Kiril Panayotov Blagoev 3753135

Homework 5

## 1 Linear Congruential Generator - revisited

### 1.1 Introduction

For this case the same implementation as in the last homework will be used, but new values for the variables will be used to mimic drand48. drand48 generates a 48bit integer, which is then normalised to a double precision floating-point number in the range $[0.0, 1.0)$ Thus the variables required to do this will be $a = 25214903917$, $c = 11$, $m = 2^{48}$ with normalisation factor $1/m$.

### 1.2 Results

The following results were obtained for the averages

| N | Average | Variance |
|---|---|---|
| $10^3$ | 0.49590 | 0.06272 |
| $10^4$ | 0.04993 | 0.07074 |
| $10^5$ | 0.50097 | 0.08273 |

The averages do still cluster around 0.5 as would be expected analytically from a uniform distribution. Because of the lack of periodicity, they cluster in a different manner around the middle for different numbers of iterations, as can be seen also in the running averages graph

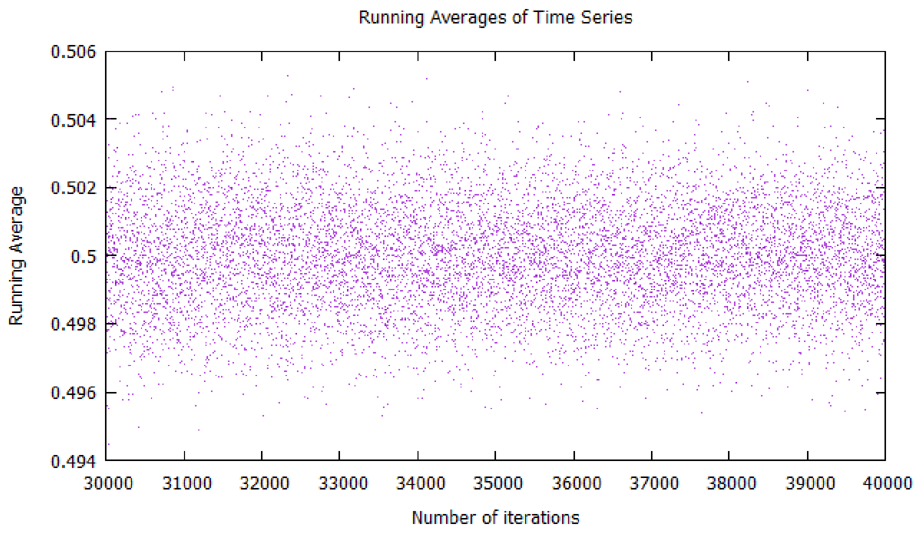


Figure 1: Non-periodic behaviour of the running averages

No fun behaviour of the evenly and oddly indexed numbers is observed either

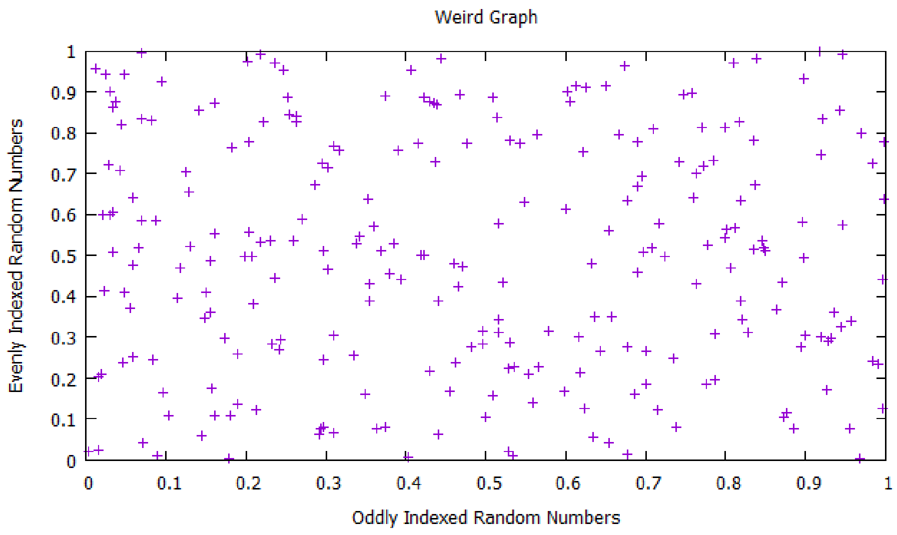


Figure 2: :(

For completeness' sake, we can show that no periodicity is observed for $10^5$ generated numbers

