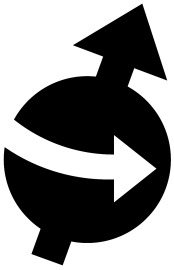
VAMPIRE

User Manual



VAMPIRE

User Manual

Software version 5.0

Manual written by Richard F. L. Evans and Andreas Biternas.

The VAMPIRE software package is principally developed and maintained by Richard F. L. Evans. Code contributors: Andrea Meo, Samuel Morris, Matthew Ellis, Oscar David Arbeláez Echeverri, Weijia Fan, Phanwadee Chureemart, Rory Pond, Sarah Jenkins, Joe Barker, Thomas Ostler, Andreas Biternas, Roy W Chantrell, Wu Hong-Ye

The VAMPIRE source code is available from www.github.com/richard-evans/vampire. This manual, software features, tutorials and more information is available from the VAMPIRE webpage at http://vampire.york.ac.uk/

Table of Contents

Introducing VAMPIRE

VAMPIRE is a state-of-the-art atomistic simulator for magnetic nanomaterials. This software is the culmination of several years of continuous development, with an aim to make atomistic simulation of magnetic materials routinely available to the non-specialist researcher. Before now, using atomistic models to simulate magnetic systems required in depth and technical knowledge of the underlying theoretical methods, computer programming skills and the ability to debug and understand intricate computational problems. The code is designed with ease of use in mind, and includes an extensive set of input parameters to control the simulations through a plain text input file. Subject to future funding it is also hoped to develop graphical user interfaces for Mac$^{\mathrm{TM}}$ OS X and Windows$^{\mathrm{TM}}$ which should make using the code more accessible.

The VAMPIRE project is still very much under active development, with an open development of all code features. The features are always available during the development stages from the develop branch of the code, but with the caveat that they are not always fully reliable. Feedback of any bugs or errors to the VAMPIRE developers is always welcome, as well as any feature requests or enhancements.

We hope that as the VAMPIRE project develops it will become a useful tool for the magnetics community for specialists and non-specialists alike.

1 Background theory

While the underlying theory behind the atomistic spin model is well known in the scientific literature, in the following a very brief overview of the fundamental theory is presented for the benefit of those who do not wish to study the methods in great detail. If more information is required then a comprehensive review of the methods implemented in VAMPIRE is available from the project website.

Atomistic Spin Models

Atomistic spin models form the natural limit of two distinct approaches, namely micromagnetics and ab-initio models of the electronic structure. In micromagnetics a material is discretized into small domains where the magnetization is assumed to be fully ordered within it. If the micromagnetic cell size is reduced to less than 1 nm, then the magnetization is no longer a true continuum, but a discrete entity considering localized moments on individual atoms. Similarly, when the electronic properties of the system are considered, the quantum mechanical properties can be mapped onto atomic cores in a manner similar to molecular dynamics, where the effective properties can often be treated with a classical approximation.

The advantage of the atomistic model over micromagnetics is that it naturally deals with atomic ordering and variation of local properties seen in real materials, such as interfaces, defects, roughness etc. The discrete formulation also allows the simulation of high temperatures above and beyond the Curie temperature, where the usual continuum micromagnetic approach breaks down. Such effects or often central to current problems in magnetism such as materials for spin electronics, heat assisted magnetic recording or ultrafast laser processes. Similarly for ab-initio calculations, mapping onto an effective spin model allows apply the full quantum mechanical deal of the properties to much larger systems and the consideration of dynamic effects on much longer timescales.

The Spin Hamiltonian

The basis of the atomistic spin model is the spin Hamiltonian, which describes the fundamental spin-dependent interactions at the atomic level (neglecting the effects of potential and kinetic energy and electron correlations). The spin Hamiltonian is typically defined as

$$\mathscr{H} = -\sum_{i<j} J_{ij}\mathbf{S}_i \cdot \mathbf{S}_j - k_2 \sum_i S_z^2 - \mu_{\mathrm{S}} \sum_i \mathbf{B}_{\mathrm{app}} \cdot \mathbf{S}_i$$

describing exchange, uniaxial anisotropy and applied field contributions respectively. Important parameters are the Heisenberg exchange J_{ij}, the anisotropy constant k_2 and the atomic spin moment, μ_{S}. $\mathbf{S}_i$ is a unit vector which describes the orientation of the local spin moment. In most magnetic materials the exchange interactions are the dominant contribution, usually by two orders of magnitude, and gives rise to the atomic ordering of the spin directions. For ferromagnetic materials (parallel alignment of spins) $J_{ij} > 0$, while for anti-ferromagnetic materials (antiparallel alignment of spins), $J_{ij} < 0$.

While the exchange interaction determines the ordering of the spins, it is usually isotropic, and so there is no preferential orientation of all the spins in the system. Most magnetic materials are anisotropic, that is the spins have a preferred orientation in space, which arises at the atomic level due to the local crystal environment, hence its full name of magnetocrystalline anisotropy. In the model this is most commonly uniaxial anisotropy, where the spins prefer to lie along a single preferred axis, known as the easy axis. The strength of the anisotropy is determined by the anisotropy constant, in our case k_2, where positive value prefer alignment along the z-axis, while negative values prefer alignment around the $x-y$ plane.

The last term describes the coupling of the spin system to an externally applied field, $\mathbf{B}_{\mathrm{app}}$, or Zeeman field. The applied field is used to reverse the orientation of the spins, and can be used in the simulation to calculate hysteresis loops, for example.

Spin Dynamics

The spin Hamiltonian describes the energetics of the system, but says nothing about the dynamic behaviour. For that the Landau-Lifshitz-Gilbert (LLG) equation is used to describe the dynamics of atomic spins. The LLG is given by

$$\frac{\partial \mathbf{S}_i}{\partial t} = -\frac{\gamma}{(1+\lambda^2)}[\mathbf{S}_i \times \mathbf{B}^i_{\mathrm{eff}} + \lambda \mathbf{S}_i \times (\mathbf{S}_i \times \mathbf{B}^i_{\mathrm{eff}})] \tag{1.1}$$

where $\mathbf{S}_i$ is a unit vector representing the direction of the magnetic spin moment of site i, γ is the gyromagnetic ratio and $\mathbf{B}^i_{\rm eff}$ is the net magnetic field on each spin. The atomistic LLG equation describes the interaction of an atomic spin moment i with an effective magnetic field, which is obtained from the negative first derivative of the complete spin Hamiltonian, such that:

$$\mathbf{B}^i_{\rm eff} = -\frac{1}{\mu_{\rm S}}\frac{\partial \mathscr{H}}{\partial \mathbf{S}_i} \tag{1.2}$$

where $\mu_{\rm S}$ is the local spin moment. The inclusion of the spin moment within the effective field is significant, in that the field is then expressed in units of Tesla, given a Hamiltonian in Joules. The LLG is integrated numerically using the Heun numerical scheme, which allows the time evolution of the spin system to be simulated.

Citations

If you use VAMPIRE for your research, it is helpful to acknowledge the authors of the code by citing relevant papers and include a statement in the paper such as the following:

The simulations in this work made use of the VAMPIRE software package [1]

and add a footnote reading:

[1] VAMPIRE software package version 5.0 available from https://vampire.york.ac.uk

In addition, it is recommended for reproducibility that you include the githash for the specific version of the code, which enables someone to checkout the specific version of the code used for the simulations. [1] VAMPIRE software package version 5.0 (Version aa842a409c68d6724e156df6cab0bcaa172f5f41) available from https://vampire.york.ac.uk

If you use the code, please cite the following article:

Atomistic spin model simulations of magnetic nanomaterials

R. F. L. Evans, W. J. Fan, P. Chureemart, T. A. Ostler, M. O. A. Ellis and R. W. Chantrell

J. Phys.: Condens. Matter 26, 103202 (2014)

If you use the constrained Monte Carlo method, in addition please cite:

Constrained Monte Carlo method and calculation of the temperature dependence of magnetic anisotropy
P. Asselin, R. F. L. Evans, J. Barker, R. W. Chantrell, R. Yanes, O. Chubykalo-Fesenko, D. Hinzke and U. Nowak
Phys. Rev. B. 82, 054415 (2010)

If you use the temperature rescaling method please cite:

Quantitative simulation of temperature-dependent magnetization dynamics and equilibrium properties of elemental ferromagnets
R. F. L. Evans, U. Atxitia, and R. W. Chantrell
Phys. Rev. B 91, 144425 (2015)

2 Installation

This chapter covers the requirements, installation and support for VAMPIRE on different platforms.

