Homework 6 Computational Physics

Giacomo Recine[*]

22 December 2025

Contents

[*]gr2640@nyu.edu

1 Introduction

In this homework, we study several applications of Markov Chain Monte Carlo (MCMC) methods, focusing in particular on the Metropolis algorithm. In Sec. 2, we apply this method to the Ising model, where we investigate the emergence of spontaneous magnetization and its dependence on temperature. In Sec. 3, we address another well-known problem in condensed matter physics: the dimer covering problem. Our goal is to determine the ground state of the system in order to understand the maximum number of dimers that can be placed on the lattice. To achieve this, we employ the simulated annealing method. In particular, we study the effect of varying the cooling schedule, whose time constant plays a crucial role in the quality of the final solution. Finally, in Sec. 4, we construct a linear congruential random number generator to produce uniformly distributed random numbers in the interval $[0, 1)$. We then use this generator to produce Gaussian random variables and apply them to the study of a random walk. In this context, we analyze the behavior of the corresponding power spectra, using the techniques introduced in Homework 3.

2 Exercise 10.9 from Newman: The Ising Model

The Ising model is a theoretical model of a magnet. The magnetization of a magnetic material arises from the collective behavior of many small magnetic dipoles distributed throughout the material. If these dipoles point in random directions, the overall magnetization of the system is close to zero. If, instead, they align so that most of them point in the same direction, the system acquires a macroscopic magnetic moment and becomes magnetized. The Ising model provides a simplified description of this phenomenon by representing the individual magnetic moments as dipoles, or *spins*, arranged on a regular grid or lattice. In this exercise we consider a two-dimensional square lattice, although the model can be defined for other lattice geometries and in any number of dimensions. In its simplest form, each spin is restricted to point in only two possible directions: up or down. Mathematically, the spin at lattice site i is represented by a variable $s_i = \pm 1$, where $+1$ corresponds to an up-pointing spin and -1 to a down-pointing spin. While real magnetic dipoles can point in arbitrary spatial directions, this two-state approximation captures much of the essential physics of magnetism while remaining analytically and computationally tractable. An important feature of many magnetic materials is that neighboring dipoles interact in such a way that it is energetically favorable for them to align in the same direction. In real magnets, the magnetic interaction energy between two dipoles is proportional to their dot product. In the Ising model, this interaction simplifies to the product $s_i s_j$ for spins located at lattice sites i and j, since the spins are scalar quantities rather than vectors. The interaction energy between two neighboring spins is therefore given by $-Js_i s_j$, where $J > 0$ is the coupling constant. The negative sign ensures that the interaction is ferromagnetic, meaning that parallel alignment of spins lowers the energy. Ferromagnetic interactions allow the system to develop a nonzero magnetization. (If $J < 0$, the interaction would be antiferromagnetic, favoring antiparallel alignment, but this case will not be considered here.) It is usually assumed that each spin interacts only with its nearest neighbors on the lattice. Under this assumption, the total energy of the system is

$$E = -J \sum_{(ij)} s_i s_j, \tag{1}$$

where the notation (ij) indicates a sum over all pairs of nearest-neighbor lattice sites. On a two-dimensional square lattice, each interior spin has four nearest neighbors, while spins at the boundaries have fewer.

Now, we wrote a program to perform a Markov chain Monte Carlo simulation of the Ising model on a 20×20 square lattice. The spin configuration is represented by a two-dimensional array whose entries take the values ± 1. The simulation proceeds as follows:

(a) we wrote a function that computes the total energy of the system using the expression given above. For a given spin configuration, we sum the contributions $s_i s_j$ from all pairs of nearest-neighbor spins and multiply the result by $-J$. Each unique pair of adjacent spins is counted only once;

(b) using this energy function, we implemented a Metropolis Monte Carlo simulation with coupling constant $J = 1$ and temperature $T = 1$, working in units where the Boltzmann constant $k_B = 1$. We initialized the system with a random configuration of spins, such that approximately half of the spins point up and half point down, resulting in an initial magnetization close to zero. We then repeatedly:

- select a lattice site at random;
- flip the spin at that site;
- compute the change in energy ΔE due to the flip;
- accept or reject the move according to the Metropolis acceptance criterion.

If a proposed move is rejected, we restore the spin to its original value. We repeat this process for many Monte Carlo steps;

(c) we computed and plot the total magnetization

$$M = \sum_i s_i, \tag{2}$$

as a function of time for a simulation consisting of one million Monte Carlo steps. We observe the emergence of a spontaneous magnetization, characterized by a nonzero value of M. While developing and testing our program, we perform shorter runs (for example, ten thousand steps) before carrying out the full simulation;

(d) we run the simulation several times and record whether the magnetization that develops is positive or negative in each run. We describe our observations and provide a brief explanation of the results;

(e) in the end we created a multi-panel plot logarithmically-spaced time intervals to show the evolution of the system.