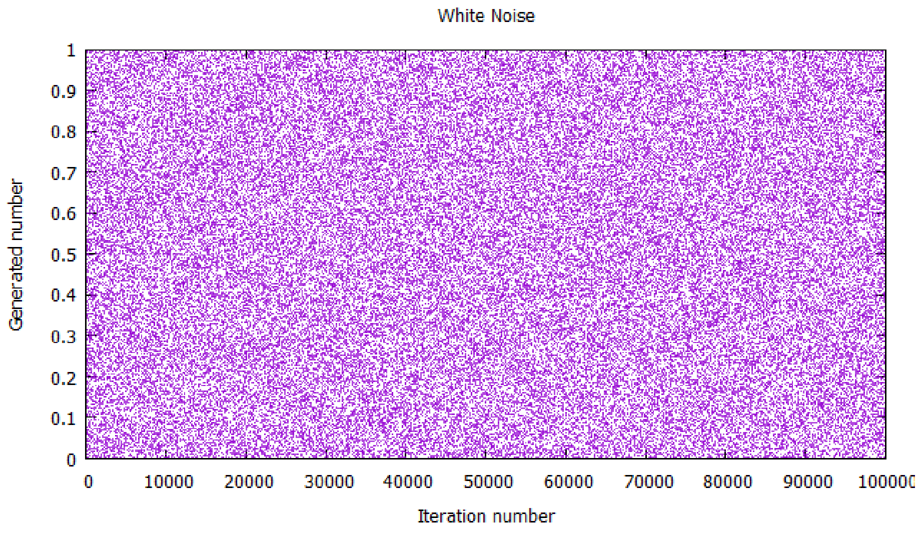


Figure 3: Non-periodic behaviour of the generated numbers

## 2 Monte Carlo simulation of the 1D Ising Model

### 2.1 The Code

The base class for the model used is listed below and heavily commented. It was written in C# in the environment of the game engine Unity, making use of it for the visual and organisational aspect.
The rest of the code contains classes for manipulating the model, sending variables to it, taking the calculations, showing the simulation and the data on the screen, and so on, but is not listed for its lack of relevance.

```
using System.Collections.Generic;
using System;
using UnityEngine;

public class IsingModel {

        // Critical Temperature
        // NOTE: Doesn't to apply to the 1D model!
        public static double critTemperature = 2d / Math.Log (1d + Math.Sqrt (2d));
        // Number of spins:
        int N;
        // A Dictionary for the spins, contains the number of the spin
        // from 0 to N+1 (see later why - has to do with
        // the free boundary conditions) and its value (+1/-1 or
        // 0 for the 0th and N+1st entries):
        public Dictionary<int, int> spins { get; protected set; }
        // Temperature of the simulation, used for computing physical properties,
        // AND for the boltzmann coefficient, thus changing che probability of
        // a successful flip:
        public double temperature = critTemperature;
        // Number of Monte Carlo steps
        public int monteCarloSteps { get; protected set; }
        // Total energy of the system
        int energy;
        // A number of successful flips of the spin:
        int successfulFlips = 0;
        // Used for calculations of physical properties:
        double energyAccumulator = 0d;
        double energySquaredAccumulator = 0d;
        int magnetisation = 0;
        double magnetisationAccumulator = 0d;
        double magnetisationSquaredAccumulator = 0d;
        // An array for the Boltzmann factors, used for changing
        // the probability of a successful flip of the spin:
        double[] w = new double[9];
```

```
// Constructor for the class:
public IsingModel(int N, double temperature)
{
        this.temperature = temperature;
        spins = new Dictionary<int, int>();
        // Set number of spins
        this.N = N;
        // Since we are going to start at minimal energy, we need
        // all spins to be pointing in the same direction,
        // in our case +1
        for (int i = 0; i < N+2; i++)
        {
                // NOTE: Here, by adding N+2 elements, instead of N, we can then
                // assign to the first and last ones a spin of zero, thus
                // implementing free boundary conditions!
                spins.Add (i, 1);
        }

        //Assigning a spin of zero to boundary elements of spins[]:
        spins[0] = 0;
        spins [N + 1] = 0;

        // ground state energy is equal to -1*the number of spins
        // TODO: In one of the pseudo-codes for 2D Ising it was -2*N. Why?
        energy = -N;
        magnetisation = N;
        // setting two of the boltzmann array elements manually
        // Because of the lack of a magnetic field, only these coefficients,
        // coresponding to their energies, will ever occur, so we can skip the
        // calculation of the others.
        // NOTE: For the 2D Ising Model, the energies +-8, +-4 and 0 will appear,
        // while for the 1D model, energies of +-4, +-2, and 0 appear.
        //w [8] = Math.Exp (-8d / temperature);
        w [4] = Math.Exp (-4d / temperature);
        w [2] = Math.Exp (-2d / temperature);

        resetData (N, temperature);
}

public double averageMagnetisation()
{
        return magnetisationAccumulator / monteCarloSteps;
}
```

```java
    public double averageEnergyTotal()
    {
            return energyAccumulator / monteCarloSteps;
    }

    public double specificHeatTotal()
    {

            double averageEnergySquared = energySquaredAccumulator / monteCarloSteps;
            double averageEnergy = energyAccumulator / monteCarloSteps;
            double averageEnergyVariance = averageEnergySquared - averageEnergy *
