme for a degenerate cross-diffusion system for ion transport

31/2018  A. Bespalov, D. Praetorius, L. Rocchi, and M. Ruggeri
Convergence of adaptive stochastic Galerkin FEM

30/2018  L. Chen, E.S. Daus, and A. Jüngel
Rigorous mean-field limit and cross diffusion

29/2018  X. Huo, A. Jüngel, and A.E. Tzavaras
High-friction limits of Euler flows for multicomponent systems

Two time discretizations for gradient flows exactly replicating energy dissipation

27/2018  A. Jüngel and M. Ptashnyk
Homogenization of degenerate cross-diffusion systems

26/2018  F. Auer and E.B. Weinmüller
Convergence of the collocation schemes for systems of nonlinear ODEs with a time singularity

25/2018  M. Karkulik, J.M. Melenk
$H$-matrix approximability of inverses of discretizations of the fractional Laplacian

24/2018  M. Bernkopf and J.M. Melenk
Analysis of the $hp$-version of a first order system least squares method for the Helmholtz equation

23/2018  T. Jawecki, W. Auzinger, O. Koch
Computable strict upper bounds for Krylov approximations to a class of matrix exponentials and $\phi$-functions
COMPUTATIONAL MICROMAGNETICS WITH COMMICS

CARL-MARTIN PFEILER, MICHELE RUGGERI, BERNHARD STIFTNER, LUKAS EXL, MATTHIAS HOCHSTEGER, GINO HRKAC, JOACHIM SCHÖBERL, NORBERT J. MAUSER, AND DIRK PRAETORIUS

Abstract. We present our open-source Python module Commics for the study of the magnetization dynamics in ferromagnetic materials via micromagnetic simulations. It implements state-of-the-art unconditionally convergent finite element methods for the numerical integration of the Landau–Lifshitz–Gilbert equation. The implementation is based on the multiphysics finite element software Netgen/NGSolve. The simulation scripts are written in Python, which leads to very readable code and direct access to extensive post-processing. Together with documentation and example scripts, the code is freely available on GitLab.

1. Introduction

Micromagnetism is a continuum theory for ferromagnetic materials located between Maxwell’s electromagnetism and quantum theory [32, 11, 56]. The magnetization distribution is modeled as a continuous vector field, where nonlocal magnetostatic interactions and local contributions are taken into account. Typical micromagnetic models exhibit length scales ranging from nanometers to few micrometers, which is often infeasible for atomistic spin dynamics simulations. In recent decades, micromagnetics evolved as a computational field, that nowadays represents a successful tool for numerical studies in materials science with important contemporary applications, e.g., data storage structures like hard disk drives [76, 54], random access memories [61], or nanowires [73, 50], magnetologic devices [24], soft magnetic sensor systems [34], and high performance permanent magnets [72, 23, 43].

1.1. Existing software. In recent years, advances in computer architecture, programming environments, and numerical methods led to the development of several micromagnetic codes, which aim at the numerical integration of the fundamental equation in micromagnetics, the Landau–Lifshitz–Gilbert equation (LLG), a well-accepted model for the magnetization dynamics [58, 48]. Well-known simulation packages based on finite difference discretizations [63] on Cartesian grids are OOMMF [39], recently extended for GPU...
usage [16] and endowed with a user-friendly Python interface [27, MuMax³ (GPU) [78, MicroMagnum [5 (CPU and GPU), magnum.fd [4, and Fidimag [31]. These implementations are both memory and computationally efficient owing to the uniform mesh and the particularly advantageous utilization of the fast Fourier transform for the long range field part [29, 9]. However, approaches based on the Finite Element Method (FEM) are geometrically more flexible [71] and provided in the scientific codes MagPar [70, TetraMag [53] and its successor tetmag2, Nmag [44] and its successor Finmag [30], as well as in commercial software like FastMag [35, FEMME [3, and magnum.fe [8].

1.2. Numerical analysis. For an introduction to the mathematical analysis of numerical integrators for dynamic micromagnetic simulations, we refer to the monographs [66, 18], the review articles [57, 47, 36], and the references therein. The ultimate goal is the development of unconditionally convergent integrators, i.e., numerical schemes for which (a subsequence of) the output converges towards a weak solution of LLG in the sense of [15] without requiring any restrictive CFL-type coupling condition on the temporal and spatial discretization parameters. In our work, we consider two types of methods characterized by such good theoretical properties: the tangent plane scheme [12, 13] and the midpoint scheme [25]; see Section 3 for more details.

1.3. Contributions. This work presents our novel open-source Python module Commics (COMputational MicroMagnetICS) to perform computational studies of the magnetization dynamics in ferromagnetic materials via micromagnetic simulations. The software is based on the multiphysics finite element software Netgen/NGSolve [6] and is made user-friendly by a high-level Python interface. While many existing codes popular in the physics community are fairly performance optimized, they often lack a thorough mathematical convergence analysis. In contrast to that, our implementation arises from recent results in the numerical analysis of unconditionally convergent LLG integrators. The code is freely available on GitLab [1] together with documentation and example scripts.

1.4. Outline. This work is organized as follows: We fix the notation and the precise micromagnetic setting in Section 2. In Section 3 the implemented algorithms are briefly presented. Section 4 demonstrates the exemplary use of Netgen/NGSolve (NGS) and discusses the integration of the boundary element library BEM++ [75]. Finally, Section 5 provides Python scripts for several benchmark problems, in order to verify our module and to demonstrate its usage.