2.1 Markov chain Monte Carlo simulation

Before presenting the solution to the exercise described in the previous section, we briefly explain the basic principles of the Markov chain Monte Carlo (MCMC) method, which is the key tool required to answer the questions posed in the exercise. The idea of a Markov chain Monte Carlo simulation is to generate a sequence of states, one after another, forming a Markov chain. Consider a single step in this process and suppose that the previous state in the chain is state i. Rather than choosing the next state completely at random from all possible states, we construct a new state by making a small change to the current one. In this way, the new state is closely related to the previous state. For example, if the system under study is a gas, one might change the quantum state of a single molecule while leaving all other molecules unchanged. The choice of the new state is made probabilistically according to a set of transition probabilities T_{ij}, which give the probability of moving from state i to state j. By far the most commonly used choice of transition probabilities leads to the so-called *Metropolis algorithm.* It is important to note that, in a Markov chain, the same state may be visited more than once, and it is even possible to visit the same state on two consecutive steps. In other words, the probability T_{ii} of remaining in the same state is allowed to be nonzero. In this case, the proposed move results in no change to the system. The Metropolis algorithm proceeds as follows. Suppose again that the current state of the system is state i. A trial state j is generated by making a random change to state i. The particular change is chosen uniformly at random from a predefined set of possible changes, often referred to as the *move set.* Returning to the example of a gas, one might choose a molecule at random and attempt to move it to a neighboring energy level, either one level higher or one level lower than its current state. A uniform choice of trial moves of this kind does not, in general, satisfy the detailed balance condition. To correct for this, the trial move is accepted or rejected according to the Metropolis acceptance probability P_a, defined as

$$P_\mathrm{a} = \begin{cases} 1, & \text{if } E_j \leq E_i, \\ \exp[-\beta(E_j - E_i)]\,, & \text{if } E_j > E_i, \end{cases}$$

where E_i and E_j are the energies of states i and j, respectively, and $\beta = 1/(K_B T)$, where T is the temperature (even if we will use with Boltzmann constant $k_B = 1$). If the proposed move is rejected, the system remains in its current state i for one additional step in the Markov chain. This is equivalent to a transition from state i to itself, which is permitted within the framework of the algorithm. If the move is accepted, the system is updated to the new state j, and the Markov chain proceeds from there. Under these assumptions, we find that the following relation always holds

$$\frac{T_{ij}}{T_{ji}} = e^{-\beta(E_j - E_i)}, \tag{3}$$

which is the constraint that the transition probabilities must satisfy in a Markov chain in order to ensure detailed balance. A complete Markov chain Monte Carlo simulation proceeds through the following steps. First, a random initial state of the system is chosen. At each Monte Carlo step, a trial move is selected uniformly at random from a predefined set of allowed moves, such as modifying the state of a single particle or spin. The acceptance probability P_a for the proposed move is then computed using the Metropolis criterion. The move is accepted with probability P_a, in which case the system is updated to the new state. Otherwise, the move is rejected and the system remains in its current state for that step. After each step, the value of the physical quantity of interest is measured and accumulated and this process is repeated for a large number of steps. After many Monte Carlo steps, the average value of the observable is obtained by dividing the accumulated sum by the total number of steps, yielding an estimate of the ensemble average. In practice, the Metropolis acceptance rule is implemented by computing the energy difference $\Delta E = E_j - E_i$ between the trial state and the current state, and then generating a random number z uniformly distributed in the interval $[0, 1]$. The move is accepted if

$$z < e^{-\beta \Delta E}.$$

If $\Delta E \leq 0$, the exponential factor is greater than or equal to one and the move is always accepted. If $\Delta E > 0$, the move is accepted with probability $e^{-\beta \Delta E}$, in full agreement with the Metropolis acceptance criterion.

2.2 The solution of the exercise

To solve the exercise, we follow the steps suggested in the text. First, we compute the total energy of the system using Eq. (2). Then, using the Metropolis acceptance criterion, we implement the Metropolis Monte Carlo simulation in order to determine the total magnetization of the system as a function of one million Monte Carlo steps. Choosing a temperature of $T = 1$ we can compute the magnetization from Eq. (2), and we get the following plot

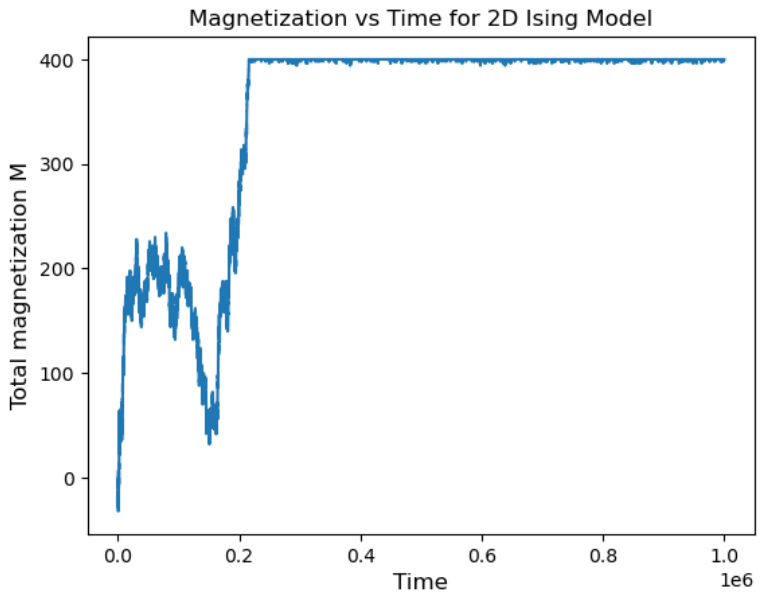


Figure 1: Total magnetization as function of the Monte Carlo time in the case of temperature $T = 1$.

