Noah Kasmanoff
nsk367

Homework 5

Computational Physics
December 15, 2019

10.9

link to code

To study the evolution of a lattice in the Ising model, we must first initialize the lattice with some random arrangement of flips at (1 or -1). This is achieve by creating an empty grid of the desired shape which in this case is 20x20, and then for each of these 400 points, predicting with probability $\frac{1}{2}$ which flip designation this point has. One quick note is that in the next section we wish to begin with a lattice split approximately 50/50 of the two magnetization types, and by the law of large numbers this empirical magnetization (sum of 1s and -1s) will tend towards this initialization of 50/50, so this will already satisfy one of the next steps.

Once doing so, we recover one such example for an initialized lattice.

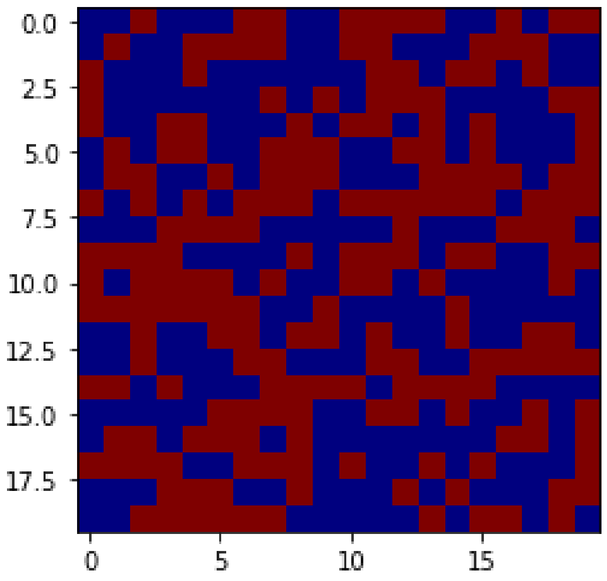

Figure 1: An randomly initialized lattice. Approximately split 50% +1 and 50% -1.

According to our calculations for energy given as $E = -J\Sigma_{<ij>}s_i s_j$ where the notation ij indicates a sum over pairs i, j that are adjacent. In the current system, $E = -18$.

Next, we code and apply the MC MC (Metropolis-Hastings) algorithm to this system and follow how the total magnetization of this system evolved. The total magnetization is given by $M = \Sigma_i s_i$, so it could simply be expressed as the sum of the individual magnetization. The energy formula provides no preference to whether or not a grid space should be +1 or -1, but does prefer for these neighboring pairs to be similar. As a result, we should expect that if this system evolves for long enough, the total magnetization will be highly positive or highly negative. Using one million time steps, we recover this result.

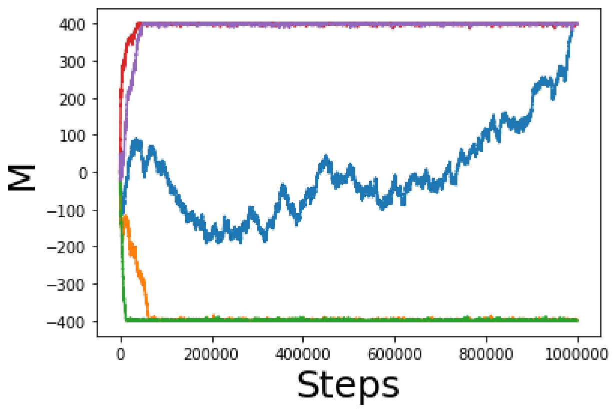

Figure 2: Total magnetization versus step count. In different simulations we find different behaviors.

Next, we can plot the evolution of one such lattice. By examining this system after approximately every new order of magnitude steps, we find that certain regions become pockets of a certain magnetization type, but are eventually overwhelmed by whatever the dominant type is.

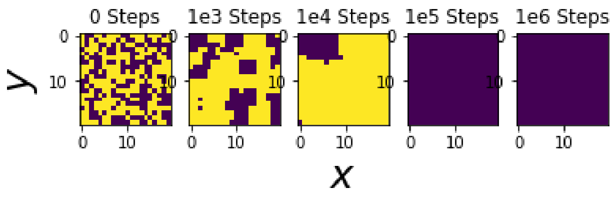

Figure 3: Time evolution of a single lattice.

In these systems, all constants were set to 1. This was in order to keep everything as simple as possible, but there are some interesting properties when T in particular is varied. This T is a part of accepted proposition stage, and as T increases, it is more likely that new permutation is accepted. In the context of simulated annealing, a large T corresponds to system that is very much still in flux, and thus is more permissive to changes which may increase the total energy. Conversely, a smaller T corresponds to the system being less accepting of these propositions, and will remain relatively selective in what new states to accept.

If we were to compare the time evolution of this system with different initial T values, we find that larger T's are preferable, especially in the early stages of the system.

