Developing a Numerical Solution to Ferromagnetic Materials

Elizabeth Drueke

May 2, 2016

Abstract

Modeling lattice-type structures under the effects of magnetization has been achieved with reasonable accuracy using the Ising model. In order to accurately describe the larger of these systems, it is important to develop computer algorithms which can do large-scale computations in small amounts of time. To this end, we develop a Monte Carlo process using the Metropolis Algorithm to study the effects of magnetization on two-dimensional lattice systems of various sizes and at various temperatures, and use the results of this analysis to determine the accuracy of our algorithm. In partiular, we focus on the effect of various random number generators on the results, to ensure that we are not encountering a strong systematic bias from this source.

1 Introduction

To say that magnetism is an important physical phenomenon is a strong understatement. Magnetism plays a role in day to day life on earth. The earth's magnetic field is perhaps the most stunning example of this. Caused by the electric fields resulting from the magma under the earth's crust [10], the earth's magnetic field has been used by explorers for centuries to navigate both on land and sea. Today, we are still using magnetization to explore the world around us, although in slightly different ways. For example, magnets are used in some of the largest physics experiments in the world, such as the Large Hadron Collider (LHC), which uses them both as a means of accelerating particles to high energies and as a means of bending charged particles in detectors in an attempt to classify them and thus answer some of the fundamental questions about the universe in which we live.

In general, there are three kinds of magnetism: diamagnetism, paramagnetism, and ferromagnetism, which is the subject of this paper. The types of magnetism are distinguished by the behavior of the spins of the atoms within the material being magnetized. In paramagnets, the spins align parallel to the magnetic field whereas in diamagnets the spins align opposite to the field. In ferromagnetic material, however, we have the special situation where the material retains its magnetized state even after the polarizing field has been removed. In this way, ferromagnets may represent a magnetic history of the fields it has encountered [4]. The simplest theoretical model we have to describe ferromagnetism is the Ising model, described in Section 2.

In this paper, we will discuss the Ising model and its importance, and then go into the Monte Carlo algorithm used to do the calculations of various statistical quantities in Section 3, taking special note of various benchmarks developed for the case of a 2×2 lattice in Section 3.1. Finally, we discuss results of our calculations in Section 4.

2 Theory

As mentioned in Section 1, the Ising model is one of the simplest models we have to describe ferromagnetic material. It does this by expressing the energy in a simple form,

$$E = -J \sum_{\langle kl \rangle}^{N} s_k s_l - B \sum_{k}^{N} s_k, \tag{2.1}$$

where $s_k = \pm 1$ is the spin of the k^{th} lattice point, J is a coupling constant expressing the strength of the interaction between the neighboring spins, B is the external magnetic field the material is in, and the first sum is taken over nearest neighbors (ie. lattice points directly above, below, to the left, or to the right of each other, assuming periodic boundary conditions). Because we are working with ferromagnetic materials, which can retain magnetization in the absence of magnetic fields, we can take $B = 0$, and so (2.1) becomes

$$E = -J \sum_{\langle kl \rangle}^{N} s_k s_l. \tag{2.2}$$

While (2.2) [6] forms the bulk of the math behind what we will discuss here, it is not the only interesting statistical mechanical quantity we will be interested in. Thus, before we go too far into things, it may behoove us to discuss briefly concepts such as probabilitiy distributions and specific heat. Statistical mechanics is a science concerned primarily with the expected behavior exhibited by large groups of interacting objects (spins). As such, it relies heavily on probability because it quickly becomes unfeasible and, in most cases, impossible, to determine exactly how each individual spin is behaving. Thus, we are often interested in the expected values of various physical quantities. In particular, the expected value of some physical quantity X is given by

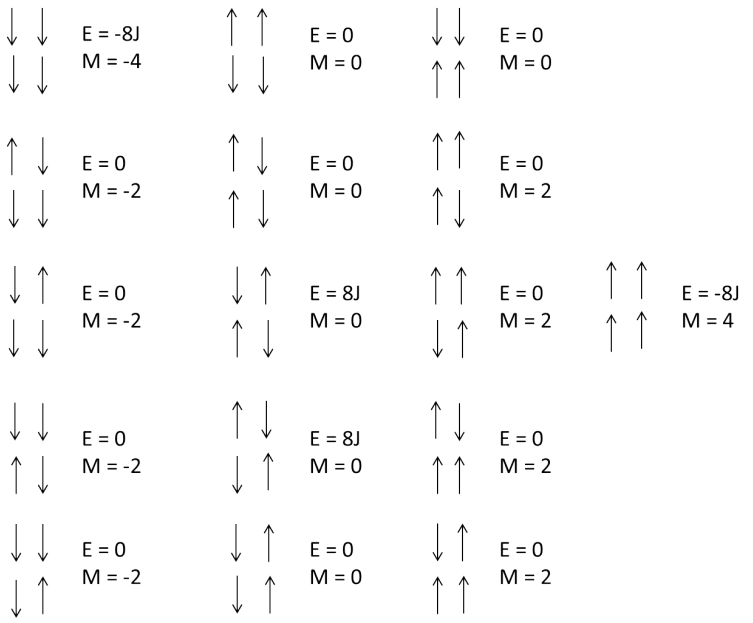
$$\langle X \rangle = \sum_{i=1}^{N} X_i P_i, \tag{2.3}$$

where the sum is over all possible states, X_i is the value of X in state i, and P_i is a probability distribution function. The probability distribution function typically used in statistical mechanics is the Boltzmann distribution,

$$P_i = \frac{e^{-\frac{E_i}{k_B T}}}{Z}, \tag{2.4}$$

where E_i is the energy of state i, k_B is the Boltzmann constant, T is the temperature, and Z is the partition function, given by

$$Z = \sum_{i=1}^{N} e^{-\frac{E_i}{k_B T}}. \tag{2.5}$$



(a)

Figure 1: The 16 possible spin orientations, with their energies and magnitizations, for a 2×2 lattice.

As an example, we might consider a 2×2 ferromagnetic lattice. In order to find any important statistical mechanical quantities, it will be important to determine the partition function. We note that there are $2^4 = 16$ possible states of this system, as any lattice point can be spin up or spin down. Already this is quite a few states to look at explicitly individually, but for the purposes of illustration we do so in Fig 1. We see that there are two states with energy $+8J$, two with energy $-8J$, and 12 with energy 0. This yields a partition function of

$$Z = 2e^{-\frac{8J}{k_B T}} + 2e^{\frac{8J}{k_B T}} + 12 = 4\cosh\frac{8J}{k_B T} + 12. \tag{2.6}$$

Using (2.3), we see also that

$$\langle E \rangle = \frac{1}{Z}\left(16Je^{-\frac{8J}{k_B T}} - 16Je^{\frac{8J}{k_B T}}\right) = \frac{-32J\sinh\frac{8J}{k_B T}}{Z}. \tag{2.7}$$

We are not only interested in the energy, however. We also want to investigate the magnetization, which is given by

$$M_i = \sum_{i=1}^{N} s_i. \tag{2.8}$$

With this simple equation, we can see that the magnetizations for each possible state in the 2×2 system are also given in Fig. 1 and that

$$\langle M \rangle = 0, \tag{2.9}$$

or, perhaps more enlighteningly,

$$\langle |M| \rangle = \frac{1}{Z} \left(8e^{\frac{8J}{k_B T}} + 4 \right). \tag{2.10}$$

The final two statistical quantities we will be interested in are the specific heat, given by

$$C_V = \frac{1}{k_B T^2} \left(\left\langle E^2 \right\rangle - \langle E \rangle^2 \right) \tag{2.11}$$

(ie. the variance of the energy of the system multiplied by some constant), and the susceptibility, given by

$$\chi = \frac{1}{k_B T} \left(\left\langle M^2 \right\rangle - \langle M \rangle^2 \right) \tag{2.12}$$

(ie. the variance of the magnetization of the system multiplied by some constant). In our 2×2 case, we will have

$$\left\langle E^2 \right\rangle = \frac{256 J^2 \cosh \frac{8J}{k_B T}}{Z}$$

and

$$\left\langle M^2 \right\rangle = \frac{1}{Z} \left(32 e^{\frac{8J}{k_B T}} + 8 \right).$$

Thus,

$$C_V = \frac{1}{k_B T^2} \left(\frac{256 J^2 \cosh \frac{8J}{k_B T}}{Z} - \left(\frac{-32 J \sinh \frac{8J}{k_B T}}{Z} \right)^2 \right) = \frac{256 J^2}{k_B T^2 Z} \left(\cosh \frac{8J}{k_B T} - 4 \sinh^2 \frac{8J}{k_B T} \right) \tag{2.13}$$

and

$$\chi = \frac{32 e^{\frac{8J}{k_B T}} + 8}{Z k_B T}. \tag{2.14}$$

3 The Algorithm

For this model, we developed a new class, the `lattice` class, which directly computes various statistical mechanical quantities for a lattice of a given size and temperature. In particular, each `lattice` object has the following objects:

- `size`, an `int` which gives the size of the square lattice (ie. n for an $n \times n$ lattice),
- `spins`, a dynamically allocated matrix of the spins in the matrix,
- `temp`, a `double` which gives the temperature of the matrix,

- `averages`, a dynamically allocated vector of the averages of E, E^2, M, M^2, and $|M|$,
- `MCcycles`, an `int` which gives the number of Monte Carlo (MC) cycles to be used in the calculation,
- `CV`, a `double` of the specific heat,
- `chi`, a `double` of the susceptibility,
- `chi_absm`, a `double` of the susceptibility calculated from $< |M| >$ rather than $< M >$ (that is, $\chi_{|M|} =< M^2 > - < |M| >^2$),
- `accepted`, an `int` of the number of Monte Carlo events accepted in the calculation, and
- `e_probs`, a `map` of an energy value to the number of times that energy appears in the calculations.