System Requirements

VAMPIRE is designed to be generally portable and compilable on Linux, Unix, Mac OSX and Windows with a range of different compilers. By design the software has a very minimal dependence on external libraries to aid compilation on the widest possible range of platforms without needing to first install and configure a large number of other packages. VAMPIRE is designed to be maximally efficient on high performance computing clusters and scalable to thousands of processors, and as such is the recommended platform if you have access to appropriate resources.

Hardware Requirements

VAMPIRE has been successfully tested on a wide variety of x86 and power PC processors. Memory requirements are generally relatively modest for most systems, though larger simulations will require significantly more memory. VAMPIRE is generally computationally limited, and so the faster the clock speed and number of processor cores the better.

Binary installation

Compiled binaries of the latest release version are available to download from:

https://vampire.york.ac.uk/download/

for Linux and Mac$^{\mathrm{TM}}$ OS X platforms. For the Linux and Mac OS X releases, a simple installation script install.sh installs the binary in /opt/vampire/ and appends the directory to your environment path. On Windows the recommended method is to use the Linux subsystem for windows developer feature which adds a linux subsystem that is capable of running the standard linux binary. A copy of qvoronoi is integrated into VAMPIRE for generating granular structures.

Compiling from source

The best way to get the vampire source code is using git, a distributed version control program which enables changes in the code to be tracked. Git is readily available on linux (git-core package on ubuntu) and Mac (via MacPorts). To get vampire from the Github repository checkout your own copy of the repository using:

```
git clone git://github.com/richard-evans/vampire.git
```

This way, updates to the code can be easily merged with the downloaded version. Compiling is generally as easy as running make in Unix platforms. In addition, on a multicore processor compilation can be parallelised using the -j N_t option, where N_t is the number of threads to use.

Compiling on Linux

In order to compile in linux, a working set of development tools are needed, which on ubuntu includes the packages build-essential and g++. VAMPIRE should compile without issue following a simple make command in the source directory.

For the parallel version, a working installation of openmpi is recommended, which must usually include a version of the development tools (openmpi-bin and openmpi-dev packages on ubuntu). Compilation is usually straightforward using make parallel.

Compiling on Mac OSX

With OS X, compilation from source requires a working installation of Xcode, available for free from the Mac App Store. In addition command line tools must also be installed. A working installation of MacPorts is recommended to provide access to a wide range of open source libraries and tools such as openmpi, rasmol and povray. For the serial version, compilation is the same as for linux, following a simple make serial-llvm command in the source directory.

Similarly for the parallel version, openmpi needs to be installed via MacPorts, and compilation is usually straightforward using make parallel-llvm.

Compiling on Windows

The recommended way to use vampire on Windows is to install Linux subsystem for windows 10 (see https://docs.microsoft.com/en-us/windows/wsl/install-win10). Older versions of windows are no longer supported. Once installed, you can download the serial linux binary as for linux and run as normal from the command line.

Compiling for ARCHER/Cray systems

ARCHER is the UK national supercomputer and includes custom compilers developed by Cray Inc. However, performance is generally better for the gnu compiler collection and so there is an optimized makefile option for compilation on the ARCHER and similar Cray XC30 systems. To compile, you need to swap the environment to the GNU compiler suite using module swap PrgEnv-cray PrgEnv-gnu. You can then compile with make parallel-archer which will compile a parallel binary.

Compiling for GPU acceleration with CUDA (beta)

The latest release includes a CUDA implementation for GPU accelerated atomistic spin dynamics. To compile the CUDA version of the code, you need to install the CUDA drivers and runtime. Once installed, compilation should be straightforward using make gcc-cuda. By default, the binary includes device code for a wide range of architectures. Depending on your device/card, you may need to modify the device code generation option in the makefile.

3 Running the code

To run the code in all version, you first need to specify an input file and material file, which must reside in the same directory where you run the code. Example files are available in the source code distribution, or from the Download section of the website (http://vampire.york.ac.uk/download/index.html).

Unix/Linux and macOS

In the directory including the input and material files, typing

```
./vampire-serial
```

will run the code in serial mode. For the parallel mode with openmpi,

```
mpirun -np 2 vampire-parallel
```

will run the code in parallel mode, on 2 CPUs. Increasing the -np argument will run on more cores.

Windows

Once you have installed Linux subsystem for Windows, you can run the code by launching bash for windows and following the instructions for Unix/Linux systems above.

Log file

When you run the program it will output some program information to screen to indicate that the program is running listing some details about the developers and some steps on the program execution. There are also options to tell the program to output simulation data to screen to see how the simulation is progressing. In addition, a log file is produced which provides more detail about the execution of the program and progress including timestamps, and errors that may occur and

certain performance metrics, such as time to update the dipole field or output a spin configuration file. A sample screen output header from the program is shown below.

```
                                        _
                                       (_)
 __   ____ _ _ __ ___  _ __  _ __ ___
 \ \ / / _` | '_ ` _ \| '_ \| | '__/ _ \
  \ V / (_| | | | | | | |_) | | | |  __/
   \_/ \__,_|_| |_| |_| .__/|_|_|  \___|
                      | |
                      |_|

                Version 5.0.0 Aug 25 2018 23:01:25

          Git commit: 4377c4a8c3b2334decf6b3892542927fcaddebc7

Licensed under the GNU Public License(v2). See licence file for details.

Lead Developer: Richard F L Evans <richard.evans@york.ac.uk>

Contributors: Andrea Meo, Rory Pond, Weijia Fan,
              Phanwadee Chureemart, Sarah Jenkins, Joe Barker,
              Thomas Ostler, Andreas Biternas, Roy W Chantrell,
              Wu Hong-Ye, Matthew Ellis, Razvan Ababei,
              Sam Westmoreland, Oscar Arbelaez, Sam Morris

              Compiled with:  LLVM C++ Compiler
              Compiler Flags:

Vampire includes a copy of the qhull library from C.B. Barber and
The Geometry Center and may be obtained via http from www.qhull.org.
```

4 Getting Started

VAMPIRE is a powerful software package, capable of simulating many different systems and the determination of parameters such as coercivity, Curie and Néel temperatures, reversal dynamics, statistical behaviour and more. This chapter contains an overview of the capabilities of VAMPIRE and how to use them.

Feature Overview

The features of the VAMPIRE code are split into three main categories: material parameters, structural parameters, and simulation parameters. Details of these parameters are given in the following chapters, but between them they define the parameters for a particular simulation.

Materials

Material parameters essentially define the magnetic properties of a class of atoms, including magnetic moments, exchange interactions, damping constants etc. VAMPIRE includes support for up to one hundred defined materials, and material parameters control the simulation of multilayers, random alloys, core shell particles and lithographically defined patterns.

Structures

Structural parameters define properties such as the system size, shape, particle size, or voronoi grain structures. In combination with material parameters they essentially define the system to be simulated.

Simulations

VAMPIRE includes a number of built-in simulations for determining the most common magnetic properties of a system, for example Curie temperature, hysteresis loops, or even a time series. Additionally the parameters for these simulations, such as applied field, maximum temperature, temperature increment, etc. can be set.

Input and Output Files

VAMPIRE requires at least two files to run a simulation, the input file and the material file. The input file defines all the properties of the simulated system, such as the dimensions or particle shape, as well as the simulation parameters and program output. The material file defines the properties of all the materials used in the simulation, and is usually given the .mat file extension. A sample material file Co.mat is included with the code which defines a minimum set of parameters for Co.

The output of the code includes a main output file, which records data such as the magnetisation, timesteps, temperature etc. The format of the output file is fully customisable, so that the amount of output data is limited to what is useful. In addition to the output file, the other main available outputs are spin configuration files, which with post-processing allow output of snapshots of the magnetic configurations during the simulation.