2. Micromagnetic setting

Let \( \Omega \subset \mathbb{R}^3 \) denote the volume occupied by a ferromagnet. In micromagnetics, the quantity of interest is the magnetization \( \mathbf{M} : \Omega \rightarrow \mathbb{R}^3 \) (in A/m). If the temperature is constant and far below the so-called Curie temperature of the material, the modulus of the magnetization is constant, i.e., it holds that \( |\mathbf{M}| = M_s \) with \( M_s > 0 \) being the saturation magnetization (in A/m). Let \( \mathbf{m} := \mathbf{M}/M_s \) denote the normalized magnetization. The
magnetic state of \( \Omega \) is described in terms of the magnetic Gibbs free energy (in J)

\[
\mathcal{E}(\mathbf{m}) = A \int_{\Omega} |\nabla \mathbf{m}|^2 \, dx - K \int_{\Omega} (\mathbf{a} \cdot \mathbf{m})^2 \, dx + D \int_{\Omega} (\nabla \times \mathbf{m}) \cdot \mathbf{m} \, dx \\
- \frac{\mu_0 M_s}{2} \int_{\Omega} \mathbf{H}_s \cdot \mathbf{m} \, dx - \mu_0 M_s \int_{\Omega} \mathbf{H}_{\text{ext}} \cdot \mathbf{m} \, dx.
\]  

(1)

The energy in (1) is the sum of exchange energy, uniaxial anisotropy, bulk Dzyaloshinskii–Moriya interaction (DMI), magnetostatic energy, and Zeeman contribution, respectively. The involved material parameters and physical constants are the exchange stiffness constant \( A > 0 \) (in J/m), the anisotropy constant \( K \geq 0 \) (in J/m\(^3\)), the easy axis \( \mathbf{a} \in \mathbb{R}^3 \) with \(|\mathbf{a}| = 1 \) (dimensionless), the DMI constant \( D \in \mathbb{R} \) (in J/m\(^2\)), and the vacuum permeability \( \mu_0 = 4\pi \times 10^{-7} \text{ N/A}^2 \). Moreover, \( \mathbf{H}_{\text{ext}} \) and \( \mathbf{H}_s \) denote the applied external field (assumed to be unaffected by variations of \( \mathbf{m} \)) and the stray field, respectively (both in A/m). The stray field (sometimes also referred to as demagnetizing or dipolar field) solves the magnetostatic Maxwell equations

\[
\nabla \cdot (\mathbf{H}_s + M_s \mathbf{m}_\chi) = 0 \quad \text{in } \mathbb{R}^3, \\
\nabla \times \mathbf{H}_s = 0 \quad \text{in } \mathbb{R}^3,
\]

(2a)

(2b)

where \((\mathbf{m}_\chi)(\mathbf{x}) = \mathbf{m}(\mathbf{x})\) in \( \Omega \) and \((\mathbf{m}_\chi)(\mathbf{x}) = 0\) elsewhere. Stable magnetization configurations are those which minimize the magnetic Gibbs free energy (1). The dynamics towards equilibrium of the magnetization is governed by LLG

\[
\partial_t \mathbf{m} = -\gamma_0 \mathbf{m} \times \left[ \mathbf{H}_\text{eff}(\mathbf{m}) + \mathbf{T}(\mathbf{m}) \right] + \alpha \mathbf{m} \times \partial_t \mathbf{m} \quad \text{in } (0, \infty) \times \Omega, \\
\partial_t \mathbf{n} = -\frac{D}{2A} \mathbf{m} \times \mathbf{n} \quad \text{on } (0, \infty) \times \partial \Omega, \\
\mathbf{m}(0) = \mathbf{m}^0 \quad \text{with } |\mathbf{m}^0| = 1 \quad \text{in } \Omega.
\]

(3a)

(3b)

(3c)

In (3), \( \gamma_0 = 2.212 \times 10^5 \text{ m/(A s)} \) is the gyromagnetic ratio of the electron, \( \alpha \in (0, 1] \) is the dimensionless Gilbert damping parameter, and \( \mathbf{n} : \partial \Omega \to \mathbb{R}^3 \) with \(|\mathbf{n}| = 1 \) denotes the outward-pointing unit normal vector to \( \partial \Omega \). The effective field \( \mathbf{H}_\text{eff}(\mathbf{m}) \) is related to the functional derivative of the energy with respect to the magnetization and takes the form

\[
\mathbf{H}_\text{eff}(\mathbf{m}) : = -\frac{1}{\mu_0 M_s} \frac{\delta \mathcal{E}(\mathbf{m})}{\delta \mathbf{m}} = \frac{2A}{\mu_0 M_s} \Delta \mathbf{m} + \frac{2K}{\mu_0 M_s} (\mathbf{a} \cdot \mathbf{m}) \mathbf{a} - \frac{2D}{\mu_0 M_s} \nabla \times \mathbf{m} + \mathbf{H}_s + \mathbf{H}_{\text{ext}}.
\]

Finally, the term \( \mathbf{T}(\mathbf{m}) \) collects all nonenergetic torque terms, which arise, e.g., when an electric current flows in a conducting ferromagnet. For instance, the Oersted field \( \mathbf{T}(\mathbf{m}) = \mathbf{H}_c \) (in A/m) is described by the magnetostatic Maxwell equations

\[
\nabla \cdot \mathbf{H}_c = 0 \quad \text{in } \mathbb{R}^3, \\
\nabla \times \mathbf{H}_c = \mathbf{J}_c \chi_\Omega \quad \text{in } \mathbb{R}^3,
\]

(4a)

(4b)

where \( \mathbf{J}_c \) denotes the electric current density (in A/m\(^2\)). Two other prominent examples are related to the so-called spin transfer torque [74, 28], which arises in the presence of
spin-polarized currents. The Slonczewski contribution \cite{Slonczewski1996}, which takes the form

\[ T(m) = \frac{\hbar J_e G(m \cdot p, P)}{e\mu_0 M_s d} m \times p, \]

is used for the simulation of switching processes in structures with current-perpendicular-to-plane injection geometries, e.g., magnetic multilayers. The involved physical quantities are the reduced Planck constant \( \hbar > 0 \) (in J s), the elementary charge \( e > 0 \) (in A s), a dimensionless polarization parameter \( P \in (0,1) \), a dimensionless unit vector \( p \in \mathbb{R}^3 \) representing the magnetization of a uniformly-magnetized polarizing layer (the so-called fixed layer), and the thickness \( d > 0 \) of the so-called free layer (in m).