We observe that after a sufficiently long time the system develops a spontaneous magnetization, in which essentially all spins become aligned in the same direction. In the case shown here, the magnetization is positive, with a value $M = +400$, corresponding to the configuration in which all spins are up, i.e. $s_i = +1$ for every lattice site. However, this is not the only possible outcome. By repeating the simulation multiple times and starting from different random initial configurations, we find that approximately half of the runs lead to a positive magnetization, while the other half result in a negative magnetization with value $M = -400$, corresponding to all spins aligned in the down direction. This behavior reflects the dependence of the final state on the initial conditions. Since the Hamiltonian of the Ising model is symmetric under a global spin inversion, the two fully magnetized states are energetically degenerate. Small fluctuations in the initial configuration break this symmetry and determine which of the two minima the system ultimately reaches. As a result, in the absence of an external magnetic field, neither final state is favored, and both occur with approximately equal probability over many independent simulations.

As a further demonstration of the emergence of spontaneous magnetization, we present the panel below, which shows the spin configuration of the system at different times during the Monte Carlo simulation

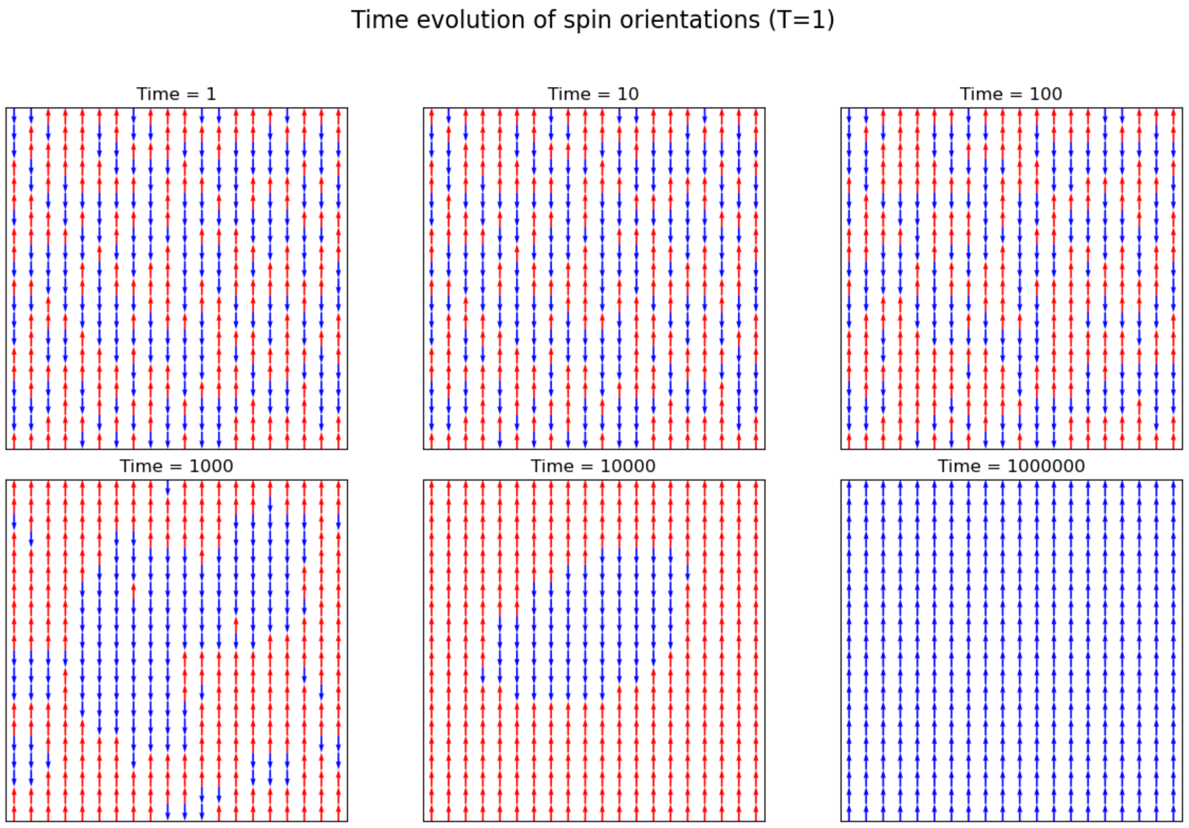


Figure 2: Panel showing the time evolution of the spin configuration of the system during the Monte Carlo simulation. Blue spins represent up spins, while red spins represent down-directed spins.

These snapshots clearly show that the system evolves toward a fully ordered state, starting from an initial lattice in which spins are randomly oriented up or down with equal probability. The ordered state is characterized by the alignment of all spins in a single direction. In the present case, the final configuration corresponds to all spins pointing upward.

However, there is another subtle point that must be taken into account. All the previous plots were obtained at temperature $T = 1$, but it is also interesting to investigate what happens when the temperature is increased. For instance, if we consider a temperature $T = 2$ and plot the total magnetization as a function of time, we obtain the following result

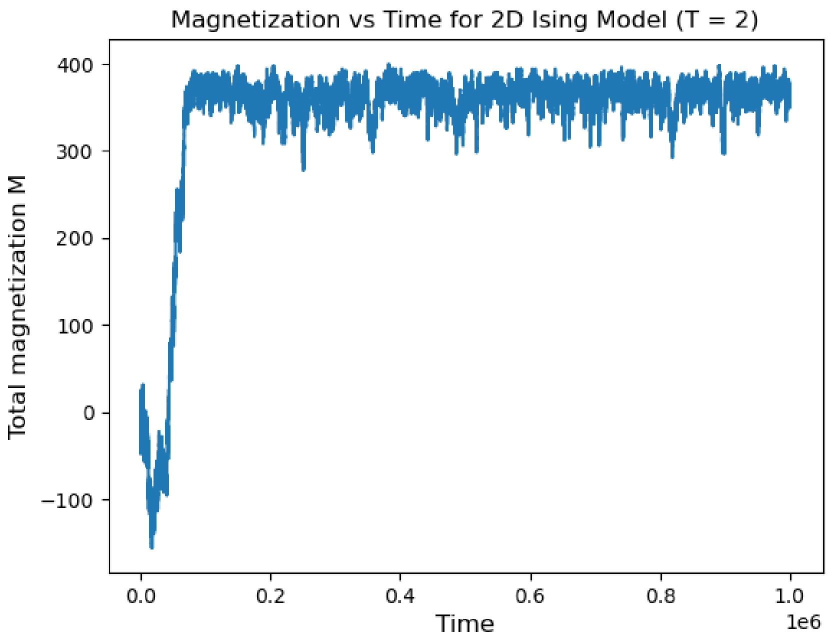


Figure 3: Total magnetization as function the Monte Carlo time in the case of temperature $T = 2$.