Sample input files

Sample input and output files are included in the source code distribution, but the files for a simple test simulation which computes the time dependence of the magnetisation of a cubic system are given here.

input

```
#----------------------------------------
# Sample vampire input file to perform
# benchmark calculation for version 5.0
#
#----------------------------------------

#----------------------------------------
# Creation attributes:
#----------------------------------------
create:crystal-structure = sc

#----------------------------------------
# System Dimensions:
#----------------------------------------
dimensions:unit-cell-size = 3.54 !A
dimensions:system-size-x = 7.7 !nm
dimensions:system-size-y = 7.7 !nm
dimensions:system-size-z = 7.7 !nm

#----------------------------------------
# Material Files:
#----------------------------------------
```

```
material:file = Co.mat

#----------------------------------------
# Simulation attributes:
#----------------------------------------
sim:temperature = 300.0
sim:time-steps-increment = 1000
sim:total-time-steps = 10000
sim:time-step = 1 !fs

#----------------------------------------
# Program and integrator details
#----------------------------------------
sim:program = benchmark
sim:integrator = llg-heun

#----------------------------------------
# data output
#----------------------------------------
output:real-time
output:temperature
output:magnetisation
output:magnetisation-length

screen:time-steps
screen:magnetisation-length
```

Co.mat

```
#===================================================
# Sample vampire material file version 5
#===================================================

#---------------------------------------------------
# Number of Materials
#---------------------------------------------------
material:num-materials=1
#---------------------------------------------------
# Material 1 Cobalt Generic
#---------------------------------------------------
material[1]:material-name = Co
material[1]:damping-constant = 1.0
material[1]:exchange-matrix[1] = 11.2e-21
material[1]:atomic-spin-moment = 1.72 !muB
material[1]:uniaxial-anisotropy-constant = 1.0e-24
material[1]:material-element = Ag
material[1]:minimum-height = 0.0
material[1]:maximum-height = 1.0
```

5 Unit Cell Files

VAMPIRE provides a selection of built-in unit cell crystal structures which can be specified with the create:crystal-structure keyword. However, there are many more crystal structures and magnetic materials than the code could possibly support, and so an advanced mode is available where the user can specify any atomic structure and exchange interactions which are then imported into the code and can be used to generate large systems and/or cut into the standard geometric shapes. It is also possible to simulate complex non-periodic structures such as nanoparticles obtained from molecular dynamics simulations with the same approach. The unit cell file is specified in the main input file using the keyword:

```
material:unit-cell-file = file.ucf
```

where "file.ucf" is the filename.

The unit cell file format

The unit cell file is split into two main parts. The first part specifies the unit cell shape, the atoms in the unit cell, and their material associations and categorisation for statistics purposes. The second part specifies all atomic exchange interactions within and between neighbouring unit cells. The general plain text format is as follows:

```
1 # Unit cell size:
2 ucx ucy ucz
3 # Unit cell lattice vectors:
4 ucvxx ucvxy ucvxz
5 ucvyx ucvyy ucvyz
6 ucvzx ucvzy ucvzz
7 # Atoms
8 num_atoms_in_unit_cell number_of_materials
9 atom_id cx cy cz [mat_id cat_id hcat_id]
10 ...
```

```
11    ...
12 # Interactions
13 num_interactions [exchange_type]
14 IID i j dx dy dz | Jij
                   | Jx Jy Jz
                   | Jxx Jxy Jxz Jyx Jyy Jyz Jzx Jzy Jzz
```

In general this format now allows the specification of any system we want, but clearly complex multi- layered systems require large file sizes. Working through line by line:

1	# defines a comment line which is ignored by the parser – so these lines are optional.
2	ucx, ucy and ucz are the unit cell size in angstroms.
$4-6$	These lines define the shape of the unit cell to be replicated, for cubic cells this is the unit matrix. Note that at present only orthogonal lattice vectors are supported by the code due to complexities relating to parallelisation.
8	Define the number of atoms and number of materials in the unit cell. Materials allow grouping of atoms by material, and have the same parameters (ie moment, damping, etc). Material specification affects the way statistics are collected and displayed, and also allows the simple creation of ordered alloys. The list of atoms must immediately follow this line.
$9-10$	These lines define the atoms in each unit cell and their parameters:
.	atom_id Number identifier of atom in unit cell, starts at 0.
.	cx,cy,cz unit cell coordinates as a fraction of unit cell size
.	mat_id material id of the atom, integer starting at 0
.	cat_id category id of the atom, used for calculating properties not categorised by material, eg height or sublattice. Integer starting at 1.
.	hcat_id Height category id used for calculating properties as a function of height
12	Defines the total number of interactions for the unit cell and the expected type of exchange (isotropic, vectorial, tensorial, normalized-isotropic, normalized-vectorial, normalized-tensorial). No lines are allowed between this line and the list of interactions.
13	These lines list all the interactions.
.	IID Interaction ID is only used for accounting purposes, starts at 0.
.	i Atom number of atom in local unit cell
.	j Atom number of atom in local/remote unit cell
.	dxuc,dyuc,dzuc relative integer coordinates of unit cell for atom j
.	Jij, Jxx... Exchange values specified in Joules.

Example: Simple Cubic System

As an example, here is a complete sample file for a simple cubic system with a single material.

```
1 # Unit cell size:
2 3.54 3.54 3.54
3 # Unit cell vectors:
4 1.0 0.0 0.0
5 0.0 1.0 0.0
6 0.0 0.0 1.0
7 # Atoms num_atoms num_materials; id cx cy cz mat cat hcat
8 1 1
9 00.00.00.0000
10 # Interactions n exctype; id i j dx dy dz Jij
11 6 isotropic
12 0
13 1
14 2
15 3
16 4
17 5
```

Here only easy axis anisotropy and isotropic exchange are defined. Since there is only a single atom in the unit cell, all i-j pairs are 0-0, but over the neighbouring unit cells ± 1 in all directions. This generally leads to a large number of interactions for increasing numbers of atoms in the unit cell, and in future I will write a program to generate some different lattices and interaction lists.

6 Input File Command Reference

The input file can accept a large number of commands, and this chapter gives a comprehensive list of all the options and what they do. Commands are in the form category:keyword=value, where value can be optional depending on the keyword.

System Generation

The following commands control generation of the simulated system, including dimensions, crystal structures etc.

create:full Uses the entire generated system without any truncation or consideration of the create:particle-size parameter. create:full should be used when importing a complete system, such as a complete nanoparticle and where a further definition of the system shape is not required. This is the default if no system truncation is defined.

create:cube Cuts a cuboid particle of size l_x = l_y = l_z = create:particle-size from the defined crystal lattice.

create:cylinder Cuts a cylindrical particle of diameter create:particle-size from the defined crystal lattice. The height of the cylinder extends to the whole extent of the system size create:system-size-z in the z-direction.

create:ellipsoid Cuts an ellipsoid particle of diameter create:particle-size with fractional diameters of dimensions:particle-shape-factor-x,dimensions:particle-shape-factor-y,d from the defined crystal lattice.

create:sphere Cuts a spherical particle of diameter create:particle-size from the defined crystal lattice.

create:truncated-octahedron Cuts a truncated octahedron particle of diameter create:particle-size from the defined crystal lattice.

create:particle Defines the creation of a single particle at the centre of the defined system. If create:particle-size is greater than the system dimensions then the outer boundary of the particle is truncated by the system dimensions.

create:particle-array Defines the creation of a two-dimensional array of particles on a square lattice. The particles are separated by a distance create:particle-spacing. If the system size is insufficient to contain at least a single entire particle of size create:particle-size then no atoms will be generated and the program will terminate with an error.

create:voronoi-film Generates a two-dimensional voronoi structure of particles, with a mean grain size of create:particle-size and variance create:voronoi-size-variance as a fraction of the grain size. If create:voronoi-size-variance=0 then hexagonal shaped grains are generated. The spacing between the grains (defined by the initial voronoi seed points) is controlled by create:particle-spacing. The pseudo-random pattern uses a predefined random seed, and so the generated structure will be the same every time. A different structure can be generated by setting a new random seed using the create:voronoi-random-seed parameter. Depending on the desired edge structure, the first row can be shifted using the create:voronoi-row-offset flag which changes the start point of the voronoi pattern. The create:voronoi-rounded-grains parameter generates a voronoi structure, but then applies a grain rounding algorithm to remove the sharp edges.