The Zhang–Li contribution \cite{ZhangLi1994} is used for the simulation of the current-driven motion of domain walls in single-phase samples characterized by current-in-plane injection geometries and, according to \cite{ZhangLi1994} (resp., \cite{ZhangLi1994a}), takes the form

\[ T(m) = -\frac{1}{\gamma_0} [m \times (u \cdot \nabla) m + \xi(u \cdot \nabla) m], \]

with \( u = -\frac{P \mu_B}{eM_s(1 + \xi^2)} J_e \) (resp., \( u = -\frac{P g_e \mu_B}{2eM_s} J_e \)).

The involved physical quantities are the spin velocity vector \( u \in \mathbb{R}^3 \) (in m/s), the dimensionless ratio of nonadiabaticity \( \xi > 0 \), the Bohr magneton \( \mu_B > 0 \) (in A m^2), and the dimensionless g-factor of the electron \( g_e \approx 2 \).

In \cite{Hillebrands2000} and \cite{Hillebrands2000a}, to fix the ideas, we considered the case of the bulk DMI as a prototype for chiral interactions \cite{Hillebrands2000b} \cite{Hillebrands2000c}. However, our implementation also covers the case of the interfacial DMI; see, e.g., \cite{Hillebrands2000d} \cite{Hillebrands2000e}. If \( D = 0 \) (no chiral interaction), then the boundary conditions \cite{Hillebrands2000d} become homogeneous Neumann boundary conditions.

3. Algorithms

The algorithms implemented in our Python module Commics employ a uniform partition of the time interval with constant time-step size \( \Delta t > 0 \). For the spatial discretization, we consider a tetrahedral mesh \( T_h \) of the ferromagnet \( \Omega \) with mesh size \( h > 0 \). The associated FEM space of piecewise affine and globally continuous functions reads

\[ V_h := \{ \varphi_h : \Omega \rightarrow \mathbb{R} \text{ continuous} : \varphi_h|_K \text{ is affine for all elements } K \in T_h \}. \]

For each time-step \( n = 0, 1, 2, \ldots \), we seek for approximations

\[ (V_h)^3 \ni m^n_h \approx m(n\Delta t) \text{ such that } |m^n_h(z)| = 1 \text{ for all nodes } z \text{ of } T_h, \]

i.e., for any time-step, the approximate magnetization satisfies the unit-length constraint \( |m| = 1 \) at the nodes of the mesh.

3.1. Tangent plane scheme. Tangent plane schemes (sometimes also referred to as projection methods) are based on variational formulations of the equivalent form of LLG

\[ \alpha \partial_t m + m \times \partial_t m = \gamma_0 (H_{\text{eff}}(m) + T(m)) - \gamma_0 [(H_{\text{eff}}(m) + T(m)) \cdot m] m. \]
The orthogonality \( \mathbf{m} \cdot \partial_t \mathbf{m} = 0 \), which characterizes any solution of (3a), is enforced at the discrete level by considering the discrete tangent space

\[ \mathcal{K}_h(\mathbf{m}^n_h) := \{ \varphi_h \in (V_h)^3 : \varphi_h(z) \cdot \mathbf{m}_h^n(z) = 0 \text{ for all nodes } z \text{ of } T_h \} \subset (V_h)^3. \]

For each time-step, one has to solve a constrained linear system to compute \( \mathbf{v}_h^n \approx \partial_t \mathbf{m}(n\Delta t) \) in \( \mathcal{K}_h(\mathbf{m}^n_h) \). With \( \mathbf{m}_h^n \) and \( \mathbf{v}_h^n \) at hand, one then computes \( \mathbf{m}^{n+1}_h \in (V_h)^3 \) by

\[
\mathbf{m}_h^{n+1}(z) := \frac{\mathbf{m}_h^n(z) + \Delta t \mathbf{v}_h^n(z)}{|\mathbf{m}_h^n(z) + \Delta t \mathbf{v}_h^n(z)|} \quad \text{for all nodes } z \text{ of } T_h.
\]

The original tangent plane scheme from [12] is formally first-order in time and was analyzed for the energy being only the exchange contribution. The scheme was extended to general lower-order effective field contributions [14, 33], DMI [52], and the coupling with other partial differential equations, e.g., various forms of Maxwell’s equations [60, 19, 59, 42], spin diffusion [10], and magnetostriction [20]. A projection-free version of the method was analyzed in [10, 67]. In a variant from [13, 38], the formal convergence order in time has been increased from one to two. Effective solution strategies and preconditioning for the resulting constrained linear system have recently been proposed in [55].

