`HyPerCarlo` - A Monte Carlo code for use in Hybrid Perovskite Research

Andrew P. McMahon[1]

[1]*Department of Physics, Imperial College London, London SW7 2AZ, UK and Centre for Doctoral Training in Theory and Simulation of Materials, Imperial College London, London SW7 2AZ, UK*

(Dated: January 13, 2017)

Here I introduce a lattice based Monte Carlo code for use in simulating condensed matter physics systems. The key points of the program's design and functionality are introduced as its usage. The program is designed to be extremely flexible in order to maximise its application, and it allows the use to choose both the terms present in the Hamiltonian and the update algorithm used. Finally, the program is used to simulate the lattice dynamics of interacting dipoles in order to understand proposed glassy dynamics in hybrid-inorganic perovskite materials.

I. INTRODUCTION

Monte Carlo (MC) simulation is one of the key techniques of computational physics and has been applied to countless problems, from quantum chromodynamics to the simulation of foams and polymers. Despite this widespread use of Monte Carlo, there are only a couple of programs available which allow flexibility in both choice of energy terms and algorithmic procedure, so that it is often more time efficient for a researcher to write their own MC code. This code seeks to alleviate the problem slightly by grouping several similar techniques together, so that several problems within condensed matter physics can be solved using one, flexible code.

II. PROGRAM STRUCTURE

The program follows quite a simple structure, containing only one class and a few methods within that class. the key difference between this and some other MC codes is that the program has been designed with flexibility in mind. The main class is the `Lattice` class, which has several methods which we will list and describe briefly:

- `int Vol()` - This returns the volume of the lattice.
- `void initialise_lattice(std::string key)` - This initialises the lattice to a predetermined configuration defined by the keyword string 'key'. Current keywords include
  - `FERRO` - Initialise to a ferroelectric state where all spins/dipoles are aligned with components $\vec{p} = (0, 0, 1)$.
  - `PARA` - Initialise the lattice to a paraelectric states where all spins/dipoles are aligned randomly.
  - `PREV` - Read in a previous state to use as initial state. This must be given in the current directory as a comma separated file with line format x, y, z, p_x, p_y, p_z, where the Cartesian coordinates and components of each spin/dipole are listed. The file must have the name `PrevState.dat`.
- `void output_lattice(std::string datafile)` - Output the current state of the lattice to a file named 'datafile' and save it in the local directory.
- `void Equilibrate(int steps, float T)` - Equilibrate the lattice by performing `steps` number of MC steps (total, not per site) at temperature `T` in Kelvin.
- `void Run(int ensemble_size, int n_ensembles float T)` - Perform a statistics gathering run (after equilibration) whereby `n_ensembles` of size `ensemble_size` steps are used in order to calculate thermodynamic quantities.
- `void MC_Step(int x, int y, int z, float T)` - Perform a (Metropolis) Monte Carlo step on dipole located at lattice coordinate (x, y, z) at temperature `T`. Currently only the Metropolis update algorithm is employed but there will soon be new functionality allowing the update to occur through the use of a cluster algorithm.
- `float site_Hamiltonian(int x, int y, int z)` - Calculates the local energy of a spin/dipole located at lattice coordinate (x, y, z). Currently only a Heisenberg model Hamiltonian

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} \vec{S}_i \cdot \vec{S}_j \qquad (1)$$

  is coded in. The exchange energy J is currently set at a default value of $J = 0.025$. This will be updated to allow other energy terms which a user can mix and match to create their own Hamiltonians (the code will eventually be designed so that the user can input their own Hamiltonian function relatively easily).
- `float dot_dipole(Lattice::dipole p1, Lattice::dipole p2)` - This functions takes two lattice dipole objects and returns their dot product.

- `float randomNumber(float min, float max)` - Returns a random number found from a Mersenne Twister algorithm which is seeded in the main program, in the range from `min` to `max`.

The lattice class also has the following data structures associated with it, which some of the above functions act on

- `struct dipole {float x; float y; float z}` - A struct which describes each spin/dipole by listing its three components (p_x, p_y, p_z) (here only labelled with `float x,y,z`). A `Lattice` class object is a `std::<vector>` of dipole objects.
- `Nx, Ny, Nz` - These are the dimensions of the lattice, set in the main program. They are private members of the `Lattice` class.

III. LATTICE HAMILTONIANS

Several Hamiltonian terms are programmed into `HyPerCarlo`, and it is easy to combine Hamiltonian terms to make new model Hamiltonians (it is left to the user's discretion as to whether or not these new Hamiltonians will make sense however). As an example one could combine the Heisenberg Hamiltonian

$$\mathcal{H} = -J\sum_{\langle i,j\rangle}\vec{S}_i\cdot\vec{S}_j + \mu_0\sum_i \vec{B}\cdot\vec{S}_i \tag{2}$$

and the Dzyaloshinskii-Moriya Hamiltonian term

$$\sum_{\langle i,j\rangle}\vec{D}\cdot\vec{S}_i\times\vec{S}_j \tag{3}$$

to produce the new Hamiltonian

$$\mathcal{H} = -J\sum_{\langle i,j\rangle}\vec{S}_i\cdot\vec{S}_j + \sum_{\langle i,j\rangle}\vec{D}\cdot(\vec{S}_i\times\vec{S}_j) - \mu_0\sum_i \vec{B}\cdot\vec{S}_i \tag{4}$$

which for a suitable choice of $\vec{D}$ can be used to model Chiral Helimagnetic Systems such as MnSi.

