scientific reports

Check for updates

OPEN

magnum.np: a PyTorch based GPU enhanced finite difference micromagnetic simulation framework for high level development and inverse design

Florian Bruckner✉, Sabri Koraltan, Claas Abert & Dieter Suess

magnum.np is a micromagnetic finite-difference library completely based on the tensor library PyTorch. The use of such a high level library leads to a highly maintainable and extensible code base which is the ideal candidate for the investigation of novel algorithms and modeling approaches. On the other hand magnum.np benefits from the device abstraction and optimizations of PyTorch enabling the efficient execution of micromagnetic simulations on a number of computational platforms including graphics processing units and potentially Tensor processing unit systems. We demonstrate a competitive performance to state-of-the-art micromagnetic codes such as mumax3 and show how our code enables the rapid implementation of new functionality. Furthermore, handling inverse problems becomes possible by using PyTorch's autograd feature.

Micromagnetic simulations are widely used in a range of applications, from magnetic storage technologies and the design of hard and soft magnetic materials, to the modern fields of magnonics, spintronics, or even neuromorphic computing. A finite difference approximation has been proven useful for many applications due to its simplicity and its high performance, compared with the more flexible finite element approach.

Currently, there are already many open-source finite difference codes available, like OOMMF[1], mumax3[2], magnum.af[3], magnum.fd[4], fidimag[5], to mention just a few. However, for the development of new algorithms or for bleeding edge simulations one often needs to modify or extend the provided tools. For example post-processing of the created data often requires the setup of a seperate tool-chain. magnum.np provides a very flexible interface which allows the combination of many of these tasks into a single framework. It should bridge the gap between development codes, which are used for the testing of new methods, and production codes which are highly optimized for one specific task.

Complex algorithms can be easily built on top of the available core functions. Possible examples include an eigenmode solver for the calculation of small magnetization fluctuations, the calculation of the dispersion relation of magnonic devices, or the string-method for the calculation of energy barriers between different energy minima[6–8].

Due to the use of PyTorch's autograd method magnum.np is also well suited for solving inverse design problems. Inverse design refers to a design approach where the desired properties and functionalities of a system are specified first, and then the optimal structure or materials are determined to achieve those properties. It involves working backwards from the desired output to determine the necessary input parameters.

Recently, some inverse micromagnetic problems have been reported[9–11], where the magnetic systems have been optimized for a specific task. Providing a gradient of the objective function with regard to the design variables allow to use very efficient gradient-based optimization methods. Using PyTorch's autograd features, it is easily possible to define the design variables as differentiable and after the micromagnetic simulation (forward problem) has been performed the corresponding gradient can be computed using reverse-mode auto-differentiation.

Magnum.np is open-source under the GPL3 licence and can be found at https://gitlab.com/magnum.np/magnum.np. Different demo scripts are part of the source code and can be tested online using Google Colab[12], without the need for local installations or specialized hardware like GPUs. A list of demos can be found on the project gitlab page https://gitlab.com/magnum.np/magnum.np#documented-demos.

Faculty of Physics, University of Vienna, Vienna, Austria. ✉email: florian.bruckner@univie.ac.at

Design

In contrast to many available micromagnetic codes magnum.np follows a high-level approach for easy readability, maintainance and development. The Python programming language combined with PyTorch offers a powerful environment, which allows to write high-level code, but still get competitive performance due to proper vectorization.

PyTorch[13] has been chosen as backend since it allows transparently switching between CPU and GPU without modification of the code. Also the use of single or double precision arithmethic can be switched easily (e.g. use `torch.set_default_dtype`). Furthermore, it offers a very flexible tensor interface, based on the the Numpy Array API. Directly using torch tensors for calculation avoids the need for custom vector classes and allows using pytorch functions without the need for any wrapping code.

As a nice benefit of using PyTorch, one can directly use inverse operations via the PyTorch's autograd feature[14]. Even the utilization of deep neural networks in combination with classical micromagnetics would become feasable[15].

One key philosophy of the magnum.np design is to utilize few well-known libraries in order to delegate work, but keep its own code clean and compact. On the other hand we try to keep the number of dependencies as small as possible, in order to improve maintainability. As an example `pyvista` is used for simple reading or writing VTK files, but also offers many additional capabilities (mesh formats, visualization, etc.).

Figure 1 summarizes the most important building blocks and features.

The `state` class contains the actual state of the simulation like time t, magnetization $\boldsymbol{m}$ or in case of an Oersted field the corresponding current density $\boldsymbol{j}$. It also contains the information about mesh and materials. The finite difference method is based on an equidistant rectangular mesh consisting of $n_x \times n_y \times n_z$ cells, with a grid spacing $(\Delta x, \Delta y, \Delta z)$ and an origin (x_0, y_0, z_0). Thus the index set (i, j, k) is sufficient to identify an individual cell center:

$$\boldsymbol{x}_{i,j,k} = \begin{pmatrix} x_i \\ x_j \\ x_k \end{pmatrix} = \begin{pmatrix} x_0 + i\ \Delta x \\ y_0 + j\ \Delta y \\ z_0 + k\ \Delta z \end{pmatrix} = \boldsymbol{x}_0 + \Delta \boldsymbol{x} \quad \text{with} \quad \begin{matrix} i = 0 \ldots n_x - 1 \\ j = 0 \ldots n_y - 1 \\ k = 0 \ldots n_z - 1 \end{matrix} \tag{1}$$

Internally, physical fields are stored as multi-dimensional PyTorch tensors, where one value is stored for each cell (e.g. scalar fields are stored as $(n_x, n_y, n_z, 1)$ tensors). Using Numpy Array API features like slicing or fancy indexing allows simple modification of the corresponding data. Furthermore, it allows the use of the same expression for constant and non-constant materials, which contains one material parameter for each cell of the mesh. This avoids additional storage in case of constant materials, without the need for independent code branches. By using overloading of the `__call__` operator, it is even possible to allow time dependent material parameters in a transparent way.

It is often very useful to select sub-regions within the mesh, e.g. for defining location dependent material parameters, or evaluate the magnetization only in a part of the geometry. We call these sub-regions "domains" and they are easily represented by boolean tensors, which can be created by low-level tensor operation or by using `SpatialCoordinate` - a list of tensors (x, y, z) which store the physical location of each cell. Using these coordinate tensors allows to specify domains by simple analytic expressions (e.g. $x^2 + y^2 < r^2$ for a circle with radius r). The same coordinate tensors can also be used to parametrize magnetic configurations like vortices or skyrmions (see e.g. Listing 1 with the corresponding magnetization visualized in Fig. 2).