### 3.2. Midpoint scheme.

The midpoint scheme is based on a variational formulation of the Gilbert form (3a) of LLG. It consists of two fundamental ingredients: the implicit midpoint rule in time and the mass-lumped \( L^2 \)-product in space, defined as

\[
\langle \varphi, \psi \rangle_h := \int_{\Omega} I_h(\varphi \cdot \psi) dx \approx \langle \varphi, \psi \rangle_{L^2(\Omega)} \quad \text{for all } \varphi, \psi : \Omega \to \mathbb{R}^3 \text{ continuous. (8)}
\]

Here, \( I_h \) is the standard nodal interpolant associated with \( T_h \). The resulting scheme is second-order in time, inherently preserves the unit-length constraint and the energy of the solutions, but requires the solution of one nonlinear system for \( \mathbf{m}_h^{n+1/2} := (\mathbf{m}_h^{n+1} + \mathbf{m}_h^n)/2 \in (V_h)^3 \) per time-step. The midpoint scheme was proposed and analyzed in [25]. The scheme was extended to lower-order terms [65], the coupling with the Maxwell equation [17], and a variant of LLG in heat-assisted magnetic recording [21, 22]. The resulting nonlinear system is usually solved with constraint-preserving fixed-point iterations, which, however, spoil the unconditional convergence.

### 3.3. Magnetostatic Maxwell equations.

For the computation of stray field and Oersted field, i.e., for the numerical solution of the magnetostatic Maxwell equations (2) and (4), we follow a hybrid FEM-BEM approach, which combines FEM with the boundary element method (BEM); see [45, 51]. The method uses the superposition principle and computes the magnetic scalar potential \( u \) such that \( \mathbf{H}_s = -\nabla u \) and the Oersted field \( \mathbf{H}_c \) by splitting the problem into two parts, where BEM techniques for the evaluation of the double-layer potential are employed; see, e.g., [65 Chapter 4.1] for details.

### 4. Implementation

Our Python module Commics is based on the Netgen/NGSolve [6] (NGS) FEM software and provides a tool to perform micromagnetic simulations with the algorithms described in Section 3 for a variety of energy contributions and dissipative effects. It is
purely Python-based, provides extensive simulation data for reproducibility and post-
processing, and automatically takes care of, e.g., the definition and the assembly of 
underlying (bi)linear forms as well as the numerical solution of the arising linear 
and nonlinear systems. Although not needed for the use of Commics, this section advertises 
some of the core features of NGS, as well as the coupling of NGS with BEM++.

4.1. Basic features. Geometry handling, mesh-generation, FEM spaces, and assembly 
routines are intentionally hidden from the user of Commics and are internally covered in 
the NGS framework. For instance, given a NGS object \texttt{mesh}, representing a tetrahedral 
mesh of the domain \(\Omega\), the discrete vector-valued product space \((V_h)^3\) is simply generated by

\[
V_h^3 = \texttt{VectorH1}(\texttt{mesh}, \text{order}=1)
\]

where the syntax \texttt{VectorH1} indicates that \(V_h^3\) is a proper subspace of the vector-valued 
Sobolev space \((H^1(\Omega))^3\).

4.2. Symbolic (bi)linear forms. Although the use of Commics requires minimal knowl-
dge of NGS, we shortly describe one feature of the library, namely the definition of 
(bi)linear forms: NGS allows the symbolic definition of (time-dependent) (bi)linear forms. 
For instance, given the space \(V_h^3\) defined in Section 4.1, the LLG-specific cross-product 
bilinear form

\[
\langle m_h^n \times \psi_h, \varphi_h \rangle_{L^2(\Omega)} \quad \text{for all } \varphi_h, \psi_h \in (V_h)^3
\]

is symbolically defined on the Python level by the following code snippet:

```python
# grid-, trial- and testfunction
psi = Vh3.TrialFunction()
phi = Vh3.TestFunction()
m = GridFunction(Vh3)
# bilinear form
LHS = BilinearForm(Vh3)
ir = IntegrationRule(TET, order=3)
LHS += SymbolicBFI(Cross(m, psi) * phi, intrule=ir)
```

In the last line, \texttt{SymbolicBFI} with the test function \texttt{phi} and the trial function \texttt{psi} realizes 
the bilinear form \([9]\) and adds it to the left-hand side \(LHS\) for further handling. Note 
that via the \texttt{intrule} option, the order of the quadrature is explicitly set to 3, since the 
realization of the bilinear form corresponds to the exact integration of piecewise cubic 
polynomials on tetrahedra, hence also the parameter \texttt{TET}. For the midpoint scheme, the 
corresponding bilinear form employs the mass-lumped \(L^2\)-product from \([8]\), i.e.,

\[
\langle m_h^n \times \psi_h, \varphi_h \rangle_h \quad \text{for all } \varphi_h, \psi_h \in (V_h)^3
\]

This bilinear form can be defined in the same way as above specifying the corresponding 
quadrature rule on the reference tetrahedron \(\text{conv}\{0, e_1, e_2, e_3\} \cup\)

```python
ir = IntegrationRule(points=[(0.0, 0.0), (1.0, 0.0), (0.1, 0.0), (0.0, 1.0)], 
weights=[1/24, 1/24, 1/24, 1/24])
```
4.3. Coupling with BEM++. For the approximate computation of the magneto-
static fields with the hybrid FEM-BEM approach from Section 3.3, the evaluation of
the double-layer potential is required. To that end, we incorporate into Commics the corre-
sponding functionality of the BEM software BEM++ [75], including matrix compression
techniques [49]. The coupling of NGS and BEM++ is done on the Python level with the
ngbem module [2]; see Appendix A.

5. Using Commics: Standard problems and numerical experiments

In this section, we present some numerical experiments performed with Commics. For each proposed example, we also include the executable Python script to run the simulation.

5.1. Using Commics. To run a micromagnetic simulation with Commics, the user has
to define an object of class commics.Integrator and call its Integrate method.

