MSD Software Documentation

Audiences: (1) End Users, and (2) Developers

# Contents:

## End Users:

1. Overview
   1. What is an MSD?
   2. What is the purpose of this software?
2. Setup
   1. Supported Platforms
   2. Download
   3. Install
   4. Update
3. Run
   1. File Structure
      1. Batch Files
      2. Parameter Files
      3. Folders
   2. Iterate
   3. Heat
   4. Magnetize
   5. Magnetize2
   6. Metropolis
      1. Extract
   7. Molecule Builder
      1. MMT Compiler
4. Technical Details
   1. Models
      1. MSD
      2. Molecule
   2. Equations
      1. Energy
      2. Other
   3. Algorithms
      1. Metropolis
5. Appendix
   1. Software License
   2. Versions
6. Acknowledgments
   1. Authors
   2. Co-Authors
   3. Special Thanks
      1. Funding
      2. Peers
      3. Open-Source Software
      4. Collaborators (Including internet forums)
   4. Related Academic Papers
   5. Sources

## Developers:

1. File Structure
2. Abstractions
   1. MSD
   2. Molecule (MolProto)
   3. Parameters & Results
   4. Iterators
   5. Vector
   6. FlippingAlgorithm
   7. Misc. Constants and Helper Functions
3. MSD Library
   1. udc.h
   2. MSD.h
   3. Vector.h
   4. SparseArray.h
   5. mmt.h
   6. MSD-export.h
   7. MSD.py
4. Compiling
5. Other Tools
   1. Build\*.bat
   2. mmb\_inspector
   3. Tests

# 1. Overview

## (1a) Purpose:

This software suite allows for the study of theoretical (simulated) behavior of **Molecular Spintronics Devices** (MSDs).

## (1b) Intended Audience:

The first part of this document (sections 1-5) are targeted towards “end-users”: e.g. students, engineers, or scientists.

The second part (sections 6-10) provide further details and documentation important for software developers.

## (1c) Background Information:

From Wikipedia, “**Spintronics**, also known as spin electronics, is the study of the intrinsic spin of the electron and its associated magnetic moment, in addition to its fundamental electronic charge, in solid-state devices.”

Traditional **Spintronics Devices** are created by layering two ferromagnets, polarized either parallel or antiparallel to each other, with a spacer or buffer layer between that prevents magnetic coupling. The resistance of the device is then measured and its strength will depend on the magnetic moment of the two ferromagnets. This effect is known as magnetoresistance.

Our lab’s novel **Molecular Spintronics Devices** (MSDs) are similar to a tradition spintronics devices; however, additional complex molecules are sandwiched between the ferromagnets creating only a partial spacer. Some coupling between the ferromagnets and molecules can occur.

The properties of such a device are experimentally varied, and largely dependent on the particular properties of the molecules and other materials used. Given the difficulty and expense required to fabricate a large quantity of various devices for study, theoretic predictions are a great asset. Moreover, by simulating such devices, information about the internal state of the device, data which would be difficult or impossible to measure, can be predicted leading to a deeper understanding of the observed macro-behaviors.

This software uses well know and empirically tested equations and models for calculating energy along with an algorithmic technique know as a Monte Carlo simulation. For more info, see section: [4. Technical Details](#_4._Technical_Details).

# 2. Setup

## (2a) Supported Platforms:

Currently the only platform supported entirely out-of-the-box is Windows (64-bit). The programs have been tested with the latest version of windows (Windows 11 at time of writing), but previous versions of Windows should work as well. There are x86 (Windows 32-bit) versions of some of the programs, but not all.

Although the software is currently targeting a Windows environment, it is written in cross-platform code and can be compiled for and run on any operating system.

## (2b) Download:

The latest version of the software can be downloaded from <https://github.com/Mathhead200/Molecular-Spintronics-Research-Project>. The GitHub archive also contains many previous versions of the software. Check the README file (and/or dates) to see which version is the most resent.

The folder suffixed “Latest Development Version” is the working folder for active development and generally not stable or ready for release. It may contain temporary development files, uncompiled code, broken code, bugs, inefficiencies, and other artifacts of the development process.

### (2c) Install:

There is currently no installer. Just download the desired version (i.e. folder) from GitHub or another distribution source, and unzip (if necessary). The folder titled “MSD Research Project (v\*.\*.\*), …” represents the root folder for the software.

## (2d) Update:

Periodically check the GitHub repository for updates. There is no official update schedule or automatic update tool at this time.

# 3. Run

## (3a) File Structure:

In the root project folder (denoted: ~), you will find a list of files and folders:

### Files:

* **~/change-log.txt**: This text file provides a detailed history of changes throughout the many versions of this software, including bug-fixes, optimizations, and additional functionality. This is the file to consult if you are not sure what version of the MSD software to use.
* **~/\*.bat**: These batch files are used to run the various C++ applications (apps) provided with this software. Further details about these apps will be provided later in this section.
* **~/parameters-\*.txt, ~/extract-input.txt**: These parameter (plain/text) files are used by their respective apps. Details about how to modify and use them will be provided later in this section along with their respective apps.
* **~/mmt\_compiler.exe**: This app compiles .mmt (MSD Molecule Text) files into .mmb (MSD Molecule Binary) files. This tool is used in conjunction with the mol-builder, and some of the provided apps. More details can be found later in this section.
* **mol-builder.html**: This web app is used to build custom molecules. Details on it’s use are given later in this section.

### Folders:

* **~/bin**: (*Not for end-users.)* These are where the compiled binaries are stored. More information for developers can be found in section [6. File Structure](#_6._File_Structure).
* **~/dev-tools**: (*Not for end-users.)* This folder contains miscellaneous utilities useful for developers.
* **~/docs**: This folder contains the current documentation for this software.
* **~/lib**: (*Not for end-users.)* This folder contains any ports or binding of the MSD software. More information for developers can be found in section [6. File Structure](#_6._File_Structure).
* **~/mol**: This folder is provided for end-users to store their custom molecules in. Its use is optional. More details about custom molecules are given later in this section.
* **~/out**: This folder is the default output location for apps. This is where the data from the various simulations will be stored when using the apps’ default configurations.
* **~/src**: (*Not for end-users.)* This folder contains almost all the source code for the core MSD libraries and apps. The only exceptions may be tools/libraries written in scripting languages stored directly in ~ or ~/lib. More details for developers in section [6. File Structure](#_6._File_Structure).