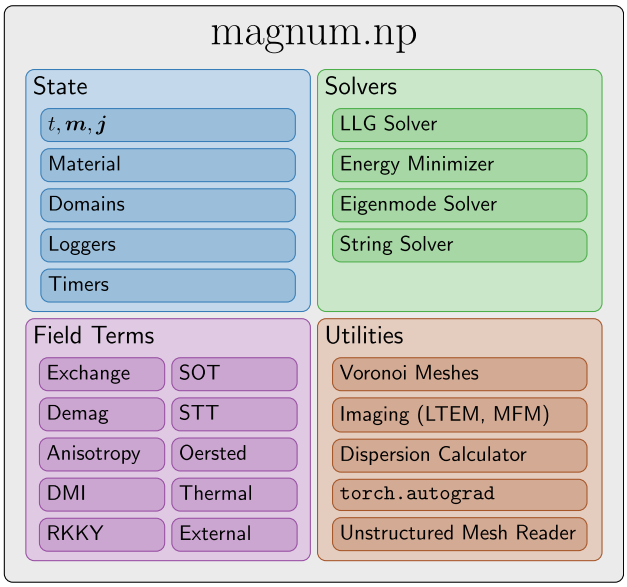

Figure 1. Overview of the high-level interface of magnum.np.

```
state = State(mesh)                    # state summarizes all informations
x, y, z = state.SpatialCoordinate()    # x, y, z are tensors which contain
disk = x**2 + y**2 < 20e-9**2          # boolean array (1:disk, 0:outside)

state.m  = torch.stack([ y, -x, 1e-9*z], dim = -1)   # parametrize vortex
state.m[~disk] = 0.                                  # set m=0 outside disk
state.m.normalize()                                  # normalize m
```

Listing 1:. Parametrization of a vortex configuration within a disk with radiusr = 20nm using `SpatialCoordinate`

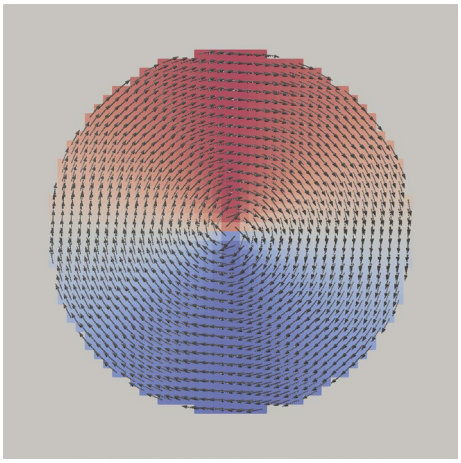

Figure 2. Resulting magnetization created using the parametrization in Listing 1. The color visualizes the x-component of the magnetization. The gray color outside of the disk shows that the magnetization is zero, outside of the magnetic domain.

The actual state can be stored by means of loggers. The `ScalarLogger` is able to log arbitrary scalar functions depending on the current `state` (e.g. average magnetization, field at a certain point, GMR signal, ...). The `FieldLogger` stores arbitrary field data using VTK.

Due to the very flexible interface it is also intendend to add utility function for various application cases to the magnum.np library. In many cases pre- and post-processing is already done in some high-level python scripts, which makes it possible to directly reuse those codes in magnum.np at least on CPU. In many cases time-critical routines can be easily translated into PyTorch code, which then also runs on the GPU, due to the common Numpy Array API. Examples of such utility functions which are already included within magnum.np are Voronoi mesh generators, several imaging tools for post-processing – like Lorentz Transmission Electron Microscopy(LTEM) or Magnetic Force Microscopy(MFM) – or the calculation of a dispersion relation from time-domain micromagnetic simulations.

Landau–Lifshitz–Gilbert equation

Dynamic micromagnetism is described by the Landau–Lifshitz–Gilbert equation

$$\dot{\boldsymbol{m}} = -\frac{\gamma}{1+\alpha^2}\left[\boldsymbol{m}\times\boldsymbol{h}^{\text{eff}} + \alpha\,\boldsymbol{m}\times\left(\boldsymbol{m}\times\boldsymbol{h}^{\text{eff}}\right)\right], \tag{2}$$

with the reduced magnetization $\boldsymbol{m}$, the reduced gyromagnetic ratio $\gamma = 2.21\times10^5\text{m/As}$, the dimensionless damping constant α, and the effective field $\boldsymbol{h}^{\text{eff}}$. The effective field may contain several contributions like the magnetostatic strayfield, or the quantummechanical exchange interaction (see "Field terms" section for the detailed descriptions of possible field terms).

For the solution of the Eq. (2) in time-domain most finite difference codes use explicit Runge–Kutta (RK) methods of different order. Magnum.np by default uses the Runge–Kutta–Fehlberg Method (RKF45)[16], which uses a 4th order approximation with a 5th order error control. Explicit RK methods, are very common, due to their simplicity and they are well suited for modern GPU computing. Additionally, third party solvers can be easily added, since many libraries already provide a proper python interface. For example wrappers for Scipy (CPU-only) and TorchDiffEq solvers are provided. Those solvers include more complicated solver methods like implicit BDF[17], which are well suited for stiff problems.

Often one is only iterested in the magnetic groundstate, in which case the LLG can be integrated with a high damping constant (and optionally without the precession term). Alternatively, the micromagnetic energy[18,19] can be minimized directly, which is often much more efficient. However, special care has to be taken since, standard conjugate gradient method may fail to produce correct results[20].

```python
class UniaxialAnisotropyField(LinearFieldTerm):
  @timedmethod
  def h(self, state):
    Ku = state.material["Ku"]
    Ku_axis = state.material["Ku_axis"]

    h = 2.*Ku*Ku_axis / (constants.mu_0 * state.material["Ms"]) \
      * torch.sum(Ku_axis * state.m, dim=3, keepdim=True)
    return torch.nan_to_num(h, posinf=0, neginf=0)
```

Listing 2:. Implementation of the first order uniaxial anisotropy field.

Field terms

The following section shows some implementation details of the effective field terms. Due to the flexible interface new field terms can easily be added even without modifying the core library.

All field terms which are linear in the magnetization $\boldsymbol{m}$ inherit from the `LinearFieldTerm` class, in order to allow a common calculation of the energy using

$$\mathcal{E}^{\text{lin}} = -\frac{1}{2}\mu_0 \int M_s\, \boldsymbol{m} \cdot \boldsymbol{h}^{\text{lin}}\, \mathrm{d}\boldsymbol{x}, \tag{3}$$

with the corresponding (continuous) field $\boldsymbol{h}^{\text{lin}}$, the saturation magnetization M_s, and the vacuum permeability μ_0.

In the following several field contributions will be described including a continuous formulation as well as the used discretization. For example the discretized version of the linear field energy can be written as

$$E^{\text{lin}} = -\frac{1}{2}\mu_0 V \sum_{\boldsymbol{i}} M_s\, \boldsymbol{m_i} \cdot \boldsymbol{h}_{\boldsymbol{i}}^{\text{lin}}, \tag{4}$$

with the cell volume $V = \Delta x\, \Delta y\, \Delta z$. $x_{\boldsymbol{i}}$ describes a discretized quantity x at the cell with index $\boldsymbol{i}$. Some indices $\boldsymbol{i}$ will be omitted for sake of better readability (e.g. for the material parameter M_s).

Anisotropy field. Spin orbit coupling gives rise to an anisotropy field which favors the alignment of the magnetization into certain axes. Depending on the crystal structure one or more of such easy axis may be observed. E.g. material with tetragonal or hexagonal structure show a uniaxial anisotropy which gives rise the the following interaction field

$$\boldsymbol{h}^{\text{u}}(\boldsymbol{x}) = \frac{2K_{\text{u1}}}{\mu_0 M_s}\, \boldsymbol{e}_{\text{u}}\, (\boldsymbol{e}_{\text{u}} \cdot \boldsymbol{m}) + \frac{4K_{\text{u2}}}{\mu_0 M_s}\, \boldsymbol{e}_{\text{u}}\, (\boldsymbol{e}_{\text{u}} \cdot \boldsymbol{m})^3, \tag{5}$$

where K_{u1} and K_{u2} are the first and second order uniaxial anisotropy constants, respectively, and $\boldsymbol{e}_{\text{u}}$ is the corresponding easy axis. Since the anisotropy is a local interaction, its discretization is straight forward and will be ommited. The corresponding source code is shown in Listing 2.