The essential inputs to initialize such an object are the geometry as a commics.Geometry
object defining the ferromagnetic domain and the meshing strategy (see Section 5.2), as
well as an object of type commics.Parameters specifying, among other things, material
parameters, an applied field or current, the initial magnetization state, and the time
discretization strategy. Moreover, the desired time integration scheme has to be chosen.
The algorithms for the numerical integration of LLG implemented in Commics described
in Section 3 can be selected in the following way:

- The first-order tangent plane scheme from [12] with explicit integration of the
  lower-order terms in time [14, 33] is provided as TPS1.
- The projection-free tangent plane scheme (with explicit integration of the lower-
  order terms in time) from [10, 67] is provided as TPS1PF.
- The second-order tangent plane scheme from [13] and its improved version from [38]
  are available as TPS2 and TPS2AB, respectively.
- The midpoint scheme from [25] and its improved version from [65] are provided
  as MPS and MPSAB, respectively.

Various example scripts are provided subsequently in this section. Further details can be
accessed with the Python built-in help() function available for Commics classes.

5.2. Geometry specification and meshing. Commics provides several ways to specify
a geometry and to generate a corresponding mesh:

- For complex geometries, the submodule netgen.csg of NGS provides a rich num-
  ber of possibilities to define geometries; see [6].
- For geometries often encountered in micromagnetics, e.g., cuboids and disks, one
can simply provide the dimensions of the sample, a scale factor, and the desired
maximum mesh size; see, e.g., the code snippets in Section 5.3.1 and Section 5.6.
Then, netgen.csg will automatically be used with appropriate settings.
- Existing NGS meshes/geometries (stored as .vol-files) can simply be loaded; see,
e.g., the code snippet in Section 5.3.2.

Meshes automatically generated by NGS are unstructured and obtained by the advancing
front method; see [69] for details. However, due to mesh quality and shape optimizations,
these meshes do not necessarily satisfy the prescribed maximum mesh size. Commics
provides two possibilities to bypass this drawback: For cuboidal geometries, setting the Commics option structuredMesh=True allows for structured meshes. First, the sample is uniformly split into cuboidal cells of prescribed size. Then, each cell is split into six tetrahedra in such a way that any tetrahedron has three mutually perpendicular edges; see Figure 1. This strategy is for example used in Section 5.4. For general geometries, Commics provides the means to repeatedly generate a new mesh (by prescribing a smaller and smaller mesh size each time) in NGS, until the initial mesh size specification is satisfied. This strategy can be enabled by setting forceNetgenMeshSize=True as done in Section 5.6.

5.3. \(\mu\)MAG standard problem \#4. To describe the key aspects of a Commics script, we consider the \(\mu\)MAG standard problem \#4 [7].

The objective is the simulation of the magnetization dynamics in a thin permalloy film of dimensions \(500\ \text{nm} \times 125\ \text{nm} \times 3\ \text{nm}\) under the influence of a constant applied external field. We split the experiment into two parts: In the first stage, we obtain the so-called equilibrium S-state, which is saved to serve as the initial configuration for the second stage, where the switching dynamics is simulated.

5.3.1. Obtaining the S-state. We consider a structured tetrahedral mesh of the given cuboid into cells of size \(h_x \times h_y \times h_z\), which are decomposed into tetrahedra as depicted in Figure 1. The dimension of the cells is chosen as \(h_x = h_y = 125/69\ \text{nm}\) and \(h_z = 1.5\ \text{nm}\). This corresponds to 228,528 elements with diameter \(h = 2.97\ \text{nm}\), 58,170 vertices, as well as 78,936 surface elements. The material parameters of permalloy read \(M_s = 8 \cdot 10^5\ \text{A/m},\ A = 1.3 \cdot 10^{-11}\ \text{J/m},\ D = 0.1\ \text{m}^2,\ \text{and}\ K = 0.1\ \text{m}^3\). To speed up the process, we deliberately choose the large value \(\alpha = 1\) for the Gilbert damping parameter. For the simulation, we use a constant time-step size \(\Delta t = 0.1\ \text{ps}\).

The problem description suggests to obtain the S-state by applying a slowly reducing external field pointing in the \((1, 1, 1)\)-direction. We start with a uniform initial state \(\mathbf{m}_h^0 \equiv (1, 0, 0)\) and let the magnitude \(|\mathbf{H}_{\text{ext}}|\) of the external field decrease linearly from \(30/\mu_0\) to 0 mT over a period of 1 ns. In Commics scripts, non-constant fields can be described using time- and space-dependent Python \texttt{lambda} -functions. Further, we relax the system for 1 ns without applying any external field and store the obtained S-state as \texttt{sp4sState.vtk} for later use.
from commics import *
# specify geometry and parameters
h_xy, h_z = 125/69, 1.5
geo = Geometry(geometry=Cuboid(500, 125, 3), meshSize=(h_xy, h_xy, h_z), 
structuredMesh=True, scaling=1e-9)
par = Parameters(A=1.3e-11, Ms=8e+5, K=0, D=0, gamma0=2.211e+5, 
alpha=1.0, m0=(1.0, 0.0, 0.0), 
T_start=0, T_end=1e-9, timeStepSize=0.1e-12)
# define time dependent applied field via lambda function
from math import sqrt
field = lambda t,x,y,z : (par.T_end - t) / (par.T_end - par.T_start) 
* 30.0e-3 / par.mu0 / sqrt(3.0)
par.H_ext = (field, field, field)
# define integrator and run simulation from T_start to T_end
sp4 = Integrator(scheme=TPS2AB, geometry=geo, parameters=par)
sp4.Integrate()
# Relax for another nanosecond and save the mesh and the result
sp4.Integrate(duration=1e-9, relax=True)
sp4.SaveMesh("sp4 mesh")
sp4.SaveMagnetization("sp4 sState")

5.3.2. Switching. We assume that the folder data contains the two files sp4sState.vtk 
and sp4mesh.vol saved from the simulation described in Section 5.3.1. As stated
in the problem description, we choose α = 0.02 and set the external field to $H_{ext} = 
(-24.6, 4.3, 0)/\mu_0$ mT. Then, we run the simulation for 3 ns.