## (3b) Variables, Notation, and Symbols:

### (3b.i) Parameters:

The standard parameters for all simulation programs are listed here for reference. Note: Some apps may have specialized parameters. See an app’s specific section below for details on running that app.

* **width, height, depth** *(integers)* — Dimensions of the MSD. *(Non-negative.)*
  + width specifies the maximum x value and includes the total width of the left ferromagnet (L), the molecule (m), and the right ferromagnet (R).
  + height specifies the maximum y value and the height of the right ferromagnet.
  + depth specifies the maximum z value and the depth of the left ferromagnet.
* **molPosL, molPosR** *(integers)* — The position of the molecule region (m). *(Non-negative, zero is okay)*
  + molPosL specifies the least/first x value (inclusive) where the molecule region (m) starts. *(Minimum: 0, Maximum: width)*
  + molPosR specifies the greatest/last x value (inclusive) where the molecule region (m) ends. *(Minimum: 0, Maximum: width)*
* **topL, bottomL, frontR, backR** *(integers)* — The boundaries where the left (L) and right (R) ferromagnet regions start and end, as well as where the molecule (m) region connects at the cross section of the two ferromagnets. *(Non-negative, zero is okay)*
  + topL specifies the least/first y value (inclusive) where the left ferromagnet (L) starts. *(Minimum: 0)*
  + bottomL specifies the greatest/last y value (inclusive) where the left ferromagnet (L) ends. *(Maximum: height - 1)*
  + frontR specifies the least/first z value (inclusive) where the right ferromagnet (R) starts. *(Minimum: 0)*
  + backR specifies the greatest/last z value (inclusive) where the right ferromagnet (R) ends. *(Maximum: depth - 1)*
* **simCount** *(integer)* — How many iterations should the simulation run for (per model). May have a slightly different interpretation for each app. Check an app’s specific details int the sub-sections below for more details. *(Non-negative, zero is okay)*
* **t\_eq** *(integer)* — Time to equilibrium. How many iterations should be run (per model) before recording data to allow the system to first reach equilibrium. *(Non-negative, zero is okay)*
* **freq** *(integer)*— Frequency, i.e., how often to record data. A frequency of 0 means no data will be recorded. Note: small frequencies (e.g., 1) may cause the app to use all available computer memory. Each recording takes about 1kB of RAM. *(Non-negative, zero is okay)*
* **kT** *(real)* — Temperature, multiplied by the Boltzmann constant k. kT = 1 corresponds roughly to a material’s Currie temperature. *(A positive, non-zero, number; Default: 0.25)*
* **B** (vector) — An external magnetic field being applied to the system. *(Default: zero vector)*
* **S** *(real)* — Spin Magnitude. All spins in the section will have the constant magnitude S throughout the simulation. *(A non-negative number, zero is okay; Default: 1)*
* **F** *(real)* — Maximum Flux Magnitude. All fluxes in the section will have a random real magnitude between 0 (inclusive) and F (exclusive). *(A non-negative number; Default: 0)*
* **J** *(real)* — Heisenberg Exchange Coupling coefficient. This effects the interaction between nearest neighboring spins. Note: JLR represents the direct coupling (i.e., tunneling) across the molecule. *(Defaults: JL, JR, Jm = 1; JmL, JmR = ±1, JLR = 0)*
* **Je0** *(real)* — Heisenberg Exchange Coupling coefficient. This effects the interaction between a spin and flux at the same location. *(Default: 0)*
* **Je1** *(real)* — Heisenberg Exchange Coupling coefficient. This effects the interaction between a spin and its nearest neighboring flux. (Default: 0)
* **Jee** *(real)* — Heisenberg Exchange Coupling coefficient. This effects the interaction between nearest neighboring fluxes. (Default: 0)
* **b** *(real)* — Biquadratic Coupling coefficient. This effects the interaction between nearest neighboring local magnetizations (i.e., spin + flux). *(Default: 0)*
* **A** *(vector)* — Anisotropy. This effects each local magnetization (i.e., spin + flux) locally. *(Default: zero vector)*
* **D** *(vector)* — Dzyaloshinskii-Moriya (i.e., Skyrmion) Interaction. This effects the interaction between nearest neighboring local magnetizations (i.e., spin + flux). This is the only non-communitive coupling affect. *(Default: zero vector)*
* **seed** *(integer)* — Seed for the pseudo-random number generator (PRNG) used by the model. This seed may be specified as an input parameter to some apps so that the results of the simulation are deterministic. Or a (weakly) unique seed may be automatically generated.

### (3b.ii) Output Variables:

The standard outputs variables for all simulation programs are listed here for reference. Note: some apps may have specialized outputs. See an app’s specific section below for details on running that app. See section [4. Technical Details](#_4._Technical_Details_1) for more information on the exact algorithms used to calculate this output variables.

