

Computational Simulations of Magnets at High Temperatures
Utilizing the Ising Model and the Monte Carlo Technique

Project SEED

Dr. Ken Ahn

Mayrolin Garcia

Introduction

How does the property of magnetism behave at high temperatures? That is the behavior my SEED partners and I tried to simulate this summer by writing our own C++ computer programs. By using the one dimensional and two dimensional Ising Model, we were able to simulate systems of atoms.

Theory

Within any material, atoms behave like little magnets. Each atom's electrons have something called a magnetic moment caused by the electron's spin and motion. It is what gives each atom a magnetic behavior. However, since each atom interacts with each other, a material as a whole may not have visible magnetic properties. This interaction is called exchange interaction. It is an additive interaction where atoms can add or cancel out their magnetic behavior between each other. This interaction is driven by the alignment of each magnetic moment. Therefore, if all the magnetic moments are aligned similarly, a material will exhibit magnetic attributes. If they align randomly, a material may show weak, if any, magnetic attributes. These alignments are not "set in stone" and can be manipulated. In the case of permanent magnets, ferromagnetism causes the alignments to change. Ferromagnetism is the method by which a material becomes a permanent magnet after interacting with a magnetic field. Once enough atoms and molecules are aligned in one direction by the pull of the external magnetic field, their small magnetic moments combine to create a much larger quantifiable magnetic field.

It is also because of this fluidity of the alignments that permanent magnets exposed to high temperatures will lose their magnetism. The Curie temperature, or Curie point, of a magnetic material is the temperature boundary at which it loses its permanent magnetism. At this temperature, unique to each material, spontaneous magnetization does not occur. This is due to the destruction of the alignment by the increase of thermal energy. Once this occurs, the material will become paramagnetic and will only show magnetic characteristics when an external magnetic field is present. When this external magnetic field is removed, the material will not retain any magnetization. This temperature boundary is an indication of how much energy is needed to disrupt the long-range ordering of the material.

Method

To find the energy of a randomly aligned magnetized system, we used the Ising Model in conjunction with the Monte Carlo technique.

Ising Model

Ernst Ising developed this mathematical model now used to describe many phenomena. By using this model in regards to ferromagnetism, we make the following assumptions:

1. Bits of information interact in pairs called spins to produce an additive effect.
2. Each spin, S_i , can take the value of 1 or -1.
3. Spins are either aligned in one or two dimensions.
4. Nearest neighbor interactions are the only interactions taken into account.
5. First and last spin sites interact in something called the Periodic Boundary Condition.

After we consider all of these conditions, we are able to measure the energy using the following equation:

$$E = \sum_{i=1}^{N-1} -J (S_i \times S_{i+1})$$

Where S_i is the spin index, and J is a positive number constant.

2D Ising Model

While the 2D Ising Model keeps the same conditions, it differs from the 1D Model in that the atoms are aligned in two dimensions and because of this there will be more periodic boundaries.

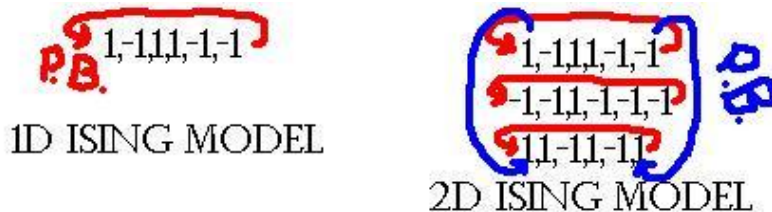


Figure 1 The blue coded periodic boundaries (P.B.) on the 2D model happen for each top and bottom element of each column.

This yields a new energy formula, which is as follows:

$$\text{General Form: } E_{2D} = -J \sum_{i \neq j} (\mathbf{S}_i \cdot \mathbf{S}_j)$$

$$\text{Simplified Form: } E_{2D} = \sum_{i \neq j} S_i (S_{i+10} + S_{i-10})$$

Magnetization

While similar to energy, magnetization describes the state of system in a current configuration. The magnetization of a material can range from $M_{\max} = 1$, where all its spins are directed to one direction, or $M_{\min} = 0$, where an equal number of spins are pointing up and down; thus canceling each other out.