Since the uniaxial anistropy field is a linear field term, only the field needs to be implemented, whereas the energy is inherited from the `LinearFieldTerm`. Material parameters are accessed from `state.material` which returns the material for each cell at the time `state.t`. The actual field expression is very close to the mathematical formulation, which makes the code easy to ready and adapt for similar use cases.

For a cubic crystal structure the corresponding cubic anisotropy field is given by

$$\boldsymbol{h}^{\text{c}}(\boldsymbol{x}) = -\frac{2K_{\text{c1}}}{\mu_0 M_s} \begin{pmatrix} m_1\, m_2^2 + m_1\, m_3^2 \\ m_2\, m_3^2 + m_2\, m_1^2 \\ m_3\, m_1^2 + m_3\, m_2^2 \end{pmatrix} - \frac{2K_{\text{c2}}}{\mu_0 M_s} \begin{pmatrix} m_1\, m_2^2\, m_3^2 \\ m_1^2\, m_2\, m_3^2 \\ m_1^2\, m_2^2\, m_3 \end{pmatrix}, \tag{6}$$

where K_{u1} and K_{u2} are the corresponding first and second order cubic anisotropy constants. m_1, m_2 and m_3 are the magnetization components in three orthogonal principal axes.

Exchange field. The quantum mechanical exchange interaction favours the parallel alignment of neigboring spins. Variation of the micromagnetic energy gives rise the the following exchange field

$$\boldsymbol{h}^{\text{ex}}(\boldsymbol{x}) = \frac{2}{\mu_0 M_s} \nabla \cdot (A\, \nabla \boldsymbol{m}), \tag{7}$$

combined with a proper boundary condition[21] for the magnetization $\boldsymbol{m}$, which can be expressed as

$$B = 2A\, \frac{\partial \boldsymbol{m}}{\partial \boldsymbol{n}} \tag{8}$$

The boundary condition is important for the correct treatment of the outer system boundaries, but also for interface between different materials. In general the jump of B over an interface Γ needs to vanish ($[\![B]\!]_\Gamma = 0$).

In case of an outer boundary this leads to the well-known $\frac{\partial \boldsymbol{m}}{\partial \boldsymbol{n}} = 0$, if no further field contibutions (like e.g. DMI) are considered.

The discretized expression of the exchange field considering spacially varying material parameters[22] is finally given by

$$\boldsymbol{h}_{\boldsymbol{i}}^{\mathrm{ex}} = \frac{2}{\mu_0 M_{s,\boldsymbol{i}}} \sum_{k=\pm x,\pm y,\pm z} \frac{2}{\Delta_k^2} \frac{A_{\boldsymbol{i}+\boldsymbol{e}_k} A_{\boldsymbol{i}}}{A_{\boldsymbol{i}+\boldsymbol{e}_k} + A_{\boldsymbol{i}}} \left(\boldsymbol{m}_{\boldsymbol{i}+\boldsymbol{e}_k} - \boldsymbol{m}_{\boldsymbol{i}}\right) \tag{9}$$

where A is the exchange constant and Δ_k is the grid-spacing in direction k. The index $\boldsymbol{i} = (i, j, k)$ indicates the cell for which the field should be evaluated, whereas the index $\boldsymbol{i} \pm \boldsymbol{e}_k$ means the index of the next neighbor in the direction $\pm \boldsymbol{e}_k$. Note that the harmonic mean of the exchange constants occurs in front of each next-neighbor difference, which makes it vanish if a cell is located on the boundary. This is important to fulfill the correct boundary conditions $\frac{\partial \boldsymbol{m}}{\partial \boldsymbol{n}} = 0$. In case of a homogeneous exchange constant this term simplifies to the well known expression

$$\boldsymbol{h}_{\boldsymbol{i}}^{\mathrm{ex}} = \frac{2A}{\mu_0 M_{s,\boldsymbol{i}}} \sum_{\mathrm{k=x,y,z}} \frac{\boldsymbol{m}_{\boldsymbol{i}+\boldsymbol{e}_k} - 2\,\boldsymbol{m}_{\boldsymbol{i}} + \boldsymbol{m}_{\boldsymbol{i}-\boldsymbol{e}_k}}{\Delta_k^2} \tag{10}$$

DMI field. Due to the spin-orbit coupling some materials show an additional antisymmetric exchange interaction called Dzyaloshinskii–Moriya interaction[23,24]. A general DMI field can be written as

$$\boldsymbol{h}^{\mathrm{dmi}}(\boldsymbol{x}) = \frac{2D}{\mu_0 M_s} \sum_{k=x,y,z} \boldsymbol{e}_k^{\mathrm{dmi}} \times \frac{\partial \boldsymbol{m}}{\partial_k}, \tag{11}$$

with the DMI strength D and the DMI vectors $\boldsymbol{e}_k^{\mathrm{dmi}}$, which describe which components of the gradient of $\boldsymbol{m}$ contribute to which component of the corresponding field. It is assumed that $\boldsymbol{e}_{-k}^{\mathrm{dmi}} = -\boldsymbol{e}_k^{\mathrm{dmi}}$.

Different kinds of DMI can be simply implemented by specifying the corresponding DMI vectors. For example the continuous interface DMI field for interface normals in z direction and DMI strength D_i is given by

$$\begin{aligned} \boldsymbol{h}^{\mathrm{dmi,i}}(\boldsymbol{x}) &= -\frac{2D_i}{\mu_0 M_s} \left[\nabla(\boldsymbol{e}_z \cdot \boldsymbol{m}) - (\nabla \cdot \boldsymbol{m})\,\boldsymbol{e}_z\right] \\ &= \frac{2D_i}{\mu_0 M_s} \left[\boldsymbol{e}_y \times \frac{\partial \boldsymbol{m}}{\partial x} - \boldsymbol{e}_x \times \frac{\partial \boldsymbol{m}}{\partial y}\right], \end{aligned} \tag{12}$$

Thus, the corresponding DMI vectors for interface DMI result in $\boldsymbol{e}^{\mathrm{dmi}} = (\boldsymbol{e}_y, -\boldsymbol{e}_x, 0)$. See Table 1 for a summary of the most common DMI types.

Finally, Eq. (11) is discretized using central finite differences. For constant D_i this results in

$$\boldsymbol{h}_{\boldsymbol{i}}^{\mathrm{dmi}} = \frac{2}{\mu_0 M_{s,\boldsymbol{i}}} \sum_{k=\pm x,\pm y,\pm z} \tilde{D}_{\boldsymbol{i},k} \frac{\boldsymbol{e}_k^{\mathrm{dmi}} \times \boldsymbol{m}_{\boldsymbol{i}+\boldsymbol{e}_k}}{2\,\Delta_k}, \tag{13}$$

where $\tilde{D}_{\boldsymbol{i},k}$ is the effective DMI coupling strength between cell $\boldsymbol{i}$ and $\boldsymbol{i} + \boldsymbol{e}_k$. Similar to the case of the exchange field, the harmonic mean is used for the avarage coupling strengths:

$$\tilde{D}_{\boldsymbol{i},k} = \frac{2\,D_{\boldsymbol{i}}\,D_{\boldsymbol{i}+\boldsymbol{e}_k}}{D_{\boldsymbol{i}} + D_{\boldsymbol{i}+\boldsymbol{e}_k}} \tag{14}$$

Note, that if DMI interactions are in place $\frac{\partial \boldsymbol{m}}{\partial \boldsymbol{n}} = 0$ does no longer hold. Instead, inhomogeneous Neumann boundary conditions occur (see e.g. Eqs. 11–15 in[2]), which leads to a coupling of exchange and DMI interaction. The exchange field could no longer be calculated independent of the DMI interaction.