                ↪ averageEnergy;

            return averageEnergyVariance / (temperature * temperature);
    }

    public double AverageSusceptibility()
    {
            double averageMagnetisationSquared = magnetisationSquaredAccumulator /
                ↪ monteCarloSteps;
            double averageMagnetisation = magnetisationAccumulator / monteCarloSteps;
            double averageMagnetisationVariance = averageMagnetisationSquared -
                ↪ averageMagnetisation * averageMagnetisation;

            return averageMagnetisationVariance / (temperature * N);
    }

    public void resetData(int n, double temperature)
    {
            this.N = n;
            this.temperature = temperature;

            monteCarloSteps = successfulFlips = 0;
            energyAccumulator = energySquaredAccumulator =
                    magnetisationAccumulator = magnetisationSquaredAccumulator = 0d;
    }
```

```
        public void doOneMonteCarloStep()
        {
                System.Random random = new System.Random ();

                // Make sure to do the appropriate number of tries, by making N tries,
                // not N-2, because we have N+2 elements.
                for (int j = 0; j < N; j++)
                {
                        // Choosing a random spin from the N available ones.
                        // NOTE: To finally implement free boundary conditions,
                        // we make sure to not pick elements 0 and N+1 in the random
                        // generator, because they are assigned a spin of zero!
                        int i = random.Next ( 1, N+1);
                        //Debug.Log ("spin number: " + i);
                        // Calculating the difference between the energy of the
                        // neighbourhood before and after a test flip of the spin.

                        int dE = 2 * spins[i] * ( spins[i+1] + spins[i-1] );

                        if (dE <= 0 || w[dE] > random.Next (0, 100000001) / 100000000d)
                        {
                                //Debug.Log ("random.Next: " + temp);
                                // If the difference between the old and new energies
                                // of the neighbourhood of spin i is less than zero, or
                                // if bigger than zero, but the boltzmann coeff. for this
                                // energy is bigger than a randomly generated number
                                // between 0 and 1, then we successfuly flip the spin.
                                spins [i] = -spins [i];
                                successfulFlips++;
                                energy += dE;
                                magnetisation += 2 * spins [i];
                        }
                        // Otherwise nothing changes.
                }
                // At the end of the cycle we just change some variables.
                energyAccumulator += energy;
                energySquaredAccumulator += energy * energy;
                magnetisationAccumulator += magnetisation;
                magnetisationSquaredAccumulator += magnetisation * magnetisation;
                monteCarloSteps++;

        }
}
```

A possible optimisation would be every time the temperature is incremented by a small amount, instead of resetting the data, to feed the current state of the model, in order to approximate the state that the model will reach after $2N$ steps, and speed up the calculation of the variables.

## 2.2 Analysis of the Data

Here are tables with data collected for the different number of spins over a range of $k_bT \in [0.5, 10]$.
The average energy and specific heat are given per spin, while the magnetisation is total, since the magnetisation per spin is expected to always be zero at temperatures above zero. Most data was recorded after 5000-15000 Monte Carlo steps, where more iterations were done for lower temperatures, because of the greater fluctuations and random magnetisation flips. For $N = 10000$ even more steps were made, since less than $2N$ steps wouldn't be enough to stabilise the variables and give good data.