To calculate magnetization, one simply needs to add all the values of the spins, and then divide by the total number of spins. Because magnetization needs to be a positive number, we take the absolute value of the answer.

$$M = \text{abs} (1/N \sum S_i)$$

Monte Carlo Technique

By using the Monte Carlo technique, we are able to introduce temperature into the system. We utilize a Boltzmann distribution as well as random numbers.

$$f = \exp (\Delta E / k_B T)$$

Where ΔE is the change in energy, k_B is Boltzmann constant, and T is temperature.

This function becomes our probability function in our C++ code:

$$\text{prob} = \exp (\Delta E / k_B T)$$

It expresses the idea that although a system may be in thermal equilibrium at a relatively low energy state, there is still the chance that the system can reach an even lower energy state.

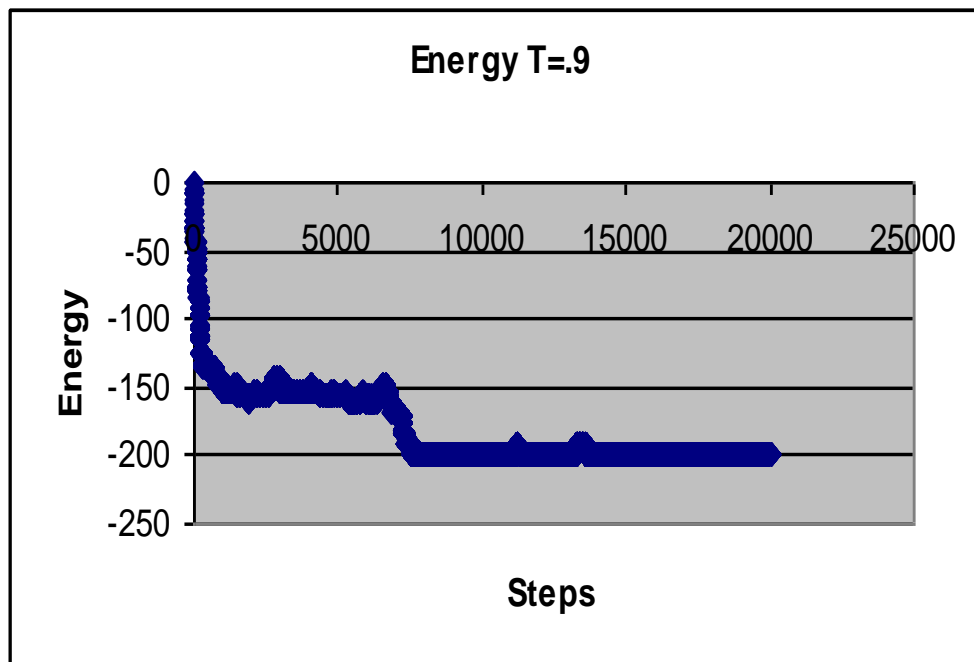
General Sequence of C++ Program

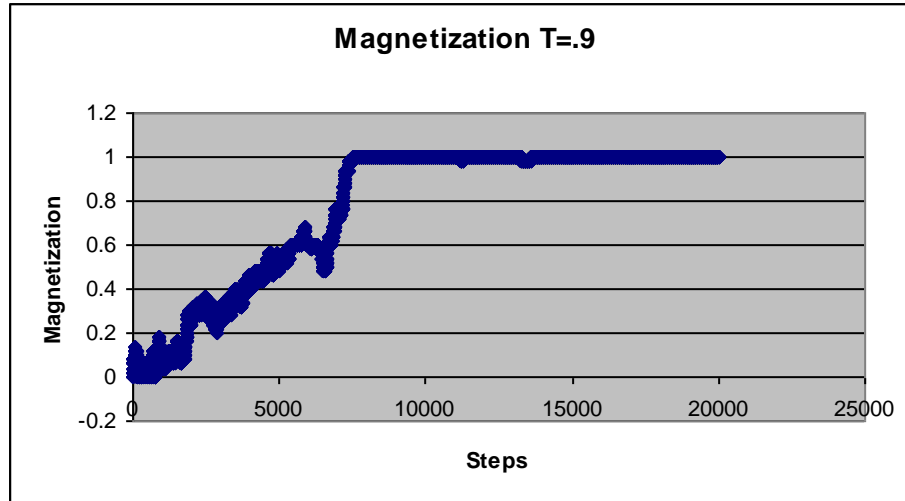
The basic steps we took to create a system and measure its energy and magnetism were:

1. Create a random spin configuration
2. Calculate its energy.
3. Randomly select a spin site.
4. If the value of that site is 1, change it to -1, or vice versa.
5. Calculate the new energy.
6. If the new energy is less than or equal to the old energy, accept the new configuration. Otherwise, calculate probability and generate a random number.
7. If the probability is less than the random number, accept the new configuration.
8. If neither, return the site to its original value, and start from step three.