We immediately observe that the system now exhibits significantly larger fluctuations, as if additional noise were present. This behavior can be understood by noting that an increase in temperature corresponds to a decrease in β. As a consequence, energy-increasing moves are accepted more frequently, since the acceptance condition $z < e^{-\beta \Delta E}$ is more easily satisfied. To further illustrate this effect, we consider an even higher temperature, $T = 3$. In this case, the total magnetization exhibits even stronger fluctuations, as shown in the following plot

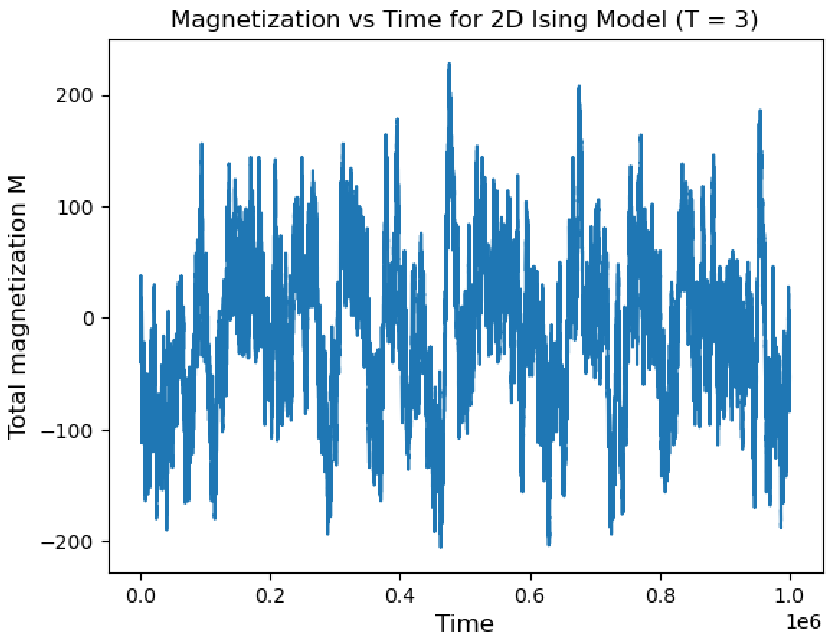


Figure 4: Total magnetization as function the Monte Carlo time in the case of temperature $T = 3$.

Here we see that a stable magnetization is not achieved, for the same reasons discussed in the previous case: at higher temperatures, thermal fluctuations are stronger and energy-increasing moves are accepted more frequently, preventing the system from reaching a fully ordered state.

3 Exercise 10.11 from Newman: The Dimers Problem

Dimer Covering Problem

A well-studied problem in condensed matter physics is the dimer covering problem. In this problem, dimers, polymers composed of two atoms, are placed on the surface of a solid, occupying the spaces between surface atoms and forming a grid. No two dimers are allowed to overlap. The question is: how many dimers can fit on an $L \times L$ square lattice? In the simplest case, the answer is approximately $\frac{1}{2}L \times L$, but for more complex lattices or differently shaped elements, the solution may be far from obvious or even unknown. We implement a simulated annealing algorithm to solve this problem on a 50×50 lattice. We define the energy of the system as the negative number of dimers, so that minimizing the energy corresponds to maximizing the number of dimers. The Markov chain moves are defined as follows:

1. we choose two adjacent lattice sites at random;
2. if both sites are empty, place always a dimer on them;
3. if both sites are occupied by a single dimer, remove the dimer with the appropriate acceptance probability (to be determined according to the Metropolis criterion);
4. otherwise, do nothing.

In the end we explore exponential cooling schedules with different time constants. A reasonable starting value is $\tau = 10000$ steps. Indeed, faster cooling schedules generally lead to poorer solutions, with a smaller fraction of the lattice covered by dimers and larger empty regions. Slower cooling schedules typically yield better lattice coverage.

So, first, before to introduce the exercise to solve, we briefly review the annealing simulation method.