* **t** *(integer)* — Time. This variable is attached to many output records. It represents the “time”, or more specifically the iteration count, when each result was calculated. Only used for results which are a function of time. *(Minimum: 0)*
* **x, y, z** *(integers)* — Position. This variable is attached to some local output variables. Represents the position of the local result. Constraints:
  + If x < molPolL, position is in L.
  + If x > molPosR, position is in R.
  + If molPosL ≤ x ≤ molPosR, position is in m.
  + If position is in L, topL ≤ y ≤ bottomL.
  + If position is in R, frontR ≤ z ≤ backR.
  + If position is in m, then
    - both molPosL ≤ x ≤ molPosR,
    - and also either y = topL or bottomL, or z = frontR or backR.
* **U** *(real)* — Internal Energy as a function of time (t). This is the primary calculation in the metropolis algorithm. The interaction between energy (U) and external temperature (kT) determines how the model mill stabilize, if at all. *(Maximum: 0)*
* **M** *(vector)* — Magnetization as a function of time (t). Represents the aggregate magnetic moment of the system (i.e., sum of all local magnetic moments, m). May be suffixed with an S (i.e., MS) when only the spins should be considered (excludes flux vectors), or F (i.e., MF) for only fluxes (excludes spin vectors).
* **s** *(vector)* — Spin as a function of position (x, y, z) and time (t). Represents only the spin component of this position’s local magnetization.
* **f** *(vector)*— Flux as a function of position (x, y, z) and time (t). Represents only the flux (i.e., free electron) component of this position’s local magnetization.
* **m** (vector) — Local magnetization as a function of position (x, y, z) and time (t). Represents the total magnetic moment of this position (i.e., ).
* **n** (integer) — The number of atoms/nodes in the MSD, or in a region of the MSD if a region suffix is present. For n\_mL, n\_mR, and n\_LR; the number of atoms is defined as twice the number of bonds/edges between the two respective regions. (i.e., the number of boundary atoms/nodes in both sections.)
* **FM\_L\_exisits, FM\_R\_exists, mol\_exists** *(boolean)* — True if the given region has any atoms/nodes. False otherwise. A region may fail to exist if certain conditions are met on the various MSD dimension parameters.
  + If molPosL = 0, FM\_L\_exists=False.
  + If molPosR = width, FM\_R\_exists=False.
  + If molPosR < molPosL, mol\_exists=False.
* **seed** *(integer)* — Seed for the pseudo-random number generator (PRNG) used by the model. If a (weakly) unique seed was automatically picked, this output variable shows which seed was pick so that repeat simulations can be performed deterministically.
* **c** *(real)* — Specific Heat as a function over a time interval.
* **x** *(real)* — Magnetic Susceptibility as a function over a time interval.
* **<M>** (vector) — Mean M as a function over a time interval.
* **<U>** *(real)*— Mean U as a function over a time interval.

### (3b.iii) Regions:

Most (but not all) parameters and output variables are suffixed with one of the following region specifiers: L, R, m, mL, mR, LR. Variables which are not suffixed by a region specified generally affect the entire system.

* **L** — Left Ferromagnet. Sometimes referred to as FML. ()
* **R** — Right Ferromagnet. Sometimes referred to as FMR. ()
* **m** — Molecule. Sometimes referred to as “mol”. ()
* **mL** — Used for the interactions between the molecule (m) and left ferromagnet (L).
* **mR** — Used for the interactions between the molecules (m) and right ferromagnet (R).
* **LR** — Used for the direct coupling (i.e., tunneling) interactions between the left (L) and right (R) ferromagnets.

If L, R, or m is used for a coupling interaction, then both vectors are in the same region. mL, mR, and LR are not used for local affects, only for interactions between two vectors.

## (3d) Iterate:

Iterate is one of the included apps. Its purpose is to take a single MSD configuration, run it a for some number of iterations, then output the results. The details of the model and algorithms being used can be found in the next section, [4. Technical Details](#_4._Technical_Details_1).

### (3d.i) Iterate — Parameters

Iterate notably does not take a t\_eq parameter. t\_eq is assumed to be 0.

Iterate can take input from a parameter file (e.g., ~/parameters-iterate.txt). Each line of the parameter file can contain

1. A parameter definition in the form: parameter\_name = value
2. A line in the form [x y z] = S, where the spin of a specific location can be modified, or even “turned off” (S = 0). Note: as of version 6.2.2, this only modifies the spin vector. Flux vectors will stay be based on the appropriate F parameter based on region.
3. A comment: line should start with the # (pound/hash) character
4. Only Whitespace: blank lines are ignored.

### (3d.ii) Iterate — Run

Once the iterate parameter file has been set up, the program can be run by executing the file: ~/iterate.bat

If any parameters were omitted in the parameter file, or if no file was given, the user will be prompted for the missing values before the simulation can begin. If any extraneous or unrecognized parameters were given in the parameter file, a warning will be displayed to the user listing the “unused” parameters, but the program will continue, ignoring these unused values.

### (3d.iii) Iterate — Configure

The ~/iterate.bat batch file may be edited in order to modify a few extra program parameters. Open the batch with a text editor. An “Edit Here” section will be located near the top of the batch file.

@rem // ---- Edit Here ----

@set model=CONTINUOUS\_SPIN\_MODEL

@set mol\_type=LINEAR

@set randomize=1

@set seed=unique

The **model** parameter (i.e., batch variable) controls the algorithm used when “flipping” a spin. The only currently maintained value for this parameter is CONTINUOUS\_SPIN\_MODEL. UP\_DOWN\_MODEL is defined for backwards compatibility with previous studies, but little work has been done to maintain its behavior other then checking that it runs.

* CONTINUOUS\_SPIN\_MODEL: Spin vectors can point in any direction.
* UP\_DOWN\_MODEL: Spin vectors can only point up or down (parallel to the y-axis).

**mol\_type** has currently has two options. Either a built-in Molecule “Type” can be used (currently the two options are LINEAR and CIRCULAR), or a file path can be provided which points to an MMB file. More information about custom molecules, MMT files, and MMB files can be found in section [(3i) Molecule Builder](#_(3i)_Molecule_Builder).

* LINEAR: The molecule will consist of a linear string of nodes with leads on either end. Parameters are uniform in mol.
* CIRCULAR: If n ≥ 2, the molecule will consist of a circular string of nodes, with leads (approx.) 180 degrees apart. Parameters will be uniform in mol. If n = 0, 1; same as LINEAR.

**randomize** is a boolean parameter. If set to 1 (on), the model’s state will be randomized at the start of every simulation. If 0 (off), the model will be initialized to the default state: all spins: up; all fluxes: zero.

**seed** can either be set to “unique” which allows the program to automatically generate a (weakly) unique seed. Or an integer can be provided which will act as the seed.

### (3d.iv) Iterate — Output

After Iterate finished, its output can be found in a CSV file in the ~/out folder. The name of the file will contain the name of the app, iterate, the date, and a sequential number identifier (starts at 1 and counts up) to avoid name conflicts.

The CSV file is broken into three sections:

**Columns 1-93** (or “A” thru “CO” in Excel) contain record data. These are macro-reading about the system as a whole and its regions over time, t (measured in iterations). For more information about each specific output variable, check the earlier sub-section [3c. Output Variable](#_(3c)_Output_Variables:).

**Columns 96-107** (or “CR” thru “DC” in Excel) contain a snapshot of the final state of the system. These are micro-readings which show the internal magnetization of the system at a specific fixed moment in time, t = simCount.

