# MERRILL

# Micromagnetic Earth Related Robust Interpreted Language Laboratory

# v2.0.1

Version date: May 2025

Documentation: June 21, 2025

# Contents

# 1 MERRILL Overview

MERRILL is the acronym for the highly contrived title 'Micromagnetic Earth Related Rapid Interpreted Language Laboratory', chosen to salute the piloting work of Ronald T. Merrill in applying micromagnetic modeling to rock magnetism `https://honors.agu.org/winners/ronald-t-merrill/`.

MERRILL is a finite-element three-dimensional micromagnetic modeling software tailored for rock magnetic applications Conbhuí et al. (2018). This is expressed in several defining features:

1. Ease of use for non-specialists in micromagnetism
2. Modeling of complex three-dimensional geometries
3. Handling several meshes for the same geometry to enable stepwise structure refinement, for example in very large magnetic particles
4. Focus on fast local energy minimization (Hubert-Minimizer) to explore magnetization structures
5. Calculation of energy barriers using a refined nudged-elastic-band (NEB) method Fabian and Shcherbakov (2018)
6. Scripting language that simplifies high-level magnetization curve calculations
7. Predefined material constants for many natural magnetic minerals

A number of examples and tutorials for MERRILL can be found at `https://blogs.ed.ac.uk/rockmag/`

## 1.1 New in Merrill 2.0.0

### 1.1.1 H2 library for demagnetizing energy

The calculation of the demagnetizing energy and its gradient in MERRILL 1 is based on a boundary element method (BEM), where size and computation time scales with $K^2$, where $K$ is the number of boundary elements in the finite element tetrahedral mesh.

The H2Lib is an open source software library for hierarchical matrices and $H^2$-matrices that is being developed in the Scientific Computing Group at Kiel University.
`http://www.h2lib.org/`

Compression of the BEM boundary matrix in micromagnetic modelling using H2Lib has been pioneered by the *tetmag* code of R. Hertel `https://github.com/R-Hertel/tetmag`.

Since version 2.0.0, Merrill depends on H2Lib and works with a compressed H2matrix representation of the boundary element matrix, reducing memory footprint from $K^2$ in number of surface elements to $K\log(K)$ during initialization and $K$ during use. This enable simulation of larger grain sizes.

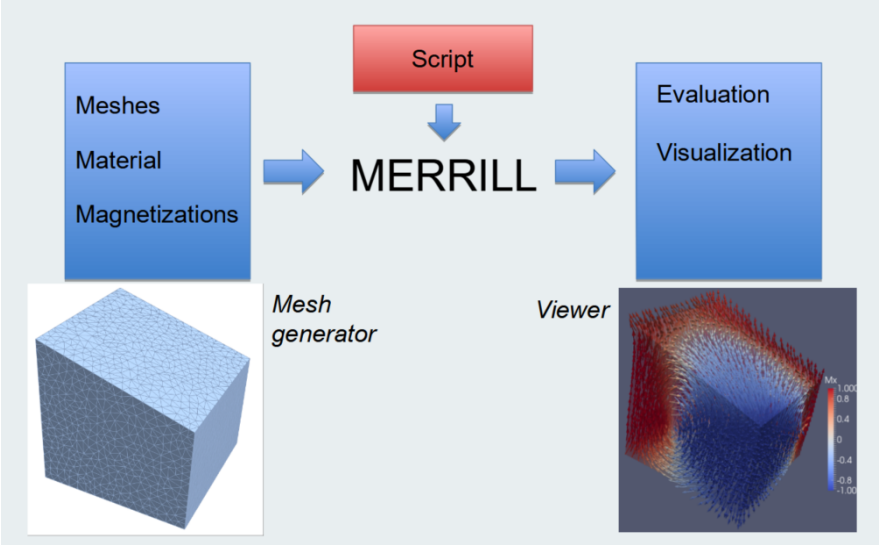


Figure 1: Structure of MERRILL.

## 1.2 General structure of MERRILL

MERRILL requires as input a three-dimensional volume mesh consisting exclusively of tetrahedral cells. This input meshes must be provided by the user and can be generated in several ways:

1. Professional 3D-mesh generators like TecPlot, SolidWorks, AutoCAD. These are expensive, but sometimes available at universities. Open-source tools like gmsh, CGal, or tetgen are extremely good, but more difficult to learn and use.

2. CAD programs can be used to generate three-dimensional shapes. Open-source codes are for example OpenSCAD or FreeCAD or Blender. These programs provide computational geometry engines and can easily produce spheres, cylinders, cuboids and many complex derived shapes. Sometimes they only export to STL files for 3D printers. Then these STL surface meshes need to be converted to volume meshes.

3. Iso2Mesh is a MATLAB based software library that uses tetgen and CGal to easily transform STL files, or image slices (e.g. from FIB-nt) to tetrahedral meshes. Iso2Mesh also works with the open-source MATLAB clone Octave. We provide a simple program merrillsave.m to export iso2mesh results in a MERRILL-readable file format. An overview of how to use Iso2Mesh can be found at `https://nanopaleomag.esc.cam.ac.uk/mesh-generation-iso2mesh/`

4. Download existing particle meshes for MERRILL from the example website: ??

Besides the input meshes, MERRILL requires the intrinsic material parameters (E.g. $M_s$, $K_1$,$A_{\mathrm{ex}}$) and external forces (e.g. stress fields, external magnetic fields) defining the micromagnetic energy terms. It also requires an initial magnetization structure, which either can be defined internally (e.g. single-domain, vortex, random), or read from an input file.

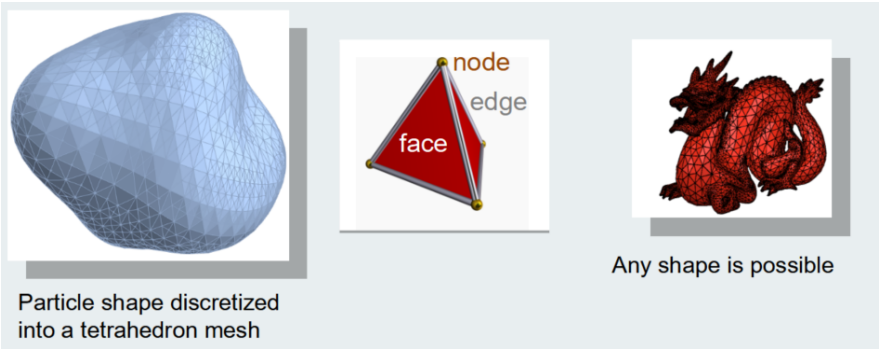


Figure 2: In MERRILL a magnetic particle is represented by a tight mesh of tetrahedra. The magnetization structure is defined by two angles $\phi, \theta$ for each node.

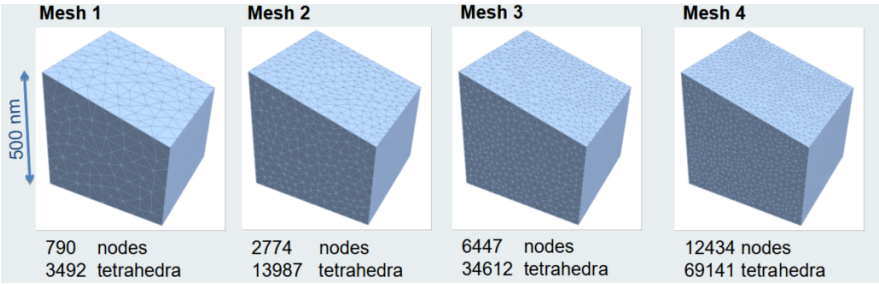


Figure 3: MERRILL can work with several meshes for the same particle geometry.

MERRILL expects a script file that contains the description of the inputs, the required minimization steps, and the output formats of the results.

The output of MERRILL is finally analyzed by external programs. A good open source 3D visualization program is for example ParaView.

## 1.3 Micromagnetic energy calculation and minimization

MERRILL is a numerical program to minimize the micromagnetic energy functional. It is based on the continuum theory of micromagnetics **?**

## 1.4 Energy barriers

## 1.5 Limits of MERRILL

The mesh related discretization error can be estimated by comparing results on two separate meshes with similar number of nodes. To test whether a minimum is really achieved one can monitor largest angular variation along the minimization route. Close to a supposed minimum one can perturb the current state $m_0$ into a state $m_1$ which is a distance d away from $m_0$. If after k minimization steps the result m2 is within ||m2 - m0|| < d/k then $m_0$ is assumed to represent an LEM.

