A Monte Carlo Study of the Classical, Isotropic, 3D Heisenberg Model

Numerical Studies of Stochastic Spin Systems

Michael Conroy
PHY 471 Capstone Project
Spring 2014
Professor: Dr. Matthew Enjalran

May 2, 2014

Southern Connecticut State University

Table of Contents

1. Introduction
   - Goal
   - Brief Overview
2. Background and Theory
   - Statistical Mechanics
   - The Heisenberg Model
   - Monte Carlo Method
3. Method
   - Implementation
4. Results and Discussion
   - Data Plots
5. Conclusion

Goal and Purpose

- Simulate the classical, isotropic, 3D Heisenberg Model on the simple cubic lattice
- Utilize the Monte Carlo method with the Metropolis Algorithm
- Compare simulation data to literature data
- Explore numerical analysis approach to the simulation

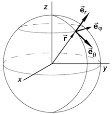
Heisenberg Model

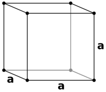
- Continuous spin model
- Hamiltonian:

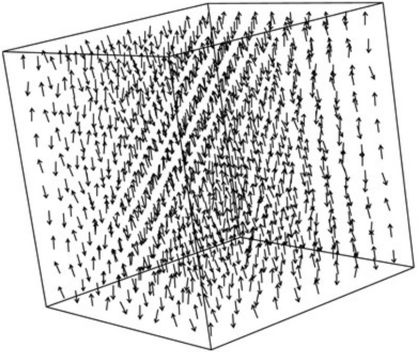
$$H = -J \sum_{<ij>}^{N} \vec{S}_i \cdot \vec{S}_j,$$

where $|\vec{S}_i| = 1.$

- Applications

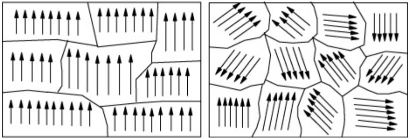






Magnetism and Magnetism in Statistical Mechanics

- Magnetic Domains



- Paramagnetism, ferromagnetism, and antiferromagnetism

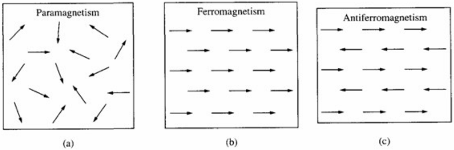


Fig. 11.58 Schematic representations of magnetic dipole arrangements in (a) paramagnetic, (b) ferromagnetic, and (antiferromagnetic materials.

Phase Transitions

- Critical Temperature
- Order Parameter

Numerical Analysis

- No analytic solutions
- Intractable problems
- Monte Carlo simulations
    - Importance sampling

Statistical Mechanics

- Canonical Ensemble
- Boltzmann Distribution: $p_\mu = \frac{1}{Z(\beta)} e^{-\beta E(\mu)}$
- Partition Function: $Z(\beta) = \sum_\mu e^{-\beta E(\mu)}$
- Most macroscopic thermodynamic variables of a system can be expressed by the partition function or its derivatives!
- For example, energy, specific heat, entropy, free energy...

Calculating the Physical Quantities

- How do we calculate the required physical quantitites of the Heisenberg Model?
- Energy and specific heat:

$$E = -J \sum_{<ij>}^{N} \vec{S_i} \cdot \vec{S_j}$$

$$C = k\beta^2(\left\langle E^2 \right\rangle - \langle E \rangle^2)$$

- Magnetization:

$$m = \sqrt{M_x^2 + M_y^2 + M_z^2},$$

$$\text{where } M_\alpha = \frac{1}{N} \sum_i \vec{S_{i\alpha}}$$

Background and Theory

- Numerical Analysis
  - No analytic solution or intractable
- Pseudo-random number generation
- Monte Carlo Simulation
  - Estimator
  - Importance Sampling
    - Markov Processes
    - Ergodicity
    - Detailed Balance
  - Acceptance Ratio

Implementation

Software

- KISS: "Keep It Simple Stupid"
- Functional Approach in C
- 3D and 4D Pseudo-arrays of Pointers
- GNU GCC Compiler, Code::Blocks IDE

```c
//Declare 4D arrays
printf("Declaring 4D arrays...\n\n");
//lattice
//double lattice[LENGTH][LENGTH][LENGTH][2]; //The Lattice[x loc][y loc][z loc][0=theta/1=phi
double ****lattice;
lattice = (double****)malloc(LENGTH * sizeof(double ***));
    if (lattice == NULL)
    {
        printf("7: Out of memory!\n");
        exit(7);
    }
for(i = 0; i < LENGTH; i++)
{
/*-----------------------------------------*/
    lattice[i] = (double***)malloc(LENGTH * sizeof(double**));
        if (lattice == NULL)
        {
            printf("8: Out of memory!\n");
            exit(8);
        }
    for(j = 0; j < LENGTH; j++)
    {
        lattice[i][j] = (double**)malloc(LENGTH * sizeof(double*));
            if (lattice == NULL)
            {
                printf("9: Out of memory!\n");
                exit(9);
            }
        for (k = 0; k < LENGTH; k++)
        {
            lattice[i][j][k] = (double*)malloc(Y_SIZE * sizeof(double));
                if (lattice == NULL)
                {
                    printf("10: Out of memory!\n");
                    exit(10);
                }
        }
    }
} //Declare 4D arrays
```

Implementation

Hardware

- Workstation
  Lenovo IdeaPad Y580
  Intel i7-3630QM, 8-thread, 3.4 GHz (max, single core)
  16 GB ram, 256 GB SSD
  GeForce GTX 660M (overclocked to 1 GHz)
  Fedora 20 Linux, Scientific Spin
- Simulation Machines
  Custom Built PCs
  AMD Opteron 6212, 16-thread, 3.2 GHz (max, $\leq$ 4 core)
  32GB ram, Fedora and Ubuntu Linux

Data Plots

Energy

Data Plots

Magnetization

Data Plots

Specific Heat

Data Plots

Magnetic Susceptibility

- **Susceptibility calculation not straightforward due to rotational invariance in 3D Heisenberg Model!**

Conclusion

- Data matches predicted behavior
- Phase transition at critical temperature of $T_c \approx 1.45K$
- Susceptibility not as straightforward to calculate as in the Ising Model
- Programmatic concerns
  - Code in Fortran for readability, debugging, and intrinsic functions (but be careful!)
  - Fix arrays and possibly implement structures
  - Improve simulation run time! Optimize code!

Next Steps and Future Work

- Code improvements
- Susceptibility
  - Correlation function calculation
- Spin dynamics
- Magnetic frustration and other lattices
- Possible project: LED Cube visualization of 3D Ising model phase change