from commics import *
# specify geometry and parameters
geo = Geometry(geometry="data/sp4mesh.vol", scaling=1e-9)
par = Parameters(A=1.3e-11, Ms=8e+5, K=0, D=0, gamma0=2.211e+5, 
alpha=0.02, timeStepSize=0.1e-12, m0="data/sp4sState.vtk")
par.H_ext = (-24.6e-3/par.mu0, 4.3e-3/par.mu0, 0.0)
# define integrator and run simulation
sp4 = Integrator(scheme=TPS2AB, geometry=geo, parameters=par)
sp4.Integrate(duration=3e-9)

For comparison, the desired output of this benchmark problem is the evolution of the $x$-, 
y- and $z$-component of the spatially averaged magnetization. Figure 2 shows, that our
results match those computed by the finite difference code OOMMF [39] available on the
µMAG homepage [7]. Further, Figure 3 visualizes the magnetization at the time when
the $x$-component of the spatially averaged magnetization first crosses zero.

5.3.3. Meshing strategy and integrator. In Section 5.3.1, we considered a structured mesh 
and the TPS2AB integrator from [38]. To compare the results, we additionally repeat
the simulation on an unstructured mesh using the midpoint scheme: To simulate the 
dynamics on an unstructured mesh generated by NGS, we replace the definition of the 
geometry in Section 5.3.1 by

g = Geometry(geometry=Cuboid(500, 125, 3), meshSize=3, scaling=1e-9)

This results in an unstructured mesh containing 48792 elements, 16682 vertices and 
33360 surface elements, which corresponds to an actual mesh size of 5.19 nm. To repeat
the simulation using the midpoint scheme from [25], we replace the definition of the integrator in Section 5.3.1 and Section 5.3.2 by

\[
\text{sp4 = Integrator(scheme=MPS, geometry=geo, parameters=par)}
\]

Although the mesh size is close to the exchange length of the material (5.69 nm), qualitatively the results match those computed by OOMMF well; see Figure 2.

5.4. \(\mu\text{MAG standard problem } \#5\). The spintronic extensions of LLG from [79, 77] are the subject of the \(\mu\text{MAG standard problem } \#5\) [7]. The sample under consideration is a permalloy film with dimensions 100 nm \(\times\) 100 nm \(\times\) 10 nm aligned with the \(x\), \(y\), and \(z\) axes of a Cartesian coordinate system, with origin at the center of the film. We consider the same material parameters as in Section 5.3 and \(\alpha = 0.1\). The initial state is obtained by solving (3) with \(T \equiv 0\) and \(m^0(x, y, z) = (-y, x, R)/\sqrt{x^2 + y^2 + R^2}\) with \(R = 10\) nm.
and maximal damping $\alpha = 1$ for 1 ns, which is a sufficiently long time for the system to reach equilibrium. Given $P J_e = (1 \cdot 10^{12}, 0, 0) \text{ A/m}^2$ and $\xi = 0.05$, we set $T$ according to the expression in (5a). Then, we solve $\dot{m}$ with the relaxed magnetization configuration as initial condition for 8 ns, which turns out to be a sufficiently long time to reach the new equilibrium; see Figure 4 and Figure 5. We consider a structured tetrahedral mesh of the given cuboid into cells of size $h_x \times h_y \times h_z$, which are decomposed into tetrahedra as depicted in Figure 1. The dimension of the cells is chosen as $h_x = h_y = h_z = 5/3$ nm. This corresponds to 129600 elements with diameter $h = 2.89$ nm, 26047 vertices, as well as 17280 surface elements.

```python
from cmmics import *
# specify geometry and parameters
h = 5 / 3
geo = Geometry(geometry=Cuboid((-50, -50, -5), (50, 50, 5)), meshSize=(h, h, h), \  structuredMesh=True, scaling=1.0e-09)
par = Parameters(A=1.3e-11, Ms=8.0e+05, K=0.0, gamma0=2.211e+05, alpha=1.0, \  spintronicsCoupling="zhangLi", g=2.0, P=1.0, \  Je=(1.0e+12, 0.0, 0.0), xi=0.05, \  timeStepSize=0.1e-12, T_start=-1.0e-09)
# space dependent initial condition (scaled domain); normalized automatically
from ngsolve import x, y
par.m0 = (-y, x, 10.0)
# define integrator and relax configuration
sp5 = Integrator(scheme=TPS2AB, geometry=geo, parameters=par)
sp5.Integrate(duration=1.0e-09, relax=True)
# set damping alpha and run simulation with specified current
sp5.SetParameter_alpha(0.1)
sp5.Integrate(duration=8.0e-09)
```

5.5. Standard problem for ferromagnetic resonance simulations. Ferromagnetic resonance (FMR) is a well-established experimental technique for the study of ferromagnetic materials. A typical application of FMR consists in perturbing the magnetization of a system from its equilibrium by a sufficiently weak excitation and studying the induced magnetization dynamics, which is basically made of damped oscillations around the initial equilibrium. The resulting resonance frequencies and the eigenmodes of the system give some insights on the magnetic properties of the material and are used, e.g., for the experimental measurement of model parameters like the Gilbert damping constant or the saturation magnetization [41, 62].