The `lattice` class also has several important functions. In addition to a default constructor, copy constructor, and destructor, `lattice` has a:

- constructor from a size, temperature, number of Monte Carlo cycles, and the option to start at a lowest-energy state,
- several "get" functions (eg. `get_E()`), which returns some statistical mechanical quantities,
- several "set" function (eg. `set_temp(double t)`), which resets and recalculates some variable,
- `plot_eprobs(string name)`, which plots the probability of various energy values E appearing in the calculations, and
- `calc_stat_quants()`, which is a private function which performs all of the calculations and sets all of the variables as the `lattice` object is constructed.

The Ising model is an excellent example of a physics model which requires the use of Monte Carlo in order to implement on a computer. The meat of this calculation is performed in the `calc_stat_quants()` function, which is called by all constructors after a random (or not) initial state is set for the lattice. This function begins by calculating the initial energy with periodic boundary conditions

```
double E=0; double M=0;
for(int i=0;i<size;i++){
  for(int j=0;j<size;j++){
    M+=spins[i][j];
    E-=spins[i][j]*(spins[(i+size-1)%size][j]+spins[i][(j+size-1)%size]);
  }
}
```

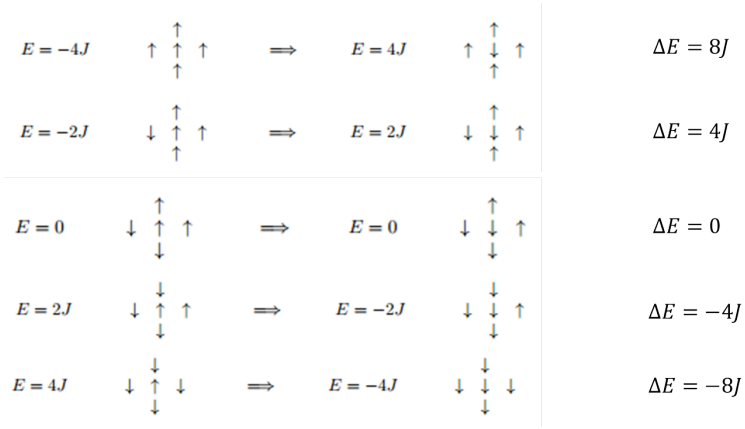
Then, it begins the Monte Carlo simulation. Iterating through integers `cycle` which are less than the `MCcycles` parameter, it first randomly changes one of the spins in the lattice:

```
int rando = rand()%(size*size);
int ct = 0;
int therow = 0; int thecol = 0;

for(int i=0;i<size;i++){
  for(int j=0;j<size;j++){
      if(ct==rando){
         spins[i][j]*=-1;
         therow = i;
         thecol = j;
       }
       ct+=1;
    }
  }
```

It then calculates the change in energy resulting from changing this spin by calling the `nearest_neighbors()` function, which takes in the row and column which were changes as well as the `spins` matrix and goes through the nearest neighbors of the changed spin to determine what ΔE should be based on Fig. 2, which shows the only possible five values of ΔE based on the number of spin-up and spin-down nearest neighbors are around the changed spin. `nearest_neighbors()` returns the ΔE value as

```
double deltaE = (-4*ups+8)*(spins[row][col]);
return deltaE;
```



(a)

Figure 2: The possible values of ΔE for the 2×2 lattice after changing one spin [6].

If the change in energy is negative, we generate a random number `myrand` (in $[0, 1]$) and compare it to $w = e^{-\frac{\Delta E}{k_B T}}$. In particular, if $w \leq$ `myrand`, we keep the change in spin. However, if $w >$ `myrand`, we keep the spin matrix as it was before. That is,

```
if(deltaE>0){
  double w = exp(-1.0*deltaE/(kB*temp));
  double myrand = (rand()%100000)/100000.0;
  if(myrand>w){
       spins[therow][thecol]*=-1;
       deltaE=0;
       go = false;
  }
}

if(go){
  accepted++;
  M+= 2*spins[therow][thecol];
  E+= deltaE;
}
```

Finally, we update the `averages` vector and `e_probs`. At every Monte Carlo cycle, we add new value of the various statistical quantities to the appropriate component of `averages` and, after all of the Monte Carlo cycles are through, we divide by `MCcycles` to get the average value. Essentially, then, `cal_stat_quants()` runs the Metropolis algorithm, one of the most important algorithms of current times. The development of the Metropolis algorithm is what first allowed for the calculation of the Ising model to be computed quickly and efficiently [6].

Finally, we can calculate our final statistical mechanical quantities:

```
CV = (1.0/(kB*pow(temp,2)))*(averages[1]-pow(averages[0],2));
chi = (1.0/(kB*temp))*(averages[3]-pow(averages[2],2));
chi_absm = (1.0/(kB*temp))*(averages[3]-pow(averages[4],2));
```

3.1 Benchmarks

To ensure that our algorithm was working properly, we ran a test on the 2×2 case discussed in Section 2. From (2.6), (2.7), (2.9), (2.10), (2.13), and (2.14), we expect that, for a temperature of T which allows $k_B T/J = 1$ for coupling $J = 1$ (ie. $T = 1/k_B$), we will have the results listed in Table 1. The results from the Monte Carlo code are shown in Fig. 3 for $< E >$, $< |M| >$, C_V, and χ. We see that, in all cases, the simulation models the expected behavior quite well very quickly ($\sim$ 1000 MC cycles) for $< E >$, $< |M| >$, and C_V, although it requires significantly more ($\sim 200 \times 10^3$) Monte Carlo cycles to achieve a somewhat reliable susceptibility. For this benchmark, the random number generator used in the Metropolis algorithm was the standard C++ random number generator seeded with `time(NULL)` and implemented as

```
double myrand = (rand()\%10000)/10000.0;
```

to ensure that the random variable was in the interval $[0, 1]$.

We also will include here a discussion of benchmarks for the random number generator itself. In particular, all computer-algorithm random number generators are inherently

Statistical Quantity	2×2 Accepted Value
Z	5973.917
$\langle E \rangle$	-7.984
$\langle \lvert M \rvert \rangle$	3.993
$\langle E^2 \rangle$	63.871
$\langle M^2 \rangle$	15.969
C_V	0.127
χ	16.001

Table 1: The accepted calculated values for the 2×2 lattice under the Ising model from the equations derived in Section 2.

deterministic. Because a random number sequence is by definition uncorrelated, computer algorithms are incapable of producing sequences of truly random numbers. Instead, they are what are known as pseudorandom number generators. For this reason, it is important to ensure that the random number generator being used is at least reasonably random. Kendall and Babington Smith [9] suggested a series of tests designed to check whether sequences of numbers were actually random: the frequency, serial, poker, and gap tests. We will focus on the first three of these.[1]

The frequency test aims to ensure that every digit occurs approximately an equal number of times in the sequence. That is, we want to ensure that the distribution is uniform. The frequency test for a sequence S over some interval $[a, b]$ is relatively straightforward: namely, ensuring that every digit occurs in the sequence approximately 10% of the time, as would be expected of a random sequence of digits. However, it is important to apply the frequency test over every segment of the sequence used to ensure that there isn't a singularly bad section. That is, we want to split the sequence into multiple blocks and ensure that, within each block, every digit still occurs about 10% of the time [3, 5, 11].

The serial test is designed to check that no two digits tend to occur in a particular order. This is essentially the frequency test again, but this time ensuring all groupings of two digits occur approximately 1% of the time within the sequence [3, 5, 11].

In the poker test, the digits in the sequence are broken into blocks of some fixed size (eg. 5 each) and then compared with might be expected in a theoretical poker hand. In [11], the only possible hands considered were pairs, three of a kind, four of a kind, and five of a kind. We might also consider occurrences such as full houses and two pairs, as in [9]. Each of these "hands" has its own likelihood of occurring. To determine what these probabilities are, we need to determine what kind of deck we are working with. We note already that, because we can conceivably have five of a kind in a hand, it cannot be the normal 52-card deck in four suits. In fact, our deck is actually an infinite one, with no suits but 10 unique cards (one for each digit). For a hand of five cards $H = (a, b, c, d, e)$, we see that each place could be any one of the 10 cards, and so there are 50 possible hands to look at. However, to be more careful we can also look at the order in which the cards are pulled. For example, we may have a pair (a, a, b, c, d) of a pair (a, b, c, a, d), and each would count as a separate

[1]The entirety of the rest of this discussion can be found in [3].