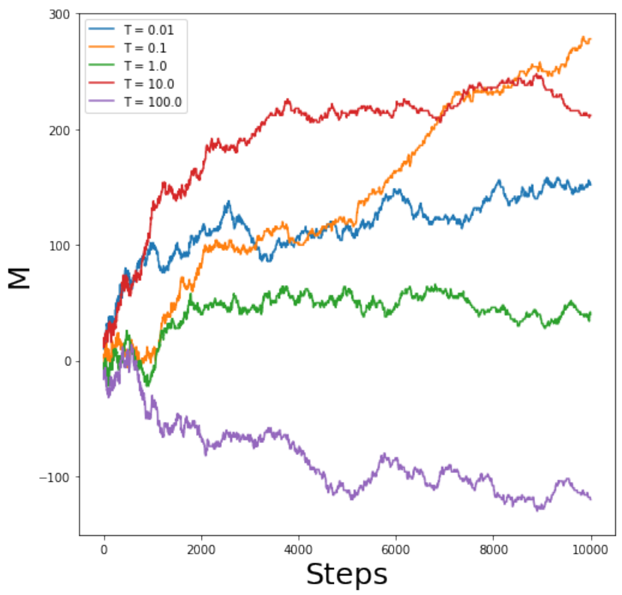

Figure 4: Time evolution of varying temperature designations.

In the early stages, these larger T values have a significant "lead" over the other T values in terms of a total magnetization.

10.11

link to code

After testing and confirming that the system to insert dimers is runs properly, we next explore what this system looks like over time. As we see, points begin to populate the lattice, and occasionally some regions are altered to accommodate a new arrangement.

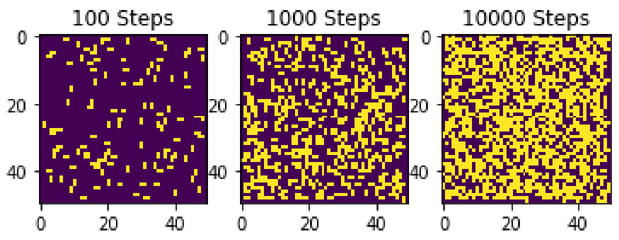

Figure 5: Time evolution of dimer placement after N steps.

In this setting, the lattice is nearly fully populated after 100000 steps. This was also for the designation of T=1, and we can examine in a manner similar to last question how varying the cooling temperature T influences the convergence rate of the system. Using a fixed value of 10,000 steps, we can observe how the system looks for an array of different cooling values.

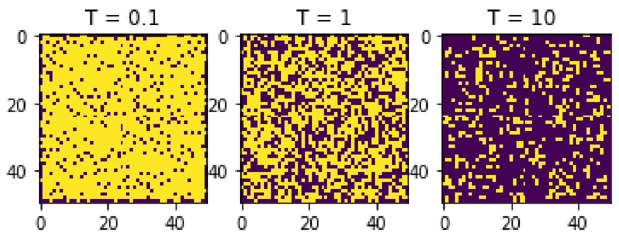

Figure 6: Dimer system after 10000 steps with different temperatures.

Surprisingly, we find that smaller temperature values yield a faster convergence. One possible reason for such a result is that dimers are less likely to be accepted in positions where a dimer already exists, forcing the dimer to find a new place to reside. For larger T's it is more probable that a proposition dimer simply replaces another, and thus may result in more replacements as opposed to new spaces, which may explain why this happens for 10000 steps into the system.

Power spectra of random numbers and random walks

link to code

Following the steps of this accordingly, we first create a linear congruential generator, a pseudo-random number generator which obeys the formula

$$x' = (a * x + c)\%m$$

Where x' is the random number produced, a and c are constants which have to obey certain rules, depending on the modulus m which is in place. In this particular case I use the same values as were provided in Newman's Computational Physics. Accordingly, my random number generator obeys the trend

$$random_i = (1664525 * random_{i-1} + 1013904223)\%4294967296$$

In these conditions, the result is an approximately uniform distribution between 0 to m. To meet the conditions requested of this being a uniform between 0 and 1, I simply divide by the modulus m.

Using this LCG, we can now sample from it in order to generate a normal distribution. In theory this is known as inverse transform sampling, where we can take the inverse CDF of a normal distribution, and compute the value of the known (LCG created) random variable many times. The resulting distribution will look like the inverse of the inverse, which in this case is our desired goal of a Gaussian.

In practice, this is done with the Box-Muller distribution. The primary challenge with this task that Box-Muller is capable of achieve is actually solving for the inverse of a Gaussian distribution, by drawing two random variables from the known distribution. The resulting Gaussian is plotted below, for ten thousand data point. For validation, is it overlayed with what the conventional standard normal distribution is generated as using numpy.