However, since the Neumann boundary conditions are only approximately fulfilled due to the finite difference approximation, magnum.np uses an alternative formulation of the discrete boundary conditions that simply ignores the non-existing values on the boundary, which is consistent with the effective coupling strengths in Eq. (14). Although, this approach seems less profound, it has been used in some well-known micromagnetic simulation packages, like *fidimag*[5] or *mumax3*(openBC)[2], and shows good agreements for many standard problems[25].

DMI type	Symmetry class	Formula	DMI vectors $\boldsymbol{e}^{\mathrm{dmi}}$
Interface	C_{nv}	$\boldsymbol{h}^{\mathrm{dmi,i}} = -\frac{2D_i}{\mu_0 M_s} \left[\nabla(\boldsymbol{e}_z \cdot \boldsymbol{m}) - (\nabla \cdot \boldsymbol{m})\,\boldsymbol{e}_z\right]$	$(\boldsymbol{e}_y, -\boldsymbol{e}_x, 0)$
Bulk	T or O	$\boldsymbol{h}^{\mathrm{dmi,b}} = -\frac{2D_b}{\mu_0 M_s} \nabla \times \boldsymbol{m}$	$(\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z)$
$D_{2\mathrm{d}}$	$D_{2\mathrm{d}}$		$(-\boldsymbol{e}_x, \boldsymbol{e}_y, 0)$

Table 1. Most common DMI types with the corresponding symmetry class and DMI vectors.

Demagnetization field. The dipole-dipole interaction gives rise to a long-range interaction. The integral formulation of the corresponding Maxwell equations can be represented as convolution of the magnetization with a proper demagnetization kernel $\boldsymbol{N}$

$$\boldsymbol{h}^{\text{dem}}(\boldsymbol{x}) = \int_{\Omega} \boldsymbol{N}(\boldsymbol{x} - \boldsymbol{x}')\, \boldsymbol{M}(\boldsymbol{x}')\ \mathrm{d}\boldsymbol{x}' \tag{15}$$

Discretization on equidistant grids results in a discrete convolution which can be efficiently solved by a Fourier method. The discrete convolution theorem combined with zero-padding of the magnetization allows to replace the convolution in real space, with a point-wise multiplication in Fourier space. The discrete version of Eq. (15) reads like

$$\boldsymbol{h}_{\boldsymbol{i}}^{\text{dem}} = \sum_{\boldsymbol{j}} \boldsymbol{N}_{\boldsymbol{i}-\boldsymbol{j}}\, \boldsymbol{M}_{\boldsymbol{j}}, \tag{16}$$

and is visualized in Fig. 3

The average interaction from one cell to another can be calculated analytically using Newell's formula[26]. More information about the implementation details can be found in[27], where the demagnetization field has been implemented using numpy.

As shown in Fig. 4 the Newell formula is prone to fluctuations if the distance of source and target cell is too large[28]. Thus, it is favourable to use Newell's formula only for the p next neighbors of a cell. For the long-range interaction one uses a simple dipole field

$$\boldsymbol{h}^{\text{dipole}}(\boldsymbol{x}) = \frac{1}{4\pi} \frac{3\boldsymbol{x}\,(\boldsymbol{M}\cdot\boldsymbol{x}) - |x|^2\,\boldsymbol{M}}{|x|^5}, \tag{17}$$

with the magnetic moment $\boldsymbol{M} = V\, M_s\, \boldsymbol{m}$ for a cell volume V.

The difference of Newell- and dipole-field is also visualized in Fig. 4. Choosing $p = 20$ as default gives accurate results for the near-field, but avoid fluctuations to the long-range interactions. One further positive effect

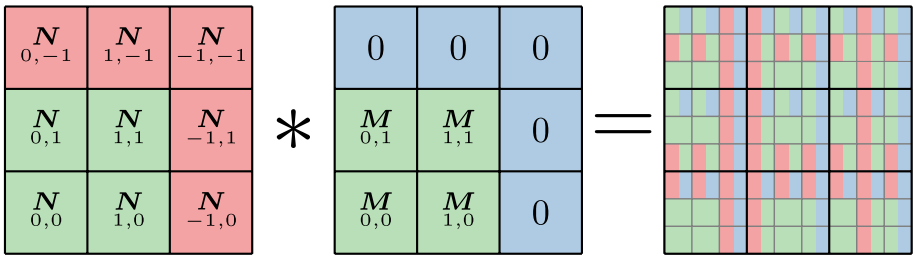

Figure 3. Discrete convolution of the magnetization $\boldsymbol{M}$ with the demagnetization kernel $\boldsymbol{N}$. The color blocks in the result matrix represent the multiplications of the respective input values. Figure taken from[21] with kind permission of The European Physical Journal (EPJ).

of using the dipole field for long-range interaction is that the setup of the demagnetization gets much faster and there is no need for caching the kernel to disk.

In case of multiple thin layers, which are not equi-distantly spaced, it is possible to only use the convolution theorem in the two lateral dimensions[3]. The asymptotic runtime in this case amounts to $\mathcal{O}(n_{xy}\ \log n_{xy}\ n_z^2)$, where n_{xy} are the number of cells within the lateral dimensions and n_z is the number of non-equidistant layers.

True periodic boundary conditions can be used to suppress the influence of the shape anisotropy due to the global demagnetization factor. This is crucial when simulating the microstructure of magnetic materials. The differential version of the corresponding Maxwell equations can be solved efficiently by means of the Fast Fourier Transfrom, which intrinsically fulfills the proper periodic boundary conditions[29].

Oersted field. For many applications like the optimization of spinwave excitation antennas[30,31] or spin orbit torque enabled devices[32,33] the Oersted field created by a given current density has an important influence. For continuous current density $\boldsymbol{j}$ it can be calculated by means of the Biot-Savart law

$$\boldsymbol{h}^{\text{oersted}}(\boldsymbol{x}) = \frac{1}{4\pi} \int \boldsymbol{j}(\boldsymbol{x}') \times \frac{\boldsymbol{x} - \boldsymbol{x}'}{|\boldsymbol{x} - \boldsymbol{x}'|^3}\ \mathrm{d}\boldsymbol{x}' \tag{18}$$

Most common finite difference micromagnetic codes offer the possibility to use arbitrary external fields, but lack the ability to calculate the Oersted field directly. Fortunately, the Oersted field has a similar structure to the demagnetization field and the occuring integral equations can be solved analytically[34]. This makes it possible to consider current densities which vary in space and time, since the corresponding field can be updated at each time-step.