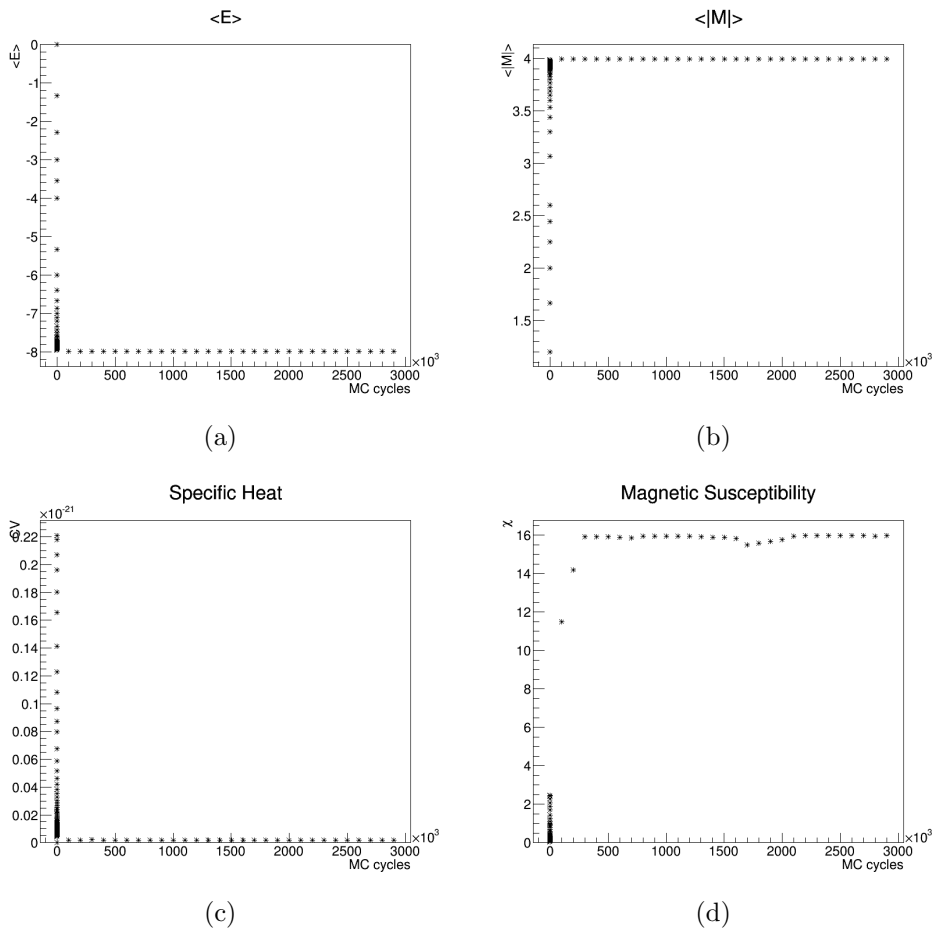


Figure 3: The (a) $< E >$, (b) $< |M| >$, (c) C_V, and (d) χ for a 2×2 lattice at temperature $T = 1/k_B$.

hand. So there are actually 10^5 possible hands in our game of poker.

How might we calculate the probability of one of the special kinds of hands of occurring? We will now present a couple of examples. The accepted probabilities of each of the hands is given in Table 2. The simplest example would be five of a kind, which must always take the form (a, a, a, a, a) for some $a \in \mathbb{Z}_n$[2]. As there are only 10 such values of a, there are only 10 such hands in the total of 10^5 hands, and so 5 of a kind should occur with a frequency of 0.01%.

Perhaps a more enlightening example is the case of four of a kind. Here, there are 5 possible hand combinations: (a, a, a, a, b), (a, a, a, b, a), (a, a, a, b, a), (a, a, b, a, a), (a, b, a, a, a), and (b, a, a, a, a). Now a can represent any of the 10 digits in $\mathbb{Z}_n$, and b can be any of those 10 digits except a, so there are 9 possible values of b for every a. As a result, we have a probability of four of a kind equal to

$$P(\text{4 of a kind}) = \frac{5 \times 10 \times 9}{10^5} = 0.45\%.$$

[2]Here, $\mathbb{Z}_n = \{x \in \mathbb{Z} : 0 \leq x < n\}$.

Continuing in this manner (determining the number of possible hand types which will give the desired hand and multiplying by the number of possible values for each variable), we can get the results of Table 2 [3].

Hand Type	Frequency (%)
5 of a Kind	0.01
4 of a Kind	0.45
3 of a Kind	7.2
2 of a Kind	50.4
Full House	0.9
Two Pair	10.8
Other	30.24

Table 2: Accepted frequencies for various poker hands [9].

To determine how well the output of the random number generator fits the expected values of the frequency, series, and poker tests, we perform a χ^2-test to check the agreement. This test is designed to determine whether there is any correlation between two variables x and y. To perform the χ^2-test, we first compute χ^2, which is given by

$$\chi^2 = \sum \frac{(v - v_e)^2}{v}, \tag{3.1}$$

where v is the value observed and v_e is the value expected [9].

Next, we determine the number of degrees of freedom of the problem, which is given by

$$DF = (n_x - 1) * (n_y - 1), \tag{3.2}$$

where n_x is the number of levels of variable x and n_y is the number of levels of variable y [1]. It is important, therefore, to note the degrees of freedom in each of the three tests used in this analysis, as shown in Table 3.

Test	Degrees of Freedom
Frequency Test	9
Serial Test	90
Poker Test	4

Table 3: The degrees of freedom of the various tests used in this analysis [9].

The final step in the χ^2-test is to consult a χ^2 table (or, in our case, an online p-value calculator [2]) to determine the p-value given the number of degrees of freedom and the computation of χ^2. This p-value gives the likelihood that differences in the observed and expected values is due to statistical fluctuations rather than a more fundamental error in the generation of the sequence. In particular, we look for p to be above a certain significance level, which we take to be 0.1 (although values of 0.01 and 0.05 are also common) [3].

The five random number generators tested here are `ran0`, `ran1`, `ran2`, and `ran3` from [12], and `rand()`, the default C++ random number generator. `ran0` is what is known as the Minimal Standard Generator, which is a multiplicative congruential generator. That is, it depends on the recurrence relation

$$I_{j+1} = aI_j \mod m \tag{3.3}$$

`ran0` specifically was proposed by Park and Miller with the settings $a = 7^5 = 16807$ and $m = 2^{31} - 1 = 2147483647$. The period of `ran0` is expected to be $2^{31} - 1 = 2.1 \times 10^9$ recipes. We found that ran0 behaves fairly well. It passes the frequency and serial tests without much of a problem. The only test which is does not appear to pass is the poker test. Here, the p-value is ~ 0.04, which is far below our significance threshold.

Statistic	Value	Significance
μ	4.50159	0.04%
σ	2.87309	0.45%
χ^2 (Frequency Test)	5.68698	0.7708
χ^2 (Serial Test)	86.2657	0.5919
χ^2 (Poker Test)	10.0296	0.0399

Table 4: Results of the frequency, serial, and poker tests performed on `ran0`, and the significance of those results (the percent difference from the expected value in the case of the mean μ and the standard deviation σ, and the p-value for the χ^2 tests).

`ran1` uses `ran0` for its random variable but also uses a shuffling algorithm developed by Bays and Durham to remove lower-order correlations. In this shuffling, a randomly generated element of the sequence, I_j, is chosen to be output as some other element of the sequence (typically I_{j+32}). According to [12], `ran1` passes all known statistical tests excepting when it is called somewhere on the order of its period ($\sim 10^8$) [12].

We found that this algorithm passes the frequency and poker tests, but not the serial test. This is an interesting result as `ran1` is designed to help rid `ran0` of correlations, and `ran0` passed the serial test without a problem.

Statsitsic	Value	Significance
μ	4.50043	0.01%
σ	2.87041	0.54%
χ^2 (Frequency Test)	7.19849	0.6165
χ^2 (Serial Test)	112.818	0.0522
χ^2 (Poker Test)	5.22794	0.2647

Table 5: Results of the frequency, serial, and poker tests performed on `ran1`, and the significance of those results (the percent difference from the expected value in the case of the mean μ and the standard deviation σ, and the p-value for the χ^2 tests).

Claimed to be a "perfect" random number generator (up to floating point errors) by [12], `ran2` adds another level of shuffling to further remove the correlations existing in `ran1`, as well

as other randomness checks originally proposed by L'Ecuyer. One such check is combining `ran0` and `ran1` such that the period of the sequence generated is the least common multiple of the periods of the first two algorithms. This is done by adding the two sequences produced by these algorithms modulo the modulus m of either of them. In this way, `ran2` is designed to have a very long period ($> 10^8$) [12].