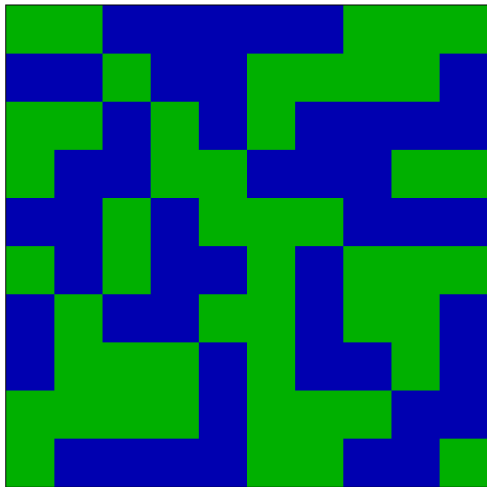
Results

Using our C++ code, we were able to print out and graph our results.





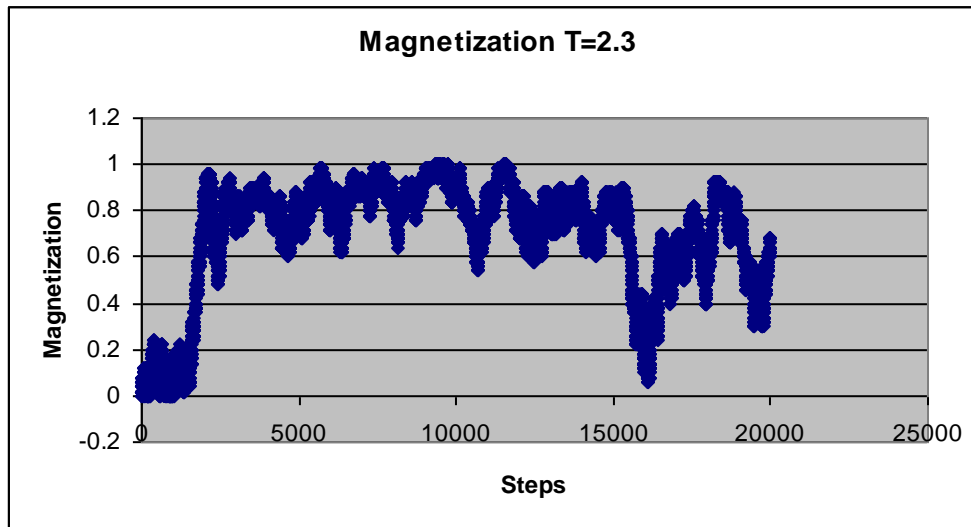
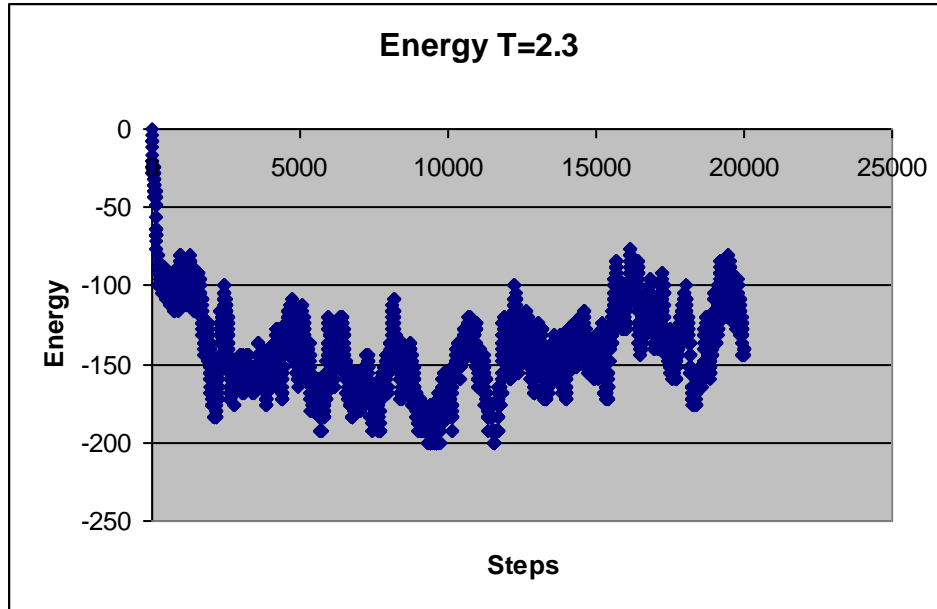
Initial Configuration-Array Size:10x10-T=.9