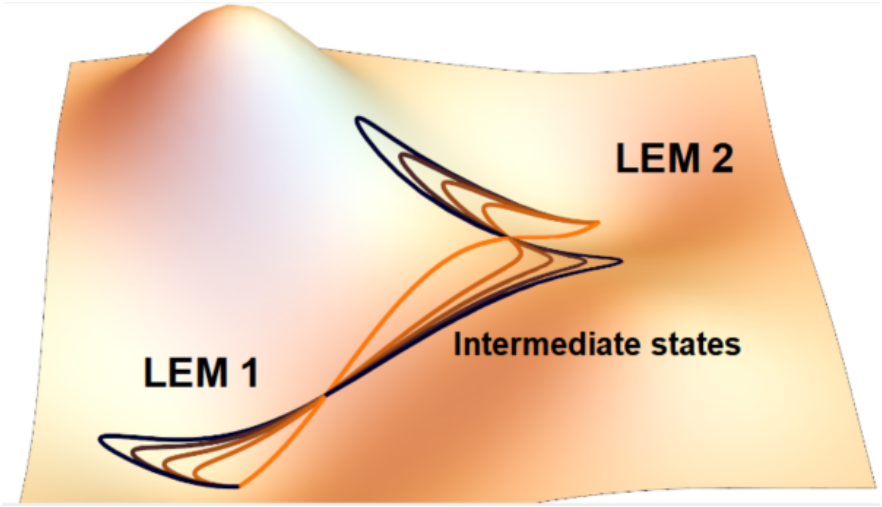


Figure 4: MERRILL can calculate energy barriers between LEM states.

Energy minimization is limited by physical, numerical and algorithmic constraints. **Numerical constraints** The different minimization algorithms have a limited precision. Minimization below this precision can lead to unwanted repetitions due to numerical noise. This can be related to the conjugate gradient method for the sparse matrices or to limited precision in internal functions, e.g. trigonometric functions used for transforming polar to cartesian coordinates.

**Algorithmic constraints** The finite mesh size or finite distances introduce grid errors and artifacts. The weak FEM solutions are elements of a finite dimensional space approximating a real physical solution. Minimizing below the corresponding approximation error is physically meaningless, even if mathematically correct.

**Physical constraints** The micromagnetic model only represents a part of the physically relevant energies. Some energy terms are neglected, like magnetostriction, elastic, and electric energies. Most notably thermal activation is disregarded. Minimization of the energy to higher precision than the real variability of the energy is physically irrelevant.

Sometimes it may be mathematically useful to perform higher precision minimizations. This could be necessary to find a complex minimization route that finally leads to a also physically better minimum.

To benchmark different algorithms it also can be useful to compare their output to unphysical degrees of precision.

In most cases it is useful to have a constantly updated estimate of the different accuracy requests to avoid unnecessary minimization steps. The thermal energy per degree of freedom is 1/2kT . In the FEM model the number of degrees of freedom (DOF) is 2 × NNODE where NNODE is the number of nodes in the mesh, because each unit vector has two DOF. The total variability in energy density ΔE is then

$$\Delta E = \frac{kT\ NNODE}{V_{tot}}$$

## 1.6 Obtaining

Download newest binary from `https://blogs.ed.ac.uk/rockmag/` or `https://bitbucket.org/wynwilliams/merrill/downloads/`.

Since version 2.0.0, Linux and MacOs executables are available. If working on Windows, run the Linux executable in WSL.

## 1.7 Using

Write/obtain a script file as detailed below.

Generate/obtain a mesh file, if used.

### 1.7.1 Linux

From bash shell, call the executable with the script file as first argument.

```
./merrill example_script.txt
```

### 1.7.2 Mac OS

### 1.7.3 Windows

# 2 MERRILL Scripts

## 2.1 Introduction

A MERRILL script is a simple ASCII file containing a sequence of lines. Empty lines, leading, trailing, and multiple spaces or tabs are ignored, as well as anything behind an exclamation mark (!). A number of keywords are used to call subroutines or perform simple assignments. All keywords are case insensitive, e.g. 'ReadMesh' and 'readmesh' are equivalent. The script file is parsed line by line. Each valid line is immediately evaluated and executed. Here is a simple example

```
! Define a new variable with name 'size' and set it to 350
define size 350
! Create a new directory with name: Trap300-Anis350
!   $size$ is replaced with the string '350'
SystemCommand mkdir Trap300-Anis$size$

! Define material constants for magnetite at 20C
Material: Type=Magnetite, TempC=20
! Set anisotropy cubic anisotropy
anisotropy: type=cubic

! Set the exteranl field conditions
external field: type=direction, vector=1;0;0
external field: type= strength, intensity=100, units=mT

set: MaxMeshNumber=2
set: MaxEnergyEvaluations = 5000

! Read two different meshes for the same particle
readmesh: mesh=1, filename=Trapezoid-500-5nm-01.pat
readmesh: mesh=2, filename=Trapezoid-500-3nm-01.pat

! First load the coarser mesh
LoadMesh 1
!Read the result of a previous calculation on this mesh
ReadMagnetization  Trap300-hys300/hys300-f65-c173.dat
! Interpolate it on mesh 2
remesh 2

! change the size of the mesh from 500 to 350 , mean scaling by factor 7/10
resize 500 #size
! Define the log file for the energy
EnergyLog Trap300-Anis$size$/anis$size$

! Disturb each magnetization vector by at most 5 degrees
randomize magnetization 5
! Run the minimization to find a local energy minimum from the start configuration
Minimize
CloseLog
WriteMagnetization Trap300-Anis$size$/anis$size$
end
```

New or modified functions are highlighted in red.

## 2.2 List of all MERRILL Script Commands

### 2.2.1 Setting Variables

- **Set:** <**variable**> = <**value**> is used to define global variables for the material, the geometry of the mesh, or program parameters. The supported variables are listed in Appendix B.
- **Define:** <**variable**> = <**value** > Defines a numeric variable that can be used like a loop variable.
- **AddTo:** <**variable**> + <**value** > Adds a number to a previously defined variable. To subtract simple add a negative number, i.e. Addto: var1 + -9.5
- **Undefine:** <**variable**> Forgets the previously defined variable.

### 2.2.2 General commands

- **SystemCommand:** <**command**>**...** Performs the system command in the remaining arguments as a line. No guarantee can be given for correct behavior. Uses FORTRAN's SYSTEM command.
- **KeyPause** Pauses script evaluation and waits for Key+Enter for continuation.
- **Stop** or **End** Stops script evaluation.
- **Loop: Name** = <**variable** >, **Start** = <**startvalue** >, **End** = <**endvalue** >, [**Step** = <**step**>] The following commands are advanced options for scripts using loops or variables. The loop command takes all commands until the next EndLoop statement and performs a loop over the enclosed commands by replacing the variable <variable> with values from <startvalue > to <endvalue > in steps of <step >. If step is not given stepsize step=1.0 is assumed. Within the loop the string #<variable> is replaced by the integer value of variable, the string %<variable> is replaced by the double precision value of variable , and the string $<variable>$ is replaced by a string of the value of variable . Nested loops are not supported! Warning: The Loop command itself must NOT contain variables! This is so because currently the parsing for replacing variables is performed only after unravelling the loops.
- **EndLoop** Delimits the set of commands inside the active loop.
- **NoWarnings** suppresses all warning messages.

### 2.2.3 Predefined Materials

- **Material: Type=<material> , TempC=<temperature in C> [, TempK=<temperature in K>, x=<x>]** Defines material to be modeled and temperature. Temperature can be specified in degrees Celsius (TempC) or Kelvin (TempK), but only one temperature may be specified. The input, x, can be used to specify a compositional value of series of materials, such as titanomagnetite.
  Supported materials include:

- **Magnetite** Defines material constants ($M_s$, $A_{Ex}$, $K_1$, $K_2$, $K_a$, $K_{aa}$, $K_b$, $K_{bb}$, $K_{ab}$, and $K_u$) for magnetite at temperature <temperature> to be used in subsequent calculations. The temperature is given in degrees Celsius and must be $-200 \leq temperature \leq 580$. Above -153°C magnetocrystalline anisotropy is cubic, at and below -153°C magnetocrystalline anisotropy is monoclinic.