We found that this algorithm readily passed all of the frequency, serial, and poker tests. In particular, it also has the lowest percent difference between the observed and expected average and standard deviation of any of the three algorithms discussed up until this point.

Statsitsic	Value	Significance
μ	4.4999	$< 0.01\%$
σ	2.87382	0.42%
χ^2 (Frequency Test)	8.62035	0.4730
χ^2 (Serial Test)	93.7899	0.3714
χ^2 (Poker Test)	5.28678	0.2591

Table 6: Results of the frequency, serial, and poker tests performed on `ran2`, and the significance of those results (the percent difference from the expected value in the case of the mean μ and the standard deviation σ, and the p-value for the χ^2 tests).

`ran3` was originally presented by Knuth and is in fact based on subtractive methods rather than acting as a linear congruential method. These subtractive methods make `ran3` a Fibonacci generator. The idea here is that the next term in the sequence comes from the subtraction of two previous terms. In particular,

$$x_k = x_{k-a} - x_{k-b} \tag{3.4}$$

for some fixed $a, b \in \mathbb{Z}$ known as lags. Fibonacci generators are typically very efficient and pass most randomness tests thrown at them, although they do require more storage than linear congruential algorithms [8] (of which `ran0`, `ran1`, and `ran2` are special cases).

We found that, indeed, `ran3` passed all three of the frequency, serial, and poker tests, just as did `ran2`. However, this algorithm had much lower p-values for the serial and poker tests than did `ran2`, indicating that perhaps this is not the stronger of the two.

Statsitsic	Value	Significance
μ	4.50161	0.04%
σ	2.87307	0.45%
χ^2 (Frequency Test)	5.92085	0.7478
χ^2 (Serial Test)	106.29	0.1157
χ^2 (Poker Test)	7.28835	0.1214

Table 7: Results of the frequency, serial, and poker tests performed on `ran3`, and the significance of those results (the percent difference from the expected value in the case of the mean μ and the standard deviation σ, and the p-value for the χ^2 tests).

The final random number generator tested in this analysis is the standard `rand()` function in the standard C++ library. According to the C++ documentation, the default underlying algorithm here is another linear congruential algorithm. We chose to test this algorithm primarily as a check that the code created to run the four randomness tests was working properly, working under the assumption that, as it comes standard, it has likely been well-tested at this point.

We found that, as expected, `ran4` passed the frequency, serial, and poker tests with flying colors. However, it failed the gap test (as did the other four random number generators tested). This result leads the author to believe that the code used in the gap test is flawed somehow, and that further study is needed before any claims can be made.

Statistic	Value	Significance
μ	4.50236	0.05%
σ	2.87138	0.51%
χ^2 (Frequency Test)	7.25112	0.6110
χ^2 (Serial Test)	79.4787	0.7784
χ^2 (Poker Test)	3.55947	0.4689

Table 8: Results of the frequency, serial, poker, and gap tests performed on `ran4`, and the significance of those results (the percent difference from the expected value in the case of the mean μ and the standard deviation σ, and the p-value for the χ^2 tests).

4 Results

To test our algorithm, we looked first at a 20×20 lattice to determine how many MC cycles would be needed to reach the steady state. Looking first at a temperature of $T = 1/k_B$, we plotted $< E >$, $< |M| >$, C_V, and χ against the number of cycles used in the computation. The results are in Fig. 4 for a random initial state and in Fig. 5 for a steady initial state.

We see that we are able to get good results for $< E >$ and $< |M| >$ after very few MC cycles in both the initial random and the initial steady states, although there was a much wider variation in the values produced for these quantities when the initial state was random. This is not suprising given that the steady state starts right near the accepted value for these quantities and only varies with a small probability from that central value, whereas the initial random state needs to find the accepted value from some random initial state. We also see much less variation once the steady state is reached for the initial steady state. Generally, though, the number of MC cycles required to reach the equilibrium values didn't seem to depend much on the initial state of the lattice for such a small lattice size, although starting in the steady state was a bit faster.

We also wanted to look at how the number of MC cycles required to reach the steady state approximation might depend on temperature. Thus, we repeated the analysis of our expectation values, C_V, and χ using $T = 2.4/k_B$. The results are shown in Fig. 6 for the random initial state and Fig. 7 for the initial steady state. Here, we see that there is a distinct difference between the number of MC cycles required to reach the equilibrium values for the

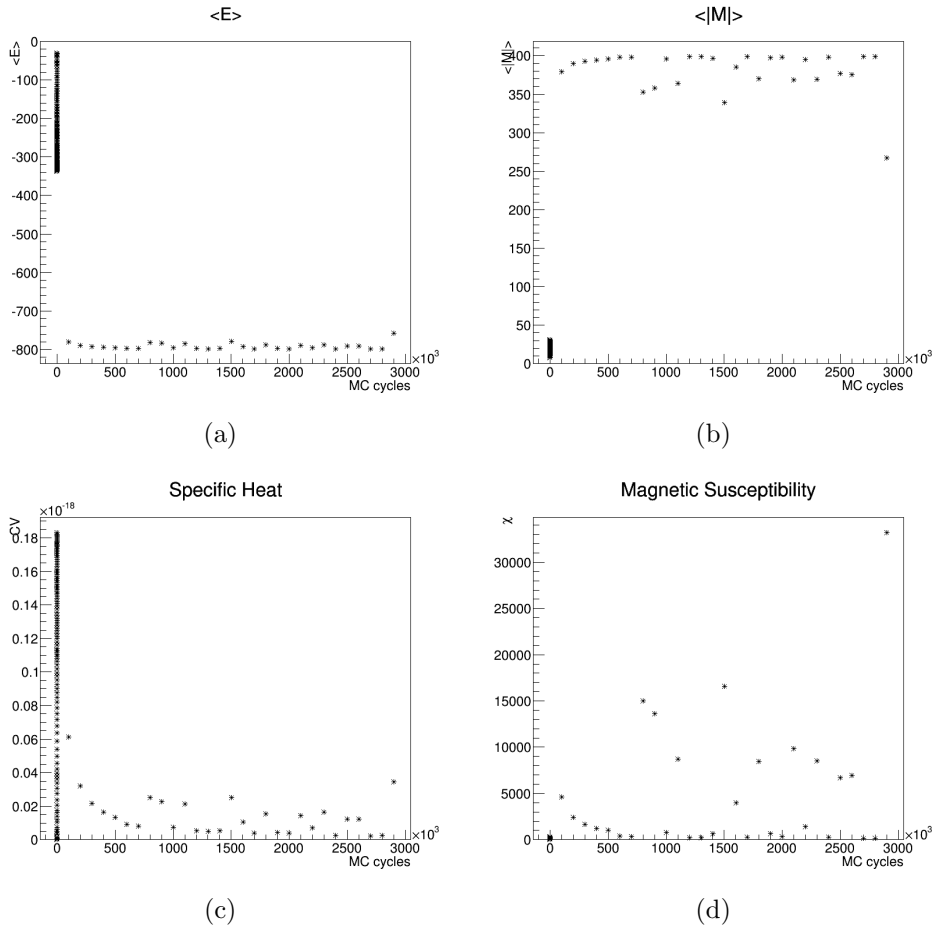


Figure 4: Statistical quantities for a 20×20 lattice at a temperature $T = 1/k_B$ with an initial state of a random selection of spins.

statistical quantities, which is particularly noticeable in the C_V plots. We see that with the random initial state, we require $\sim 500 \times 10^3$ MC cycles to reach the equilibrium value for C_V, but only $\sim 400 \times 10^3$ or less for the initial steady state. Both of these values, though are much larger than the number of cycles required with $T = 1/k_B$, which is $\sim 100 \times 10^3$ for the initial steady state and $\sim 300 \times 10^3$ for the random initial state.

One interesting feature of Fig. 6 and Fig. 7 is that the magnetic susceptibility never seems to reach a reasonable equilibrium value, regardless of the number of MC cycles. Even in Fig. 4 and Fig. 5, χ has a large variation and doesn't approach the equililbrium value strongly. It may be possible that this is due to the fact that $< M >$, rather than $< |M| >$, was used in the calculation.