Finally, **columns 110-169** (or “DF” thru “FM” in Excel) contain the input parameters for this simulation, as well as the PRNG seed, and MSD model version.

### (3d.v) Iterate — Errors

<To-do>

## (3e) Heat

Heat is one of the included apps. Its purpose is to run a single model through multiple simulations at varying “heat” (kT) levels.

### (3e.i) Heat — Parameters

Unlike [Iterate](#_(3d)_Iterate:) and [Metropolis](#_(3h)_Metropolis), Heat does not take a parameters file. All parameters are entered at the command line. The user will be prompted to enter them one at a time after starting the program.

There are three unique parameters specific to the Heat app:

* kT\_min: Minimum (inclusive) kT value for the simulation.
* kT\_max: Maximum (inclusive) kT value for the simulation.
* kT\_inc: Increment or step size between kT values. This value may be positive or negative. Negative values will cause the simulation to start at the kT\_max, and “cool” to kT\_min.

e.g., kT\_min = 0.3, kT\_max = 1, kT\_inc = 0.1 would simulate for the following set of kT values: 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0.

kT\_min ≤ kT\_max, and kT\_inc ≠ 0.

In version 6.2.2, Heat has 58 parameters. The order is as follows:

(1-3) width, height, depth;

(4-5) molPosL, molPosR;

(6-9) topL, bottomR, frontR, backR;

(10-12) t\_eq, simCount, freq;

(13-15) kT\_min, kT\_max, kT\_inc;

(16) B;

(17-22) SL, SR, Sm, FL, FR, Fm;

(23-28) JL, JR, Jm, JmL, JmR, JLR;

(29-31) Je0L, Je0R, Je0m;

(32-37) Je1L, Je1R, Je1m, Je1mL, Je1mR, Je1LR;

(38-43) JeeL, JeeR, Jeem, JeemL, JeemR, JeeLR;

(44-46) AL, AR, Am;

(47-52) bL, bR, bm, bmL, bmR, bLR;

(53-58) DL, DR, Dm, DmL, DmR, DLR.

For the above parameters which are vectors, 3 values (x, y, z) should be enter on the same line separated by spaces: e.g., B = 1.2 0 -0.25 would specify that the external magnetic field is a vector with x=1.2, y=0, and z=-0.25.

Each line expects just one parameters value (or 3 for vectors).

If no (or not enough) values are given on a line, the program will expect the missing value(s) to be entered on the subsequent line.

If too many values are entered on a line, the extra values will be queued, and automatically entered into the subsequent fields until the queued values are exhausted. By using this feature, it is possible to enter all the parameter values onto the first line, or to feed them in through STDIN redirection.

### (3e.ii) Heat — Run

To run the Heat app, execute the batch file: ~/heat.bat

As detailed above, the user will be prompted to enter all parameters.

### (3e.iii) Heat — Configure

As with [Iterate](#_(3d.iii)_Iterate_—), the Heat batch file contains a few extra program parameters that can be modified. Open the ~/heat.bat file with a text editor. Near the top, there is an “Edit Here” section.

@rem // ---- Edit Here ----

@set model=CONTINUOUS\_SPIN\_MODEL

@set reset=noop

@set mol\_type=LINEAR

The **model** parameter acts the same as in [Iterate](#_(3d.iii)_Iterate_—); it controls the algorithm used when “flipping” a spin.

**reset** is a unique parameter to Heat and [Magnetize](#_(3f)_Magnetize). It has three options:

* **noop**: No operation is performed when kT changes. The system will remain in the final state of the previous kT simulation. This preserves continuity between each step of the heating/cooling process.
* **reinitialize**: The system is reinitialized to the default “all up” state when kT changes. This breaks continuity between each step of the heating/cooling process.
* **randomize**: The system is randomized when kT changes. This breaks continuity between each step of the heating/cooling process.

**mol\_type** acts the same as in Iterate; it controls the type of molecule used, either LINEAR or CIRCULAR. Custom molecules input via MMT or MMB files are not supported in Heat yet as of version 6.2.2.

### (3e.iv) Heat — Output

The output for Heat will be stored in the ~/out folder in a CSV file titled with “Heat”, the (start) date of the simulation, and a number identifier (for uniqueness). This is the same output process as [Iterate](#_(3d.iv)_Iterate_—), Magnetize, and Magnetize2.

The output can be broken into three sections:

**Columns 1-106** (or “A” thru “DB” in Excel) contain aggregate results calculated over a range of t values from t = t\_eq to t = t\_eq + simCount. Each row/record represents a different simulation run at the given kT value. (See section [3c. Output Variables](#_(3c)_Output_Variables:) for more details on each variables.)

**Columns 108-198** (or “DD” thru “GP” in Excel) contain final results calculated at the specific final time in each simulation, t = t\_eq + simCount.

Finally, **Columns 200-259** (or “GR” thru “IY” in Excel) contains the input parameters used, as well as the PRNG seed, and MSD model version.

### (3e.v) Heat — Errors

<To-do>

## (3f) Magnetize

Magnetize is one of the included apps. The purpose of this app is to study the effects of a discretely changing external magnetic field, B. For a continuously changing magnetic field, see [3g. Magnetize2](#_(3g)_Magnetize2).

### (3f.i) Magnetize — Parameters

Like Heat, Magnetize has no parameters file. All parameters must be entered on the command line (i.e., standard input).