3.1 Simulated annealing

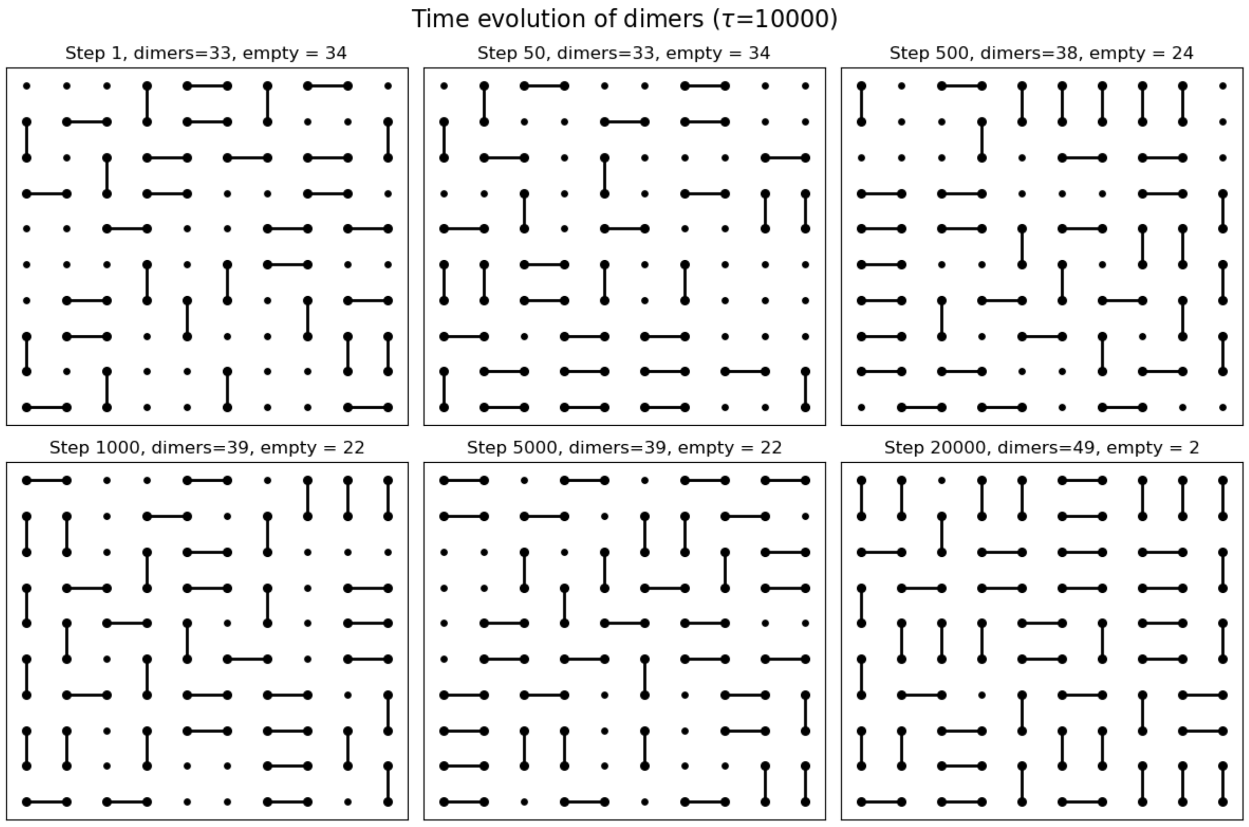
Simulated annealing is a computational technique that mimics the slow cooling of a material to find its ground state. It uses a Monte Carlo simulation with a temperature parameter that is gradually lowered from a high initial value towards zero. If the cooling is sufficiently slow, the method is guaranteed to find the ground state. In practice, however, cooling slowly enough to guarantee the exact ground state is often impractical, so the system is typically cooled as slowly as reasonably possible, yielding a state that is close to the ground state. Although inspired by physics, simulated annealing is widely used in engineering, computer science, biology, economics, statistics, and other fields. Many optimization problems can be formulated as finding the global minimum or maximum of a function, and simulated annealing provides an effective method for solving such problems. Implementation is straightforward. One performs a Monte Carlo simulation of the system of interest while gradually lowering the temperature according to a cooling schedule. The initial temperature should be high enough that $k_B T$ exceeds typical energy changes of individual Monte Carlo moves, ensuring that most moves are accepted and the system is well randomized. The cooling schedule is often exponential

$$T(t) = T_0 \, e^{-t/\tau}, \tag{4}$$

where T_0 is the initial temperature and τ is a time constant controlling the cooling rate. Slower cooling (larger τ) generally produces better results, but requires longer simulation time. In practice, the choice of τ balances accuracy with computational efficiency. The final state of the simulation, once the system stops evolving, is taken as an estimate of the ground state.

3.2 Solution of the exercise

By applying the simulated annealing method (starting with $T_0 = 1$), we first present a panel for a lattice of size $L = 10$, as this case is much easier to visualize graphically. In this exercise, we use an exponential cooling schedule, given by Eq. (4). For a time constant $\tau = 10000$, we obtain the following evolution



Time evolution of dimers (τ=10000)
Step 1, dimers=33, empty = 34
Step 50, dimers=33, empty = 34
Step 500, dimers=38, empty = 24
Step 1000, dimers=39, empty = 22
Step 5000, dimers=39, empty = 22
Step 20000, dimers=49, empty = 2

Figure 5: Panel showing the time evolution of the dimers formation.

obtained after performing 20 000 Monte Carlo steps. From this panel, we can observe how the dimers gradually form during the simulation as a result of the annealing cooling method. It is also interesting to examine how the energy evolves as a function of the Monte Carlo steps. Indeed, we find