We next looked at the probability of various energies in the calculation, $P(E)$, defined to be the number of times a given energy E occurs in the MC process divided by the number of cycles used. The results starting from the steady state are shown in Fig. 8. As we can see, at both temperatures, the most likely energy is the equlibrium energy. However, there are far fewer energies used in the calculation for the lower-temperature system. It is reasonable

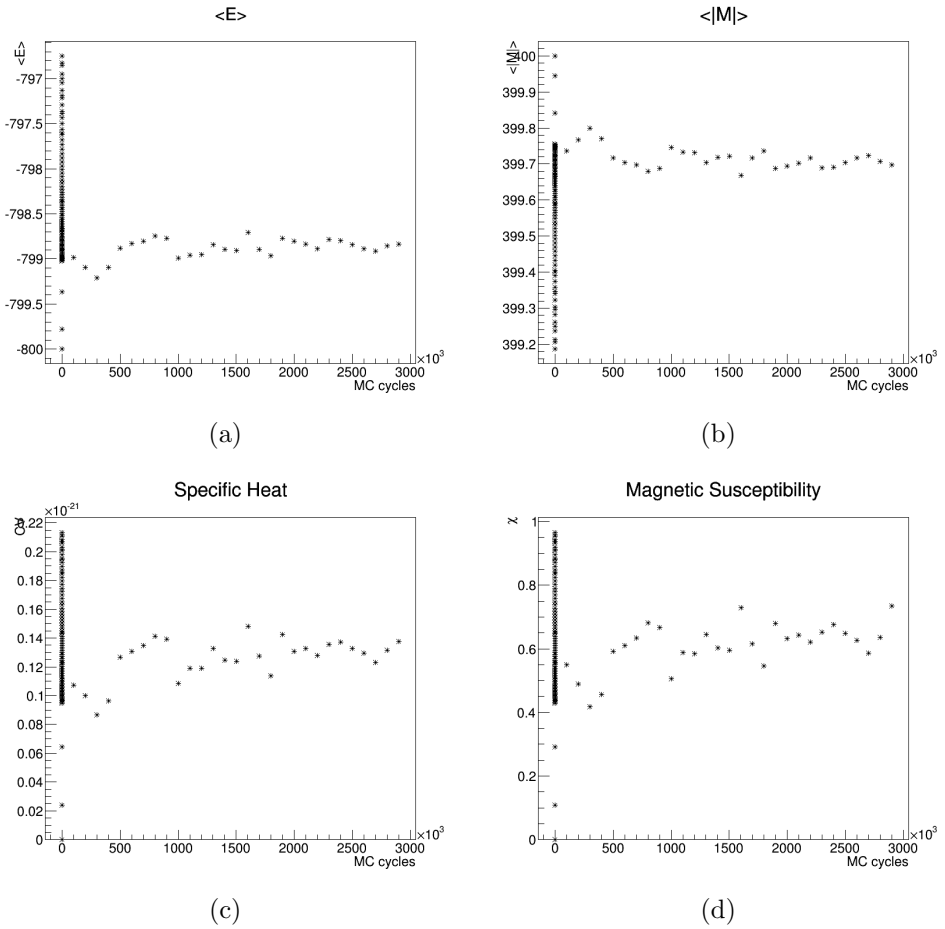


Figure 5: Statistical quantities for a 20×20 lattice at a temperature $T = 1/k_B$ with a steady-state initial state.

to look to the C_V plots in Fig. 5 and Fig. 7, because C_V is the variance of E up to a multiplicative factor. We see that $C_V \approx 0.12$ for $T = 1/k_B$ but $C_V \approx 8$ for $T = 2.4/k_B$. This implies that we would expect a far larger variation in possible energy values for the higher temperature system.

This is all well and good, but we would like now to do some physics with our Ising model calculation. In particular, we can look at phase transitions by investigating the various statistical mechanical properties which we been focusing on. A phase transition is when the matter undergoes a change of phase (eg. liquid to gas or solid to liquid, etc.). It occurs at the critical temperature of the system, denoted T_C. Near T_C, many physical quantities can be characterized by a power law exponent. For example, the mean magnetization is given by

$$< M(T) > \sim (T - T_C)^{1/8}, \tag{4.1}$$

the heat capacity by

$$C_V(T) \sim |T_C - T|^0, \tag{4.2}$$

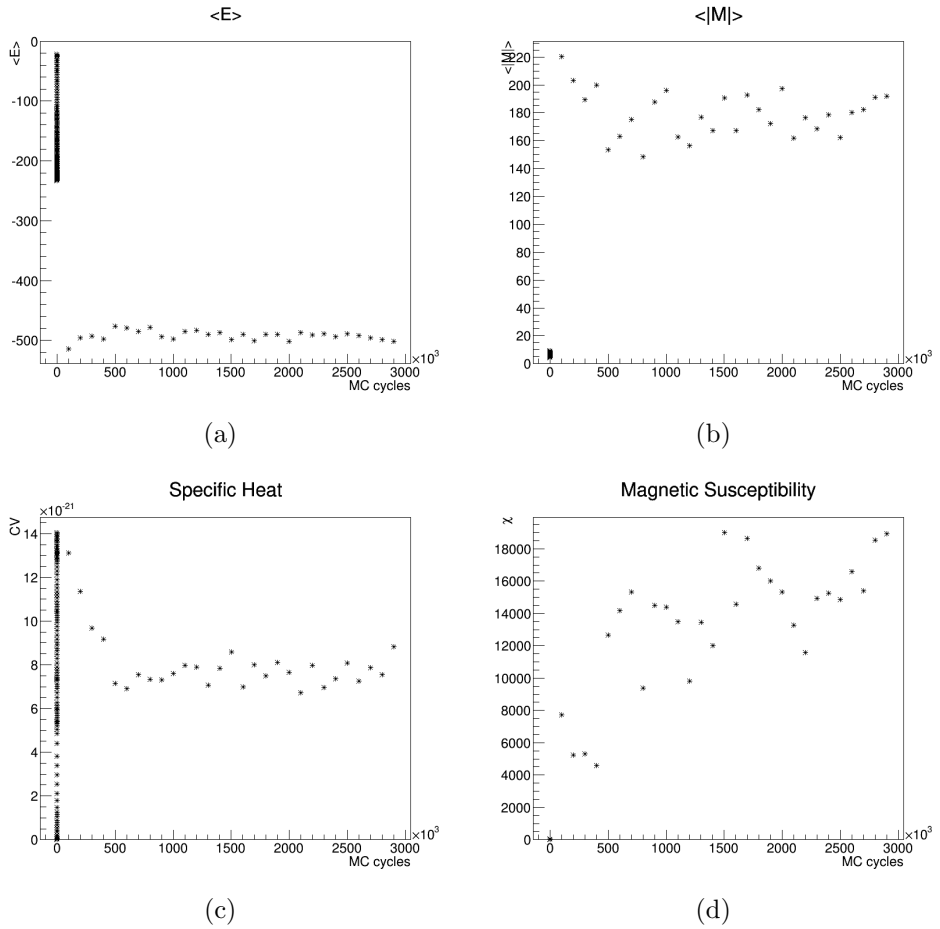


Figure 6: Statistical quantities for a 20×20 lattice at a temperature $T = 2.4/k_B$ with an initial state of a random selection of spins.

and the susceptibility by

$$\chi(T) \sim |T - T_C|^{\frac{7}{4}}. \tag{4.3}$$

In addition, with the introduction of T_C, we can introduce the correlation length ξ, which is a measure of how correlated the spins within the lattice are to one another. It is expected that ξ is on the order of the lattice spacing for $T >> T_C$, but increases drastically as

$$\xi(T) \sim |T_C - T|^{-\nu} \tag{4.4}$$

for T near T_C because near T_C the spins begin to correlate more and more [6].

At a second-order phase transition, we expect ξ to span the whole system. This becomes a problem because we are restricted to finite-sized lattices. However, this means that ξ will scale with the size of the lattice, and so we can use a finite lattice to approximate an infinite system. In particular, the critical temperature will scale as