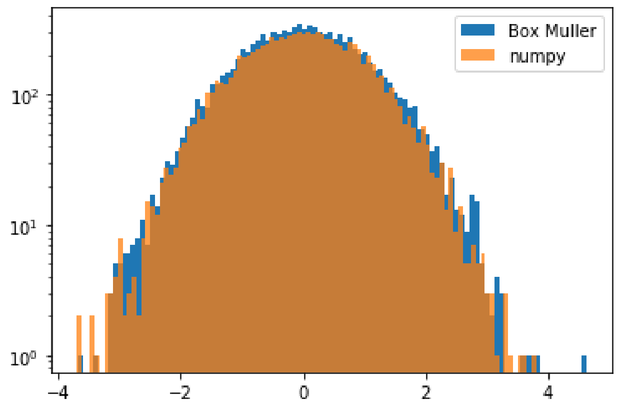

Figure 7: Numpy normal versus inverse transform sampling produced normal.

Based on this image alone, it appears that this method has successfully re-created a Gaussian distribution.

Next, we apply the discrete Fourier transform function created for homework 3 to this data of Gaussian random variables, and visualize the power spectrum. The result is shown below.

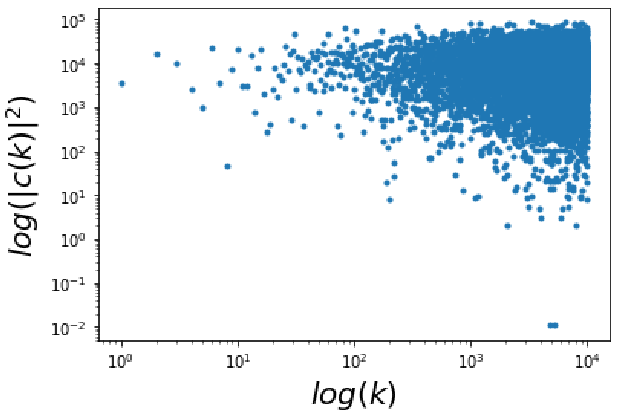

Figure 8: Power spectrum of random variables drawn from a normal distribution.

This result matches what we would expect for a the drawing of many Gaussian random variables.

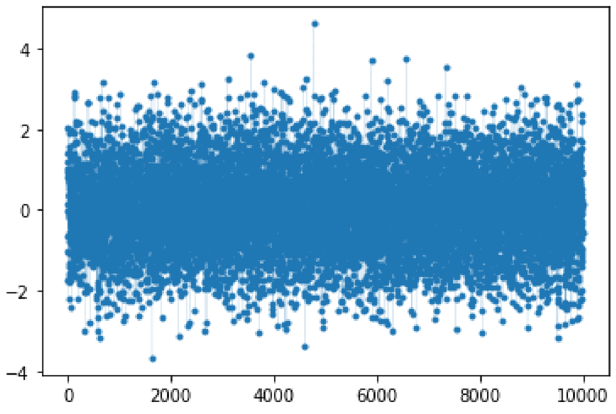

Figure 9: White noise visualization of the drawn Gaussian random variables.

As we see in the image of white noise, the majority of points are close to the empirical mean of 0, and this manifests itself in Fourier space as high frequency (and high k) values. We observe exactly this.

Finally, we compute the Gaussian random walk. To do so, we create a running sum of the series of Gaussian random variables already drawn, and follow the trajectory of this particle. The result of two distance Gaussian random walks are plotted below. In either case, we see the walker take a different path.

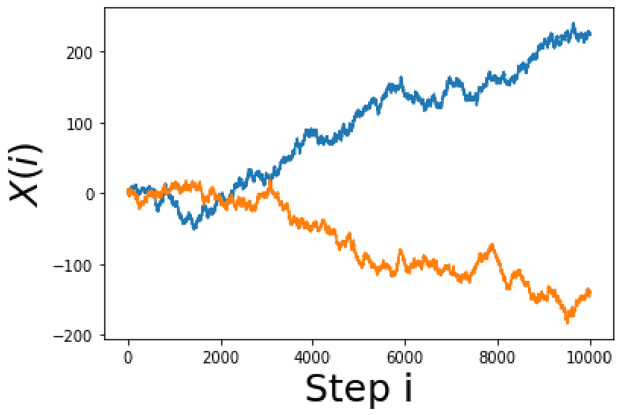

Figure 10: Two unique random walks drawing from a Gaussian distribution.

Taking the power spectrum of one such Gaussian random walk, we recover an approximately k^{-2} relationship.

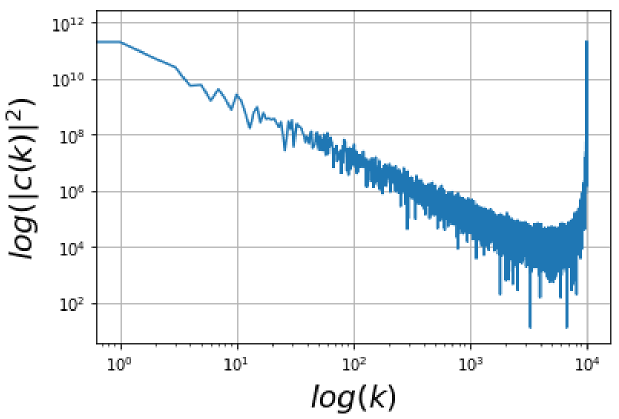

Figure 11: Power spectrum of a Gaussian random walk.