- **MagnetiteLegacy** Defines material constants ($M_S$, $K_1$, $A_E$ and $\lambda_s$) for magnetite at temperature <temperature> to be used in subsequent calculations. The temperature is given in degrees Celsius and must be positive and below magnetite's Curie temperature at 580°C. Note that although we now set the linear saturation magnetostriction constant , it will have no effect unless a non-zero value is set for the compressive stress $\sigma$ . **NOTE: This is the original "magnetite" function of MERRILL prior to v1.5.0**.

- **TM** Defines material constants for titanomagnetite of titanium composition <x> (as a percentage) at temperature <temperature> to be used in subsequent calculations. The temperature is given in degrees Celsius and must be positive and below the Curie temperature of the relevant composition. Compositions with <x> greater than 65 are less reliable.

- **Iron** Defines material constants for iron at temperature <temperature> to be used in subsequent calculations. The temperature is given in degrees Celsius and must be positive and below iron's Curie temperature at 770°C.

### 2.2.4 Mesh related commands

- **LoadMesh: Mesh=<index>** Loads a previously read mesh and its finite element arrays from location <index>. This mesh will then be used in subsequent operations.

- **Resize: OldSize=<old length>, NewSize=<new length>** changes the length scale of the mesh such that after this rescaling the length <old length> will become <new length> .

- **LoadMagState: Mesh=<index>** Loads a previously saved magnetic state associated with saved mesh at location <index>. This should ONLY be used immediately after an appropriate mesh is loaded using the **LoadMesh** command.

- **ReMesh: Mesh=<index> [, Scaling=<y | n>]** Takes the current magnetization array and interpolates it at the nodes of the previously read mesh at location <index>. This mesh from location <index> is then loaded and will be used in subsequent operations. If you want to enforce that the old and new meshes overlap then use the option **scaling=y**. This centres

and linearly scales the old mesh to the centre and dimensions of the new mesh, so that the scaling along x, y and z may be different if the dimensions of the old and new mesh are different along the different axes. This is the default if you do not use the scaling option. If you dot want rescaling specify **scaling=n**

- **RenumberBlockSD: OldBlock=<index1>, NewBlock=<index2>** For the currently loaded mesh this command renumbers elements with SubDomain (Block) ID's <index1> to <index2>. The new index (<index2>) can (usefully) be the same as an existing BlockSD number. In this way you can merge Subdomain's. The modified mesh numbering is saved to the current mesh index (see **ReadMesh** command), and so can be re-loaded at a later point in MERRILL script using **LoadMesh**. If you need to restore the original SubDomain ID's you will need to re-read the mesh using **ReadMesh**. This will report an error if the current mesh does not contain SubDomain ID's of value <index1>. This command is useful to redefine element blocks that are subsequently remeshed using **ReMeshSD**.

- **ReMeshSD: Mesh=<index>** Takes the current magnetization array and interpolates it at the nodes of the previously read mesh at location <index>. All and Each subdomain in the mesh is iteratively interpolated to the identically numbered subdomain in the previously read mesh. The magnetization and mesh outside each subdomain is ignored for each interpolation. If a subdomain does not exist in the new mesh it's magnetization is set to zero. This mesh from location <index> is then loaded and will be used in subsequent operations. **NOTE** If one or more subdomains in the previously read mesh (the mesh you are interpolating on to) have different SubDomain numbers from the current mesh, then they will have their magnetization set to zero. You then need to set the magnetisation for these subdomains using commands such as **Randomize all moments SD = <Material Number>** or **Uniform magnetization <x> <y> <z> SD = <Material Number>**

- **ReplaceSD: savedmesh=<mesh number> , block=<block number>** Takes the current magnetization array and interpolates it on to a subdomain with block number =<block number> of a previously loaded mesh that has been saved with a particular <mesh number> . The current magnetization state node locations are translated so it is centred and scaled to the dimensional extent of the chosen subdomain of the saved mesh. Note that the both the chosen mesh and the chosen subdomain number must exist. Once the magnetization of the subdomain has been replaced the magnetization associated with that mesh is permanently changed, and can be altered by a subsequent ReplaceSD command. Note, where the mapped magnetization nodes fall outside the mesh used for interpolating the magetization, then the magnetization is set to a random unit vector for that node.

- **FixBlockSD: SD=<index1; index2; ...>** This fixes the current values of the magnetization for all nodes in any specified SubDomain so that they do not change during optimization. You must specify all SubDomain indices on a single command line. At present, a maximum of 100 subdomains can be specified. Subdomain indices must be separated by a semi-colon (";"). For example **FixBlockSD: sd=101; 167; 134** will constrain the values of the magnetization in the three subdomain blocks **101 167 134**. Note that subsequent calls to **FixBlockSD** will reset the constrained blocks to the new SubDomain index list. To free all nodes chose a single negative block index, e.g **FixBlockSD: SD=-1**

### 2.2.5 External field control

- **External field: Type=direction, Vector=<x> <y> <z>** Sets the direction vector for an external homogeneous magnetic field. Intrinsically sets the field to B = 0 T.
- **External field: Type=Fibonacci, Direction=<i> , Max=<N>** Sets the direction vector for an external homogeneous magnetic field. Intrinsically sets the field to B = 0 T. Field direction is defined as the $i^{th}$ unit vector from a sequence of N directions from a Fibonacci sphere.
- **External Field: Type=strength, Intensity= <B> [, units=<unit>]** Sets the strength of the external homogeneous magnetic field as B in units of unit. Possible values for unit are 'muT', 'mT', or 'T', with the default being 'mT'. The field strength command must be set after defining the direction. Subsequent calls reset the field to B without changing the direction. Can be used for hysteresis modeling.

### 2.2.6 Magnetocrystalline anisotropy control

- **Anisotropy: Type=<anisotropy type> [, SD=<subdomain number>]** Sets the magnetocrystalline anisotropy to specified type. Optional argument **sd** scnbe used to specify the subdomain number to be modified. **NOTE:** All anisotropy constants default to zero and MUST be set using the **Set** command. The following anisotropy types are supported:

  **Uniaxial** where the easy axis defaults to the X axis, but can be changed using the **easyaxis** command. Uses the K1 anisotropy constant.
  **Cubic** where the cubic axis are the [1 0 0], [0 1 0] and [0 0 1]. Uses the K1 and K2 anisotropy constants. Thus a negative K1 will produce easy <1 1 1> axes.
  **Monoclinic** for magnetite. You can specify the first order (Ka, Kb and Ku) second order (Kaa, Kab and Kbb) anisotropy constants. Ka and Kaa are along [ 1 -1 0 ], Kb and Kbb are along [1 1 0] and Ku along [1 1 1] (see Abe, K. et al., 1976., J. Phys. Soc. Japan, 41, 1894–1902).
  **MonoclinicPyrrhotite** Sets the magnetocrystalline anisotropy to be monoclinic and UNIAXIAL in the basal plane for Pyrrhotite. You can specify the first K1p K2p, K3p, K4p and K5p anisotropy constants with reference to the [1 0 0], [0 1 0], [0 0 1] axis (See Bin, M. and Pauthenet, R. Magnetic Anisotropy in Pyrrhotite. Journal of Applied Physics, 34(4), 1161–1162. 1963).
  **HexPyrrhotite** Sets the magnetocrystalline anisotropy to be monoclinic AND hexagonal in the basal plane for Pyrrhotite. You can specify the first K1p K2p, K3p, K4p and K5p anisotropy constants with reference to the [1 0 0], [0 1 0], [0 0 1] axis (See Bin, M. and Pauthenet, R. Magnetic Anisotropy in Pyrrhotite. Journal of Applied Physics, 34(4), 1161–1162. 1963).

  **Example usage:**

  **anisotropy: Type = cubic**
  **anisotropy: Type = monoclinic, sd= 2**

You can change the default anisotropy axis by using the **EasyAxis** for uniaxial anisotropy or **CubicRotation** for all other anisotropy types.

- **CubicRotation: Alpha=<alpha>, Theta= <theta>, Phi=<phi>, [ units = ¡Radians | Degrees¿, SD =¡SD¿]** Performs a 3D rotation of the cubic or monoclinic anisotropy axis. Variables are real numbers that specify the rotation angles in radians. Alpha, theta, phi, corresponds to rotations around the x-, y-, and z-axes, respectively. Positive angles correspond to rotation in a clockwise sense when viewed from the origin along the specified axis.