Magnetize has five unique parameters:

* B\_min *(real)*: The minimum magnitude (i.e., rho) of the external magnetic field, B. This is a single real value, *not* a vector like B. The direction of the vector will be entered later as B\_theta and B\_phi. B\_min can be (and often is) negative, which starts the vector anti-parallel.
* B\_max *(real)*: The maximum magnitude (i.e., ρ = |B|) of the external magnetic field, B. Like B\_min, this is a single real value, and can be negative. *(B\_min ≤ B\_max)*
* B\_inc *(real)*: The increment or step between discrete B, values. *(B\_inc > 0)*
* B\_theta (angle, degrees): The θ angle of the vector on the xy-plane measure in degrees. 0° corresponds to the positive x-axis, and 90°: the positive y-axis. *(0 ≤ B\_theta < 360)*
* B\_phi *(angle, degrees)*: The φ angle of the vector off the xy-plane measured in degrees. 0° corresponds to a 0° angle between the vector and the xy-plane; i.e., the vector is on the xy-plane; Bz = 0. 90° corresponds to the vector being parallel to the positive z-axis. -90° corresponds to the vector being parallel to the negative-z axis. If B\_phi = ±90, B\_theta does not matter. *(-90 ≤ B\_phi ≤ 90)*

In version 6.2.2, Magnetize has 60 parameters. The order is as follows:

(1-3) width, height, depth;

(4-5) molPosL, molPosR;

(6-9) topL, bottomR, frontR, backR;

(10-12) t\_eq, simCount, freq;

(13) kT;

(14-18) B\_min, B\_max, B\_inc, B\_theta, B\_phi;

(19-24) SL, SR, Sm, FL, FR, Fm;

(25-30) JL, JR, Jm, JmL, JmR, JLR;

(31-33) Je0L, Je0R, Je0m;

(34-39) Je1L, Je1R, Je1m, Je1mL, Je1mR, Je1LR;

(40-45) JeeL, JeeR, Jeem, JeemL, JeemR, JeeLR;

(46-48) AL, AR, Am;

(49-54) bL, bR, bm, bmL, bmR, bLR;

(55-60) DL, DR, Dm, DmL, DmR, DLR.

### (3f.ii) Magnetize — Run

Run Magnetize by executing the ~/magnetize.bat batch file. As detailed above, the user will be prompted to enter all parameters.

### (3f.iii) Magnetize — Configure

As with [Iterate](#_(3d.iii)_Iterate_—) and [Heat](#_(3e.iii)_Heat_—), Some extra configuration options are present in ~/magnetize.bat

To configure, open the batch file with a text editor. Near the top, there will be an “Edit Here” section:

@rem // ---- Edit Here ----

@set model=CONTINUOUS\_SPIN\_MODEL

@set reset=noop

@set mol\_type=LINEAR

The **model** parameter is the same as [Iterate](#_(3d.iii)_Iterate_—) and [Heat](#_(3e.iii)_Heat_—). Either CONTINUOUS\_SPIN\_MODEL or UP\_DOWN\_MODEL.

**reset** can take one of 3 values (same as in [Heat](#_(3e.iii)_Heat_—)):

* **noop**: No operation is performed when B changes. The system will remain in the final state of the previous B-value simulation. This preserves continuity between each step as the external magnetic field changes.
* **reinitialize**: The system is reinitialized to the default “all up” state when B changes. This breaks continuity between the discrete steps as the external magnetic field changes.
* **randomize**: The system is randomized when B changes. This breaks continuity between the discrete steps as the external magnetic field changes.

**mol\_type** acts the same as in [Heat](#_(3e.iii)_Heat_—) and [Iterate](#_(3d.iii)_Iterate_—); it controls the type of molecule used, either LINEAR or CIRCULAR. Custom molecules input via MMT or MMB files are not supported in Heat yet as of version 6.2.2.

### (3f.iv) Magnetize — Output

The output for Magnetize will be stored in the ~/out folder in a CSV file titled with “Magnetize”, the start date of the simulation, and a number identifier (for uniqueness). This is the same output process as [Iterate](#_(3d.iv)_Iterate_—), [Heat](#_(3e.iv)_Heat_—), and Magnetize2.

The output can be broken into three sections:

**Columns 1-109** (or “A” thru “DE” in Excel) contain aggregate results calculated over a range of B values from t = t\_eq to t = t\_eq + simCount. Each row/record represents a different simulation run at the given B value. (See section [3c. Output Variables](#_(3c)_Output_Variables:) for more details on each variable.)

**Columns 111-201** (or “DG” thru “GS” in Excel) contain final results calculated at the specific final time in each simulation, t = t\_eq + simCount.

Finally, **Columns 203-267** (or “GU” thru “JG” in Excel) contains the input parameters used, as well as the PRNG seed, and MSD model version.

### (3f.v) Magnetize — Errors

<To-do>

## 

## (3g) Magnetize2

Magnetize2 is one of the included apps. The purpose of this app is to study the effects of a continuously changing external magnetic field, B. For a discretely changing magnetic field, see [3f. Magnetize](#_(3f)_Magnetize).

A continuously changing external magnetic field is approximated by modifying B in very small increments of B\_rate (i.e., dB/dt) at each step in the simulation. Instead of allowing the model to stabilize at each new B value, the magnetic field is continuously changed on the same model, traveling from B\_max to B\_min and back. This can create a hysteresis loop.

### (3g.i) Magnetize2 — Parameters

Like [Heat](#_(3e.i)_Heat_—) and [Magnetize](#_(3f.i)_Magnetize_—), Magnetize2 has no parameters file. All parameters must be entered on the command line (i.e., standard input).

Magnetize has five unique parameters:

* B\_min *(real)*: The minimum magnitude (i.e., ρ = ±|B|) of the external magnetic field, B. This is a single real value, *not* a vector like B. The direction of the vector will be entered later as B\_theta and B\_phi. B\_min can be (and often is) negative, which starts the vector anti-parallel.
* B\_max *(real)*: The maximum magnitude (i.e., ρ = ±|B|) of the external magnetic field, B. Like B\_min, this is a single real value, and can be negative. *(B\_min ≤ B\_max)*
* B\_rate *(real)*: The continuous rate of change, dB/dt. This is normally a small value (e.g., 1e-6) since it will be applied for every step in the simulation. The effective simCount on one pass from B\_min to B\_max is (B\_max – B\_min) / B\_rate. So B\_rate and simCount are inversely proportional. This one-pass simCount is multiplied by either 2 (or 2.5 depending on the configuration) since this app moves B in both directions. *(B\_rate > 0)*
* B\_theta (angle, degrees): The θ angle of the vector on the xy-plane measure in degrees. 0° corresponds to the positive x-axis, and 90°: the positive y-axis. *(0 ≤ B\_theta < 360)*
* B\_phi *(angle, degrees)*: The φ angle of the vector off the xy-plane measured in degrees. 0° corresponds to a 0° angle between the vector and the xy-plane; i.e., the vector is on the xy-plane; Bz = 0. 90° corresponds to the vector being parallel to the positive z-axis. -90° corresponds to the vector being parallel to the negative-z axis. If B\_phi = ±90, B\_theta does not matter. *(-90 ≤ B\_phi ≤ 90)*

There is no simCount parameter in Magnetize2 since the simCount is determined by B\_rate. Otherwise, the parameters are in the same order as [Magnetize](#_(3f.i)_Magnetize_—) except B\_inc is replaced with B\_rate.