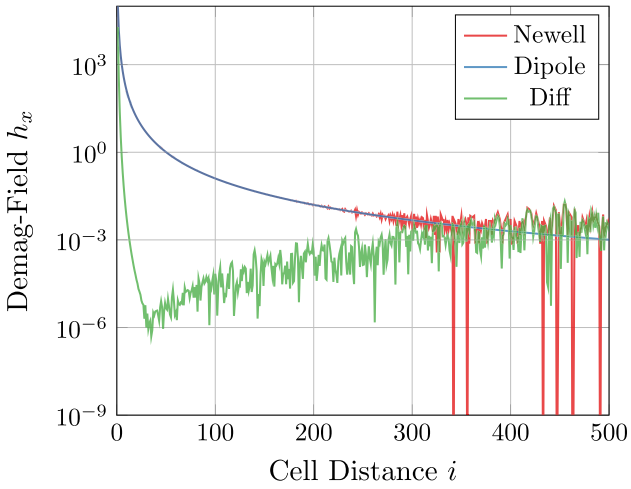

Figure 4. Comparison of the numerical strayfield caluclation using Newell's equations[26], the Dipole appoximation (17), and the differnce of both increasing cell distance.

As with the demagnetization field the far-field is approximated by the field of a singular current density, which avoids numerical fluctuations.

Spin-torque fields. Modern spintronic devices are based on different kinds of spin-torque fields[35,36], which describe the interaction of the magnetization with the electron spin. An overview about models and numerical methods used to simulate spintronic devices can be found in[21].

In general arbitrary spin torque contributions can be described by the following field

$$\boldsymbol{h}^{\mathrm{st}}(\boldsymbol{x}) = -\frac{j_e \hbar}{2e\mu_0 M_s}\left[\eta_{\mathrm{damp}}\, \boldsymbol{m} \times \boldsymbol{p} + \eta_{\mathrm{field}}\, \boldsymbol{p}\right], \tag{19}$$

with the current density j_e, the reduced Planck constant $\hbar$, the elementary charge e, and the polarization of the electrons $\boldsymbol{p}$. η_{damp} and η_{field} are material parameters which describe the amplitude of damping- and field-like torque[37].

In case of Spin–Orbit-Torqe (SOT) η_{field} and η_{damp} are constant material parameters, whereas for the Spin-Transfer-Torque inside of magnetic multilayer structures those parameters additionally depend on ϑ—the angle between $\boldsymbol{m}$ and $\boldsymbol{p}$. Expressions for the angular dependence are e.g. introduced in the original work of Slonczewski[38] or more generally in[39].

Spin-Transfer-Torque can also occur in bulk material inside regions with high magnetization gradients like domain walls, or vortex-like structures. The following field has been proposed by Zhang and Li[40] for this case:

$$\boldsymbol{h}^{\mathrm{stt,zl}}(\boldsymbol{x}) = \frac{b}{\gamma}\left[\boldsymbol{m} \times (\boldsymbol{j}_e \cdot \nabla)\boldsymbol{m} + \xi\ (\boldsymbol{j}_e \cdot \nabla)\boldsymbol{m}\right], \tag{20}$$

with the reduced gyromagnetic ratio γ, the degree of nonadiabacity ξ. b is the polarization rate of the conducting electrons and can be written as

$$b = \frac{\beta \mu_B}{eM_s(1+\xi^2)}, \tag{21}$$

with the Bohr magneton μ_B, and the dimensionless polarization rate β.

The muMAG Standard Problem #5 is included in the magnum.np source code for demonstration of the Zhang-Li spin-torque.

Interlayer-exchange field. The Ruderman–Kittel–Kasuya–Yosida (RKKY) interaction[41] gives rise to an exchange coupling of the magnetic layers in multilayer structures which are separated by a non-magnetic layer. The corresponding continuous interaction energy can be written as

$$E^{\mathrm{rkky}} = -\int_{\Gamma} J_{\mathrm{rkky}}\, \boldsymbol{m}_1 \cdot \boldsymbol{m}_2\ \mathrm{d}\boldsymbol{A}, \tag{22}$$

where Γ is the interface between two layers with magnetizations $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$, respectively. J_{rkky} is the coupling constant which oscillates with respect to the spacer layer thickness.

When discretizing the RKKY field using finite difference in many cases the spacer layer is not discretized. Instead the interaction constant J_{rkky} is scaled by the spacer layer thickness. Additionally, one has to make sure

that the two layers are not coupled by the classical exchange interaction. In magnum.np the corresponding exchange field can be defined on subdomains, so there is no coupling via the interface.

The magnetizations $\boldsymbol{m}_1$, $\boldsymbol{m}_2$ should be evaluated directly at the interface. Since the magnetization is only available at the cell centers, most finite difference codes use a lowest order approximation which directly uses those center values. magnum.np also allows to use higher order approximations, which show significantly better convergence if partial domain walls are formed at the interface[42].

For $\boldsymbol{m}_1$ the following expression can be found:

$$\boldsymbol{m}_1 = \begin{cases} \boldsymbol{m_i} & \text{if order} = 0 \\ \frac{3}{2}\,\boldsymbol{m_i} - \frac{1}{2}\,\boldsymbol{m}_{i-1} & \text{if order} = 1 \\ \frac{15}{8}\,\boldsymbol{m_i} - \frac{5}{4}\,\boldsymbol{m}_{i-1} + \frac{3}{8}\,\boldsymbol{m}_{i-2} & \text{if order} = 2 \end{cases} \tag{23}$$

where $\boldsymbol{m_i}$ denotes the magnetization of the cell adjacent to the interface insided of layer 1, where the field should be evaluated. $\boldsymbol{m}_{i-1}$ and $\boldsymbol{m}_{i-2}$ are its first and second next neighbor, respectively. A similar expression is given for $\boldsymbol{m}_2$, but indices $\boldsymbol{i}$ are replaced with the corresponding indices $\boldsymbol{j}$ of cells inside of layer 2.

Finally, the discretization of the RKKY field corresponding to the energy Eq. (22) yields

$$\boldsymbol{h}_{\boldsymbol{i}}^{\text{rkky}} = \frac{J_{\text{rkky}}}{\mu_0\, M_s\, \Delta_z}[\boldsymbol{m}_2 - (\boldsymbol{m}_1 \cdot \boldsymbol{m}_2)\boldsymbol{m}_1], \tag{24}$$

with the cell thickness Δ_z and the indices $\boldsymbol{i}$ and $\boldsymbol{j}$ of two adjacent cells in layer i and j. Note that the second term stems from a modified boundary condition for the classical exchange field, if higher order approximations are used.

Thermal field. Thermal fluctuation can be considered in micromagnetic simulations by adding a stochastic thermal field $\boldsymbol{h}^{\text{th}}$, which is characterized by

$$\begin{aligned} \langle \boldsymbol{h}_{\boldsymbol{i}}^{\text{th}} \rangle &= 0 \\ \langle \boldsymbol{h}_{\boldsymbol{i}}^{\text{th}}(t_0)\; \boldsymbol{h}_{\boldsymbol{j}}^{\text{th}}(t_1) \rangle &= \frac{2\alpha k_B T}{\mu_0 M_s \gamma\, V \Delta t}\; \delta(t_1 - t_0)\; \delta_{\boldsymbol{ij}} \end{aligned} \tag{25}$$

with the Boltzmann constant k_B, the temperature T, the dimensionless damping parameter α, the cell volume V, and the timestep Δt. $\langle . \rangle$ denotes the ensemble average. The two delta functions indicate that the thermal noise is spatially and temporally uncorrelated. The actual thermal field can then be calculated by

$$\boldsymbol{h}_{\boldsymbol{i}}^{\text{th}} = \boldsymbol{\eta_i} \sqrt{\frac{2\alpha k_B T}{\mu_0 M_s \gamma\, V \Delta t}}, \tag{26}$$

where $\boldsymbol{\eta_i}$ is a random vector drawn from a standard normal distribution for each time-step.