  To rotate [111] to [100]:
  CubicRotation 0 0.6154 -0.7854
  To rotate [111] to [110]:
  CubicRotation -0.4636 0.4205 0.1007

- **ReadCubicRotation <filename>** Reads in a list of rotation angles for each subdomain. Useful for matching rotations of mesh geometries to rotation of anisotropy axis.The file MUST have the following format:

  First line of text header (ignored by MERRILL)
  <subdomain number> <alpha> <theta> <phi>
  e.g.
  first line of any text
  1 0.4636 0.4205 0.1007
  2 -0.4636 0.4205 0.1007
  3 0 0.6154 -0.7854

- **Easyaxis <x> <y> <z>** Sets the easy axis for uniaxial anisotropy (defaults to <1 0 0>).

- **Easyaxis <x> <y> <z> SD = <Material number>** Sets the easy axis for uniaxial anisotropy per subdomain in a multi-phase model.

- **Stressaxis <x> <y> <z> SD = <Material number>** Sets the compressional stress axis for uniaxial stress per subdomain in a multi-phase model. If **SD** is not set then this commands sets the stress axis for all subdomains

- **ConstantStressvVoigt** $< \sigma_x x > < \sigma_y y > < \sigma_z z > < \sigma_y z > < \sigma_x z > < \sigma_x y >$ Sets the uniform stress according to the Voigt convention (T. Belytschko, W.K. Liu, and B. Moran, Finite Elements for Nonlinear Continua and Structures, Wiley, 1996). This is applied to whole mesh irrespective of any defined sub-domains.

- **ConstantStressvVoigt** $< \sigma_x x > < \sigma_y y > < \sigma_z z > < \sigma_y z > < \sigma_x z > < \sigma_x y >$ Sets the uniform stress according to the Voigt convention (T. Belytschko, W.K. Liu, and B. Moran, Finite Elements for Nonlinear Continua and Structures, Wiley, 1996). This is applied to whole mesh irrespective of any defined sub-domains.

- **AddEdgeDislocation <G> $< \nu >$ <x> <y> <z> $< dir_x > < dir_y > < dir_z > < B_x > < B_y > < B_z >$** . Introduces an edge dislocation at $(x, y, z)$ in an isotropic media or cubic, in the direction of the vector $(dir_x, dir_y, dir_z)$. $< \nu_y y >$ is the Poisson's ratio, and G is the elastic constant $C_{44}$ in units of Pa, and $(B_x, B_y, B_z)$ is the Burgers vector. An EDGE dislocation must have a Burgers vector perpendicular to the line direction ! NOTE : All lengths are given in microns, including the Burgers vector.

- **AddScrewDislocation <G> <x> <y> <z>** $< B_x > < B_y > < B_z >$ . Introduces an screw dislocation at $(x, y, z)$, in the direction of the Burgers vector $(B_x, B_y, B_z)$. G is the elastic constant $C_{44}$ in units of Pa. NOTE : All lengths are given in microns, including the Burgers vector.

### 2.2.7 Initial magnetization structure control

- **Vortex <Axis>** [ **<Origin>** ] [ **<Chirality>** ] [**<CoreStrength>**] This sets a vortex-type initial state. It requires that <axis> be set, but other parameters are optional. <Axis> is desired direction of the vortex core specified as a 3D cartesian vector. The origin of the axis is assumed to be [ 0 0 0 ], otherwise you can explicitly state the <Origin>, again as a 3D cartesian vector. The Chirality of the vortex is assumed to be Right-Handed, otherwise you can explicitly request left handed using the optional parameter 'LH'. Finally you may bias the size of the vortex core by a scalar value <CoreStrength>, which simply multiplies the magnetisation along the vector core direction.

- **RotateMag [alpha=<angle about X axis> beta=<angle about Y axis> gamma=<angle about Z axis>]**. All angles are specified in degrees and default to zero. The command takes the current magnetization array and rotates it by the specified angles about the centroid of the geometry. If multiple angles are defined then rotation is first about X then Y and finally Z. The rotated magnetization is scaled and interpolated on to the original mesh and saved to its original meshnumber. Note (i) that since interpolation is used, rapid changes in magnetization may look slightly different, and (ii) Where the rotated magnetization nodes do not map within the original mesh, the magnetization is set to a random unit vector.

- **InvertMag**. The command takes no optional parameters. It simply multiplies the magnetization components by -1, so the that the magnetization direction is reversed.

- **Uniform magnetization <x> <y> <z>** Creates a uniform magnetization for the current mesh pointing in the normalized direction <x>,< y>, <z>. The optional parameter b is the index of the block of nodes in the mesh that should be set. By definition block 1 contains the free nodes, while higher block numbers can be used to define fixed nodes. These block have to be defined in the Patran file. Any previous magnetization is lost!

- **Uniform magnetization <x> <y> <z> SD = <Material number>** Creates a uniform magnetization for the current mesh pointing in the normalized direction <x>,< y>, <z> where <Material Number> is an integer that applies the uniform direction to a specified material of a multi-phase model. Note you need to include the SD = , where SD stands for

Sub Domain and a space is needed between 'SD' and '='. The variable it is set to MUST match a BLOCK ID set in your PATRAN mesh file.

- **RandomSaturation [SD = <Material Number>]** Creates a uniform magnetization in a random direction, which is printed to STDOUT. On setting the uniform magnetization the externally applied direction is set in the same direction. When used with the subdomain (SD) call, that last randomly generated direction will be set as the field direction.
- **Randomize magnetization <angle>** Randomly changes each current magnetization vector by at most <angle> degrees. The previous magnetization is (partly) lost!
- **Randomize magnetization <angle> SD = <Material number>** Randomly changes each current magnetization vector by at most <angle> degrees, where <Material Number> is an integer that applies randomisation to a specified material of a multi-phase model. Note you need to include the SD = , where SD stands for Sub Domain and a space is needed between 'SD' and '='. The variable it is set to MUST match a BLOCK ID set in your PATRAN mesh file.
- **Randomize all moments** Replaces the current magnetization by randomly distributed unit vectors. Any previous magnetization is lost!
- **Randomize all moments SD = <Material Number>** Replaces the current magnetization of specified subdomain by randomly distributed unit vectors, where ¡Material Number ¿ is an integer that applies to randomisation to a specified material of a multi-phase model. Note you need to include the SD = , where SD stands for Sub Domain and a space is needed between 'SD' and '='. The variable it is set to MUST match a BLOCK ID set in your PATRAN mesh file.

### 2.2.8 Minimization control

- **Minimize** Calls the minimization routine for the current mesh and initial magnetization. The spherical polar axis is switched form Z to Y to X and then Cartesian back to Z for a number of steps equal to **SwitchCount** or **SwitchPercentage** of **MaxEnergyEvaluations**. Thereafter the polar axis is switched for each restart of the energy minimization. This call does not save the final result!
- **ConjugateGradient** Uses conjugate gradient steps during the accelerated descent.
- **SteepestDescent** Uses normal gradient steps during the accelerated descent.
- **MinimizePolX** Calls the minimization routine for the current mesh and initial magnetization. Minimizes using spherical polar coordinates with polar axis X. This call does not save the final result!
- **MinimizePolY** Calls the minimization routine for the current mesh and initial magnetization. Minimizes using spherical polar coordinates with polar axis Y. This call does not save the final result!

- **MinimizePolZ** Calls the minimization routine for the current mesh and initial magnetization. Minimizes using spherical polar coordinates with polar axis Z. Prior to MERRILL 1.36 this was the default. This call does not save the final result!

- **MinimizeCart** Calls the minimization routine for the current mesh and initial magnetization. Minimizes using Cartesian coordinates. This call does not save the final result!

- **LLG <Euler | RK4 | RKadapt > [ < Options > ]** Sovlves the Landau Lifshitz Gilbert equation using either Euler, 4th order Runge-Kutta or Adaptive step Runge-Kutta time stepping, for the current mesh and initial magnetization. Default convergence criterion set to be a change of magnetization vector on all nodes to less than $10^{-4}$ per time step. While solving it outputs iteration time, model time (seconds), convergence rate (i.e. max diff between iterations), and the average normalized Mx, My and Mz.
  Optional parameters in any order, set using <**key**>=<**value**> pairs with no gap before or after the =. Default values in parenthesis ().