The t\_eq parameter, will cause the model to run for the given number of iterations (may be 0) in order to reach equilibrium at the initial B value before starting the simulation.

The freq parameter is how often a reading will be taken (every freq iterations) during the simulation.

In version 6.2.2, Magnetize has 59 parameters. The order is as follows:

(1-3) width, height, depth;

(4-5) molPosL, molPosR;

(6-9) topL, bottomR, frontR, backR;

(10-11) t\_eq, freq;

(12) kT;

(13-17) B\_min, B\_max, B\_rate, B\_theta, B\_phi;

(18-23) SL, SR, Sm, FL, FR, Fm;

(24-29) JL, JR, Jm, JmL, JmR, JLR;

(30-32) Je0L, Je0R, Je0m;

(33-38) Je1L, Je1R, Je1m, Je1mL, Je1mR, Je1LR;

(39-44) JeeL, JeeR, Jeem, JeemL, JeemR, JeeLR;

(45-47) AL, AR, Am;

(48-53) bL, bR, bm, bmL, bmR, bLR;

(54-59) DL, DR, Dm, DmL, DmR, DLR.

### (3g.ii) Magnetize2 — Run

Run Magnetize2 by executing the ~/magnetize2.bat batch file. As detailed above, the user will be prompted to enter all parameters.

### (3g.iii) Magnetize2 — Configure

As with [Iterate](#_(3d.iii)_Iterate_—), [Heat](#_(3e.iii)_Heat_—), and [Magnetize](#_(3f.iii)_Magnetize_—), some extra configuration options for Magnetize2 are present in ~/magnetize2.bat

To configure, open the batch file with a text editor. Near the top, there will be an “Edit Here” section:

@rem // ---- Edit Here ----

@set model=CONTINUOUS\_SPIN\_MODEL

@set randomize=0

@set startAtMaxB=0

@set mol\_type=LINEAR

The **model** parameter is the same as in [Iterate](#_(3d.iii)_Iterate_—), [Heat](#_(3e.iii)_Heat_—), and [Magnetize](#_(3f.iii)_Magnetize_—). Either CONTINUOUS\_SPIN\_MODEL or UP\_DOWN\_MODEL.

**randomize** is a boolean parameter (same as in [Iterate](#_(3d.iii)_Iterate_—)) and can either be 0 or 1:

* **1** (on): the model’s state will be randomized at the start.
* **0** (off): the model will be initialized to the default state: all spins: up; all fluxes: zero.

**startAtMaxB** is a boolean parameter and can either be 0 or 1:

* **1** (yes): The simulation will start with Bρ = B\_max. The simulation will then precede to make two complete passes, first decreasing Bρ from B\_max to B\_min, then increasing Bρ back again from B\_min to B\_max.
* **0** (no): The simulation will start with Bρ = 0. The simulation will start by making a half-pass increasing Bρ from 0 to B\_max. The simulation then proceeds two make two complete passes as described above (same as if startAtMaxB=1). In this mode, startAtMaxB=0, the simCount will include a total of 2.5 passes instead of the 2.0 if startAtMaxB=1.

**mol\_type** acts the same as in [Iterate](#_(3d.iii)_Iterate_—), [Heat](#_(3e.iii)_Heat_—), and [Magnetize](#_(3f.iii)_Magnetize_—); it controls the type of molecule used, either LINEAR or CIRCULAR. Custom molecules input via MMT or MMB files are not supported in Heat yet as of version 6.2.2.

### (3g.iv) Magnetize2 — Output

The output for Magnetize2 will be stored in the ~/out folder in a CSV file titled with “Magnetize2”, the start date of the simulation, and a number identifier (for uniqueness). This is the same output process as [Iterate](#_(3d.iv)_Iterate_—), [Heat](#_(3e.iv)_Heat_—), and [Magnetize](#_(3f.iv)_Magnetize_—).

The output can be broken into two sections:

**Columns 1-96** (or “A” thru “CR” in Excel) contain the results calculated at a specific time in each simulation. Since B is a function of t in this app, B is output, but not t. Each row/record represents the same simulation at different times. (See section [3c. Output Variables](#_(3c)_Output_Variables:) for more details on each variable.)

**Columns 98-162** (or “CT” thru “FF” in Excel) contains the input parameters used, as well as the PRNG seed, and MSD model version.

### (3g.v) Magnetize2 — Errors

<To-do>

## (3h) Metropolis

Metropolis is one of the included apps. (It’s named after the algorithm this software uses.) The purpose of this app is varying any number of parameters over many parallel simulations. This app supports native multi-threading, and is more complex than the other simulation apps: [Iterate](#_(3d)_Iterate:), [Heat](#_(3e)_Heat), [Magnetize](#_(3f)_Magnetize), or [Magnetize2](#_(3g)_Magnetize2_1).

### (3h.i) Metropolis — Parameters

The parameters for the Metropolis app must be enter through a parameters file:

~/parameters-metropolis.txt

If parameters are missing, the program will not run. (This be haviour is different from Iterate where the missing parameter values will be prompted.)

The parameter file format is similar to [Iterate](#_(3d.i)_Iterate_—), but with more features. On each line you can type one of the following:

* A constant parameter can be specified with parameter\_name label = value.
* A variable parameter can be specified with a range: parameter\_name label = start\_value end\_value inc where
  + label: An optional label/name. Two variable parameters sharing the same name will vary together, whereas parameters without a name or with different names will vary independently. Multiple parameters varying independently increases the number of parameter combination exponentially.
  + start\_value: The first value for the given parameter.
  + end\_value: The last value for the given parameter.
  + inc: An increment. i.e., What steps to iterate by when counting from start\_value to end\_value.
* A variable parameter can also be specified with an enumeration: parameter\_name label { first\_value, second\_value, …, last\_value }

e.g., kT { 0.1 0.3 1.0 1.1 } will cause kT to have 4 different values.