In this section, we compute with Commics a problem for FMR simulations recently proposed in [16]. The computational domain is a cuboid of permalloy with dimensions 120 nm $\times$ 120 nm $\times$ 10 nm. The material parameters are the same as in Section 5.3. During the first stage, we set $\alpha = 1$ and consider a constant applied external field of magnitude $|H_{ext}| = 8 \cdot 10^4$ A/m pointing in the direction $(1, 0.715, 0)$. We initialize the system with a uniform ferromagnetic state $m^0_h \equiv (1, 0.0)$ and let the system evolve for 5 ns. The resulting state is then used as initial condition for the second stage, in which we set $\alpha = 0.008$, change the direction of the applied external field to $(1, 0.7, 0)$ but keep $|H_{ext}| = 8 \cdot 10^4$ A/m, and let the system evolve to the new equilibrium for 20 ns. We consider a structured tetrahedral mesh of the given cuboid into cells of size $h_x \times h_y \times h_z$, which are decomposed into tetrahedra as depicted in Figure 1. The dimension of the cells is chosen as $h_x = h_y = h_z = 2$ nm. This corresponds to 108000 elements, 22326 vertices,
Figure 4. µMAG standard problem #5: Magnetization in the \(xy\)-plane viewed from the top at different times. Starting from the relaxed configuration at \(t = 0.0\) ns, the vortex (red) follows a spiral-like motion. After \(t = 8.0\) ns no further movements are observed.

Figure 5. µMAG standard problem #5: Evolution of the spatially averaged \(x\)- and \(y\)-component of \(m\). as well as 16,800 surface elements, and yields a mesh size of \(h = 3.46\) nm. We compare our results obtained with Commics to those presented in [16]. There, the authors use the finite difference code OOMMF [39] and investigate the evolution of the \(y\)-component of the spatially averaged magnetization, as well as its power spectrum \(S_y\). The power spectrum is obtained by a discrete Fourier transform as described in [16, Section 2.3.1]. Our results match well with those of [16]; see Figure 6 and Figure 7.
from commics import *
# specify geometry and parameters
geo = Geometry( geometry=Cuboid(120, 120, 10), meshSize=(2,2,2), \structuredMesh=True, scaling=1.0e-09)
par = Parameters(A=1.3e-11, Ms=8.0e+05, K=0.0, gamma0=2.210173e+05, \alpha=1.0, timeStepSize=0.1e-12, T_start=-5.0e-9, \m0=(0, 0, 1))
# define integrator, choose number of threads, specify folder for results
fmr = Integrator(scheme=TPS2AB, geometry=geo, parameters=par, numthreads=8)
fmr.SetResultsFolder(“FMR_Result”)
# obtain initial condition
amplitude = 80.0e+03
e = (1.0, 0.715, 0.0)
e_length = (sum(e[j]**2 for j in range(3)))**0.5
H_ext = (amplitude*e[0]/e_length, amplitude*e[1]/e_length, 0.0)
fmr.SetParameter_H_ext(H_ext)
fmr.Integrate(duration=5.0e-09)
# change direction of applied field and run simulation
fmr.SetParameter_alpha(0.008)
e = (1.0, 0.7, 0.0)
e_length = (sum(e[j]**2 for j in range(3)))**0.5
H_ext = (amplitude*e[0]/e_length, amplitude*e[1]/e_length, 0.0)
fmr.SetParameter_H_ext(H_ext)
fmr.Integrate(duration=20.0e-09)

![Figure 6. Ferromagnetic resonance simulation from Section 5.5: Time evolution of ⟨m_y⟩ obtained with Commics compared to the results computed with OOMMF.](image)

5.6. Current-induced dynamics of skyrmions in nanodisks. With this experiment, we aim to show how Commics can be used to numerically investigate the stability and the induced dynamics of magnetic skyrmions in helimagnetic materials in response to spin-polarized currents.

We consider a magnetic nanodisk of diameter 120 nm (x_1x_2-plane) and thickness d = 10 nm (x_3-direction). We use the material parameters of iron-germanium (FeGe), i.e., M_s = 3.84 · 10^5 A/m, A = 8.78 · 10^{-12} J/m, D = 1.58 · 10^{-3} J/m^2, and K = 0 J/m^3; see, e.g., [26]. The initial condition for our experiment is obtained by relaxing a uniform out-of-plane ferromagnetic state m^0 ≡ (0, 0, 1) for 2 ns. For the relaxation process, we
choose the large value \( \alpha = 1 \) for the Gilbert damping constant. The resulting relaxed state is the skyrmion depicted in Figure 8.

Starting from this configuration, we apply a perpendicular spin-polarized current pulse \( \mathbf{J}_e(t) = (0, 0, J(t)) \) of maximum intensity \( J_{\text{max}} \) for 150 ps; see Figure 9. To model the resulting spin transfer torque, we include \( \mathbf{T} \) from (5). Then, we turn off the current density and let the system evolve for 20 ns. In order to capture all possible excitation modes during the application of the pulse and the subsequent relaxation process, we set the value of the Gilbert damping constant to \( \alpha = 0.002 \), which is considerably smaller than the experimental value of \( \alpha \approx 0.28 \) measured for FeGe; see [20].