$$T_C(L) - T_C(L = \infty) = aL^{\frac{1}{\nu}} \tag{4.5}$$

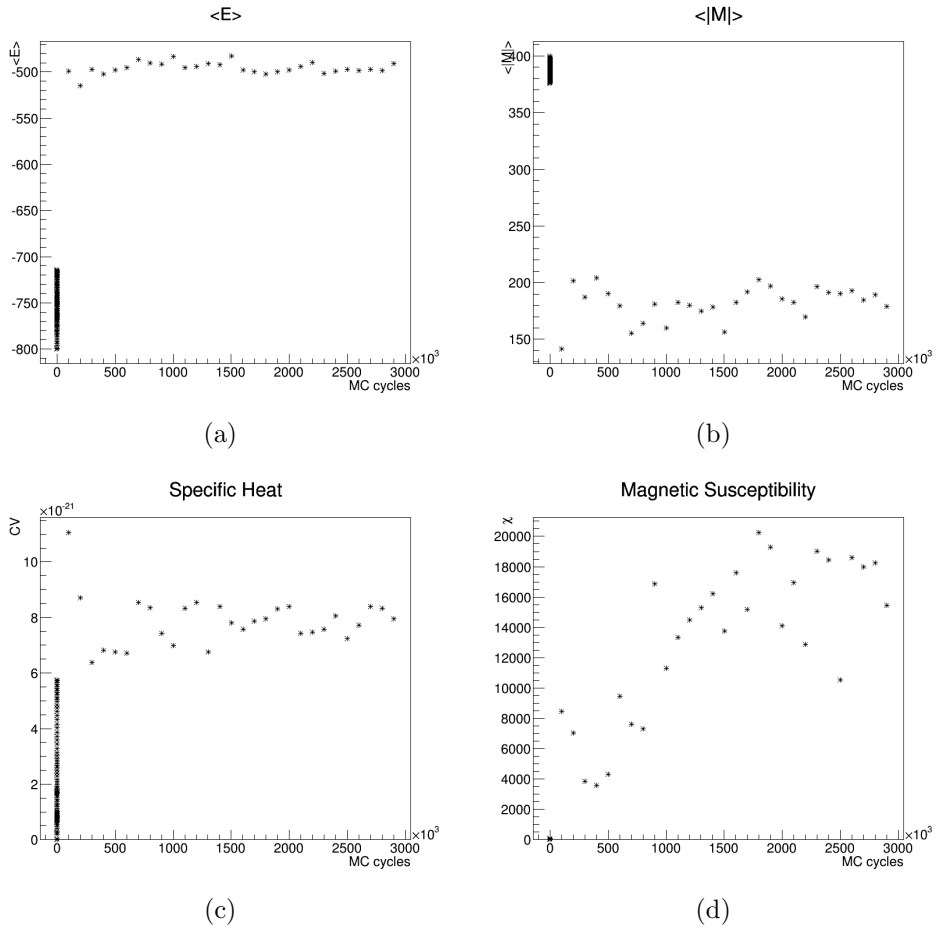


Figure 7: Statistical quantities for a 20×20 lattice at a temperature $T = 2.4/k_B$ with a steady-state initial state.

for some constant a for a lattice of size L. By setting $T = T_C$, we can then see that (4.1), (4.2), and (4.3) become [6]

$$< M(T) > \sim (T - T_C)^{\frac{1}{8}} \to L^{\frac{1}{8\nu}} \tag{4.6}$$

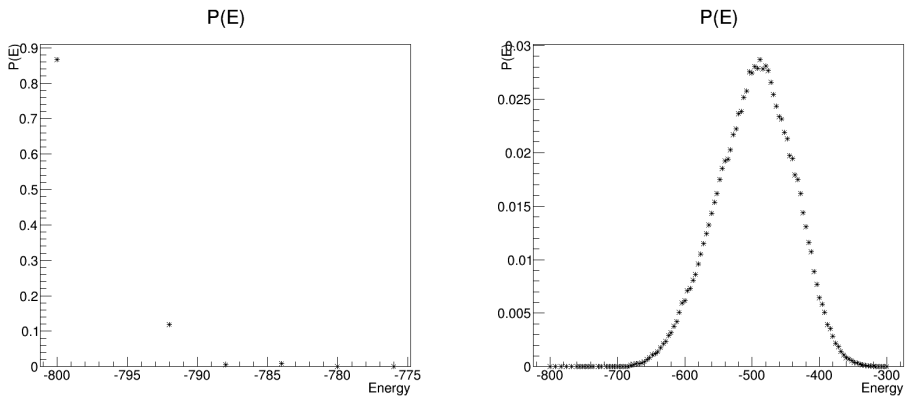
$$C_V(T) \sim |T_C - T|^{-\frac{7}{4}} \to L^0 \tag{4.7}$$

$$\chi(T) \sim |T_C - T|^0 \to L^{\frac{7}{4\nu}}. \tag{4.8}$$

In order to study T_C, we look for indications of a phase transition in the graphs of $< E >$, $< |M| >$, C_V, and χ[3] for various lattice sizes. These plots are shown in Fig. 9, Fig. 10, Fig 11, and Fig. 12. They are made with 10^6 MC cycles at each temperature.

In all of the plots, the evidence of a phase transition is obvious. Just before $T = 2.5/k_B$, the $< E >$ curve slopes sharply upward, the $< |M| >$ curve slopes sharply downward, and both C_V and χ peak sharply. In particular, we can note that the peak is sharper to the left in these cases, and has a somewhat longer tail.

[3]Here, χ is calculated with $< |M| >$.



(a) (b)

Figure 8: $P(E)$ for a 20×20 lattice at a temperature of (a) $T = 1/k_B$ and (b) $T = 2.4/k_B$.

We wish to use these results to determine T_C in the limit $L \to \infty$, using $\nu = 1$. The accepted value is given by $T_C = 2.269/k_B$ [6]. We can use (4.5) to calculate $T_C(L = \infty)$ from our plots. To do this, we need the information of $T_C(L)$. This is found in Table 9.

L	T_C $(1/k_B)$ (from C_V)	T_C $(1/k_B)$ (from χ)	Average T_C $(1/k_B)$
20	2.25	2.35	2.30
40	2.30	2.35	2.33
60	2.35	2.40	2.38
80	2.40	2.45	2.43

Table 9: Approximate values for $T_C(L)$ for $L = 20, 40, 60$, and 80 used with (4.5) to determine $T_C(L = \infty)$.

Now we know

$$T_C(L = \infty) = T_C(L) - aL^{-1},$$

for some $a \in \mathbb{R}$. So

$$\begin{aligned} T_C(L = \infty) &= \frac{2.30}{k_B} - \frac{a}{20} \\ T_C(L = \infty) &= \frac{2.33}{k_B} - \frac{a}{40} \\ T_C(L = \infty) &= \frac{2.38}{k_B} - \frac{a}{60} \\ T_C(L = \infty) &= \frac{2.43}{k_B} - \frac{a}{80}. \end{aligned}$$

Because (4.5) gives us to equations in 2 unknowns, and there we are looking at four different lattice sizes, we can get any number of 8 possible solutions for T_C and a. After performing these calculations in Mathematica [7], we see get the possible solutions shown

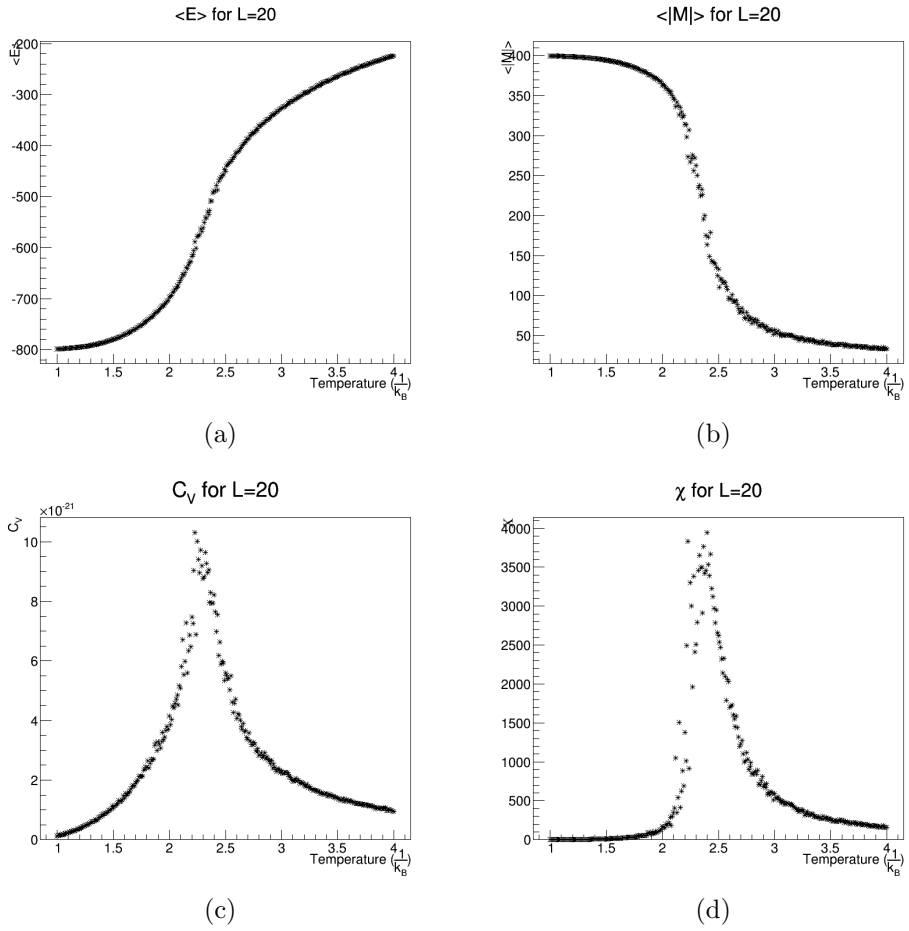


Figure 9: Statistical quantities for a 20×20 lattice with a steady initial state plotted against the temperature.

in Fig. 13. That is, we have $\frac{2.35}{k_B} \leq T_C \leq \frac{2.575}{k_B}$. This is an error of anywhere between $3.57\% \leq \sigma \leq 13.49\%$ compared with the accepted result of $2.269/k_B$ [6]. Given that our values for $T_C(L)$ were estimated by eye from the graphs, this is a very reasonable result.