When numerically integrating stochastic differential equations, a drift term can occur if not using the correct statistics within the numerical methods. Although some higher-order Runge-Kutta schemes exist, they become increasingly complex. Fortunately, it has been proven that in case of the LLG the drift term only changes the length of the magnetization, which is fixed anyway. Thus, it is possible to straight forwardly use available adaptive higher order schemes for the solution of the stochastic LLG[43].

Timings. Benchmarks of the field terms are presented in Fig. 5. The results show that for systems larger than about $N = 10^6$ elements, the demagnetization field is the dominating field term and it is less than a factor 2 slower than the mumax3 version. However, these timings have been performed without any low-level optimization. Instead magnum.np utilizes high-level optimization, that does not influence the simplicity of the code. For example just-in-time compilers (like PyTorch-compile, numba, nvidia-warp, etc.) are used to improve the performance of the code. For all local field contributions this works increadibly well and the resulting timings are even outperforming mumax3. Optimized timing using `torch.compile` of the recently published version 2.0 of PyTorch are included in Fig. 5. Unfortunately, `torch.compile` does not yet support complex datatypes, which prevents it from being used to calculate the demagnetization field.

In case of the demagnetization field an optimized padding for the 3D FFT which is not yet provided by PyTorch, could give some further speedup.

Examples

The following section provides some examples which should demonstrate the ease of use and the power of the magnum.np interface. Due to the python/PyTorch interface pre- and post-processing can be done in a single script (or at least in the same scripting language) and allows to keep the complete simulation framework as simple as possible. The presented code focuses on complex examples which would be more elaborate to setup with other micromagnetic codes. In the magnum.np source code[44] several other examples are included, such as hysteresis loop calculations, simulation of soft magnetic composites, an RKKY standard problem and the muMAG standard problems. Further examples will be continously added.

Spintronic devices. The first example demonstrates the creation and manipulation of skyrmions in magnetic thin films, that can be patterned by means of ion radiation techniques to locally alter the magnetic materi-

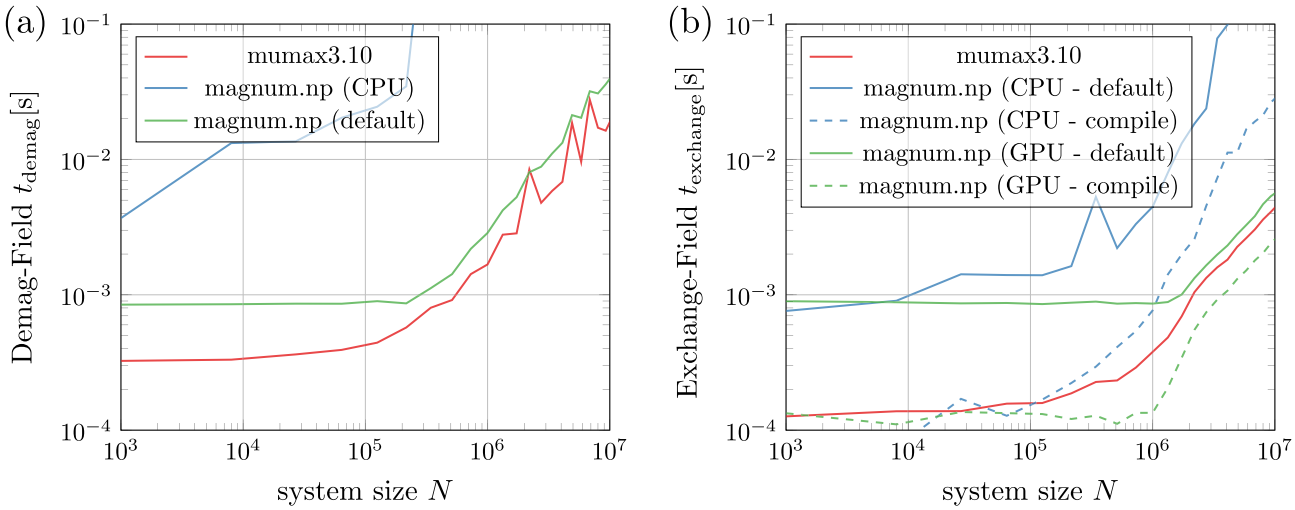

Figure 5. Benchmarking (**a**) demagnetization field and (**b**) exchange field for different system sizes N on an Intel(R) Xeon 6326 CPU @ 2.90 GHz using one NVIDIA A100 80GB GPU (CUDA Driver 11.8). An average of 10000 evaluations has been measured for each field term. Before measurent begins, 1000 warm-up loops are used to ensure that the GPU has reached its maximum performance state. Single precision arithmetics are used for comparison with mumax3.

```
mesh = Mesh(n = (1500, 300, 1), dx = (10e-9, 10e-9, 20e-9)) # mesh
domains = read_mesh(mesh, "mesh.msh", scale = 1e-9)         # read domain-ids
state = State(mesh)                                         # define state
irradiated = (domains == 1)                                 # boolean array

state.material = {                                  # set material everywhere
    "Ms": 400e3,                                    #
    "A": 4e-12,                                     #
    "Ku": 100e3,                                    #
    "Ku_axis": (0,0,1),                             #
    "alpha": 1.0}                                   #
state.material["Ku"][irradiated] = 50e3             # reset material on sub-domain

write_vti(state.material, "material.vti")           # store material parameters
```

Listing 3:. Mesh creation and boolean domains read from an external unstructuredgrid file "mesh.msh", which are used to define location dependent materialparameters.

als of the system[45]. This simulation technique is also useful for the numerical modeling of structued Pt-layers on top of the thin-film that create a location-dependent DMI interaction as realized recently in an experimental work[46].

Listing 3 shows the material definition for the spintronic demo, where the anisotropy constant is altered in the irradiated region. A rectangular mesh with $\boldsymbol{n}$ cells and a grid spacing $\boldsymbol{dx}$ is created and integer domain-ids are read from an unstructured mesh file by means of the `mesh_reader`. Boolean domain arrays can then be derived and in turn be used to set location dependent material parameters, which will influences the local skyrmion densities.

A random initial magnetization is set and the default RKF45 solver is used for time-integration. Several logging capabilities allow to flexibly log scalar- and field-data to files. Custom python functions that return derived quantities, such as the Induction Map (IM) or the Lorentz Transmission Electron Microscopy (LTEM) image of the magnetization state, can simply be added as log entries. Listing 4 shows the corresponding code and the results are visualized in Fig. 6. One can see that the density of skyrmions in the irradiated region is increased significantly compared to the outside region. The lower anisotropy allows the nucleation of not only skyrmions, but also trivial type-II bubbles, and antiskyrmions[47].

Inverse design. Finding the optimal shape of magnetic components for certain applications is an essential, but quite challenging task. An automated topology optimization requires the efficient calculation of the so-called forward problem, as well as the corresponding gradients (compare e.g.[48,49]). The following example should demonstrate how magnum.np can be used to solve inverse problems, by utilizing PyTorch's autograd mechanism.