create:voronoi-size-variance=[float] Controls the randomness of the voronoi grain structure. The voronoi structure is generated using a hexagonal array of seed points appropriately spaced according to the particle size and particle spacing. The seed points are then displaced in x and y according to a gaussian distribution of width create:voronoi-size-variance times the particle size. The variance must be in the range 0.0-1.0. Typical values for a realistic looking grain structure are less than 0.2, and larger values will generally lead to oblique grain shapes and a large size distribution.

create:voronoi-row-offset flag [default false] Offsets the first row of hexagonal points to generate a different pattern, e.g. 2,3,2 grains instead of 3,2,3 grains.

create:voronoi-random-seed = int Sets a different integer random seed for the voronoi seed point generation, and thus produces a different random grain

structure.

create:voronoi-rounded-grains flag [default false] Controls the rounding of voronoi grains to generate more realistic grain shapes. The algorithm works by expanding a polygon from the centre of the grain, until the total volume bounded by the edges of the grain is some fraction of the total grain area, defined by create:voronoi-rounded-grains-area. This generally leads to the removal of sharp edges.

create:voronoi-rounded-grains-area = float [0.0-1.0, default 0.9] Defines the fractional grain area where the expanding polygon is constrained, in the range 0.0-1.0. Values less than 1.0 will lead to truncation of the voronoi grain shapes, and very small values will generally lead to circular grains. A typical value is 0.9 for reasonable voronoi variance.

create:particle-centre-offset shifts the origin of a particle to the centre of the nearest unit cell.

create:crystal-structure = string [sc,fcc,bcc,hcp,rocksalt; default sc] Defines the default crystal lattice to be generated.

create:single-spin flag Overrides all create options and generates a single isolated spin.

create:periodic-boundaries-x flag creates periodic boundaries along the x-direction.

create:periodic-boundaries-y flag creates periodic boundaries along the y-direction.

create:periodic-boundaries-z flag creates periodic boundaries along the z-direction.

create:select-material-by-height specifies that materials are preferentially assigned by their height specification.

create:select-material-by-geometry specifies that materials are preferentially assigned by their geometric specification (eg in core-shell systems).

create:fill-core-shell-particles

create:interfacial-roughness specifies that a global roughness is applied to the material height specification (eg from a non-flat substrate).

create:material-interfacial-roughness specifies that a material-specific roughness is applied to the material height specification (eg from differences in local deposition rate).

create:interfacial-roughness-random-seed specifies the random seed for generating the roughness pattern, where different numbers generate different random patterns. Number should ideally be large and around 2,000,000,000.

create:interfacial-roughness-number-of-seed-points determines the undulation for the roughness, where more points gives a larger undulation.

create:interfacial-roughness-type determines whether the roughness is applied as peaks or troughs in the material-specific material heights. Valid options are "peaks" or "troughs".

create:interfacial-roughness-seed-radius

create:interfacial-roughness-seed-radius-variance

create:interfacial-roughness-mean-height

create:interfacial-roughness-maximum-height

create:interfacial-roughness-height-field-resolution

create:alloy-random-seed integer [default 683614233] Sets the random seed for the psuedo random number generator for generating random alloys. Simulations use a predictable sequence of psuedo random numbers to give repeatable results for the same simulation. The seed determines the actual sequence of numbers and is used to generate a different alloy distribution. Note that different numbers of cores will change the structure that is generated.

create:grain-random-seed integer [default 683614233] Sets the random seed for the psuedo random number generator for generating random grain structures.

create:dilution-random-seed integer [default 683614233] Sets the random seed for the psuedo random number generator for diluting the atoms, leading to a different realization of a dilute material. Note that different numbers of cores will change the structure that is generated.

create:intermixing-random-seed integer [default 683614233] Sets the random seed for the psuedo random number generator for calculating intermixing of materials. A different seed will lead to a different realization of a dilute material. Note that different numbers of cores will change the structure that is generated.

System dimensions

The commands here determine the dimensions of the generated system.

dimensions:unit-cell-size = float [0.1 Å- 10 μ m, default 3.54 Å] Defines the size of the unit cell.

dimensions:unit-cell-size-x Defines the size of the unit cell if asymmetric.

dimensions:unit-cell-size-x Defines the size of the unit cell if asymmetric.

dimensions:unit-cell-size-z Defines the size of the unit cell if asymmetric.

dimensions:system-size Defines the size of the symmetric bulk crystal.

dimensions:system-size-x Defines the total size if the system along the x-axis.

dimensions:system-size-y Defines the total size if the system along the y-axis.

dimensions:system-size-z Defines the total size if the system along the z-axis.

dimensions:particle-size = float Defines the size of particles cut from the bulk crystal.

dimensions:particle-spacing Defines the spacing between particles in particle arrays or voronoi media.

dimensions:particle-shape-factor-x = float [0.001-1, default 1.0] Modifies the default particle shape to create elongated particles. The selected particle shape is modified by changing the effective particle size in the x direction. This property scales the as a fraction of the particle-size along the x-direction.

dimensions:particle-shape-factor-y = float [0.001-1, default 1.0] Modifies the default particle shape to create elongated particles. The selected particle shape is modified by changing the effective particle size in the y direction. This property scales the as a fraction of the particle-size along the y-direction.

dimensions:particle-shape-factor-z = float [0.001-1, default 1.0] Modifies the default particle shape to create elongated particles. The selected particle shape is modified by changing the effective particle size in the z direction. This property scales the as a fraction of the particle-size along the z-direction.

dimensions:particle-array-offset-x [0-10^4 Å] Translates the 2-D particle array the chosen distance along the x-direction.

dimensions:particle-array-offset-y Translates the 2-D particle array the chosen distance along the y-direction.

dimensions:double macro-cell-size determines the macro cell size for calculation of the demagnetizing field and output of the magnetic configuration. Finer discretisation leads to more accurate results at the cost of significantly longer run times. The cell size should always be less than the system size, as highly asymmetric cells will leads to significant errors in the demagnetisation field calculation.

Anisotropy calculation

The following commands control the calculation of the magnetic anisotropy energy for the system. **anisotropy:surface-anisotropy-threshold** integer default [native] Determines minimal number of neighbours to classify as surface atom. The default value is the number of neighbours specified by the crystal or unit cell file. You can set this as a lower threshold.

anisotropy:surface-anisotropy-nearest-neighbour-range float default [∞] Sets the interaction range for the nearest neighbour list used for the surface anisotropy calculation.

anisotropy:enable-bulk-neel-anisotropy bool default false Enables calculation of the Néel pair anisotropy in the bulk, irrespective of the number of neighbours, enabling the effect of localised spin-orbit interactions. Internally this sets a large threshold, and so specifying anisotropy:surface-anisotropy-threshold will override this flag.