  **Converge** Sets the integration convergent criterion, needed for all methods (1.0E-4) .
  **Damp** Sets the damping parameter, needed for all methods (1.0) .
  **DeltaT** Sets the time step length. Required for Euler (8.0E-4) and RK4 (8.0E-3).
  **MaxStepLength** Sets the maximum step length. Required for RKadapt (8.0E-2).
  **FirstStep** Sets the first attempted step length. Required for RKadapt (8.0E-3).
  **MinStepLength** Sets the minimum step length. Required for RKadapt (0.0).
  **StartTime** Sets the initial (start time). Required for RKadapt (0.0).
  **Accuracy** Sets the accuracy requirement for integration of magnetisations. Required for RKadapt (1.0E-8).
  **Filename** If set, will write out intermediate LLG 'solutions' every **interval** units (see below). The solution 'time' is written as tecplot zone title. If not set, no intermediate out is written.
  **Format** Sets the format of the tecplot file. Must be either 'block' or 'point'. If nots set , defaults to 'block'
  **Filename** If set, will write out intermediate LLG 'solutions' every **interval** units (see below). The solution 'time' is written as tecplot zone title. If not set, no intermediate out is written.
  **Interval** Sets the minimum interval between saving intermediate 'solution'. Defaults to 4.0. NOTE if you use a value of 0.0, this will result in all intermediate 'solution' being saved, and this might result in a very large file.

  LLG example use:
  LLG RKadapt Damp=3 FirstStep=1.E-3 MaxStepLength=5.0E-2 filename=afilename interval=4.1

### 2.2.9 Energy barrier path calculation control

- **MakeInitialPath** Assumes that a path is defined by set PathN <number > and that the first and last magnetization patterns are defined. Then proceeds by stepwise **Multi-polar and Cartesian** minimization to construct an initial path for subsequent optimization by the NEB method. Use **Set SwitchCount** and **Set SwitchPercent** commands to set the switch frequency between Cartesian and the X, Y and Z polar axis.

- **MakeInitialPathCart** Assumes that a path is defined by set PathN <number > and that the first and last magnetization patterns are defined. Then proceeds by stepwise **Cartesian** minimization to construct an initial path for subsequent optimization by the NEB method.

- **PathMinimize** Assumes that an initial path is defined and minimizes the action integral using a variant of the NEB method.

  **PathLogfile <filename>** Starts logging all subsequent path minimization calculations into three logfiles <filename>.enlog <filename>.grlog, and <filename>.dlog. They contain energies along the path, norms of the gradients along the path and cumulative distances along the path. Logging can be stopped by EndLog or CloseLogfile.

- **PathStructureEnergies <filename>** Computes the energies for each structure along the minimum energy path. The output is written to <filename>, however if this is omitted then the output is written to standard out.

- **PathStructureEnergyComponents <filename>** Computes the energies for each structure along the minimum energy path, and also reports the separate energies for each distinct blocks defined by unique BLOCK ID's in the mesh file. NOTE that this only works for BLOCKS that do NOT share a boundary, such as when you have a different BLOCK ID for each particle in a multi-particle simulation. If you use this command where you have multi-phase model with a shared boundary between BLOCKS, then the reported energies associated with each BLOCK maybe unreliable. The output has N+1 energy columns, the first is the total energy and the reminder is the energy associated with each BLOCK ID that was defined in the mesh file. The output is written to <filename>, however if this is omitted then the output is written to standard out.

- **PathStructureAllComponents <filename>** Computes and outputs all energy components for each structure along the minimum energy path, and also reports the separate energies for each distinct blocks defined by unique BLOCK ID's in the mesh file. Block Zero reports the component energies for total structure, i.e the sum of all the other blocks. NOTE that this only works for BLOCKS that do NOT share a boundary, such as when you have a different BLOCK ID for each particle in a multi-particle simulation. If you use this command where you have multi-phase model with a shared boundary between BLOCKS, then the reported energies associated with each BLOCK maybe unreliable. The output has N+1 energy columns, the first is the total energy and the reminder is the energy associated with each BLOCK ID that was defined in the mesh file. The output is written to <filename>, however if this is omitted then the output is written to standard out.

- **MagnetizationToPath <index >** Saves the current magnetization in the path at location <index>. This allows to assemble a path from individual magnetization states that have to fit to the current mesh! After assembling a path it must be renewed before further operations can be performed.
- **PathToMagnetization <index>** Moves the path magnetization state at location <index> to the current magnetization. This allows to change individual magnetizations in the path. E.g. Initial and final states of a path read from a file can be minimized for new material constants.
- **RenewPath** Defines all path variables, like distances and tangent vectors, assuming that all magnetizations have been correctly filled.
- **RefinePathTo <newlength>** Refines the current path to a new number of states by linear interpolation in the magnetization angles. This also resets PathN to the new value and renews the path. Of course, the new number of states can also be less than the previous PathN .

#### 2.2.10 High level magnetization processes

- **HystLoop <max. field> <min. field> <field step> <filename> [InitialGeess Flag]** Function to simulate a hysteresis loop. By default The routine imparts a saturation magnetization in the set field direction, then loops through the specified field strengths from maximum to minimum, using the specified field step. Field step must be negative if going from a greater to lower field. Default units are mT. Magnetization results are output to <filename>.loop file, domain structures are output to a multi-zone Tecplot file. An optional flag **InitialGuess** can be used to override the setting of a saturated initial guess. If **InitialGuess** is set to any number less than 0, then no initial guess is set, and so should be set explicitly by the user.
- **IRM <max. field> <field step> <filename>** Simulates an isothermal remanence (IRM) acquisition curve from zero to maximum field using the specified field step. Default units are in mT. The routine starts from a randomized initial guess for the initial state of the zero field solution. The routine tracks in-field moments and when needed will sweep to zero-field by finding solutions at fields equivalent to 75%, 50%, and 25% of the saturation moment. Magnetization results are output to <filename>.irm file, domain structures are output to a multi-zone Tecplot file. Both in-field and zero-field results are saved.
- **DCD <max. negative field> <field step> <filename>** Simulates a DC demagnetization (DCD, or back-field) curve from zero to maximum field using the specified field step. Default units are in mT. The routine starts from saturation along the applied field direction and determines solutions at 80, 20 and 5 mT as the field decreases to zero for remanence states. The routine tracks in-field moments and when needed will sweep to zero-field by finding solutions at fields equivalent to 75%, 50%, and 25% of the saturation moment. This smoothed sweep to zero field ensures stable solutions. Magnetization results are output to <filename>.dcd file, domain structures are output to a multi-zone Tecplot file. Both in-field and zero-field results are saved.

- **IRMDCD <max. field> <field step> <filename>** Simulates both an isothermal remanence (IRM) acquisition curve and a DC demagnetization (DCD, or back-field) curve from zero to maximum using the specified field step. Default units are in mT. See the IRM and DCD commands for details. This routine uses the final solution of the IRM acquisition as the initial state for the DCD curve.

- **SimpleFORC <saturation field> <nFORC> <filename> [<save FORC Tecplot file?>] [<save last complete state?>]** A simple first-order reversal curve (FORC) routine that measures nFORC number of curves from reversal fields between positive and negative saturation fields (in mT). Field step is determined by the number of curves specified. An initial hysteresis measurement is performed with magnetization results output to <filename> Hysteresis.loop, and domain structures are output to a multi-zone Tecplot file. FORC magnetization results are output to <filename>.frc, domain structures are output to a multi-zone Tecplot file. <save FORC Tecplot file?> is a text flag ("yes" or "no") specifying if the Tecplot solution file for the FORCs is saved. The FORC .tec files can be extremely large, so the default is "no", but the hysteresis Tecplot file is always saved. <save last complete state?> is a text flag ("yes" or "no") specifying if the if the last complete solution is saved to a temporary file to aid recovery if MERRILL crashes (see SimpleFORCRestart). The default is "yes". A LastState file is saved every 5 hours.
  RESTARTS of partially computed FORC's use the same command, and SimpleFORC simply looks for LastState files. Two such files are usually saved in case of a system crash during writing. If it finds one, then it will resume the FORCs calculations from the most recent complete saved state. During a restart FORC parameters are determined form the hysteresis loop and saved states, and the **saturation field** and **nFORC** input parameters are ignored.