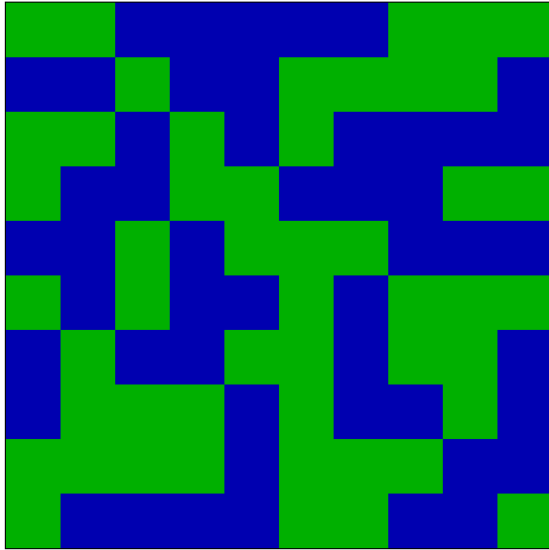
Final Configuration-Array Size:10x10-T=.9



As demonstrated by the graphs for $T=0.9$, we can see that a system at such low temperature will easily find its ground state, and give the maximum value for magnetism. This model tells us that the spins are easily aligned at low temperatures, thus creating visible magnetic properties.