4.1 Testing Random Number Generators

As discussed in Section 3.1, we are interested particularly in looking at the effect of random number generators on our results, looking particularly at `ran0`, `ran1`, `ran2`, `ran3`, and `rand()`, the standard C++ random number generator, with which all of the data in Section 4 was created. Overall, we find that the choice of random number generator didn't have major contributions to the results determined. However, small differences were observed.[4] For example, it was observed that, if we plotted the number of accepted MC against the total number of MC cycles used in the calculation, the slopes were slightly different. In

[4]Throughout this discussion, we are working with the generation of a 2×2 lattice with an initial random configuration at a temperature of $T = 1/k_B$.

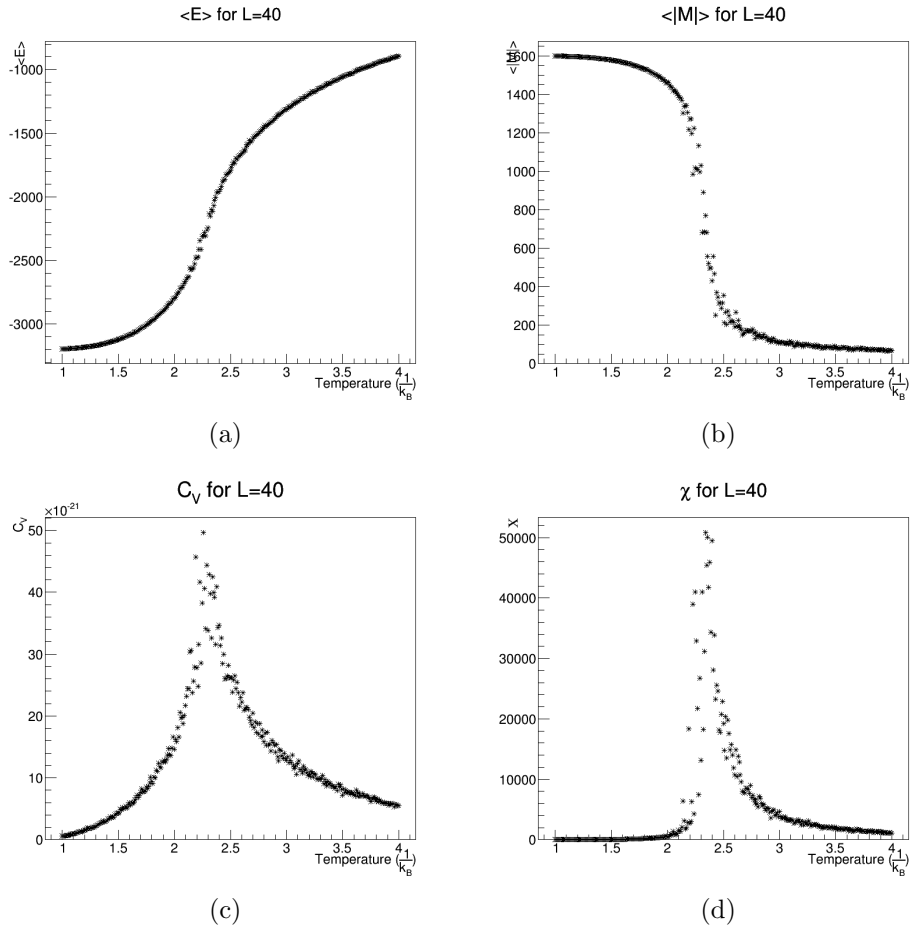
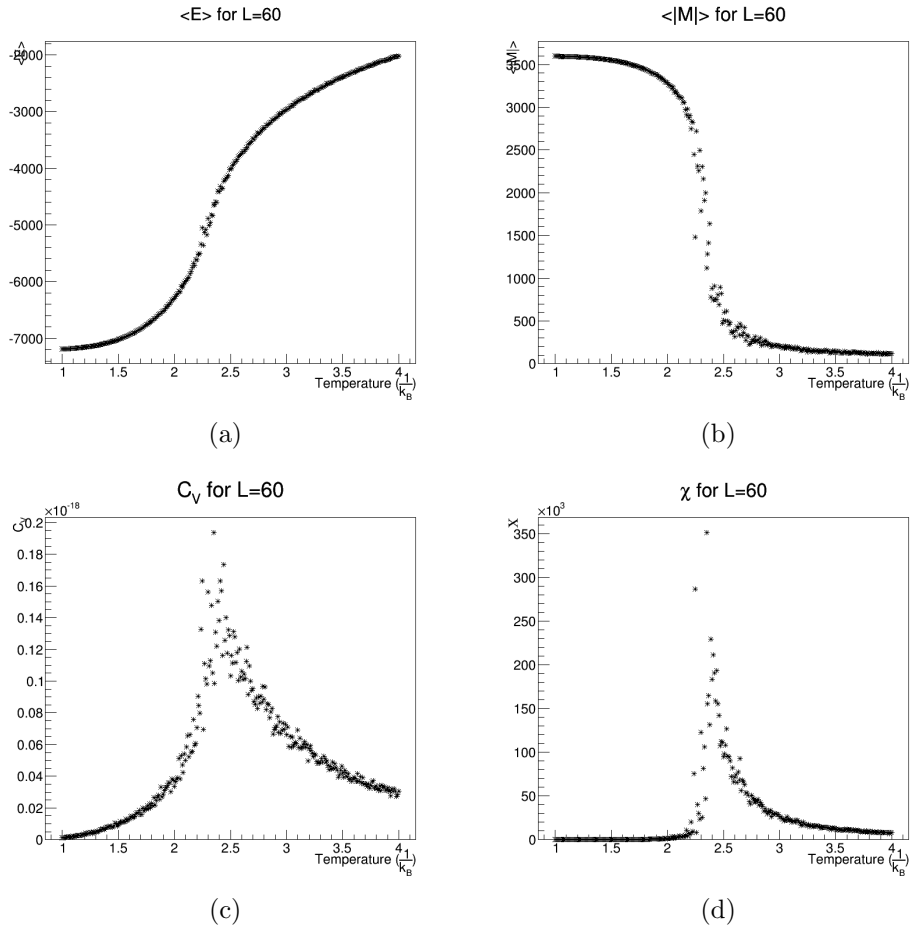


Figure 10: Statistical quantities for a 40×40 lattice with a steady initial state plotted against the temperature.

particular, `ran1`, `ran3`, and `rand()` have more accepted MC than do `ran0` or `ran2`.

But what is really important is whether or not changing the random number generator would effect the physics. As an example, see Fig. 14, where we have plotted the magnetic susceptibility χ as a function of the total number of MC cycles. Recall from Section 4 that it appeared that χ was the most unstable of the statistical quantities being observed. In fact, it does appear that the variation – or the number of MC cycles needed to get to the equilibrium value – may depend on the random number generator used. In particular, the more random the generator (according the number of tests passed as discussed in Section 3.1 and the significance of the associated χ^2 tests), the more MC cycles were required before the susceptibility approached a constant value. Still, regardless of the random number generator used, the final value of χ was the same.

Because no effect was more pronounced than that on χ, we assume that it is safe to use any of the five random number generators tested. It is expected, however, that, as the number of MC cycles increases, this will no longer be true. This is because the period of the random number generators will eventually play a roll in the calculations, which is an undesired result



(a) (b)

(c) (d)

Figure 11: Statistical quantities for a 60×60 lattice with a steady initial state plotted against the temperature.

(at that point the numbers can be considered correlated and thus, by definition, no longer random). Statistical biases in the form of patterns may begin to creep up in the data when the number of MC cycles nears the period of the random number generator being used, and so it is important to find a generator with a remarkably long period (such as the Mersenne Twister, which has a period of roughly 2^{19937} [6]).

5 Conclusions

The Ising model in two dimensions is an immensely useful tool in statistical mechanics as a means by which to calculate expected values (such as that of the energy and magnetization), as well as fundamental properties of matter (such as the heat capacity and magnetic susceptibility). From these quantities, it is possible to determine the critical temperature, T_C, at which phase transitions occur. We were able (see Section 4) to do this calculation for an infinite lattice to within 14% accuracy by estimating the T_C from plots generated of χ and C_V of various sized lattices.