If the user wishes to simulate a lattice of electrostatically interacting dipoles, as may be relevant for ferroelectric materials, they can select a serious of dipole based terms including the dipole-dipole electrostatic interaction for site i

$$\mathcal{H}_i = \sum_j\left(\frac{\vec{p}_i\cdot\vec{p}_j - 3(\vec{p}_i\cdot\hat{r}_{ij})(\vec{p}_j\cdot\hat{r}_{ij}))}{r_{ij}^3}\right) \tag{5}$$

In this case an Ewald summation is used in order to efficiently calculate this sum.

There is also the possibility to study relaxor ferroelectrics by selecting a Hamiltonian of the form

$$\mathcal{H} = -\sum_{\langle i,j\rangle}J_{ij}\vec{p}_i\cdot\vec{p}_j - \sum_i \vec{E}\cdot\vec{p}_i \tag{6}$$

where the J_{ij} is a random interaction energy drawn from a Gaussian distribution ?

$$\mathcal{P}(J_{ij}) \propto \left(-\frac{J_{ij}^2}{2\sigma_J^2}\right) \tag{7}$$

where σ_J is the standard deviation of the distribution of J_{ij}.

Glassy systems can be studied with similar models such as the Hamiltonian

$$\mathcal{H} = -J\sum_{\langle i,j\rangle}\vec{p}_i\cdot\vec{p}_j - \sum_i \vec{e_i}\cdot\vec{p}_i \tag{8}$$

where $\vec{e_i}$ is a random local electric field on the lattice, drawn again from a Gaussian distribution of values

$$\mathcal{P}(e_i) \propto \left(-\frac{e_i^2}{2\sigma_e^2}\right) \tag{9}$$

and distributed on the lattice at the beginning of the simulation. The value σ_e is of course the standard deviation of the distribution of random fields, e_i.

IV. SIMULATION WORKFLOW

The basic outline of how the program should be used in order to calculate relevant quantities in an MC simulation is shown in Figure 1.

V. TIMING INFORMATION

This section is to keep me right as I progress with development. See table V for some example timing information.

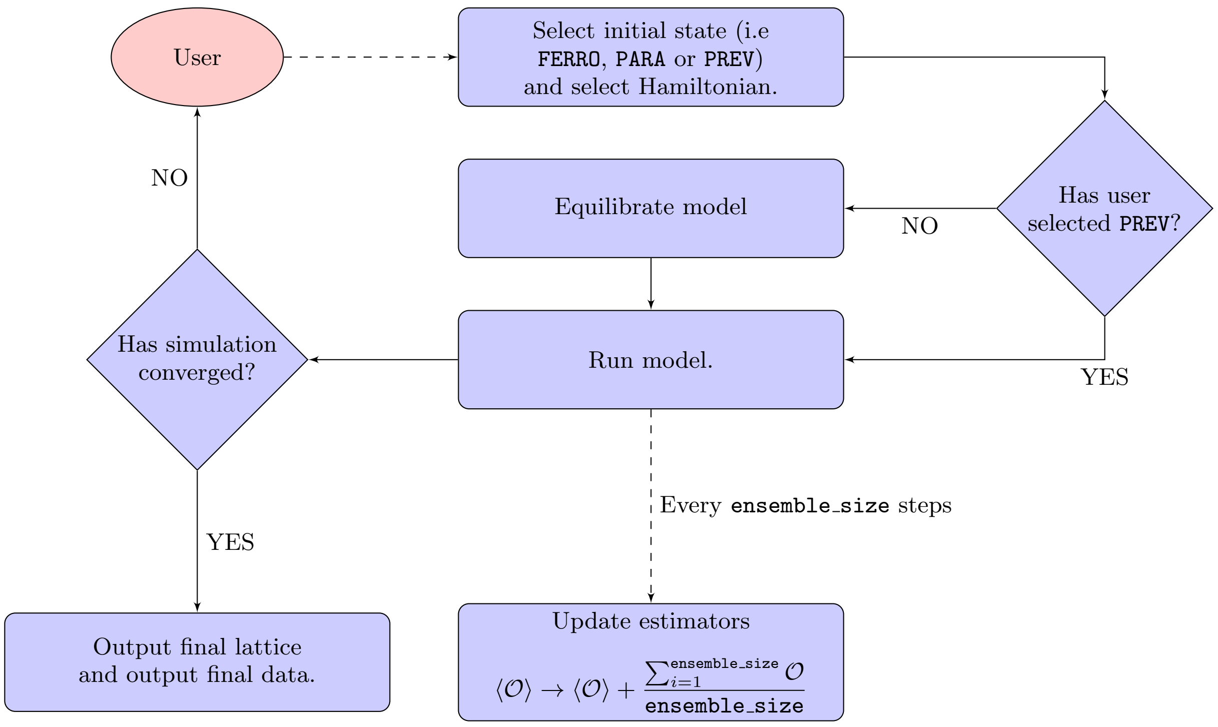
User
Select initial state (i.e FERRO, PARA or PREV) and select Hamiltonian.
Has user selected PREV?
NO
Equilibrate model
YES
Run model.
Has simulation converged?
NO
YES
Every ensemble_size steps
Output final lattice and output final data.
Update estimators
$\langle \mathcal{O} \rangle \to \langle \mathcal{O} \rangle + \frac{\sum_{i=1}^{\texttt{ensemble\_size}} \mathcal{O}}{\texttt{ensemble\_size}}$

FIG. 1. Basic workflow of a Monte Carlo simulation performed using `HyPerCarlo`.

Date	$(T_{min}, T_{max}, \Delta T)$	$(E_{min}, E_{max}, \Delta E)$	sampleDistance (MCS)	nSamples	equilStepsPerSite (MCS)	Time (mins)
13/1/2017	(1000,2000,50) K	N/A	150	1000	10000	170 for 70%