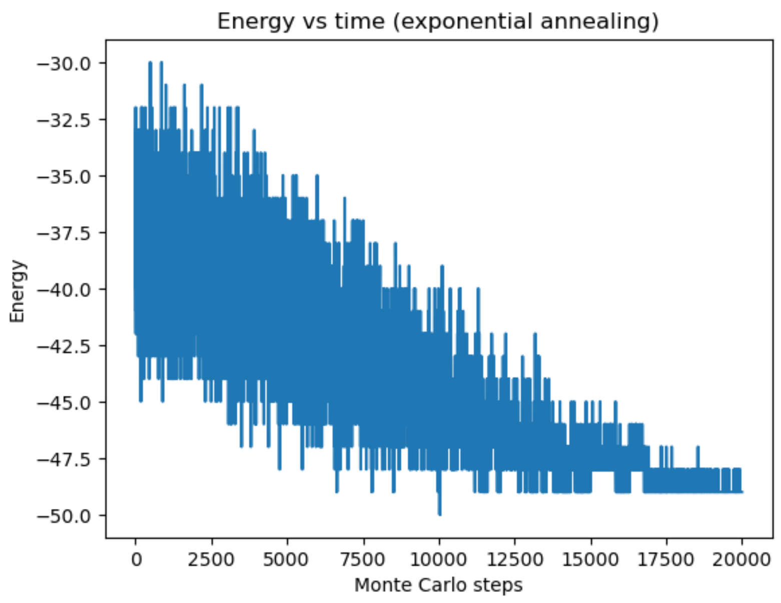


Figure 6: Energy function of the system in function of Monte Carlo steps for $\tau = 10000$.

From this plot, we can immediately observe that as the number of Monte Carlo steps increases, and consequently the temperature decreases, the energy of the system decreases. This behavior is precisely the mechanism underlying the simulated annealing method, whose goal is to drive the system toward its ground state. At certain stages of the simulation, the system reaches the maximum possible number of dimers, corresponding to the minimum (ground-state) energy. However, the algorithm does not stop at this point, since in general the exact value of the ground-state energy is not known a priori. As a result, the simulation continues and the final configuration typically corresponds to a state whose energy is close to, but not necessarily equal to, the true ground-state energy. In our case, the final configuration yields an estimated ground-state energy

$$E_{\text{g.s.}} = -49,$$

which is very close to the true ground-state energy,

$$E_{\text{g.s.}}^{\text{true}} = -50.$$

This result indicates that the simulated annealing algorithm successfully finds a configuration close to the global energy minimum, although it does not reach the exact ground state in this particular run. Note that, to evaluate the performance of the algorithm, we consider the energy. This choice is entirely natural, since for this particular system the energy is defined as the negative of the number of dimers. Therefore, interpreting the results in terms of energy is equivalent to analyzing the number of dimers formed during the simulation.

So far, we have used a lattice of size $L = 10$ in order to visualize the dynamics of the system more easily. From this point onward, we consider a larger lattice with $L = 50$. For this larger system, we no longer display the full time evolution of the dimer configuration, but instead focus on the behavior of the energy as a function of the Monte Carlo steps. In the case $\tau = 10\,000$ and $L = 50$, the final state of the system exhibits the following dimer configuration

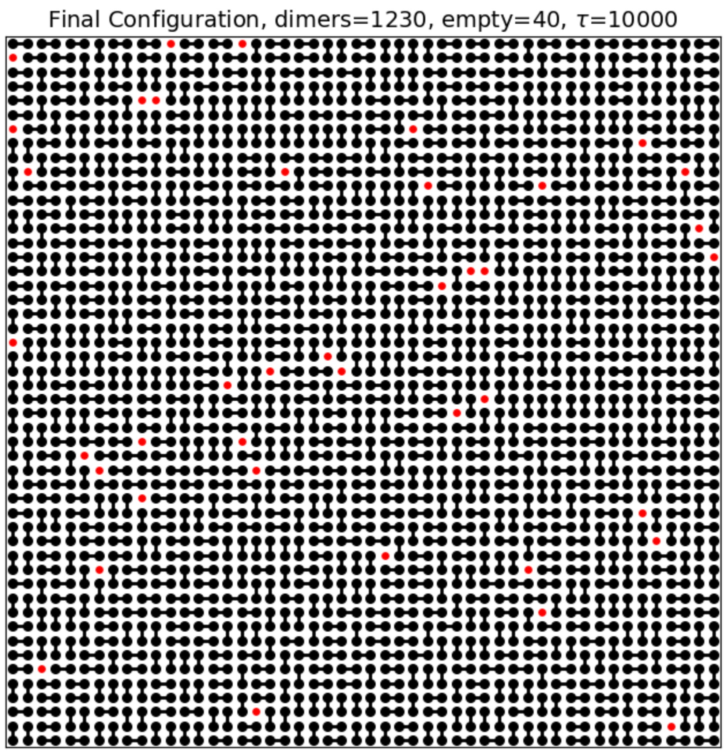


Figure 7: Final configuration of the system in function of Monte Carlo steps for $\tau = 10000$, and for a lattice of $L = 50$.

The energy as a function of the Monte Carlo steps is shown below

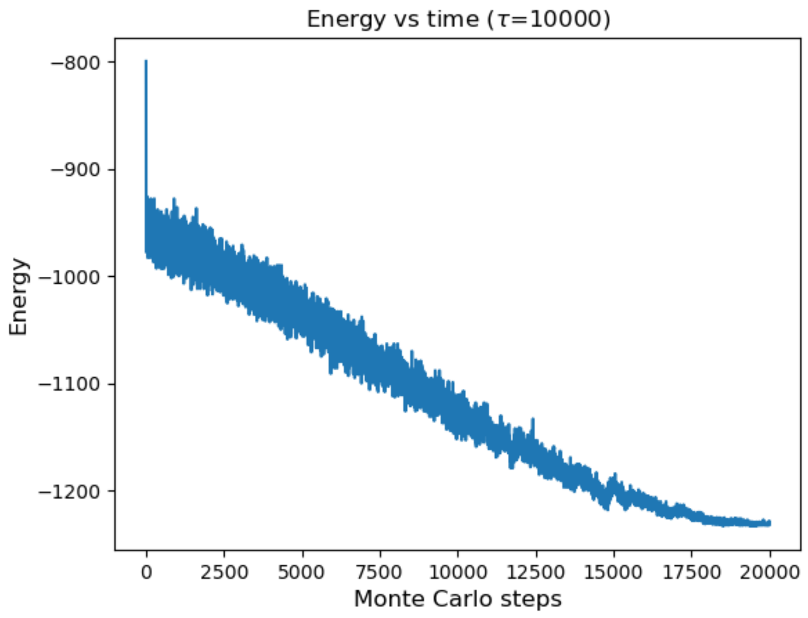


Figure 8: Energy function of the system in function of Monte Carlo steps for $\tau = 10000$, and for a lattice of $L = 50$.