There must be space(s) around the { braces }

* A line in the form [x y z] = S, where the spin of a specific location can be modified, or even “turned off” (S = 0). Note: as of version 6.2.2, this only modifies the spin vector. Flux vectors will stay be based on the appropriate F parameter based on region.
* # A line starting with a pound sign is a comment and will be ignored.
* Blank lines are ignored.

Unlike other apps, vector parameters are entered on three separate lines, e.g.,

B\_x = 1

B\_y \_B : -1.0 1.0 0.1

B\_z { 0 1 -1 }

The \_x, \_y, and \_z suffixes are used to specify each component of a vector value. This is done so that different components of the vector can be iterated independently or together.

### (3h.ii) Metropolis — Run

To run Metropolis, execute the ~/metropolis.bat file. The program will use the parameters found in ~/parameters-metropolis.txt as described above. If any parameters are missing, or there is an error in the parameters file, the program will not execute, and an error message will be displayed.

### (3h.iii) Metropolis — Configure

As with the other apps, the Metropolis batch file has extra configuration options. Open ~/metropolis.bat with a text editor. Near the top, there will be an “Edit Here” section:

@rem // ---- Edit Here ----

@set model=CONTINUOUS\_SPIN\_MODEL

@set mode=RANDOMIZE

@set mol\_type=LINEAR

@set threadCount=3

**model**, as with the other apps can either be CONTINUOUS\_SPIN\_MODEL or UP\_DOWN\_MODEL.

**mode** has two options:

* RANDOMIZE: The model should start with an initial random state; all spin and flux vectors randomized.
* REINITIALIZE: The model should be default initialized to the “all up” state at the start of each simulation.

**mol\_type**, same as the other apps, has two options: LINEAR and CIRCULAR. As of version 6.2.2, Metropolis doesn’t yet support custom MMT or MMB files.

**threadCount** is an integer which specifies the number of processor (CPU) threads this app can use for simulations. Since this app runs many independent simulations with different parameter combinations, these simulations can be run concurrently to save time.

* If threadCount=1, the app will run in single threaded mode. Each simulation will be run in sequence on the main thread.
* If threadCount>1, a number of concurrent simulation equal to the given thread count will run. In multi-threaded mode, the main thread is not counted in the thread count since its CPU time is negligible.
* If threadCount is omitted (or set equal to nothing), the program will run with a default number of threads, and a warning will be displayed.

### (3h.iv) Metropolis — Output

Metropolis stores its output in the ~/out folder in XML format. (This is different from the CSV format used by other apps.) The XML format stores information hierarchically, which is different than the table-based format of CSV.

There is a utility, ~/extract.bat, which allows data from the CSV file to be extracted into a CSV file.

This program takes as input:

1. An XML file in the Metropolis output format (msd.dtd). This would include any file output from the Metropolis app.
2. A list of variables (input parameters and output variables) which are to be extracted from the XML. (e.g., You might list kT, M\_x, M\_y, M\_z here if you wanted to see a graph/table of M vs kT.) This list can have any number of variables. You move on to the next input section by entering a blank line.
3. A list of “filters”. A filter is made up of three parts:
   1. A parameter/output variable name
   2. A comparison operator: =, !=, >, <, >=, <=

( == may be used in place of = )

* 1. A numeric value

e.g., kT = 0.1 would only extract data from simulations where kT = 0.1. Note: there must be spaces between the variable, operator, and value.

Again, a blank line moves to the next section.

1. A filename (or file path) for storing the output CSV file. The filename should have a .csv extension.

~/extract.bat can be configured such that its input is entered either on the command or from a text file. To change the configuration, open the batch file in a text editor. The batch file is very simple. The first line (as of v6.2.2) should contain either one of the following:

1. bin\extract <extract-input.txt
2. bin\extract

In mode 1, the < less-than symbol tells the operating system (e.g., Windows) to use a text file instead of the command prompt for input. In the above example, that text file is called “extract-input.txt”. A text file with this name should be in the ~ directory. This mode is useful if you want to extract lots of variables multiple times, and don’t want to keep retyping the information into the extract program. Pay careful attention to type to info into the extract-input.txt file in exactly the way you would the command prompt, including spaces, and line-breaks. Unlike with the “parameters” files, prompts should not be included in the extract-input.txt file.

In mode 2, the extract app will prompt for data from the command line, and not read any input file. This is the recommended mode for your first use of this tool.

### (3h.v) Metropolis — Errors

<To-do>

## (3i) Molecule Builder

Molecule Builder, unlike the other apps, is not used to run simulations. It is a utility to build custom molecule prototypes.

### (3i.i) Molecule Builder — Run

To open the Molecule Builder tool, open the ~/mol-builder.html file in a web browser. (On most systems, HTML files open with a web browser by default.) You will be greeted with a web app (an app running as a web page in your web browser).

There are four sections to the app (as of version 6.2.2):

1. The main section is the **canvas**: a blank white square with a red “x” in the corner. This is where you will build your molecule visually.
2. The **config** section can be found above the canvas. It contains buttons for saving your molecule, starting over (clear), and changing the size of the canvas. (Note: the “load” option is unfinished as of version 6.2.2)
3. The **form** section, or editor pane, can be found either to the right side or below the canvas (depending on the size of your window/screen). It looks like a rectangle with rounded corners. By default, there will be nothing displayed there except a placeholder, “(blank)”. This area is used to display and edit to specific properties (i.e., parameters or variables) of each component (nodes and edges) of the molecule.
4. Lastly, the **instructions** section can be found at the bottom of the page, below the canvas. Please refer to those instructions of this document when there is an inconsistency as they will also be up to date.

Once pressing the save button, your molecule will be converted into MMT (MSD Molecule Text), a format for storing custom MSD molecule prototypes as text. You will see this text appear in the form section, along with a download link which saves the MMT to a file on your computer.

The MMT data can be edited dynamically in the form section, and, if valid, will update the molecule in the canvas. However, these changes will not be saved unless the save button is clicked again.

Saving a molecule means that the next time you open the Molecule Builder, the previously saved molecule prototype will be loaded onto the canvas automatically.

To clear a previously saved molecule, press the clear button, and confirm. The canvas will be cleared; however, the cleared canvas will not be saved unless you also press the save button. If you simply refresh the browser (or reopen the tool) after clearing, but without saving, the previously cleared molecule will still be reloaded.