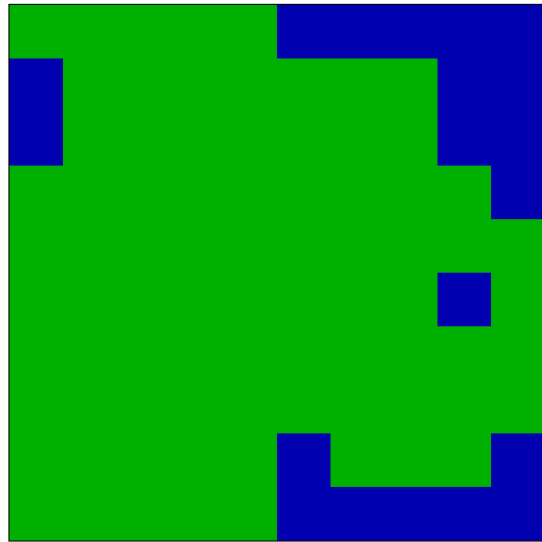


Initial Configuration-Array Size:10x10-T=2.3



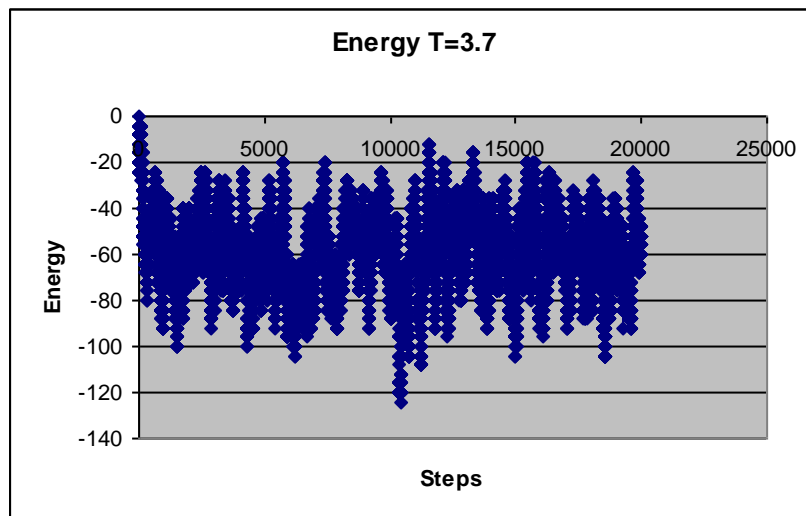
Copyright Dept. of Physics, NJIT

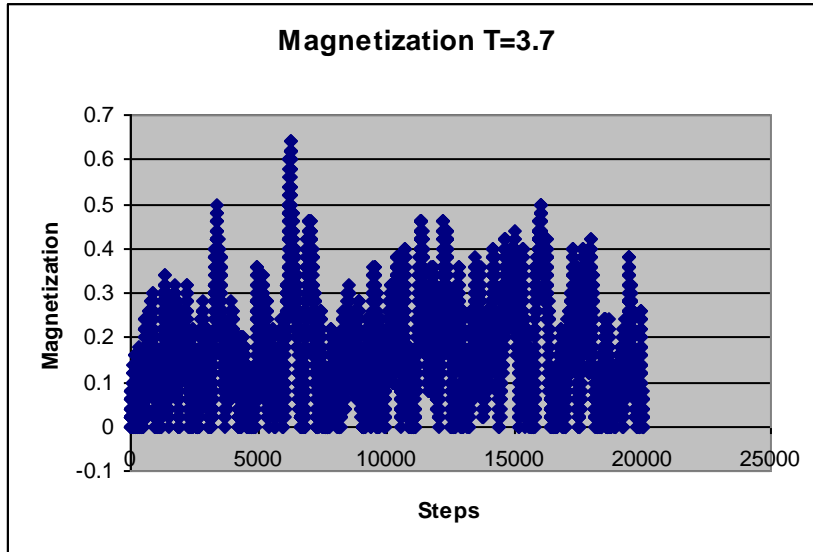
Final Configuration-Array Size:10x10-T=2.3



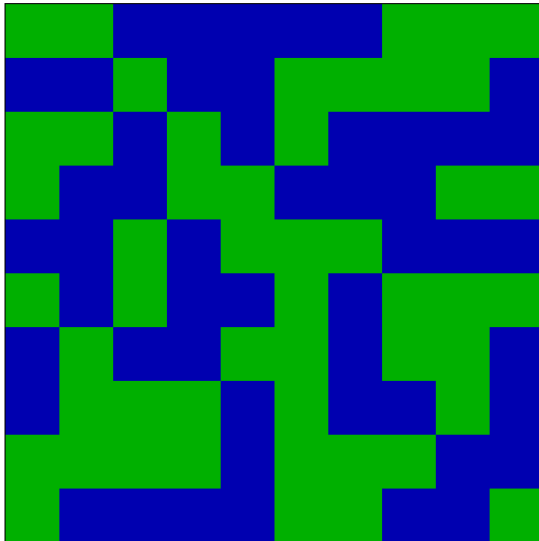
Copyright Dept. of Physics, NJIT

At a point very close to the Curie temperature, we can start to see signs of the decay in alignment. The system is able to reach its ground state, but not maintain it. In terms of magnetism, it is considerably declining.