Table 1: Data for $N = 20$ — Table 2: Data for $N = 100$

| | Table 1: Data for $N = 20$ | | | | Table 2: Data for $N = 100$ | | | |
|---|---|---|---|---|---|---|---|---|
| $k_bT$ | $< e >$ | $< c >$ | $< M >$ | $< \chi >$ | $< e >$ | $< c >$ | $< M >$ | $< \chi >$ |
| 0.5 | -0.967 | 0.26 | -3.3 | 30.85 | -0.959 | 0.25 | -2.1 | 61.89 |
| 0.7 | -0.894 | 0.40 | -1.1 | 14.69 | -0.895 | 0.41 | -2.1 | 24.23 |
| 1.0 | -0.771 | 0.42 | -0.3 | 6.50 | -0.763 | 0.41 | -1.2 | 7.20 |
| 1.2 | -0.699 | 0.37 | -0.2 | 4.03 | -0.685 | 0.37 | -0.1 | 4.26 |
| 1.5 | -0.600 | 0.28 | 0.3 | 2.19 | -0.587 | 0.29 | -1.1 | 2.45 |
| 2.0 | -0.489 | 0.19 | 0.0 | 1.27 | -0.467 | 0.20 | 0.3 | 1.28 |
| 2.5 | -0.411 | 0.13 | 0.0 | 0.85 | -0.386 | 0.14 | 0.2 | 0.86 |
| 3.0 | -0.354 | 0.10 | -0.1 | 0.62 | -0.329 | 0.10 | -0.2 | 0.68 |
| 3.5 | -0.317 | 0.07 | -0.1 | 0.49 | -0.286 | 0.07 | 0.6 | 0.49 |
| 4.0 | -0.286 | 0.06 | 0.0 | 0.41 | -0.252 | 0.06 | -0.1 | 0.41 |
| 4.5 | -0.256 | 0.04 | 0.0 | 0.34 | -0.225 | 0.05 | 0.0 | 0.36 |
| 5.0 | -0.232 | 0.04 | 0.0 | 0.28 | -0.203 | 0.04 | 0.2 | 0.28 |
| 5.5 | -0.219 | 0.03 | 0.1 | 0.26 | -0.187 | 0.03 | -0.1 | 0.26 |
| 6.0 | -0.207 | 0.02 | 0.0 | 0.23 | -0.176 | 0.03 | 0.1 | 0.23 |
| 7.0 | -0.181 | 0.02 | 0.1 | 0.19 | -0.149 | 0.02 | -0.1 | 0.19 |
| 8.0 | -0.165 | 0.01 | 0.0 | 0.16 | -0.131 | 0.01 | -0.1 | 0.16 |
| 9.0 | -0.160 | 0.01 | 0.0 | 0.14 | -0.118 | 0.01 | 0.2 | 0.14 |
| 10.0 | -0.149 | 0.01 | 0.1 | 0.12 | -0.111 | 0.01 | 0.2 | 0.12 |

Table 3: Data for $N = 1000$ Table 4: Data for $N = 10000$

| $k_bT$ | $< e >$ | $< c >$ | $< M >$ | $< \chi >$ | $< e >$ | $< c >$ | $< M >$ | $< \chi >$ |
|---|---|---|---|---|---|---|---|---|
| 0.5 | -0.964 | 0.27 | -8.5 | 102.80 | -0.965 | 0.32 | 140.0 | 228.20 |
| 0.7 | -0.891 | 0.42 | 4.6 | 28.20 | -0.891 | 0.45 | 24.0 | 36.01 |
| 1.0 | -0.762 | 0.43 | 5.1 | 8.10 | -0.762 | 0.44 | -1.7 | 9.28 |
| 1.2 | -0.683 | 0.40 | -0.1 | 4.83 | -0.682 | 0.38 | 1.4 | 4.9 |
| 1.5 | -0.583 | 0.29 | -0.4 | 2.73 | -0.583 | 0.31 | 5.0 | 3.14 |
| 2.0 | -0.463 | 0.20 | 3.0 | 1.42 | -0.462 | 0.21 | 3.7 | 1.67 |
| 2.5 | -0.380 | 0.14 | 1.4 | 0.92 | -0.380 | 0.15 | 0.3 | 1.20 |
| 3.0 | -0.322 | 0.10 | -0.9 | 0.73 | -0.321 | 0.10 | -1.2 | 0.76 |
| 3.5 | -0.279 | 0.08 | 1.9 | 0.54 | -0.278 | 0.08 | 1.5 | 0.62 |
| 4.0 | -0.245 | 0.06 | 0.6 | 0.44 | -0.245 | 0.06 | 1.7 | 0.46 |
| 4.5 | -0.220 | 0.05 | 0.1 | 0.36 | -0.219 | 0.05 | 1.2 | 0.36 |
| 5.0 | -0.198 | 0.04 | 0.0 | 0.30 | -0.197 | 0.04 | 0.6 | 0.35 |
| 5.5 | -0.181 | 0.03 | 1.0 | 0.26 | -0.180 | 0.03 | 0.6 | 0.27 |
| 6.0 | -0.167 | 0.03 | 0.5 | 0.24 | -0.165 | 0.03 | 1.5 | 0.24 |
| 7.0 | -0.143 | 0.02 | 0.6 | 0.19 | -0.142 | 0.02 | 0.1 | 0.20 |
| 8.0 | -0.125 | 0.01 | 0.6 | 0.16 | -0.124 | 0.02 | 0.5 | 0.17 |
| 9.0 | -0.112 | 0.01 | 0.5 | 0.14 | -0.111 | 0.01 | 0.4 | 0.14 |
| 10.0 | -0.100 | 0.01 | 0.1 | 0.12 | -0.100 | 0.01 | 0.1 | 0.13 |