In Figure 9, we plot the time evolution of the first component of the spatially averaged magnetization of the sample after a current pulse with \( J_{\text{max}} = 1 \cdot 10^{12} \text{ A/m}^2 \), \( \mathbf{p} = (0, 1, 0) \), and \( P = 0.4 \). The induced dynamics is a damped precession of the skyrmion around the center of the sample; see Figure 8. We consider an unstructured tetrahedral mesh of the nanodisk generated by NGS. For a desired mesh size of 7.5 nm, the automatically generated mesh consists of 35390 elements with maximum diameter \( h = 6.1 \text{ nm} \), 8436 vertices, as well as 9586 surface elements.
**Figure 9.** Simulation of the skyrmion dynamics from Section 5.6. Structure of the applied current pulse (left). Time evolution of the $x$-component of the spatially averaged magnetization (right).

```python
from commics import *

# specify geometry and parameters
geo = Geometry( geometry=Disk(120.0, 10.0), meshSize=7.5, \ forceNetgenMeshSize=True, scaling=1.0e-09)
par = Parameters(A=8.78e-12, Ms=3.84e+05, D=1.58e-03, dmCoupling="bulk", \ spintronicsCoupling="slonczewski", d=10.0e-09, P=0.4, \ p=(0.0, 1.0, 0.0), \ alpha=1.0, timeStepSize=1e-13, theta=1.0, m0=(0, 0, 1))

# define integrator, choose number of threads
helimag = Integrator(scheme=TPS1, geometry=geo, parameters=par, numthreads=8)

# choose to save magnetization as .vtk-file every X seconds, run simulation
helimag.RecordMagnetization(every=10.0e-12)

# prepare JeMax, alpha, and points in time
helimag.SetParameter_alpha(0.002)
JeMax = 1.0e+12
T0, T1, T2, T3 = 0, 40.0e-12, 110.0e-12, 150.0e-12

# set increasing current, run simulation
helimag.SetParameter_Je(lambda t,x,y,z : (t-T0)/(T1-T0)*JeMax)

# set constant current, run simulation
helimag.Integrate(duration=T1-T0)

# set decreasing current, run simulation
helimag.SetParameter_Je(JeMax)

# final relaxation
helimag.Integrate(duration=20.0e-9, relax=True)
```
In this section we show how NGS, BEM++, and the ngbem module can be used to perform stray field computations following the approach proposed in [45]:

- **NGS** is used for mesh generation and the FEM problems.
- **BEM++** provides boundary integral operators and an interface for solving the BEM problem.
- **ngbem** provides the means to extract a boundary element mesh from the volume mesh together with a mapping between the corresponding degrees of freedom.

To test the procedure we consider the uniformly magnetized unit ball. Then, the stray field is given by \( H_s(M) \equiv -M/3 \) in \( \Omega \). The complete code is provided in the following Python script.

```python
# Demonstration of the Fredkin-Koehler approach for stray field computations
# using ngsolve, ngbem, bempp
import ngsolve, ngbem, bempp.api, commics, numpy
# mesh, femSpaces, GridFunctions
geo = commics.Geometry(commics.UnitBall(), meshSize=0.3)
geo.GetReady()
mesh = geo.GetMesh()
Vh3 = ngsolve.VectorH1(mesh, order=1)
Vh = ngsolve.H1(mesh, order=1)
VhD = ngsolve.H1(mesh, order=1, dirichlet="bc_dirichlet")
m = ngsolve.GridFunction(Vh3)
m.components[0].vec.FV().NumPy()[:] = 1.0
u1 = ngsolve.GridFunction(Vh)
u2 = ngsolve.GridFunction(VhD)
hs = -u1.Deriv() - u2.Deriv()
# Neumann FEM problem
a1 = ngsolve.BilinearForm(Vh)
a1 += ngsolve.Laplace(1.0)
c1 = ngsolve.Preconditioner(a1, "local")
a1.Assemble()
solverU1 = ngsolve.CGSolver(mat=a1.mat, pre=c1.mat)
phi1 = Vh.TestFunction()
f1 = ngsolve.LinearForm(Vh)
f1 += ngsolve.SymbolicLFI(phi1.Deriv() * m)
measOmega = ngsolve.Integrate(ngsolve.CoefficientFunction(1.0), mesh)
# BEM problem
bemSpace, op_NgToBem = ngbem.H1_trace(Vh)
op_NgToBem.eliminate_zeros()
bemToNgIdx = numpy.full(op_NgToBem.shape[0], fill_value = Vh.ndof, dtype=int)
bemToNgIdx[op_NgToBem.row] = op_NgToBem.col
bem_K = bempp.api.operators.boundary.laplace.double_layer(bemSpace, bemSpace, \ bemSpace)
bem_I = bempp.api.operators.boundary.sparse.identity(bemSpace, bemSpace, \ bemSpace)
bemRhsOp = bem_K - 0.5*bem_I
bemLhs = bem_I
# Dirichlet FEM problem
a2 = ngsolve.BilinearForm(VhD)
a2 += ngsolve.Laplace(1.0)
c2 = ngsolve.Preconditioner(a2, "bddc")
a2.Assemble()
f2 = ngsolve.LinearForm(VhD)
```

16
\begin{verbatim}
f2.Assemble()
bvp2 = ngsolve.BVP(bf=a2, lf=f2, gf=u2, pre=c2)
# solve Neumann FEM problem
f1.Assemble()
u1.vec.data = solverU1 * f1.vec
u1.vec.FV()[:,i] = - ngsolve.Integrate(u1, mesh) / measOmega
# solve BEM problem
bem_u1 = bempp.api.GridFunction(bemSpace,
    coefficients=u1.vec.FV().NumPy()[bemToNgIdx])
bem_rhs = bemRhsOp * bem_u1
bem_g = bempp.api.linalg.iterative_solvers.gmres(bemLhs, bem_rhs)[0]
# solve Dirichlet FEM problem
u2.vec.FV()[bemToNgIdx] = bem_g.coefficients
bvp2.Do()
# check results
TOL = 0.01
HS = ngsolve.GridFunction(Vh3)
for d in range(3): HS.components[d].Set(hs[d])
passed = all(abs(-1/3 - HS.components[0].vec.FV().NumPy()) < TOL) \
    and all(abs(HS.components[1].vec.FV().NumPy()) < TOL) \
    and all(abs(HS.components[2].vec.FV().NumPy()) < TOL)
print("PASSED STRAYFIELD TEST: ", passed)
\end{verbatim}

REFERENCES


\[ \text{M. Beg, R. A. Pepper, and H. Fangohr. User interfaces for computational science: A domain specific language for OOMMF embedded in Python. } \text{AIP Adv.}, 7(5):056025, 2017. \]