We observe that the estimated ground state is reached after approximately Step $\simeq 17\,500$, with an estimated ground-state energy

$$E_{\text{g.s.}} = -1233,$$

which is reasonably close to the true ground-state energy

$$E_{\text{g.s.}}^{\text{true}} = -1250.$$

An interesting point is how these results change when varying the time constant of the cooling schedule. For a faster cooling rate, corresponding to $\tau = 5\,000$, we obtain the following energy evolution

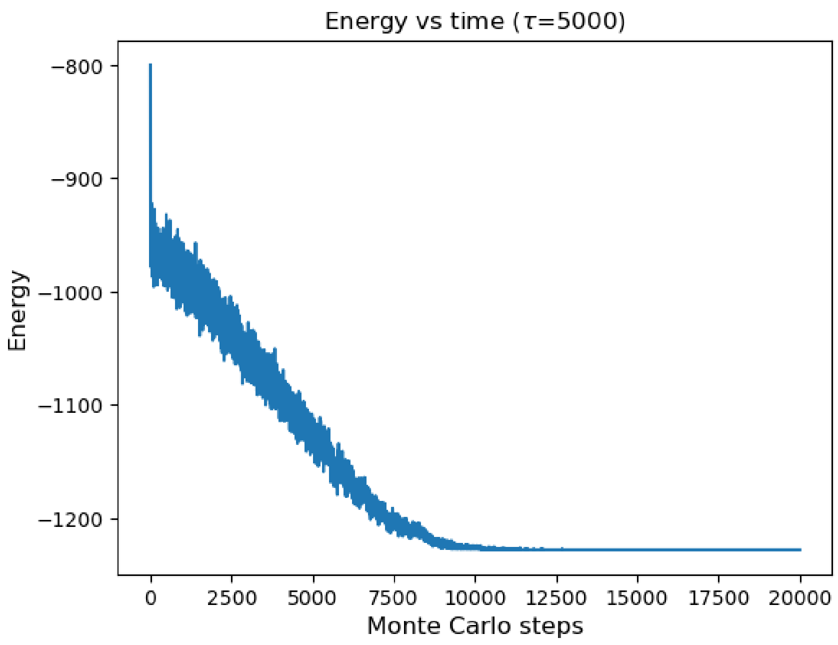


Figure 9: Energy function of the system in function of Monte Carlo steps for $\tau = 5000$, and for a lattice of $L = 50$.

In this case, we observe that the system reaches an approximate ground state earlier than in the slower cooling schedule with $\tau = 10\,000$. However, the final result is less accurate, with an estimated ground-state energy of $E_{\text{g.s.}} = -1228$. If we further decrease the time constant, for instance to $\tau = 2\,000$, we obtain

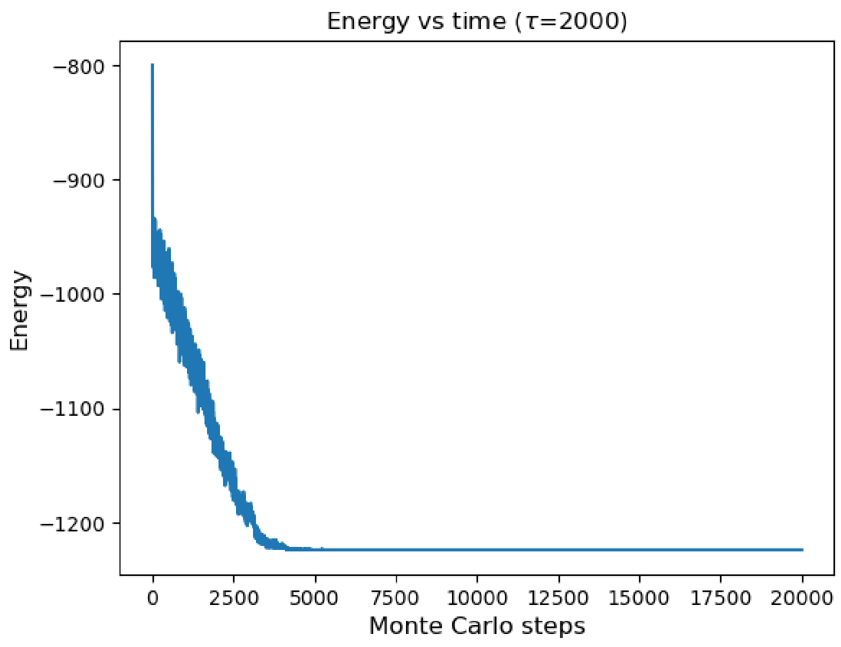


Figure 10: Energy function of the system in function of Monte Carlo steps for $\tau = 2000$, and for a lattice of $L = 50$.

As expected, we observe once again that the system reaches an approximate ground state earlier; however, the accuracy of the final estimated ground-state energy decreases further, yielding $E_{\text{g.s.}} = -1224$.