### 2.2.11 Input/Output control

- **ReadMesh: mesh=<index>, filename=<filename> [, filetype = <filetype> ]** Reads the Patran file <filename>, and stores the corresponding mesh and finite element arrays at location <index>. The index must be less or equal to the previously set MaxMeshNumber. Optionally filetype = <filetype> can be specified where <filetype> takes the value `patran` or `tecplot`. If not specified `patran` is assumed. A Patran mesh file can be specified simply as

  ```
  ReadMesh: mesh=1, filename=patranfile.pat
  ```

  In the case of TecPlot files, both `block` and `point` formats are supported. Old and new Tecplot header formats are accepted. The old format contains headers such as `N`, `E`, `F` and `ET` for number of nodes, number of elements, datapacking and zone type. These are replaced by more explicit header names in newer formats (see example below). Additionally, a tecplot mesh file can contain no magnetization data, which is useful when starting a simulation.

  A TecPlot file can be specified as

```
ReadMesh: mesh=1, filename=tecplotfile.tec,  filetype = tecplot
```

An example of a TecPlot mesh file is

```
TITLE = "My tecplot mesh"
VARIABLES = "X", "Y", "Z", "SD"
ZONE NODES = 5688, ELEMENTS = 29188,
DATAPACKING = BLOCK, ZONETYPE = FETETRAHEDRON,
VARLOCATION = ([4] = CELLCENTERED)
 0.04000  0.00000  0.00000 -0.04000  0.00000  0.00000  0.03985  0.03939  0.03864  0.03759
 0.03625  0.03464  0.03277  0.03064  0.02828  0.02571  0.02294  0.02000  0.01690  0.01368
 ...
1 1 1 1 1 1 1 1 1 1
 ...
2612 415 5485 303
 ...
```

New in Merrill 2 : if a `.nc` file with the same path and name stem as the mesh file is found, the H2 matrix is read from the file. Otherwise a new H2 matrix is constructed and saved in an ".nc" file with the same name stem as the mesh file.

- **CompressionSettings** <**clf**> <**eta**> <**method**> <**eps_hmatrix**> <**eps_h2matrix**> Set parameters `clf`, `eta`, `method`, `eps_hmatrix`, `eps_h2matrix` controlling compression settings during H2 matrix construction. See H2Lib documentation for details.
  - `clf` integer, default `32`: max leafcluster size in `build_bem3d_cluster`
  - `eta` real, default `2.0`: distance cutoff in `admissible_2_cluster`.
  - `method` string, default `"aca"`: choice of function for hmatrix construction, choose `hca`, `aca`, or `inter`.
  - `eps_hmatrix` real, default `10e-8`: accuracy for H-matrix construction.
  - `eps_h2matrix` real, default `10e-6`: accuracy for H2-matrix construction.

  Example:

  ```
  CompressionSettings 32 2.0 aca .000001 .000001
  ```

  Must be set before `ReadMesh` line, which triggers construction of h2 matrix from mesh file. Has no effect if the `ReadMesh` command detects a pre-existing `.nc` file and does not construct a new H2 matrix.

- **ncFile** [<**filename**>] [**force**] Sets name of `.nc` file to save the H2 matrix to or read the H2 matrix from. Overrides the default behavior of looking for / writing an nc file matching the mesh file name. Must be declared before `readMesh`. Presence of the keyword `force` means

that even when a file with the name is found, it will be ignored; a new H2 matrix will be constructed and written to file, overwriting the previous file contents.

Examples:

```
ncFile my_file_name

ncFile my_file_name.nc

ncFile path/to/my_file_name

ncFile force

ncFile my_file_name.nc force
```

- **ReportEnergy: filename = ¡filename¿** Makes a report on the model parameters (mesh size, material parameters, exchange length etc, as well as the component and total magnetic energy both in reduced units of (Kd. V) and in Joules. If not filename is specified it writes to standard out , i.e. the screen

- **ReadMagnetization: filename=<filename> [, zone=<zone number>, meshno=<mesh number>, format=<TECPLOT | DAT> ]**, Reads a magnetization file (.dat or .tec) into the current mesh magnetization array. The data format is automatically chosen from the file suffix, but you can override this by using **format** option. It will report an error if the data file has a different number of nodes than the current mesh file. If you have multiple meshes loaded you can associate the data with a particular mesh using the **meshno** option. For TECPLOT format files you can optionally chose to read in a particular data zone using the **zone** option. The default zone number = 1. Command abbreviation **ReadMag: file=<filename>**

- **WriteTecPlotPath: filename=<filename> [packing = <BLOCK | POINT>]** Exports the current path to a TecPlot file with name <filename> where '.tec' is appended to the end of the filename if not already specified. All states along the path are individual zones in the TecPlotFile.

- **ReadTecPlotPath: filename=<filename>** Reads a new path from a TecPlot file with name <filename>. All states along the path are individual zones in the TecPlotFile and the PathN parameter (see Set PathN), is automatically set to the number of zones in the Tecplot File. The command also reads in the mesh so no other ReadMesh command should exist in the script. All mesh related quantities are thus recalculated. You must NOT read in a mesh separately, or set PathN separately. Make sure that all material parameters are correctly assigned, since those are not read !

- **AppendTecplotZone: filename = <filename> [, packing =<BLOCK | POINT>]** Saves the magnetization and the mesh to a file <**filename**>_mult.dat. For the first call to this command, or when the filename is changed, it will write the magnetization and the finite-element mesh. Subsequent calls will write the magnetization only. This is useful for multiple solutions that use the same mesh (such as field or volume hysteresis). Each Zone can be named using the ZoneName command. Files are written in the TecPlot file format containing

the data and the mesh for visualization using TecPlot only. Paraview can currently only reads Tecplot format files that contain only **one** zone, and so files created using this command generally cannot be read by paraview.
The optional <packing> parameter takes the value of either BLOCK or POINT. BLOCK format is required to save the sub-domain regions of a multi-phase solution to a Tecplot file. The subdomains are numbered by an 'SD' variable for each element. BLOCK is now the default output format.

- **ZoneName:** <**string**> Provides a Zone name that is written to any Tecplot format file, so works with command WriteMagnetization, WriteDemag and AppendTecplotZone. The string is limited to a maximum of 80 characters.

- **EnergyLog: filename** = <**filename**> Starts logging all subsequent energy calculations into the logfile <filename>.log. For multi-phase material the magnetization reported at each node is the box-volume average of relative Ms*m for each of the elements to which the node belongs. Logging can be stopped by EndLog or CloseLogfile

- **CloseLogfile** Ends the previous logging of energy calculations or path minimizations.

- **WriteMagnetization: filename=**<**filename**> [**, format** = <**TECPLOT** | **DAT** | **BOTH** > **, packing** = <**BLOCK** | **POINT**>]. This writes the solution magnetization structure to a file. By default this produces two files: (1) <**filename**>.dat, which contains vertex coordinates and magnetization vectors only, but no mesh and (2) <**filename**>_mult.tec which is TecPlot format file containing the data and the mesh for visualization using ParaView or TecPlot. Any suffix in the original filename is removed. You may optionally chose to write only the Tecplot or Dat format file. The optional <packing> parameter affects only the tecplot files. <packing> takes the value of either BLOCK or POINT. BLOCK format is the default and is required to save the sub-domain regions of a multi-phase solution to a Tecplot file (readable by Paraview). Only Tecplot files format can contain subdomains (i.e. different blocks of elements), and Dat files do not contain any mesh information at all.

- **WriteDemag** <**filename**> [<**packing**> ] Saves the internal grain demagnetization field, the magnetization vector and the magnetic scalar potential at each node, the and the Finite Element mesh: <filename> demag.tec TecPlot file for visualization using ParaView or TecPlot. Contains mesh geometry and demagnetization field for one or more magnetization states. The output is of the form "x, y, z, Hd_x, Hd_y, Hd_z, M_x, M_y, M_z, POT, SD" Saves in POINT or BLOCK format. H values are in units of A/m and M value are unit vector directions multiplied by the local Ms value. Note SD parameter is only output in BLOCK format The demag field is in A/m.
  The optional <packing> parameter takes the value of either BLOCK or POINT. BLOCK format is required to save the sub-domain regions of a multi-phase solution to a Tecplot file (readable by Paraview). The subdomains are numbered by an 'SD' variable for each element. BLOCK is now the default output format.