The field created by a magnetization at a certain location $\boldsymbol{x}_0$ should be maximized. The objective function J which should be minimized could thus be defined as $J[\boldsymbol{m}] = h_y(\boldsymbol{x}_0))$. The forward problem is simply an

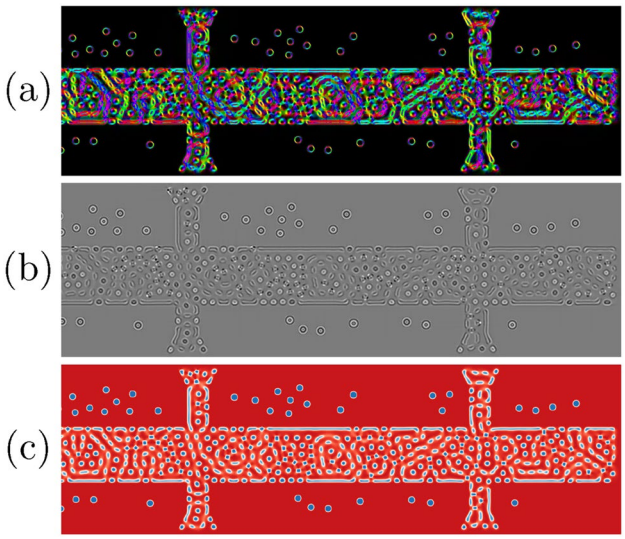

Figure 6. Visualization of the created skyrmions at $\mu_0 H_z = 250\text{mT}$ using (**a**) an Induction Map, (**b**) an underfocus Lorentz Transmission Electron Microscopy image, or (**c**) the z-component of the magnetization.

```
state.m = RandomUnitSphere(state)           # start with random magnetization

demag    = DemagField()                     # define field contributions
aniso    = UniaxialAnisotropyField()        #
exchange = ExchangeField()                  #
external = ExternalField(Hext)              #

llg = LLGSolver([demag, aniso, exchange, external])   # use RKF45 solver
logIM = ('IM', lambda state: IMImage(state))          # evaluate IM from state
logLTEM = ('LTEM', lambda state: LTEMImage(state))    # evaluate LTEM from state

logger = Logger("data", ['t','m'], ['m', logIM, logLTEM])
while state.t < 20e-9:          # integrate until 20ns
    logger << state             # log [t,m] as scalars, and [m,IM,TEM] as VTI
    llg.step(state, 1e-11)      # perform time-integration for 10ps
```

Listing 4:. Setup of time-integration for 20 ns and logging. Scalar data, like time$t$ and avarage magnetization $\langle \boldsymbol{m} \rangle$, will be written to a column based text field.Field data, like the magnetization $\boldsymbol{m}$ as well as a corresponding LTEM image,will be written to .vti files utilizing pyvista.

evaluation of the demagnetization field. The optimization requires the calculation of the gradient $\boldsymbol{g} = \frac{\partial J}{\partial \boldsymbol{m}}$. The magnetization should always point in y direction, and its magnitude m_y saturates at M_s.

The optimal magnetization which leads to the maximum field at the evaluation point can be found by using an gradient-based optimization method (e.g. Conjugate Gradient). Since this simple example is linear, the optimal solution is found after a single iteration. Depending on the sign of the gradient the optimal magnetization within each cell is 1, if the calculated gradient is positive and 0 otherwise. Listing 5 summarizes how the gradient calculation is performed. The optimized magnetization is visualized in Fig. 7 and shows perfect agreement with the analytical result.

Conclusion

An overview of the basic design ideas of magnum.np has been given. Equations and references for the most important field contributions as well as solving methods are included for clarification. Some typical applications are provided in order to demonstrate the ease of use and the power of the provided python-base interface. Furthermore the use of PyTorch extends magnum.np's capabilities to inverse probems and allows seamlessly running applications on CPU and GPU without any modification of the code. The openness of the project should encourage other developers to contribute code and use magnum.np as a framework for the development and testing of new algorithms, while still getting reasonable performance and generality.

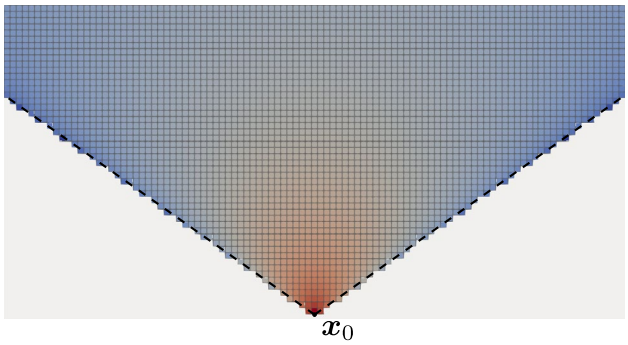

Figure 7. Optimal topology that maximizes the z-component of the strayfield at the marked cell. Only cells with a positive gradient are shown. The logarithmic color scheme represents the sensitivity of the objective function on the magnetization within the corresponding cell (red means a large sensitivity). The dotted line shows the analytic result $x < \sqrt{2}\,y$.

```
from magnumnp import *

n  = (101, 51, 1)             # define forward problem
dx = (5e-9, 5e-9, 5e-9)       #
mesh = Mesh(n, dx)            #
state = State(mesh)           #
state.material = {"Ms": 1.}   #

state.m = state.Constant([0,1,0], requires_grad = True) # define design parameter
h = DemagField().h(state)       # calculate demag field
J = h[n[0]//2, 0, n[2]//2, 1]   # evaluate objective function J

J.backward()                    # evalutate gradient dJ/dm
write_vti(state.m.grad, "data/m_grad.vti")   # output
```

Listing 5:. Full topology optimization example which solves the inverse strayfieldproblem utilizing PyTorch's autograd mechanism.

Data availability

magnum.np is Open Source Software published under the GPL3 Licence. Its complete source code, demos and unit tests can be found at https://gitlab.com/magnum.np/magnum.np.

Received: 7 March 2023; Accepted: 21 July 2023
Published online: 25 July 2023

References

1. Donahue, M. J. & Donahue, M. Oommf user's guide, version 1.0 (1999).
2. Vansteenkiste, A. *et al.* The design and verification of mumax3. *AIP Adv.* **4**(10), 107133 (2014).
3. Heistracher, P., Bruckner, F., Abert, C., Vogler, C. & Suess, D. Hybrid FFT algorithm for fast demagnetization field calculations on non-equidistant magnetic layers. *J. Magn. Magn. Mater.* **503**, 166592 (2020).
4. Abert, C. magnum.fd—a finite-difference/fft package for the solution of dynamical micromagnetic problems. https://github.com/micromagnetics/magnum.fd (2013).
5. Bisotti, M.-A. *et al.* Fidimag—a finite difference atomistic and micromagnetic simulation package. *J. Open Res. Softw.* **6**(1), 22. https://doi.org/10.5334/jors.223 (2018).
6. Weinan, E., Ren, W. & Vanden-Eijnden, E. Simplified and improved string method for computing the minimum energy paths in barrier-crossing events. *J. Chem. Phys.* **126**(16), 164103 (2007).
7. Koraltan, S. *et al.* Dependence of energy barrier reduction on collective excitations in square artificial spin ice: A comprehensive comparison of simulation techniques. *Phys. Rev. B* **102**(6), 064410 (2020).
8. Hofhuis, K. *et al.* Thermally superactive artificial kagome spin ice structures obtained with the interfacial dzyaloshinskii-moriya interaction. *Phys. Rev. B* **102**(18), 180405 (2020).
9. Wang, Q., Chumak, A. V. & Pirro, P. Inverse-design magnonic devices. *Nat. Commun.* **12**(1), 2636 (2021).
10. Papp, Á., Porod, W. & Csaba, G. Nanoscale neural network using non-linear spin-wave interference. *Nat. Commun.* **12**(1), 1–8 (2021).
11. Kiechle, M. *et al.* Experimental demonstration of a spin-wave lens designed with machine learning. *IEEE Magn. Lett.* **13**, 1–5 (2022).
12. Google. Colaboratory.https://colab.research.google.com/ (2023). Accessed 02 June 2023.
13. Paszke, A. *et al.* Pytorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.* **32** (2019).
14. Paszke, A., *et al.* Automatic differentiation in pytorch (2017).
15. Kovacs, A. *et al.* Magnetostatics and micromagnetics with physics informed neural networks. *J. Magn. Magn. Mater.* **548**, 168951 (2022).