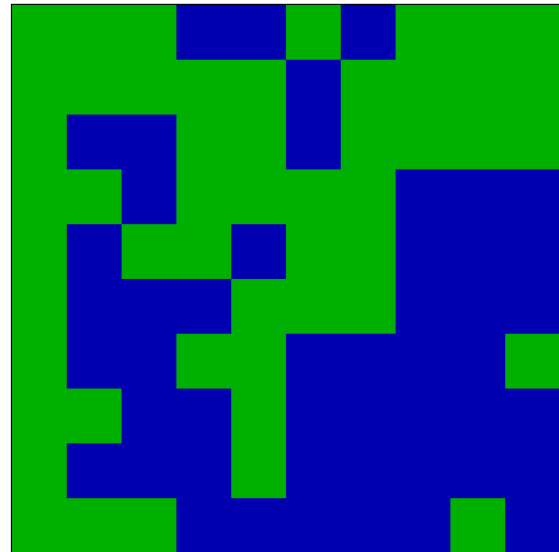




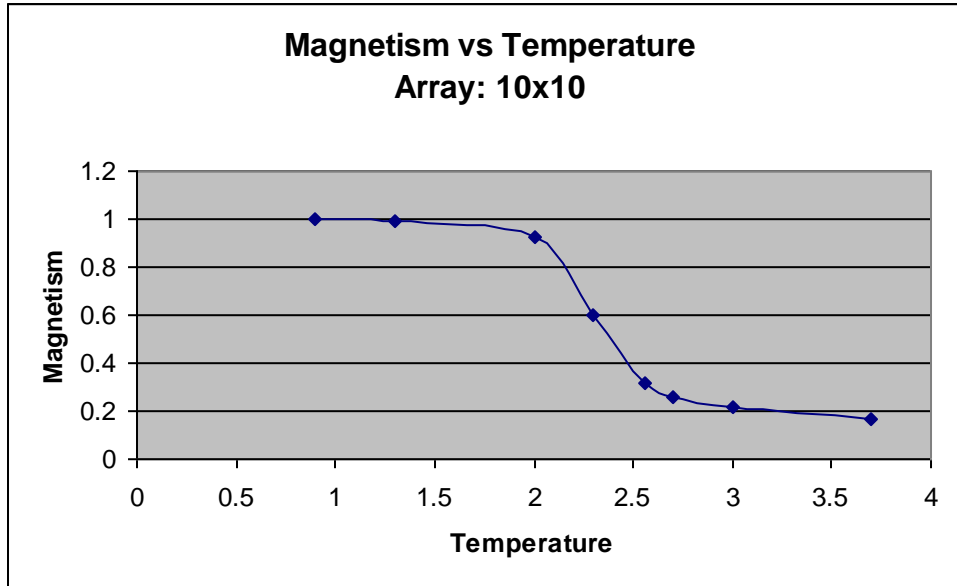
Initial Configuration-Array Size:10x10-T=3.7



Final Configuration-Array Size:10x10-T=3.7



At such high temperature, the system has too much energy that it can't find its ground state. It just varies from -20 to -80. Its magnetism is considerably low, since barely any spins are aligned with each other.

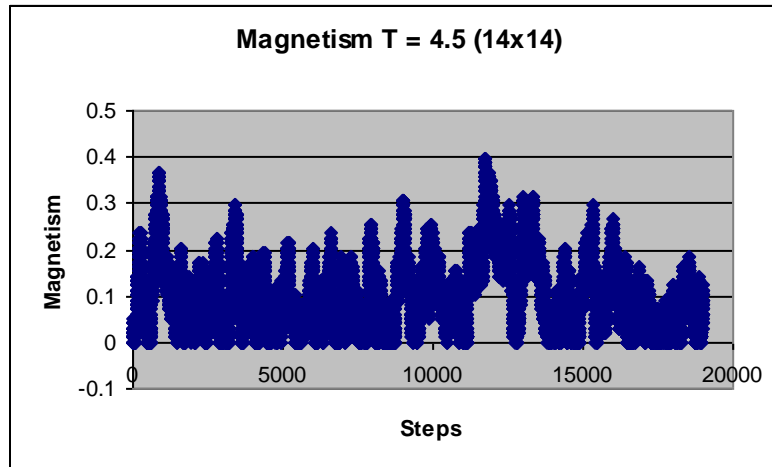


This graph correlates my results for magnetism at different temperatures. As was expected, magnetism drops immensely once the temperature hits a certain range. This is the Curie point: the start of the destruction of the alignments due to an increase in thermal energy.

Discussion

As can be seen in Magnetism vs. Temperature graph, my T_c was off. My decline started at about 2.16, whereas the accepted value is 2.26. My percent error is -4.42%.

In addition, our simulation does not clearly exemplify real world conditions. In real materials, magnetism is zero at temperatures above T_c . However, in our simulation, magnetism is reduced to very low numbers, but not zero.



This is because in our C++ code only takes into account spin interactions between up, down, left, and right neighbors, not in front or behind. In addition, the “random” generator used by C++ is actually a pseudorandom generator, and because of this our probability will never be below our random number.

Conclusion

Through our simulations, we were able to get very close results that acted like real material systems. Although our percent error was -4.42% and our magnetism is unable to reach zero, the program was able to reproduce the mechanism that imitates real magnetic systems closely.

Acknowledgements

I would like to thank the American Chemical Society for making this great opportunity possible. I would also like to show my gratitude to Dr. Ken Ahn and Tsezar Seman for allowing me to be part of their research. My thanks also go to the New Jersey Institute of Technology, as well as Dr. Fahrenholtz for arranging the summer internship.