Dipole field calculation

The following commands control the calculation of the dipole-dipole field. By default the dipole fields are disabled for performance reasons, but for large systems (> 10 nm) the interactions can become important. The VAMPIRE code implements several different solvers balancing accuracy and performance. The default in V5+ is the tensor method, which approximates the dipole dipole interactions at the macrocell level but calculating a dipole-dipole tensor which is exact if the magnetic moments in each cell are aligned.

dipole:solver = exclusive string [default tensor] Declares the solver to be used for the dipole calculation. Available options are:

macrocell

tensor

Simulation Control

The following commands control the simulation, including the program, maximum temperatures, applied field strength etc.

sim:integrator = exclusive string [default llg-heun] Declares the integrator to be used for the simulation. Available options are:

llg-heun

monte-carlo

llg-midpoint

constrained-monte-carlo

hybrid-constrained-monte-carlo

sim:program = exclusive string defines the simulation program to be used.

sim:program = benchmark program which integrates the system for 10,000 time steps and exits. Used primarily for quick performance comparisons for different system architectures, processors and during code performance optimisation.

sim:program = time-series program to perform a single time series typically used for switching calculations, ferromagnetic resonance or to find equilibrium magnetic configurations. The system is usually simulated with constant temperature and applied field. The system is first equilibrated for sim:equilibration-time-steps time steps and is then integrated for sim:time-steps time steps.

sim:program = hysteresis-loop program to simulate a dynamic hysteresis loop in user defined field range and precision. The system temperature is fixed and defined by sim:temperature. The system is first equilibrated for sim:equilibration time-steps time steps at sim:maximum-applied-field-strength applied field. For normal loops sim:maximum-applied-field-strength should be a saturating field. After equilibration the system is integrated for sim:loop-time-steps at each field point. The field increments from +sim:maximum-applied-field-strength to =sim:maximum-appli -field-strength in steps of sim:applied-field-increment, and data is output after each field step.

sim:program = static-hysteresis-loop program to perform a hysteresis loop in the same way as a normal hysteresis loop, but instead of a dynamic loop the equilibrium condition is found by minimisation of the torque on the system. For static loops the temperature must be zero otherwise the torque is always finite. At each field increment the system is integrated until either the maximum torque for any one spin is less than the tolerance value (10^{-6} T), or if sim:loop-time-steps is reached. Generally static loops are computationally efficient, and so sim:loop-time-steps can be large, as many integration steps are only required during switching, i.e. near the coercivity.

sim:program = curie-temperature Simulates a temperature loop to determine the Curie temperature of the system. The temperature of the system is increased stepwise, starting at sim:minimum temperature and ending at sim:maximum-

temperature in steps of sim:temperature-increment. At each temperature the system is first equilibrated for sim:equilibration-steps time steps and then a statistical average is taken over sim:loop-time-steps. In general the Monte Carlo integrator is the optimal method for determining the Curie temperature, and typically a few thousand steps is sufficient to equilibrate the system. To determine the Curie temperature it is best to plot the mean magnetization length at each temperature, which can be specified using the output:mean-magnetisation-length keyword. Typically the temperature dependent magnetization can be fitted using the function

$$m(T) = \langle \sqrt{\sum_i \mathbf{S}} \rangle = \left(1 - \frac{T}{T_{\mathrm{C}}}\right)^{\beta} \tag{6.1}$$

where T is the temperature, T_{C} is the Curie temperature, and $\beta \sim 0.34$ is the critical exponent.

sim:program = field-cooling

sim:program = temperature-pulse

sim:program = cmc-anisotropy

sim:enable-dipole-fields flag enables calculation of the demagnetising field.

sim:enable-fmr-field

sim:enable-fast-dipole-fields Bool default false Enables fast calculation of the demag field by pre calculation of the interaction matrix.

sim:dipole-field-update-rate Integer default 1000 Number of timesteps between recalculation of the demag field. Default value is suitable for slow calculations, fast dynamics will generally require much faster update rates.

sim:time-step

sim:total-time-steps

sim:loop-time-steps

sim:time-steps-increment

sim:equilibration-time-steps

sim:simulation-cycles

sim:maximum-temperature

sim:minimum-temperature

sim:equilibration-temperature

sim:temperature

sim:temperature-increment

sim:cooling-time

sim:laser-pulse-temporal-profile square two-temperature double-pulse-two-temperature double-pulse-square

sim:laser-pulse-time

sim:laser-pulse-power

sim:second-laser-pulse-time

sim:second-laser-pulse-power

sim:second-laser-pulse-maximum-temperature

sim:second-laser-pulse-delay-time

sim:two-temperature-heat-sink-coupling

sim:two-temperature-electron-heat-capacity

sim:two-temperature-phonon-heat-capacity

sim:two-temperature-electron-phonon-coupling

sim:cooling-function exponential gaussian double-gaussian linear

sim:applied-field-strength

sim:maximum-applied-field-strength

sim:equilibration-applied-field-strength

sim:applied-field-strength-increment

sim:applied-field-angle-theta

sim:applied-field-angle-phi

sim:applied-field-unit-vector

sim:demagnetisation-factor = float vector [default (000)] vector describing the components of the demagnetising factor from a macroscopic sample. By default this is disabled, and specifying a demagnetisation factor adds an effective field, such that the total field is given by:

$$\mathbf{H}_{\mathrm{tot}} = \mathbf{H}_{\mathrm{ext}} + \mathbf{H}_{\mathrm{int}} - \mathbf{M} \cdot \mathbf{N}_{\mathrm{d}}$$

where $\mathbf{M}$ is the magnetisation of the sample and $\mathbf{N}_{\mathrm{d}}$ is the demagnetisation factor of the macroscopic sample. The components of the demagnetisation factor must sum to 1. In general the demagnetisation factor should be used without the dipolar field, as this results in counting the demagnetising effects twice. However, the possibility of using both is not prevented by the code.

sim:integrator-random-seed Integer [default 12345] Sets a seed for the psuedo random number generator. Simulations use a predictable sequence of psuedo random numbers to give repeatable results for the same simulation. The seed determines the actual sequence of numbers and is used to give a different realisation of the same simulation which is useful for determining statistical properties of the system.

sim:constraint-rotation-update

sim:constraint-angle-theta = float (default 0) When a constrained integrator is used in a normal program, this variable controls the angle of the magnetisation of the. Whole system from the x-axis [degrees]. In constrained simulations (such as c,c anisotropy) this has no effect.

sim:constraint-angle-theta-minimum float (default 0)

sim:constraint-angle-theta-maximum

sim:constraint-angle-theta-increment = float 0.001-360 (default 5) Incremental Change of angle of m from z-direction in constrained simulations. Controls the resolution of

sim:constraint-angle-phi

sim:constraint-angle-phi-minimum

sim:constraint-angle-phi-maximum

sim:constraint-angle-phi-increment

sim:monte-carlo-algorithm

spin-flip

uniform

angle

hinzke-nowak

sim:checkpoint flag [default false] Enables checkpointing of spin configuration at end of simulation sim:save-checkpoint=end sim:save-checkpoint=continuous sim:save-checkpoint-rate=1 sim:load-checkpoint=restart sim:load-checkpoint=continue

sim:preconditioning-steps integer [default 0] defines a number of preconditioning steps to thermalise the spins at sim:equilibration-temperature prior to the main simulation starting. The preconditioner uses a Monte Carlo algorithm

to develop a Boltzmann spin distribution prior to the main program starting. The method works in serial and parallel mode and is especially efficient for materials with low Gilbert damping. The preconditioning steps are applied after loading a checkpoint, allowing you to take a low temperature starting state and thermally equilibrate it.

Data output

The following commands control what data is output to the output file. The order in which they appear is the order in which they appear in the output file. Most options output a single column of data, but some output multiple columns, particularly vector data or parameters related to materials, where one column per material is output.

output:time-steps outputs the number of time steps (or Monte Carlo steps) completed during the simulation so far.

output:real-time outputs the simulation time in seconds. The real time is given by the number of time steps multiplied by sim:time-step (default value is 1.0×10^{-15} s. The real time has no meaning for Monte Carlo simulations.

output:temperature outputs the instantaneous system temperature in Kelvin.

output:applied-field-strength outputs the strength of the applied field in Tesla. For hysteresis simulations the sign of the applied field strength changes along a fixed axis and is represented in the output by a similar change in sign.

output:applied-field-unit-vector outputs a unit vector in three columns $\hat{h_x}$, $\hat{h_y}$, $\hat{h_z}$ indicating the direction of the external applied field.

output:applied-field-alignment outputs the dot product of the net magnetization direction of the system with the external applied field direction $\hat{\mathbf{m}} \cdot \hat{\mathbf{H}}$.

output:material-applied-field-alignment outputs the dot product of the net magnetization direction of each material defined in the material file with the external applied field direction $\left[\hat{\mathbf{m}}_1 \cdot \hat{\mathbf{H}}\right]$, $\left[\hat{\mathbf{m}}_2 \cdot \hat{\mathbf{H}}\right]$...$\left[\hat{\mathbf{m}}_n \cdot \hat{\mathbf{H}}\right]$.