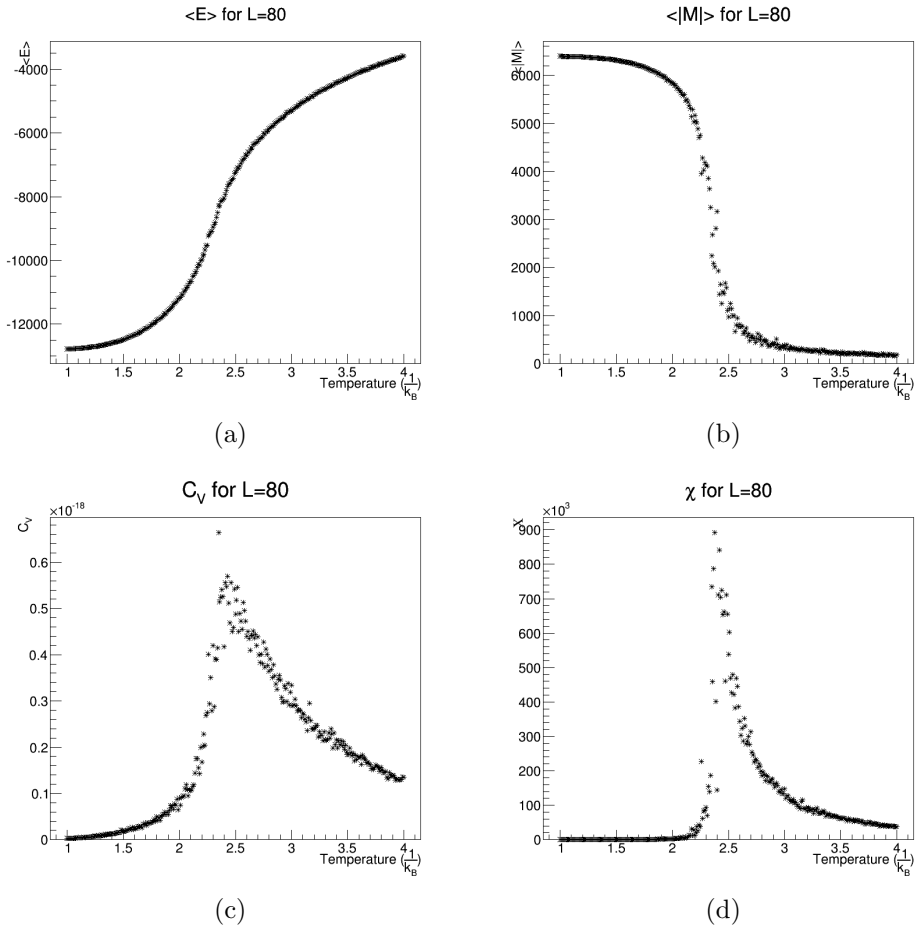


Figure 12: Statistical quantities for a 80×80 lattice with a steady initial state plotted against the temperature.

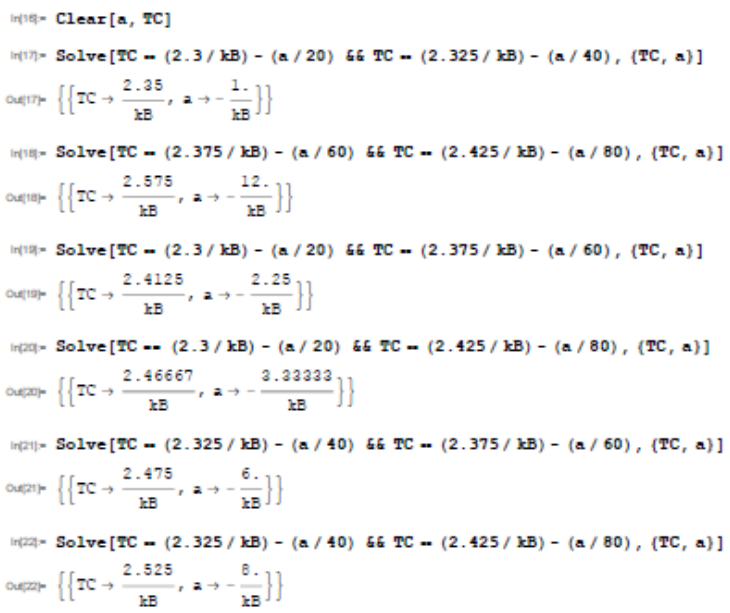
In addition to its physical importance, the Ising model proves to be an excellent introduction to Monte Carlo generation. We were able to develop a code to implement Monte Carlo under the guise of the Metropolis Algorithm citelecture within a new `lattice` class, which performed the calculations of our various important statistical mechanical quantities for us. Because random number generation is key in any Monte Carlo simulation, we were also able to run some benchmark tests on five common random number generators to ensure that they were, indeed, random, and that the physics produced was not affected by the random number generator chosen. Overall, the results of this project followed what was expected.

References

[1] Chi-square test for independence. web, 2016.

[2] P value calculator. web, 2016.

[3] E. Drueke. Pseudorandom sequences and random number generators. April 2016.



(a)

Figure 13: Using Mathematica [7] to solve for the possible values of $T_C(L = \infty)$ given the data from Table 9 and (4.5).

[4] D. J. Griffiths. *Introduction to Electrodynamics*. Prentice Hall, Upper Saddle River, NJ 07458, 3 edition, 1999.

[5] F. Gruenberger. Notes. *Mathematics of Computation*, 4:244–245, 1950.

[6] M. Hjorth-Jensen. Computational physics lecture notes fall 2015, August 2015.

[7] W. R. Inc. *Mathematica 8.0*, 2010.

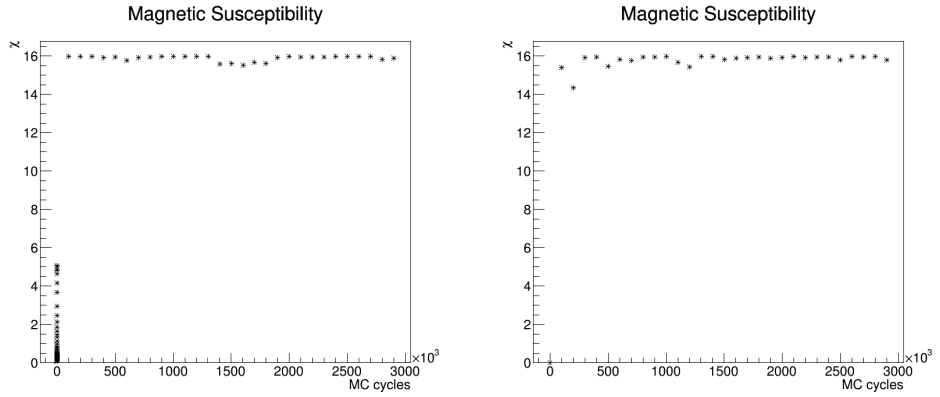
[8] D. Kahaner, C. Moler, and S. Nash. *Numerical Methods and Software*. Prentice Hall, 1989.

[9] M. G. Kendall and B. B. Smith. Randomness and random sampling numbers. *Journal of the Royal Statistical Society*, 101(1):147–166, 1938.

[10] C. R. Nave. Magnetic field of the earth. http://hyperphysics.phy-astr.gsu.edu/hbase/magnetic/magearth.html, 1999.

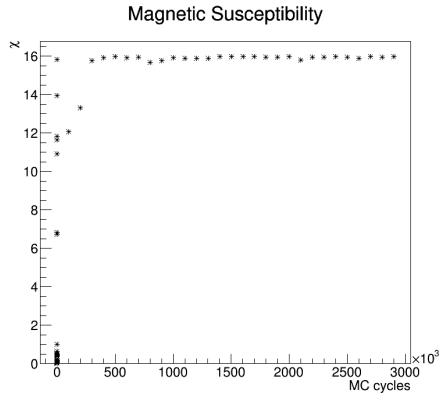
[11] T. G. Newman and P. L. Odell. *The Generation of Random Variates*, volume 29 of *Griffin's Statistical Monographs and Courses*. Hafner Publishing Company, 42 Drury Lane, London, WC2B 5RX, 1971.

[12] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 3 edition, September 2007.

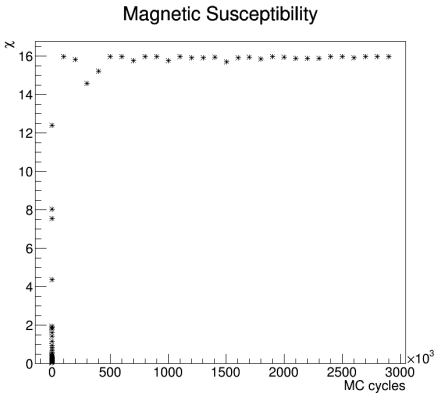


(a)

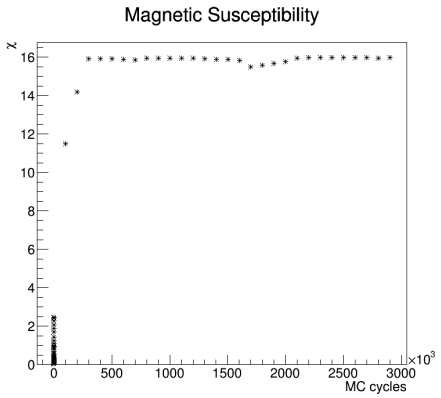
(b)



(c)



(d)



(e)

Figure 14: Magnetic susceptibility χ as a function of the number of MC cycles used in the calculation for (a) `ran0`, (b) `ran1`, (c) `ran2`, (d) `ran3`, and (e) `rand()`.