Here are all the graphs imposing the data gathered through the simulator over the analytic data from the enumeration technique

Figure 4:

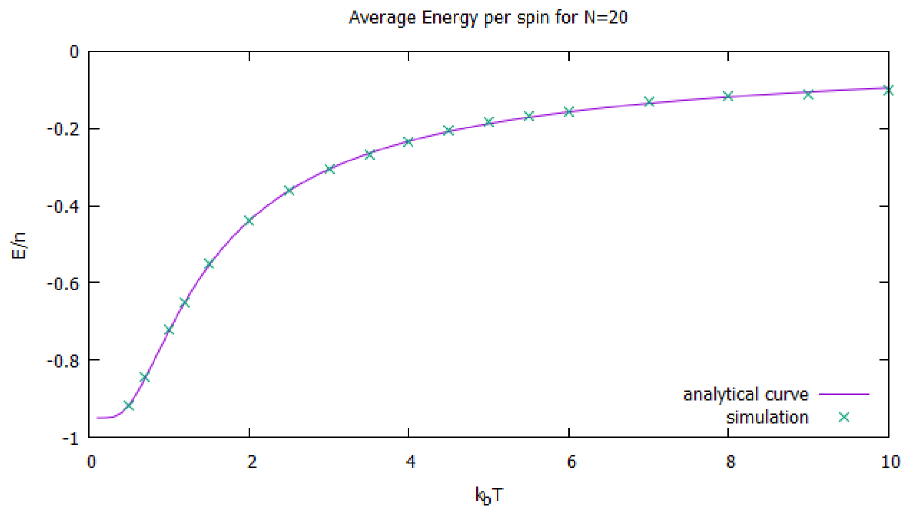


Figure 5:

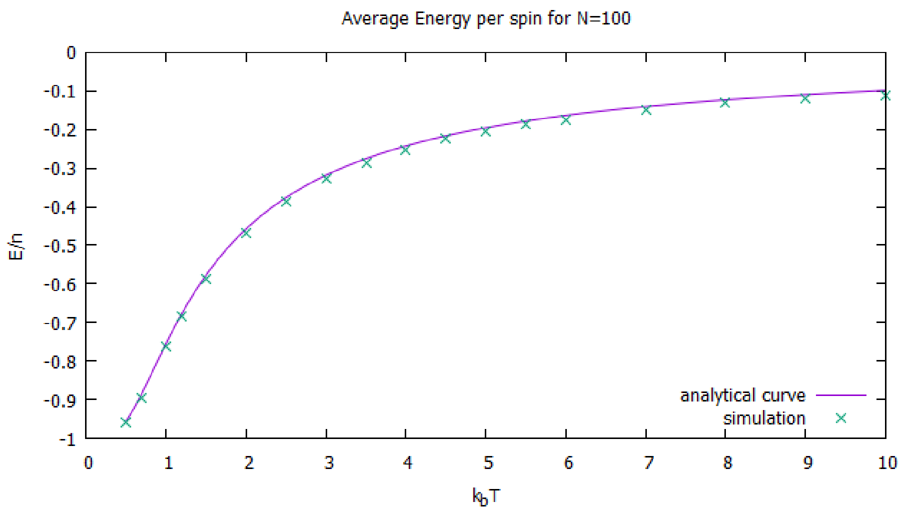


Figure 6:

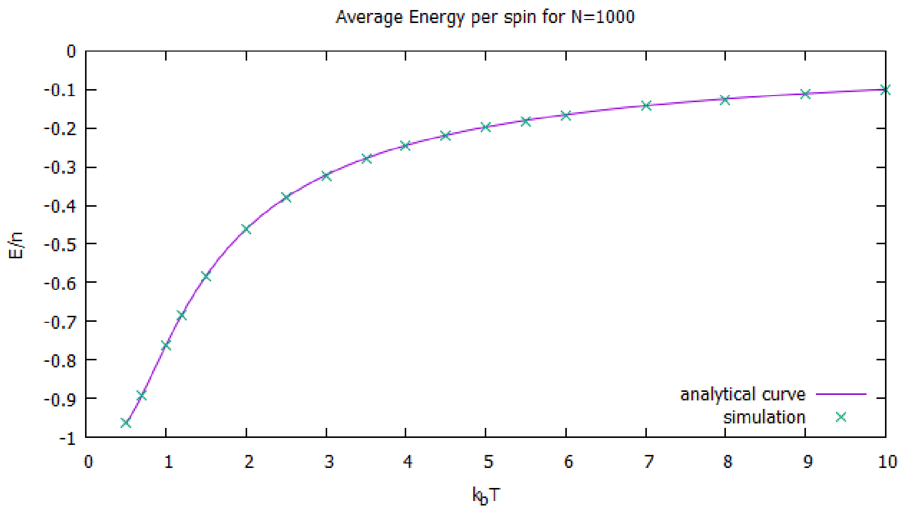


Figure 7:

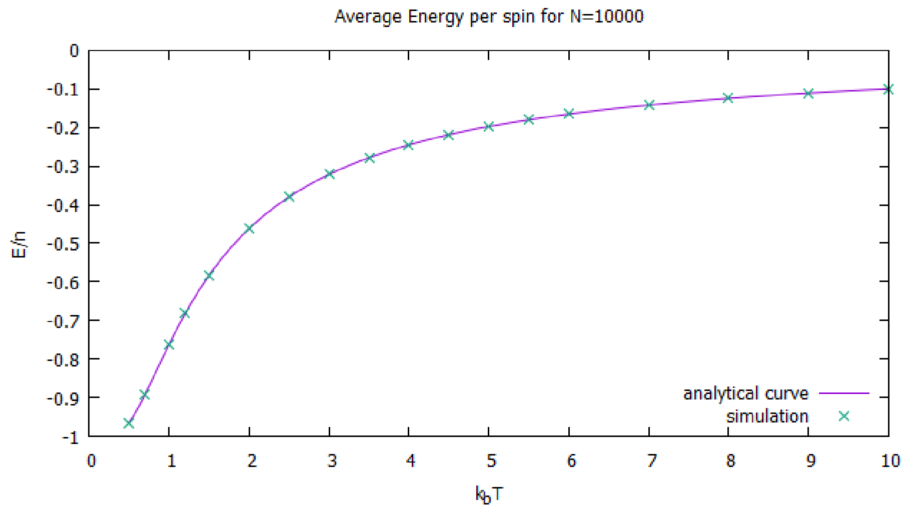


Figure 8:

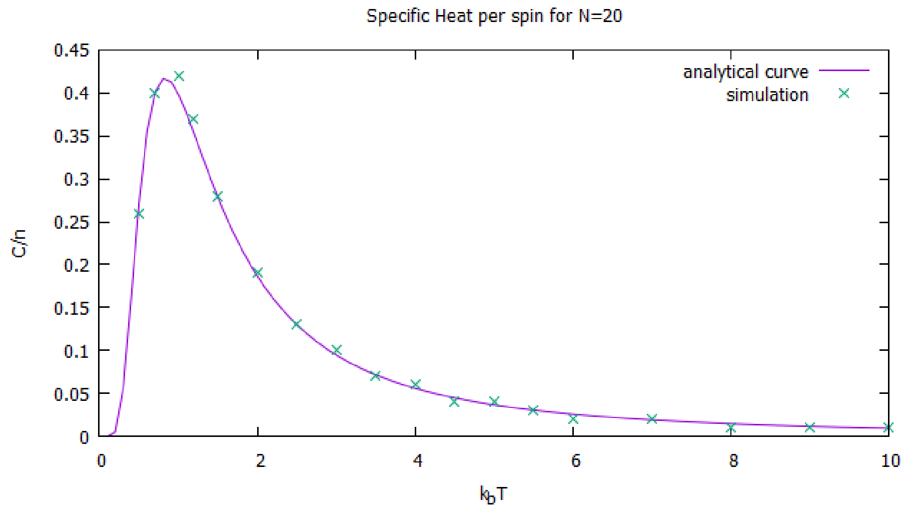


Figure 9:

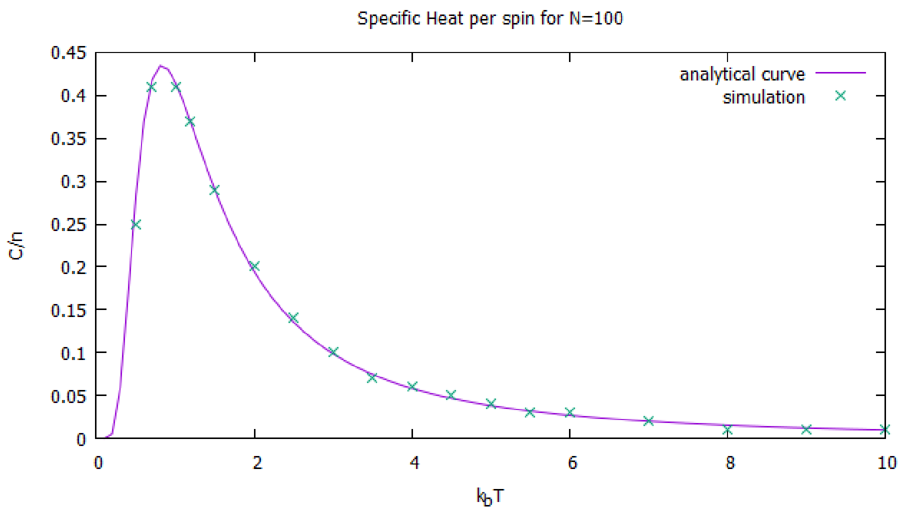


Figure 10:

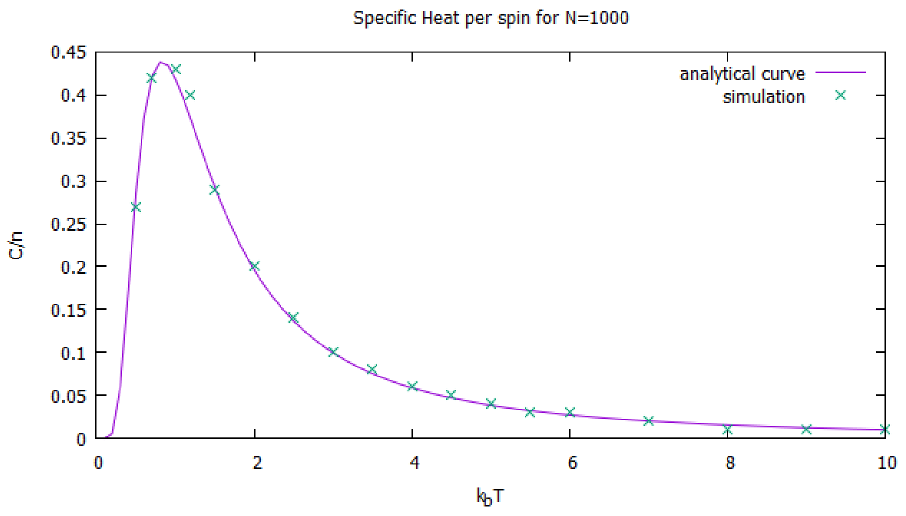
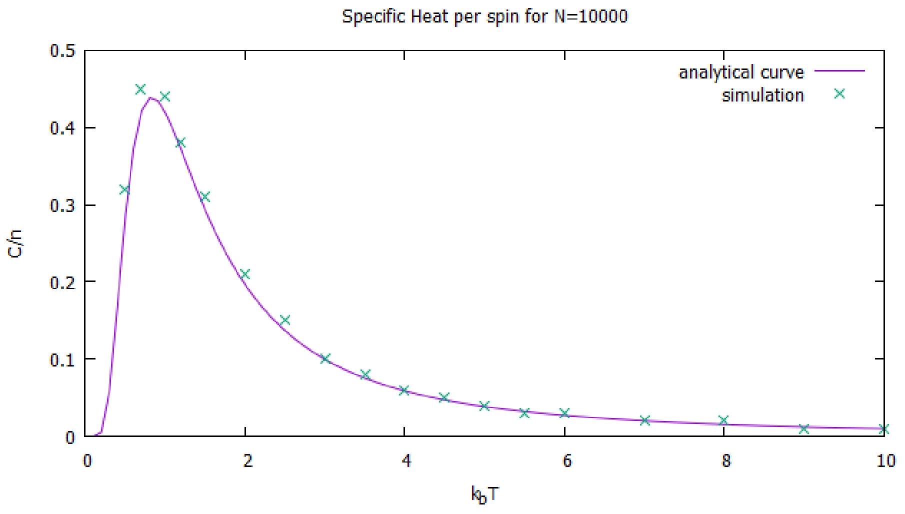
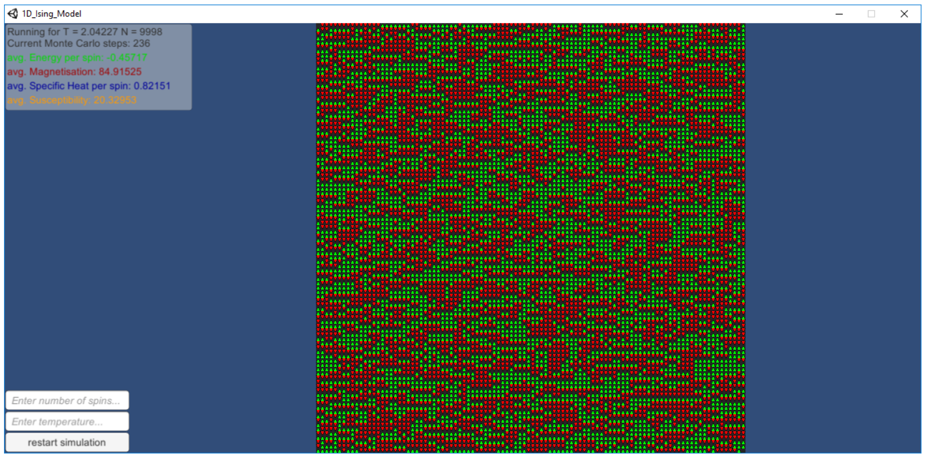