output:magnetisation outputs the instantaneous magnetization of the system. The data is output in four columns $\hat{m_x}$, $\hat{m_y}$, $\hat{m_z}$, $|m|$ giving the unit vector direction of the magnetization and normalized length of the magnetization respectively. The normalized length of the magnetization $|m| = |\sum_i \mu_i S_i| / \sum \mu_i$ is given by the sum of all moments in the system assuming ferromagnetic alignment of all spins. Note that the localized spin moments μ_i are taken into account in the summation.

output:magnetisation-length outputs the instantaneous normalized magnetization length $|m| = |\sum_i \mu_i S_i| / \sum \mu_i$, where the saturation value is defined by ferromagnetic alignment of all spins in the system. Note that the localized spin moments μ_i are taken into account in the summation.

output:mean-magnetisation-length outputs the time-averaged normalized magnetization length $\langle |m| \rangle$.

output:mean-magnetisation outputs the time-averaged normalized magnetization vector $\langle |\mathbf{m}| \rangle$.

output:material-magnetisation outputs the instantaneous normalized magnetization for each material in the simulation. The data is output in blocks of four columns, with one block per material defined in the material file, e.g.

$$\left[\hat{m_1^x},\hat{m_1^y},\hat{m_1^z},|m_1|\right],\ \left[\hat{m_2^x},\hat{m_2^y},\hat{m_2^z},|m_2|\right]\ \ldots\ \left[\hat{m_n^x},\hat{m_n^y},\hat{m_n^z},|m_n|\right]$$

Note that obtaining the actual macroscopic magnetization length from this data is not trivial, since it is necessary to know how many atoms of each material are in the system. This information is contained within the log file (giving the fraction of atoms which make up each material). However it is usual to also output the total normalized magnetization of the system to give the relative ordering of the entire system.

output:material-mean-magnetisation-length outputs the time-averaged normalized magnetization length for each material, e.g. $\langle |m_1| \rangle$, $\langle |m_2| \rangle$...$\langle |m_n| \rangle$.

output:material-mean-magnetisation outputs the time-averaged normalized magnetization length for each material, e.g. $\langle |\mathbf{m}_1| \rangle$, $\langle |\mathbf{m}_2| \rangle$...$\langle |\mathbf{m}_n| \rangle$.

output:total-torque outputs the instantaneous components of the torque on the system $\tau = \sum_i \mu_i \mathbf{S}_i \times \mathbf{H}_i$ in three columns τ_x, τ_y, τ_z (units of Joules).

In equilibrium the total torque will be close to zero, but is useful for testing convergence to an equilibrium state for zero temperature simulations.

output:mean-total-torque outputs the time average of components of the torque on the system $\langle \tau \rangle = \langle \sum_i \mu_i \mathbf{S}_i \times \mathbf{H}_i \rangle$ in three columns $\langle \tau_x \rangle$, $\langle \tau_y \rangle$, $\langle \tau_z \rangle$. In equilibrium the total torque will be close to zero, but the average torque is useful for extracting effective anisotropies or exchange using constrained Monte Carlo simulations.

output:constraint-phi outputs the current angle of constraint from the z-axis for constrained simulations using either the Lagrangian Multiplier Method (LMM) or Constrained Monte Carlo (CMC) integration methods.

output:constraint-theta outputs the current angle of constraint from the x-axis for constrained simulations using either the Lagrangian Multiplier Method (LMM) or Constrained Monte Carlo (CMC) integration methods.

output:material-mean-torque outputs the time average of components of the torque on the each material system $\langle \tau \rangle$ in blocks of three columns, with one block for each material defined in the material file e.g.

$$\left[\langle \tau_1^x \rangle, \langle \tau_1^y \rangle, \langle \tau_1^z \rangle\right], \left[\langle \tau_2^x \rangle, \langle \tau_2^y \rangle, \langle \tau_2^z \rangle\right] \ldots \left[\langle \tau_n^x \rangle, \langle \tau_n^y \rangle, \langle \tau_n^z \rangle\right]$$

Computing the torque on each material is particularly useful for determining equilibrium properties of multi-component systems with constrained Monte Carlo simulations. In certain cases the components of a system (different materials) can exert equal and opposite torques on each other, giving a total system torque of zero. The decomposition of the torques for each material allows the determination of internal torques in the system.

output:mean-susceptibility outputs the components of the magnetic susceptibility χ. The magnetic susceptibility is defined by

$$\chi_\alpha = \frac{\sum_i \mu_i}{k_\mathrm{B} T} \left(\langle m_\alpha^2 \rangle - \langle m_\alpha \rangle^2 \right)$$

where $\alpha = x, y, z, m$ giving the directional components of the magnetization in x, y and z respectively as well as the longitudinal susceptibility χ_m. The data is output in four columns χ_x, χ_y, χ_z, and χ_m. The susceptibility is useful for identifying the critical temperature for a system as well as atomistic parametrization of the

micromagnetic Landau-Lifshitz-Bloch (LLB) equation.

output:material-mean-susceptibility outputs the components of the magnetic susceptibility χ for each defined material in the system. The data is output in sets of four columns χ_x, χ_y, χ_z, and χ_m for each material. In multi-sublattice systems the susceptibility of each sublattice can be different.

output:electron-temperature outputs the instantaneous electron temperature as calculated from the two temperature model.

output:phonon-temperature outputs the instantaneous phonon (lattice) temperature as calculated from the two temperature model.

output:total-energy

output:mean-total-energy

output:anisotropy-energy

output:mean-anisotropy-energy

output:exchange-energy

output:mean-exchange-energy

output:applied-field-energy

output:mean-applied-field-energy

output:magnetostatic-energy

output:mean-magnetostatic-energy

output:mpi-timings

output:gnuplot-array-format

output:output-rate = integer [default 1] controls the number of data points

written to the output file or printed to screen. By default VAMPIRE calculates statistics once every sim:time-steps-increment number of time steps. Usually you want to output the updated statistic (e.g. magnetization) every time, which is the default behaviour. However, sometimes you may want to plot the time evolution of an average, where you want to collect statistics much more frequently than you output to the output file, which is controlled by this keyword. For example, if output:output-rate = 10 and sim:time-steps-increment = 10 then statistics (and average values) will be updated once every 10 time steps, and the new statistics will be written to the output file every 100 time steps.

output:precision = integer [default 6] controls the number of digits to be used for data written to the output file or printed to screen. The default value is 6 digits of precision.

output:fixed-width = flag [default false] controls the formatting to be used for data written to the output file or printed to screen. The default is false which ignores trailing zeros in the output.

Configuration output

These options enable the output of spin configuration snapshots during the simulation. The configurations can then be visualised using povray or other software generated with the vampire data converter (vdc) utility.

config:atoms flag enables the output of atomic spin configurations.

config:atoms-output-rate = int [0+, default 1000] determines the rate configuration files are outputted as a multiple of sim:time-steps-increment.

The following options allow a cubic slice of the total configuration data to be output to the configuration file. This is useful for reducing disk usage and processing times, especially for large sale simulations.

config:atoms-minimum-x = float [0.0 - 1.0] determines the minimum x value (as a fraction of the total system dimensions) of the data slice to be outputted to the configuration file.

config:atoms-minimum-y determines the minimum y value (as a fraction of the total system dimensions) of the data slice to be outputted to the configuration file.

config:atoms-minimum-z determines the minimum z value (as a fraction of the total system dimensions) of the data slice to be outputted to the configuration file.

config:atoms-maximum-x determines the maximum x value (as a fraction of the total system dimensions) of the data slice to be outputted to the configuration file.

config:atoms-maximum-y determines the maximum y value (as a fraction of the total system dimensions) of the data slice to be outputted to the configuration file.

config:atoms-maximum-z determines the maximum z value (as a fraction of the total system dimensions) of the data slice to be outputted to the configuration file.

config:macro-cells enables the output of macro cell spin configurations.

config:macro-cells-output-rate
determines the rate configuration files are outputted as a multiple of sim:time-steps-increment.

config:output-format = exclusive string [default text] specifies the format of the configuration data. Available options are:

text

binary

The text option outputs data files as plain text, allowing them to be read by a wide range of applications and hence the highest portability. There is a performance cost to using text mode and so this is recommended only if you need portable data and will not be using the vampire data converter (vdc) utility. The binary option outputs the data in binary format and is typically 100 times faster than text mode. This is important for large-scale simulations on large numbers of processors where the data output can take a significant amount of time. Binary files are generally not compatible between operating systems and so the vdc tools generally needs to be run on the same system which generated the files.

config:output-mode = exclusive string [default file-per-node] specifies how configuration data is outputted to disk. Available options are:

file-per-node

file-per-process

mpi-io

Using this option is important for obtaining good performance on Tier-0 (European) and Tier-1 (National) supercomputers for simulations typically using more than 1000 cores. Large scale supercomputers have high performance parallel file systems with a peak bandwidth typically over 10 GB/s. VAMPIRE supports three different modes of data output: file-per-node (FPN), file-per-process (FPP) and mpi-io. Note that high performance requires config:output-format = binary to be set to output the data in binary format, but this is not default behaviour for easier data analysis and portability for the casual user.