- **AppendDemagZone** <**filename**> [<**packing**>] Saves the internal grain demagnetization field, magnetisation (normalised for each material region) and the mesh to a : <filename>_

demag.tec TecPlot file in the same way as the WriteDemag command, for visualization using ParaView or TecPlot. Subsequent calls will write the demag field and magnetization only. This is useful for multiple solutions that use the same mesh (such as field or volume hysteresis). Each Zone can be named using the ZoneName command. Files are written in the TecPlot file format containing the data and the mesh for visualization using TecPlot only. Paraview can currently only reads Tecplot format files that contain only **one** zone, and so files created using this command generally cannot be read by paraview.
The optional <packing> parameter takes the value of either BLOCK or POINT. BLOCK format is required to save the sub-domain regions of a multi-phase solution to a Tecplot file. The subdomains are numbered by an 'SD' variable for each element. BLOCK is now the default output format.

- **WriteHyst <filename>** Saves hysteresis data in 5 columns of MOD(H), M$H_{ext}$ and the 3 components on the average unit M vector, where M is the magnetization and $H_{ext}$ is the external field. M$H_{ext}$ is normalized to the saturated magnetization in the direction of the applied field. For multi-phase material the magnetization at each node is the box volume average of relative Ms*m for each of the elements to which the node belongs. The first line of the file states the total saturated magnetization of the model, used to normalize the reported M$H_{ext}$ values. Output file is <filename >.hyst

- **WriteStressHyst <filename>** Saves stress hysteresis data in 5 columns of MOD(Sigma), MSt-dir, and the 3 components on the average unit M vector, where M is the magnetization and St-$dir_{ext}$ is the (normalized) stress direction. MSt-dir is normalized to the saturated magnetization in the direction of the applied field. For multi-phase material the magnetization at each node is the box volume average of relative Ms*m for each of the elements to which the node belongs. The first line of the file states the total saturated magnetization of the model, used to normalize the reported Mst-dir values. Output file is <filename >.stresshyst

- **WriteLoopData <filename> <Name1> <Value1> <Name2> <Value2>...** Save field, moment, total volume and user specified variables to file <filename>.loop. <Name1> specified the column header printed to the file and must not contain spaces. <Value1> is the real number parameter value to be saved. Useful when looping over user defined variables.
Example usage:
Loop Br -100 10 100
WriteLoopData MyOuputFile Reversal_Field $Br$
EndLoop

- **WriteBoxData <filename >** Writes the magnetization per node along with the node associated volume x,y,z, $m_x$, $m_y$, $m_z$, vbox. Where x,y,x is the node location, $m_x$, $m_y$, $m_z$ is the magnetization unit vector components, and vbox is the volume associated with a node (in units of microns3). This is useful if you want to compute the absolute magnetization associated with each node.

- **SDenergySurface < filename > [ < Options > ]** Creates a table of Energy for a uniform (SD) magnetization pointing in different directions, written as a simple CSV file to <filename>_surf.dat .The Energy outputs are all given in units of Joules. The 10 columns of

data are x,y,z,theta, phi, total energy, anisotropy, demag, extnal field, stress. Where x,y and z are the cartesian directions, theta and phi the polar and azimuthal polar coordinates, and the rest are the component magnetic energies.
Optional parameters in any order, set using <**key**>=<**value**> pairs with no gap before or after the =. Default values in parenthesis ().

**ThetaMin** The minimum value of the polar angle stated in degrees (0) .
**ThetaMax** The maximum of the polar angle stated in degrees (180) .
**DeltaTheta** Theta increment stated in degrees (1.0).
**PhiMin** The minimum value of the azimuthal angle stated in degrees (0).
**PhiMax** The minimum value of the azimuthal angle stated in degrees (359).
**DeltaPhi** Phi increment stated in degrees (1.0)

# 3 Developer - Compiling MERRILL

This section is just for developers. If you just want to use MERRILL, precompiled binaries are available for LINUX and macOS , and can be downloaded from the MERRILL homepage at: `http://www.rockmag.org` , `https://bitbucket.org/wynwilliams/merrill/downloads/`.

If you want to modify the way MERRILL behaves, then you will need to download the source code from the repository at:

`https://bitbucket.org/wynwilliams/merrill`

Every computer and operating system is unique and software and compilers may operate or install differently on different systems. These guidelines are there provided for general guidance, but should work in the majority of instances.

## 3.1 Windows

### 3.1.1 Requirements

This approach uses the MinGW (GNU ports) of **gcc/gfortran, gmake** and **cmake** for Windows. Most users will need to install these packages as they are not usually installed on Windows.

1. Install MinGW (GNU port of gcc/gfortran for Windows). The basic default basic install is fine.

   `http://www.mingw.org`

2. Set the environment variables to add the compilers to the Windows Command Line or PowerShell (I like the new PowerShell in recent versions of Windows, but you can use the standard cmd.exe if you prefer.)

   (a) This step varies slightly depending on your version of Windows (7/8/10), but you are looking for the **Environment Variables**, which is usually found in the **Advanced Settings** of the System Properties dialogue box, in the control panel or by right-clicking on the 'Computer' desktop or start menu item. A useful visual guide is here:

   `https://www.rose-hulman.edu/class/csse/resources/MinGW/installation.htm`

   (b) In the **Environment Variables** dialogue box, look for the **System Variables** box, select the **Path** variable, and click **Edit**. (Do not delete anything!)

   (c) Navigate to the end of the Edit System Variable dialogue box, and append the following to the very end of the Variable Value for Path: (Assuming the location of the MinGW installation is here)

   **C:\MinGW\bin**

   (d) Click OK and Apply the settings.

   (e) In a shell or command line, the gcc command, and related commands should now be available.

3. Install CMAKE
   (a) `https://cmake.org/download`
4. Create a build folder in the Merrill directory and change into that directory (we called it "build")
5. Run CMake from within that build directory just created
   (a) Note we are not using the unix.sh script shipped with Merrill
   (b) Remember to specify the above directory with "..", as we are using the CMakeLists.txt file in the top-level MERRILL directory, one directory above us now.
   (c) The -G flag specifies the generator system name. We want to use the MinGW system that we have previously installed.
   (d) A full list of available generator systems is displayed by running cmake –help

### 3.1.2 Compile commands

1. (a) To make a **dynamic build** (the usual option, creates small build size of about 0.5Mb but needs runtime libraries to loaded dynamically) use:
   `cmake -G "MinGW Makefiles" ..`
   the two dots at the end are important!
   OR
   (b) To make a **static build** (bigger build size of about 1Mb, but easily moved to other windows machines):
   `cmake -G "MinGW Makefiles" -DCMAKE_EXE_LINKER_FLAGS:STRING="-static"`
   `-DCMAKE_BUILD_TYPE:STRING="Release" -DBUILD_SHARED_LIBS:BOOL=OFF ..`
   remember the two dots at the end!

   These create a Makefile in the build directory.
2. We now issue the make command to use this generated makefile:
   `mingw32-make`
   Which will produce the **merrill.exe** executable file and libmerrill.a. In the case of a static build the libmerrill.a is not needed for merrill.exe to run.

## 3.2 OS X

### 3.2.1 Requirements

1. Install Command Line Tools
   This requires the MacOS App Xcode installed. Xcode contains the tools needed to compile native code on your machine. It can be downloaded from the App Store or from `https://developer.apple.com/xcode/downloads/`.

This is a large app and can take sometime to download and install. Once installed open an terminal window and and execute the command

```
xcode-select {install
```

and press enter/return. You may be prompted for your password. Then wait for the command to finish.

2. Install HomeBrew.

   Homebrew makes it very easy to install and maintain various command line tools and is highly recommended if you intend to any software development. To install open a terminal window and execute:

   ```
   ruby -e "$(curl -fsSL
   https://raw.githubusercontent.com/Homebrew/install/master/install)"
   < /dev/null 2> /dev/null
   ```

   type the full text and then press enter/return. You may be prompted for your password.

   If you get an error "Ivalid certificate chain" when downloading from github, the following will usually solve this. At the terminal prompt type:

   ```
   curl https://raw.githubusercontent.com/Homebrew/install/master/install
   ```

   and press enter/return. Then re-execute the previous command:

   ```
   ruby -e "$(curl -fsSL
   https://raw.githubusercontent.com/Homebrew/install/master/install)"
   < /dev/null 2> /dev/null
   ```

3. Install cmake.

   At the terminal prompt type:

   ```
   brew install cmake
   ```

   and press enter/return.