16. Mathews, J. H. *et al. Numerical Methods Using MATLAB* Vol. 4 (Pearson Prentice Hall, Upper Saddle River, 2004).
17. Suess, D. *et al.* Time resolved micromagnetics using a preconditioned time integration method. *J. Magn. Magn. Mater.* **248**(2), 298–311 (2002).
18. Exl, L. *et al.* Labonte's method revisited: An effective steepest descent method for micromagnetic energy minimization. *J. Appl. Phys.* **115**(17), 17D118 (2014).
19. Exl, L. *et al.* Preconditioned nonlinear conjugate gradient method for micromagnetic energy minimization. *Comput. Phys. Commun.* **235**, 179–186 (2019).
20. Fischbacher, J. *et al.* Nonlinear conjugate gradient methods in micromagnetics. *AIP Adv.* **7**(4), 045310 (2017).
21. Abert, C. Micromagnetics and spintronics: Models and numerical methods. *Eur. Phys. J. B* **92**(6), 1–45 (2019).
22. Heistracher, P., Abert, C., Bruckner, F., Schrefl, T. & Suess, D. Proposal for a micromagnetic standard problem: domain wall pinning at phase boundaries. *J. Magn. Magn. Mater.* **548**, 168875 (2022).
23. Dzyaloshinsky, I. A thermodynamic theory of weak ferromagnetism of antiferromagnetics. *Phys. Chem. Solids* **4**, 241 (1958).
24. Moriya, T. Anisotropic superexchange interaction and weak ferromagnetism. *Phys. Rev.* **120**(1), 91 (1960).
25. Cortés-Ortuño, D. *et al.* Proposal for a micromagnetic standard problem for materials with Dzyaloshinskii–Moriya interaction. *New J. Phys.* **20**(11), 113015 (2018).
26. Newell, A. J., Williams, W. & Dunlop, D. J. A generalization of the demagnetizing tensor for nonuniform magnetization. *J. Geophys. Res. Solid Earth* **98**(B6), 9551–9555 (1993).
27. Abert, C. *et al.* A full-fledged micromagnetic code in fewer than 70 lines of numpy. *J. Magn. Magn. Mater.* **387**, 13–18 (2015).
28. Krüger, B., Selke, G., Drews, A. & Pfannkuche, D. Fast and accurate calculation of the demagnetization tensor for systems with periodic boundary conditions. *IEEE Trans. Magn.* **49**(8), 4749–4755 (2013).
29. Bruckner, F., Ducevic, A., Heistracher, P., Abert, C. & Suess, D. Strayfield calculation for micromagnetic simulations using true periodic boundary conditions. *Sci. Rep.* **11**(1), 1–8 (2021).
30. Demidov, V. E. *et al.* Excitation of microwaveguide modes by a stripe antenna. *Appl. Phys. Lett.* **95**(11), 112509 (2009).
31. Chumak, A. V. *et al.* Advances in magnetics roadmap on spin-wave computing. *IEEE Trans. Magn.* **58**(6), 1–72 (2022).
32. Talmelli, G. *et al.* Spin-wave emission by spin–orbit-torque antennas. *Phys. Rev. Appl.* **10**(4), 044060 (2018).
33. Woo, S. *et al.* Spin–orbit torque-driven skyrmion dynamics revealed by time-resolved X-ray microscopy. *Nat. Commun.* **8**(1), 15573 (2017).
34. Krüger, B. *Current-Driven Magnetization Dynamics: Analytical Modeling and Numerical Simulation*. PhD thesis, Staats-und Universitätsbibliothek Hamburg Carl von Ossietzky (2011).
35. Garello, K. *et al.* Symmetry and magnitude of spin–orbit torques in ferromagnetic heterostructures. *Nat. Nanotechnol.* **8**(8), 587–593 (2013).
36. Avci, C. O. *et al.* Current-induced switching in a magnetic insulator. *Nat. Mater.* **16**(3), 309–314 (2017).
37. Abert, C. *et al.* Fieldlike and dampinglike spin-transfer torque in magnetic multilayers. *Phys. Rev. Appl.* **7**(5), 054007 (2017).
38. Slonczewski, J. Currents and torques in metallic magnetic multilayers. *J. Magn. Magn. Mater.* **247**(3), 324–338 (2002).
39. Xiao, J., Zangwill, A. & Stiles, M. D. Macrospin models of spin transfer dynamics. *Phys. Rev. B* **72**(1), 014446 (2005).
40. Zhang, S. & Li, Z. Roles of nonequilibrium conduction electrons on the magnetization dynamics of ferromagnets. *Phys. Rev. Lett.* **93**(12), 127204 (2004).
41. Ruderman, M. A. & Kittel, C. Indirect exchange coupling of nuclear magnetic moments by conduction electrons. *Phys. Rev.* **96**(1), 99 (1954).
42. Suess, D. *et al.* Accurate finite-difference micromagnetics of magnets including RKKY interaction: Analytical solution and comparison to standard micromagnetic codes. *Phys. Rev. B* **107**(10), 104424 (2023).
43. Leliaert, J. *et al.* Adaptively time stepping the stochastic Landau–Lifshitz–Gilbert equation at nonzero temperature: Implementation and validation in MuMax3. *AIP Adv.* **7**(12), 125010 (2017).
44. magnum.np. magnum.np. https://gitlab.com/magnum.np/magnum.np (2023). Accessed 31 Jan 2023.
45. Kern, L.-M. *et al.* Deterministic generation and guided motion of magnetic skyrmions by focused he+-ion irradiation. *Nano Lett.* **22**(10), 4028–4035 (2022).
46. Vélez, S. *et al.* Current-driven dynamics and ratchet effect of skyrmion bubbles in a ferrimagnetic insulator. *Nat. Nanotechnol.* **17**(8), 834–841 (2022).
47. Heigl, M. *et al.* Dipolar-stabilized first and second-order antiskyrmions in ferrimagnetic multilayers. *Nat. Commun.* **12**(1), 2611 (2021).
48. Abert, C. *et al.* A fast finite-difference algorithm for topology optimization of permanent magnets. *J. Appl. Phys.* **122**(11), 113904 (2017).
49. Huber, C. *et al.* Topology optimized and 3d printed polymer-bonded permanent magnets for a predefined external field. *J. Appl. Phys.* **122**(5), 053904 (2017).

Acknowledgements

This research was funded in whole, or in part, by the Austrian Science Fund (FWF) P 34671 and (FWF) I 4917. For the purpose of open access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

Author contributions

F.B. and C.A. developed the core components of magnum.np. S.K. and D.S. contributed and improved the parts of the code. S.K. performed a large number of simulations using magnum.np and provided most of the demos. All authors contributed to the paper writing.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to F.B.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.