The first (default) option of file-per-node collates data from different processes onto a defined number of config:output-nodes before outputting to disk, with the total data spread out with a different file per output node. This has good performance for medium-scale simulations and above (>100 cores) with a reasonable number of output nodes (typically 1 per physical node). Small simulations (<100 cores) benefit from a larger number of output processes. to maximise bandwidth fro the independent write operations. For typical simulations with > 40,000 atoms per core striping of the parallel file system improves performance, while for less atoms striping can be detrimental and should be disabled. This option is also best for distributed file systems, typical on local resources such as university clusters.

The file-per-process option means every process in the simulation outputs its own data to disk independent of all others. The option is available for advanced tuning but is generally not recommended for typical simulations due to the large number of small files generated, complicating data analysis and generally having very poor performance.

The mpi-io option uses the MPI library routines to output a large, single shared file. Enabling this option automatically forces binary data output. In general this option gives good performance for large systems with a single file for each configuration snapshot. In general this gives worse performance than the file-per-node option except for the largest system sizes.

config:output-nodes = int [default 1] Specifies the number of files to be generated per snapshot. For typical small scale simulations (on a single physical node) the default value of 1 is fine. For larger scale simulations more output nodes are beneficial to achieve maximum performance, with one output node per physical node being a sensible choice, but this can be specified up to the

maximum number of processes in the simulation.

7 Material File Command Reference

The material file defines all the magnetic properties of the materials used in the simulation, including exchange, anisotropy, damping etc. Material properties are defined by an index number for each material, starting at one. Material properties are then defined as follows:

material[index]:keyword = value !unit

followed by a carriage return, so that each property is defined on a separate line. The defined keywords are listed below. The material file is largely free-format, apart from the first line which must specify the number of materials for the simulation. The material properties can be defined in any order, and if omitted the default value will be used. When the same property for a particular material is defined in the file, the last definition (reading top to bottom) will be used. Comments can be added to the file using the # character, which moves the file parser to the next line.

material:num-materials = int [1-100; default 1] defines the number of materials to be used in the simulation, and must be the first uncommented line in the file. If more than n materials are defined, then only the first n materials are actually used. The maximum number of different materials is currently limited to 100. If using a custom unit cell then the number of materials in the unit cell cell must match the number of materials here, otherwise the code will produce an error.

material:material-name = string [default material#n] defines an identifying name for the material with a maximum length of xx characters. The identifying name is only used in the output files and does not affect the running of the code.

material:damping-constant = float [0.0-10.0; default 1.0] defines the phenomenological relaxation rate (damping) in dynamic simulations using the LLG equation. For equilibrium properties the damping should be set to 1 (critical

damping), while for realistic dynamics the damping should be representative of the material. Typical values range from 0.005 to 0.1 for most materials.

material:exchange-matrix[index] = float [default 0.0 J/link] Defines the pairwise exchange energy between atoms of type index and neighbour-index. The pair wise exchange energy is independent of the coordination number, and so the total exchange integral will depend on the number of nearest neighbours for the crystal lattice. The exchange energy must be defined between all material pairs in the simulation, with positive values representing ferromagnetic coupling, and negative values representing anti ferromagnetic coupling. For a ferromagnet with nearest neighbour exchange, the pairwise exchange energy can be found from the Curie temperature by the meanfield expression:

$$J_{ij} = \frac{3k_{\mathrm{B}}T_{\mathrm{C}}}{\epsilon z}$$

where J_{ij} is the exchange energy, k_{B} is the Boltzmann constant, T_{C} is the Curie temperature, z is the coordination number (number of nearest neighbours) and ϵ is a correction factor to account for spin wave fluctuations in different crystal lattices. If a custom unit cell file is used the exchange values defined here are ignored.

material:atomic-spin-moment = float [0.01+ μ_{B}, default 1.72 μ_{B}] Defines the local effective spin moment for each atomic site. Atomic moments can be found from ab-initio calculations or derived from low temperature measurements of the saturation magnetisation. The atomic spin moments are related to the macroscopic magnetisation by the expression:

$$\mu_{\mathrm{S}} = \frac{M_{\mathrm{S}}a^3}{n}$$

where a is the lattice constant, n is the number of atoms per unit cell, and M_{S} is the saturation magnetisation in units of J/T/m^3 (A/m). Note that unlike micromagnetic simulations, atomistic simulations always use zero-K values of the spin moments, since thermal fluctuations of the magnetisation are provided by the model. Small values ($< 1\mu_{\mathrm{B}}$) will typically lead to integration problems for the LLG unless sub-femtosecond time steps are used.

material:uniaxial-anisotropy-constant = float [default 0.0 J/atom] Defines the local second order single-ion magnetocrystalline anisotropy constant at each atomic site. The anisotropy energy is given by the expression

$$E_i = -k_2(\mathbf{S}_i \cdot \mathbf{e}_i)^2$$

where $\mathbf{S}_i$ is the local spin direction and $\mathbf{e}_i$ is the easy axis unit vector. Positive values of k_2 give a preferred easy axis orientation, and negative v alues give a preferred easy plane orientation of the spin perpendicular to the easy axis direction.

material:second-order-uniaxial-anisotropy-constant = float [default 0.0 J/atom] Has the same meaning and is the preferred form for material:uniaxial-anisotropy-constant.

material:fourth-order-uniaxial-anisotropy-constant = float [default 0.0 J/atom] implements fourth order uniaxial anisotropy as implemented with spherical harmonics.

material:cubic-anisotropy-constant = float [default 0.0 J/atom] Defines the local cubic magnetocrystalline anisotropy constant at each atomic site. The anisotropy energy is given by the expression

$$E_i = -\frac{k_c}{2}(S_x^4 + S_y^4 + S_z^4)$$

where $S_{x,y,z}$ are the components of the local spin direction and k_c is the cubic anisotropy constant. Positive values of k_c give a preferred easy axis orientation along the [001] directions, medium-hard along the [110] directions and hard along the [111] directions. Negative values give a preferred easy direction along the [111] directions, medium ahead along the [110] directions and hard along the [100] directions.

material:fourth-order-cubic-anisotropy-constant = float [default 0.0 J/atom] Has the same meaning and preferred form for material:cubic-anisotropy-constant.

material:uniaxial-anisotropy-direction = float vector [default (0,0,1)] A unit vector $\mathbf{e}_i$ describing the magnetic easy axis direction for uniaxial anisotropy. The vector is entered in comma delimited form - For example:

material[1]:uniaxial-anisotropy-direction = 0,0,1

The unit vector is self normalising and so the direction can be expressed in standard form (with length $r = 1$) or in terms of crystallographic directions, e.g. [111].

Random anisotropy can be specified using

material[1]:uniaxial-anisotropy-direction = random

which allocates each atom in the material a random anisotropy vector in three dimensions. This may applicable to truly amorphous Rare earth alloys where no local anisotropy direction is preferred.