Figure 11:

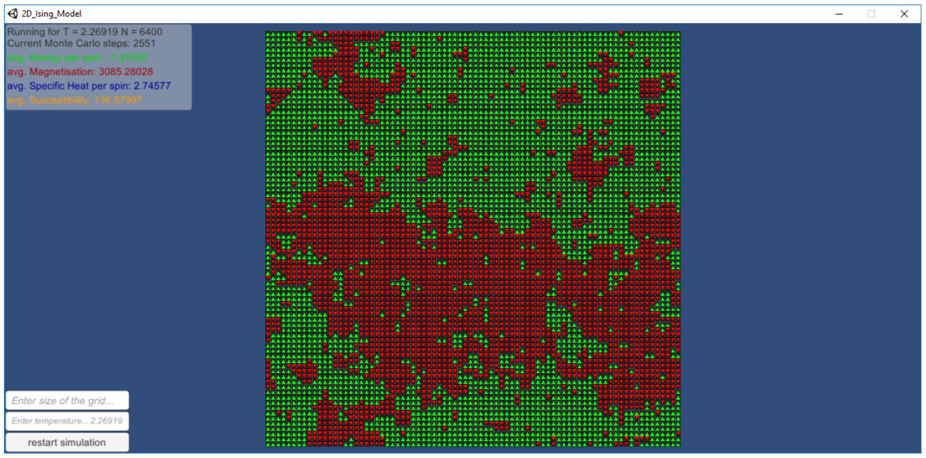


An agreement can be seen between the data points and analytic results for the most part. Where there are points above the line for the specific heat, this can be melded by having the simulator run for more steps.
And finally, a couple of screenshots from the 1D simulator used here and a 2D simulator that was just one step away, after the 1D was finished



```
Running for T = 2.04227 N = 9998
Current Monte Carlo steps: 236
avg. Energy per spin: -0.45717
avg. Magnetisation: 84.91525
avg. Specific Heat per spin: 0.82151
avg. Susceptibility: 20.32953
```

Figure 12: 1D Ising Model simulator



```
Running for T = 2.26919 N = 6400
Current Monte Carlo steps: 2551
avg. Energy per spin: -1.41092
avg. Magnetisation: 3085.28028
avg. Specific Heat per spin: 2.74577
avg. Susceptibility: 136.87997
```

Figure 13: 2D Ising Model simulator