### (3i.ii) Molecule Builder — Output

Once you have built a molecule prototype, the molecule can be saved as a .mmt file. It is recommended that these .mmt files be stored in the provided ~/mol folder for organizational purposes, but it’s not required.

These files are in a text format so that they can be easily read and edited by humans. However, to use them in a simulation, they first need to be compiled, i.e., converted into a binary format: MMB (MSD Molecule Binary).

To compile a .mmt file, open it with the provided MMT Compiler: ~/mmt\_compiler.exe

To use the MMT Compiler, simply open any valid .mmt file with ~/mmt\_compiler.exe. On Windows you can do this by

1. Right clicking the .mmt file
2. Selecting the “Open with” option
3. Selecting “Choose another app”
4. Selecting “More apps ↓”
5. Selecting “Look for another app on this PC” at the bottom, below the list
6. Navigating to the MSD Research Project folder you are using
7. Selecting the file mmt\_compiler.exe

Note: once you have done this process once, Windows will remember the mmt\_compiler app, and you will see it listed under “Open with” after step 2. You can simply click it from there.

Moreover, after step 3, you can check the box “Always use this app to open .mmt files” before selecting mmt\_compiler. If you do this, Windows will run the compiler when ever you open (i.e., double click) on a .mmt. But you may instead wish to make a text editor (e.g., notepad) the default program for opening .mmt files in case you find yourself regularly opening them to check or edit your molecule details.

Note: if you make changes to a .mmt file with a text editor, you must save and recompile the file into a new .mmb file for those changes to be reflected into the MMB file. If you already have a compiled .mmb file, and you recompile the .mmt file with the same name, a new .mmb file will silently overwrite the old file. If you do not want this to happen, change the name either the MMB file or the MMT file to have different names. The name of an MMB file will always be the same as the MMT file name when it was compiled. You can safely change the name of an MMB file after it is created, however. The filename does not affect the molecule data.

The MMB (not the MMT) is the file that is used by apps like [Iterate](#_(3d.iii)_Iterate_—).

# 4. Algorithms

The algorithms, including models and equations, used by this MSD simulation software are based on the Heisenberg model, which is itself based on the Ising model, for ferromagnets.

## (4a) Current Models

### (4a.i) 1D Ising Model

Early versions of this software started as a simple 1D Ising model. In this model, atoms in a ferromagnet are represented as nodes or vertices in an array (i.e., a 1D lattice). Each node has a “spin” which can be either up (+1 or +ĵ) or down (-1 or -ĵ).

Energy between the nodes is calculated with the equation:

where *J* is the Heisenberg exchange coupling constant, |*i* - *j*| is the distance between position *i* and *j*, and *si* is the spin of the *i*th atom in the array. i.e., The energy is proportional to the negative sum of products of adjacent spins.

In the traditional Ising Model, the material is assumed uniform, i.e., *J* is constant.

This is the major difference in MSDs; *J* is a function of position, *i*. We started by defining three regions: L, R, and m (see section [3b.iii, Regions](#_(3d)_Regions:)).

We then create five J constants: JL, JR, Jm, JmL, JmR. The formular is now:

where *Jij* is a function of the two positions, i and j, of the spins. *Jij* will equal one of JL, JR, Jm, JmL, JmR.

Another way to think about this is that *i* and *j* represent nodes in a weighted graph/network. *Si* is a value at each node, and *Jij* is a value/weight for each edge. In this specific model, edges only exist between adjacent nodes, but we could generalize to a *J* value for any pair of nodes.

### (4a.ii) 2D/3D Ising Model

We can also expand this model to a 2D or 3D rectangular lattice, with each atom having six adjacent atoms instead of just two, two for each direction: x, y, and z. (Except to the boundary atoms which may have less.)

### (4a.iii) Heisenberg Model

In the Heisenberg model, we generalize by allowing the spins for each atom to be vectors instead of simply +1 or -1. At first, we allow spins to be unit vectors in any direction, but eventual we add a parameter, S (a function of location), which specifies the magnitude of each atom’s spin. This allows spins in different regions to vary in strength.

### (4a.iv) Changing Shape

Up to this point, the left and right ferromagnets and molecule all have the same height and depth.

Whereas physical MSD devices have a shape that can be better approximated by the two ferromagnets overlapping perpendicularly. That is, allowing the heights and depths of the two ferromagnets to be independent of each other.

Moreover, a full sheet (or brick) of molecules is a little unrealistic. The physical devices attach molecule along the exterior only.

Now that the dimensions of the two ferromagnets are different, they have an intersection of (y, z) positions that fall within both ferromagnets. It is along this intersection that the molecule will be defined. Moreover, only the perimeter of this region will be molecules, and the interior will be omitted. (This area represents the physical spacer.)

### (4a.v) Molecules

Up to this point, the properties of the molecules have been uniform, and the shape of the molecule has been 1 or more nodes in a linear array, similar to a 1D Heisenberg model.

As alluded to before, however, the equations used such far can be generalized to a graph/network model. At this step, the “atoms” or parts of the molecules are each represented as nodes in a graph with unique properties. And these nodes can be connected in any arbitrary way with edges, each of which contain unique properties as well.

These graph-based molecules then interface with the two ferromagnets at the same contact points as before. Two of the nodes in each molecule must be demarcated as “leads” (leftLead, rightLead). These nodes are how the molecule interfaces with the two ferromagnets.

### (4a.vi) Free Electrons

<To-do>

## (4b) Equations

Along with the changes to the model, many more parameters have been added to the model to better predict the behavior of materials different then simple ferromagnets. A list of all the parameters, output variables, and other notation can be found in section [3b. Variables, Notation, and Symbols](#_(3b)_Variables,_Notation,).

, means:

* Case 1: and are positions in one of the ferromagnet regions. Then or or . In this case, two of the components will be equal; i.e., the line containing and is parallel to either the , , or axis.
* Case 2: and are nodes in the molecule region. Then and share a directed edge such that .
* Case 3: one of and is in the ferromagnet region, and one in the molecule region. Then .

# 5. Acknowledgments

# 6. File Structure

# 7. Abstractions

# 8. MSD Library

# 9. Compiling

# 10. Other Tools