Random anisotropy for each grain in the simulation (specified by generating a voronoi structure or particle array) can be specified using

material[1]:uniaxial-anisotropy-direction = random-grain

which allocates each atom in the material a random anisotropy vector in three dimensions, giving all atoms of the material in each grain a different anisotropy vector. Both random anisotropies are compatible with higher order uniaxial anisotropy but not lattice anisotropy or cubic anisotropy.

material:surface-anisotropy-constant = float default 0.0 (J/atom) Describes the surface anisotropy constant in the Néel pair anisotropy model. The anisotropy is given by a summation over nearest neighbour atoms given by

$$E_i = +\frac{1}{2}\sum_j^{\mathrm{nn}} k_{\mathrm{s}}(\mathbf{S}\cdot\mathbf{r}_{ij})^2$$

where k_{s} is the surface anisotropy constant between atoms i and j and $\mathbf{r}_{ij}$ is a unit vector between sites i and j.

neel-anisotropy-constant[index] = float [default 0.0 J] Has the same meaning and is the preferred form for material:surface-anisotropy-constant.

lattice-anisotropy-constant = float [default 0.0 J/atom] Defines anisotropy arising from temperature dependent lattice expansion such as in RETM alloys. The temperature dependence of the lattice anisotropy is defined with a user defined function specified in the parameter lattice-anisotropy-file.

lattice-anisotropy-file = string Defines a file containing the temperature dependence of anisotropy arising from lattice expansion. The first line of the file specifies the number of points, followed by a list of pairs indicating the temperature (in K) and normalised lattice anisotropy, typically of order 1 at $T = 0K$. The specified points are linearly interpolated by the code and so excessively high resolution is not required for high accuracy, and 1 K resolution is typically

sufficient for most problems.

material:relative-gamma float [default 1] defines the gyromagnetic ratio of the material relative to that of the electron $\gamma_e = 1.76\ \mathrm{T}^{-1}\mathrm{s}^{-1}$. Valid values are in the range 0.01 - 100.0. For most materials $\gamma_r = 1$.

material:initial-spin-direction float vector /bool [default (001) / false] determines the initial direction of the spins in the material. Value can wither be a unit vector defining a direction in space, or a boolean which initialises each spin to a different random direction (equivalent to infinite temperature). As with other unit vectors, a normalised value or crystallographic notation (e.g. [110]) may be used.

material:material-element string [default "Fe"] defines a purely descriptive chemical element for the material, which gives visual contrast in a range of interactive atomic structure viewers such as jmol, rasmol etc. In rasmol, Fe is a gold colour, H is white, Li is a deep red, O is red, B is green and Ag is a medium grey. This parameter has no relevance to the simulation at all, and only appears when outputting atomic coordinates, which can be post-processed to be viewable in rasmol. The contrast is particularly useful in inspecting the generated structures, particularly ones with a high degree of complexity.

material:geometry-file string [default ""] specifies a filename containing a series of connected points in space which is used to cut a specified shape from the material, in a process similar to lithography. The first line defines the total number of points, which must be in the range 3-100 (A minimum of three points is required to define a polygon). The points are normalised to the sample size, and so all points are defined as x, y pairs in the range 0-1, with one point per line. The last point is automatically connected first, so need not be defined twice.

material:alloy-host flag [default off] is a keyword which, if specified, scans over all other materials to replace the desired fraction of host atoms with alloy atoms. This is primarily used to create random alloys of materials with different properties (such as FeCo, NiFe) or disordered ferrimagnets (such as GdFeCo).

material:alloy-fraction[index] = float [0-1 : default 0.0] defines the fractional number of atoms of the host material to be replaced by atoms of material $index$.

material:minimum-height = float [0-1 : default 0.0] defines the minimum

height of the material as a fraction of the total height z of the system. By defining different minimum and maximum heights it is easy to define a multilayer system consisting of different materials, such as FM/AFM, or ECC recording media.

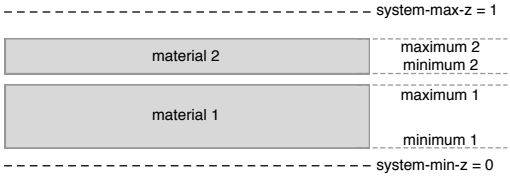


Figure 7.1: Schematic diagram showing definition of a multilayer system consisting of two materials. The minimum-height and maximum-height are defined as a fraction of the total z-height of the system.

The heights of the material are applied when the crystal is generated, and so in general further geometry changes can also be applied, for example cutting a cylinder shape or voronoi granular media, while preserving the multilayer structure. The code will also print a warning if materials overlap their minimum/maximum ranges, since such behaviour is usually (but not always) undesirable.

material:maximum-height = float [0-1 : default 1.0] defines the maximum height of the material as a fraction of the total height z of the system. See material:minimum-height for more details.

material:core-shell-size = float [0-1 : default 1.0] defines the radial extent of a material as a fraction of the particle radius. This parameter is used to generate core-shell nanoparticles consisting of two or more distinct layers. The core-shell-size is compatible with spherical, ellipsoidal, cylindrical, truncated octahedral and cuboid shaped particles. In addition when particle arrays are generated all particles are also core-shell type. This option is also comparable with the minimum/maximum-height options, allowing for partially filled or coated nanoparticles.

material:interface-roughness = float [0-1 : default 1.0] defines interfacial roughness in multilayer systems.

material:intermixing[index] = float [0-1 : default 1.0] defines intermixing between adjacent materials in multilayer systems. The intermixing is defined as

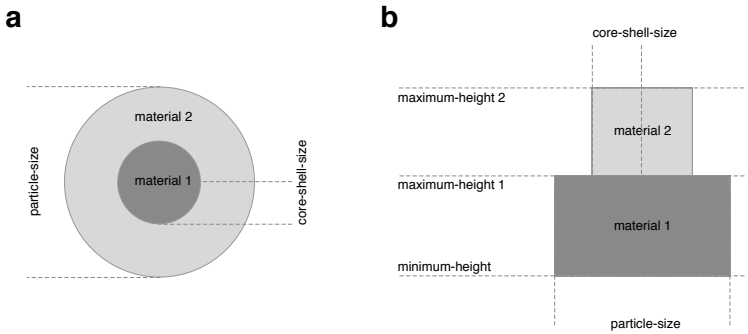


Figure 7.2: (a) Schematic diagram showing definition of a nanoparticle with two materials with different radii. core-shell-size is defined as a fraction of the particle radius (particle-size/2). (b) Schematic diagram showing side-on iew of a cylinder, consisting of two materials with different core-shell-size and different maximum heights. Part of the core material is exposed, while the other part is covered with the other material.

a fraction of the total system height, and so small values are usually used. The intermixing defines the mixing of material $index$ into the host material, and can be asymmetric (a -> b != b -> a).

material:density = float [0-1 : default 1.0] defines the fraction of atoms to remove randomly from the material (density).

material:continuous = flag [default off] is a keyword which defines materials which ignore granular CSG operations, such as particles, voronoi media and particle arrays.

material:fill-space = flag [default off] is a keyword which defines materials which obey granular CSG operations, such as particles, voronoi media and particle arrays, but in-fill the void created. This is useful for embedded nanoparticles and recording media with dilute interlayer coupling.

material:couple-to-phononic-temperature = flag [default off] couples the spin system of the material to the phonon temperature instead of the electron temperature in pulsed heating simulations utilising the two temperature model. Typically used for rare-earth elements.

material:temperature-rescaling-exponent = float [0-10 : default 1.0] defines the exponent when rescaled temperature calculations are used. The higher the exponent the flatter the magnetisation is at low temperature. This parameter must be used with temperature-rescaling-curie-temperature to have any effect.

material:temperature-rescaling-curie-temperature = float [0-10,000 : default 0.0] defines the Curie temperature of the material to which temperature rescaling is applied.

material:non-magnetic flag [default remove] defines atoms of that material as being non-magnetic. Non-magnetic atoms by default are removed from the simulation and play no role in the simulation. If configuration output is specified then the positions of the non-magnetic atoms are saved and processed by the vdc utility. This preserves the existence of non-magnetic atoms when generating visualisations but without needing to simulate them artificially. The "keep" option preserves the non-magnetic atoms in the simulation for parallelization efficiency but instructs the dipole field solver to ignore them for improved accuracy.

Example material files

Bibliography