Appendix: C++ Code

```
//2D Ising Model
```

```
//Mayrolin Garcia ~ August 27, 2009
```

```
#include <iostream>
```

```
#include <cstdlib>
```

```
#include <fstream>
```

```
#include <cmath>
```

```
#include <stdio.h>
```

```
using namespace std;
```

```
#define NX 14
```

```
#define NY 14
```

```
#define N (NX*NY)
```

```
////////////////////////////////////
```

```
// Function to Calculate Energy
```

```
//
```

```
double EnergyCalc(int spins[][NX])
```

```
{
```

```
    double energy = 0.0;
```

```
    int ir, ic, row, col;
```

```
    for (row = 0; row < NY; row++) {
```

```
        for (col = 0; col < NX; col++){
```

```

        // periodic... boundary...
        if (row == NY-1) ir = 0;
        else ir = row + 1;
        if (col == NX-1) ic = 0;
        else ic = col + 1;

        energy = energy + spins[row][col] * (spins[row][ic] + spins[ir][col]);
    }
}
return -1.0 * energy;
}

```

```

////////////////////////////////////

```

```

// Function to Calculate Magnetization

```

```

//

```

```

double Magnetization(int spins[][NX])

```

```

{

```

```

    int MG = 0;

```

```

    int row, col;

```

```

    for (row = 0; row < NY; row++){

```

```

        for (col=0; col < NX; col++){

```

```

            MG += spins[row][col];

```

```

        }

```

```

    }

```

```

    if (MG < 0) MG = MG * -1;

```

```

    return (double)MG / N;
}

/////////////////////////////////////////////////////////////////

// Function to Export Spin-Map
//
void SaveSpins(int spins[][NX], char filename[])
{
    ofstream fout(filename, fstream::out);
    int row, col;
    for (row = 0; row < NY; row++){
        for (col = 0; col < NX; col++){
            fout << spins[row][col];
            if (col < NX-1) fout << ",";
        }
        fout << "\n";
    }
    fout.close();
}

/////////////////////////////////////////////////////////////////

// Function to Get Spins
//
int GetRandomSpin()
{
    if ( ((double)rand() / RAND_MAX) < 0.5)

```

```
        return 1;
    else
        return -1;
}

/////////////////////////////////////////////////////////////////

// Function to Initialize Array
//
void Initialize(int spins[][NX])
{
    int row, col;
    for (row = 0; row < NY; row++) {
        for (col = 0; col < NX; col++) {
            spins[row][col] = GetRandomSpin();
        }
    }
}
```

```
/////////////////////////////////////////////////////////////////

// Random Function Generator
//

double UnitRandom ()
{
```

```

    return (double)rand()/RAND_MAX;
}

/////////////////////////////////////////////////////////////////

// Code entry point
//
int main()
{
    double en_old, en_new, T;
    int p,q;
    int steps = 0;
    int spins[NY][NX];
    //energy gound state
    int en_grst = -2 * N;
    int counter = 0;

    srand (77);

    Initialize(spins);
    SaveSpins(spins, "init-config .8.txt");

    ofstream fout("ground-state .8.txt", fstream::out);
    ofstream fout2("magentization .8.txt", fstream::out);

```

```

en_old = EnergyCalc(spins);

while (steps < 50000)//en_old>en_grst &&
{
    // pick random spin site
    p = rand() % NY;
    q = rand() % NX;
    spins[p][q] = -1 * spins[p][q];

    // calculate the new energy of the new configuration
    en_new = EnergyCalc(spins);

    // calculate delta Energy
    double dEnergy = en_new-en_old;

    T = .8;

    if(dEnergy <= 0)
    {
        en_old = en_new;
        fout << steps << "\t" << en_old << "\n";
        fout2 << steps << "\t" << Magnetization(spins) << "\n";
        //totalMag += calcMagnetization(ddArray);
    }
}

```

```

        else
    {
double prob = exp(-dEnergy/T);
    if (prob > UnitRandom() )//(newEnergy <= oldEnergy)
        {
            en_old = en_new;

            fout << steps << "\t" << en_old << "\n";
            fout2 << steps << "\t" << Magnetization(spins) << "\n";
            //totalMag += calcMagnetization(ddArray);
        }
    else

        spins[p][q] = -1 * spins[p][q];

        fout << steps << "\t" << en_old << "\n";
        fout2 << steps << "\t" << Magnetization(spins) << "\n";
    }
    steps++;
}

```

```

SaveSpins(spins, "final-config .8.txt");

```

```

fout.close();

```

```

fout2.close();

```

```

// pausing the screen, comment this out when don't need it

```

```
system("PAUSE");  
return 0;  
}
```

References

Britannica Online, www.britannicaonline.com

C. Kittel, "Introduction to Solid State Physics." 6th edition