4. Install gfortran (if necessary)

   Xcode will normally install gfortran. You can test this by typing `gfortran` at the terminal window command prompt. If installed it will complain `gfortran: fatal error: no input files`, and your good to go. If it is NOT installed it will complain `gfortran: command not found`, and you need to install gfortran, by issuing the following command at the terminal window prompt:

   ```
   brew install gcc
   ```

   and press enter/return.

### 3.2.2 Compile commands

1. In directory Merrill create a directory **build**, will contain the compiled code.
2. (a) For a **dynamic** build (the usual option, creates small build size of about 0.5Mb but needs runtime libraries to loaded dynamically):
   From the build directory issue the command:
   `cmake ..`
   note the two dots at the end are important.
   When that comand has finished , issue the build command:
   `make`
   This should compile and link and create an executable file called **merrill**.
   OR
   (b) To make a **static build** (bigger build size of about 1Mb, but easily moved to other macOS machines), issue the commands:
   `export FFLAGS=` "–lz –static-libgfortran –static –libgcc"
   then
   `cmake ..`
   and finally
   `make`
3. Running MERRILL.
   It is useful to have the location of the executable code in your path. This can be done by ensuring that the path of the directory is listed in your environment variable **$PATH** stored in your **.bash_profile** or **.profile** file in your home directory. It should contain a line such as
   `export PATH="/Users/wynwilliams/merrill/build:$PATH"`
   you can n then issue the command
   `merrill <scriptfilename>`
   from any directory

## 3.3 Linux

### 3.3.1 Requirements

git, cmake, gfortran (for now only option - add TODO other compiler's flag options in cmake) should be present on the system.

For the H2Lib (new and default since 2.0.0 ) variant, the gcc C compiler should also be present. Libraries lapack, OpenBLAS, and netcdf need to be found in a standard location or via library path . With admin permissions, install

```
apt install libopenblas-dev liblapack-dev libnetcdf-dev
```

(ubuntu) / equivalent packages on other linux flavors and package managers. Without admin permission, build lapack, openBLAS, and netcdf libraries locally and add to library path, or contact the facilities' admin.

### 3.3.2 Obtaining source code

Merrill is hosted at `https://bitbucket.org/wynwilliams/merrill`.

Clone merrill with

```
git clone --recurse-submodules https://bitbucket.org/wynwilliams/merrill
```

. Merrill now includes H2Lib as an external submodule in "src/interfaces/H2Lib"; the "–recurse-submodules" flag is needed to also clone the H2Lib submodule.

On an already cloned directory, run

```
rm -r src/interfaces/H2Lib
git submodule update --init --recursive
```

to populate the H2Lib submodule .

Contents of the H2Lib directory will be cloned from their respective developers, `https://github.com/H2Lib/H2Lib`.

### 3.3.3 Compiling

Use standard cmake and make build system commands,

```
mkdir build
cd build
cmake ..
make
```

### 3.3.4 Legacy / fallbacks / build options

If unable to obtain or compile H2Lib, the legacy version of merrill using native sparse matrix formats and SLATEC routines can be accessed with

```
cmake .. -DMERRILL_USE_H2LIB=OFF
```

. This version has a larger memory footprint.

If OpenBLAS and/or lapack cannot be found, H2Lib without BLAS can be used with

```
cmake .. -DH2LIB_USE_BLAS=OFF
```

.

Use cmake arguments `-DCMAKE_BUILD_TYPE=RELEASE` (default) for optimized version and `-DCMAKE_BUILD_TYPE=DEBUG` for no compiler optimization `-O0`, compiler warnings, and enabling usage with `gdb` and `gprof`.

## References

Conbhuí, P. Ó., Williams, W., Fabian, K., Ridley, P., Nagy, L., and Muxworthy, A. R. (2018). MERRILL: Micromagnetic Earth related robust interpreted language laboratory. *Geochemistry, Geophysics, Geosystems*, 19.

Fabian, K. and Shcherbakov, V. P. (2018). Energy barriers in three-dimensional micromagnetic models and the physics of thermoviscous magnetization. *Geophysical Journal International*, 215(1):314–324.

# A List of Variables

### A.1 Material constants

- **Ms** saturation magnetization in A/m.
- **Aex** exchange constant in J/m.
- **K1** 1st anisotropy constant for uniaxial or cubic anisotropy in J/m$^3$.
- **K2** 2nd anisotropy constant for uniaxial or cubic anisotropy in J/m$^3$.
- **Ka** 1st anisotropy constant for monoclinic anisotropy along **a** [1 -1 0] axis, in J/m$^3$.
- **Kb** 1st anisotropy constant for monoclinic anisotropy along **b** [1 1 0] axis, in J/m$^3$.
- **Ku** 1st anisotropy constant for monoclinic anisotropy along **u** [1 1 1] axis, in J/m$^3$.
- **Kaa** 2nd anisotropy constant for monoclinic anisotropy along **a**, in J/m$^3$.
- **Kbb** 2nd anisotropy constant for monoclinic anisotropy along **b**, in J/m$^3$.
- **Kab** 2nd anisotropy constant for monoclinic anisotropy along **a, b**, in J/m$^3$.
- **K1p** 1st anisotropy constant for Pryyhotite monoclinic anisotropy, in J/m$^3$.
- **K2p** 2nd anisotropy constant for Pryyhotite monoclinic anisotropy, in J/m$^3$.
- **K3p** 3rd anisotropy constant for Pryyhotite monoclinic anisotropy, in J/m$^3$.
- **K4p** 4th anisotropy constant for Pryyhotite monoclinic anisotropy, in J/m$^3$.
- **K5p** 5th anisotropy constant for Pryyhotite monoclinic anisotropy, in J/m$^3$.
- **LamdaS** the linear saturation magnetostriction constant $\lambda_S$ for magnetite (dimensionless).
- **SigmaS** the uniaxial stress constant $\sigma$, in N/m$^2$. For the ***compressional*** stress regime, $\sigma$ should be $\geq 0$ and the magnetostrictive energy is given by $E_\sigma = \frac{3}{2}\lambda_S \sigma V cos^2\phi$, where $\phi$ is the angle that the compressive stress $\sigma$ makes with the magnetization $\boldsymbol{M}$. For the ***tensional*** stress regime, $\sigma$ should be $< 0$ and the magnetostrictive energy is given by $E_\sigma = \frac{3}{2}\lambda_S \sigma V sin^2\phi$.

## A.2 Internal Variables for meshes

- **MaxMeshNumber** Maximal number of finite element meshes stored. Must be set once before loading meshes.
- **Zone** Current Zone to be written into the TecPlot output file (double). Zone can be set before each output, or can be used with automatic increment.
- **Ls** inverse length scale 1/m. Internally $Ls^2$ is used. Not recommended to set directly. Rather use the command resize that changes Ls in a better understandable way.
- **ZoneIncrement** Automatic increment of zone (default=1.0).

## A.3 Internal variables for energy minimization

- **MaxRestarts** Maximum number of restarts during energy minimization.
- **MaxEnergyEvaluations** Maximum number of energy calculations during energy minimization (typical: 10000). Afterwards energy minimization is aborted.
- **ExchangeCalculator** Chooses the exchange energy discretization method used. The available choices are $1 = m\Delta m$, $2 = \phi^2$ along edges, $3 = (\nabla\theta)^2 + \sin^2(\theta)(\nabla\phi)^2$.
- **SwitchCount** In the default multi-axis minimization this sets how often the energy minimization switches between different coordinate axis and systems. The switchcount counts energy evaluations. Defaults to 300.
- **SwitchPercent** In the default multi-axis minimization this sets how often the energy minimization switches between different coordinate axis and systems. SwitchPercent counts energy evaluations as a percentage of the defined MaxEnergyEvaluations. Defaults to 101% (i.e. is switched off).

## A.4 Internal variables for energy barrier paths

- **PathN** Number of structures along the magnetization path. *Warning*: The mesh must have been defined previously – Use only after ReadMesh.
- **MaxPathEvaluations** Maximum number of path energy calculations during path minimization (typical: 2000).
- **NEBSpring** Spring constant for nudged-elastic band method (NEB) .
- **CurvatureWeight** Weight of curvature contribution for nudged-elastic band method (NEB).

# B Test routines for MERRILL

## B.1 Micromagnetic benchmarks

### B.1.1 MuMag #3

## B.2 Unit tests