If, instead, we consider a slower cooling schedule, such as $\tau = 15\,000$, we expect the opposite behavior. Indeed, plotting the energy as a function of the Monte Carlo steps, we obtain

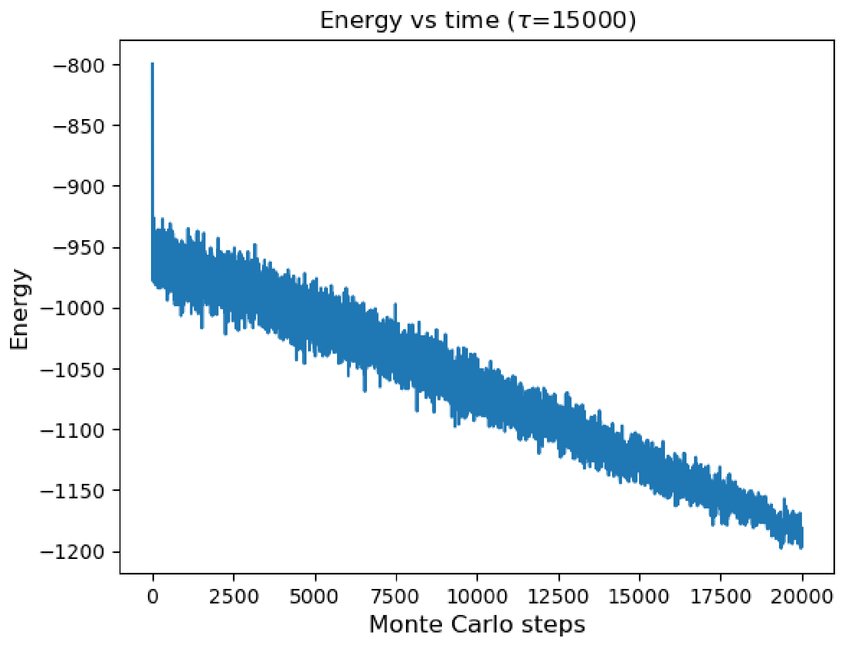


Figure 11: Energy function of the system in function of Monte Carlo steps for $\tau = 15\,000$, and for a lattice of $L = 50$.

From this plot, we see that the ground state is not reached during the Monte Carlo steps performed. As expected, the accuracy of the estimated ground-state energy is worse compared to the previous cases. In this run, we obtain $E_{\text{g.s.}} = -1198$, which is less accurate than the earlier results.

But this is not the end of the story. If we increase the number of Monte Carlo steps, the system eventually reaches a ground state. Indeed, we obtain the following behavior

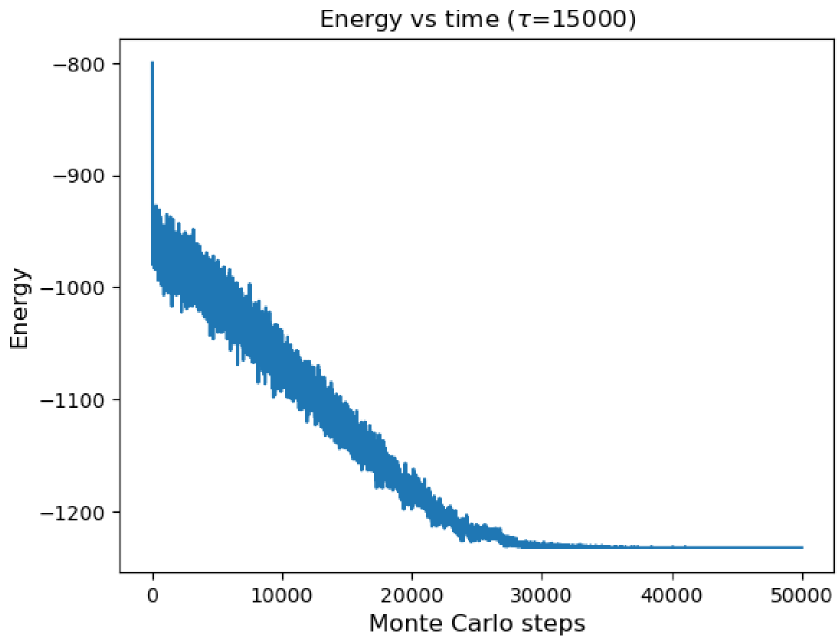


Figure 12: Energy function of the system in function of Monte Carlo steps for $\tau = 15\,000$, and for a lattice of $L = 50$ but with more number of Monte Carlo steps.

Here we clearly see the achievement of an approximate ground state, with $E_{\text{g.s.}} = -1232$. We notice that, in fact, slower cooling schedules allow the algorithm to find better solutions, even if more is the time required. Although the time required to reach this result is longer compared to faster schedules, the final estimated ground-state energy is more accurate in the case of slower cooling than in the case of faster cooling.

Another aspect to consider is that, as seen in the plot generated by the code (linked in Sec. 6), faster cooling schedules result in a smaller fraction of the lattice being covered by dimers, with larger empty spaces between them. In contrast, slower cooling schedules allow the algorithm to find much better lattice coverings, producing more compact and complete dimer arrangements.

4 Power spectra of random numbers and random walks

The linear congruential generator (LCG) is a simple method for generating sequences of pseudo-random numbers. It is defined by the recurrence relation

$$x' = (ax + c) \bmod m, \tag{5}$$

where a, c, and m are integer constants, and x is an integer variable. Given an initial value x, this equation produces a new integer x'. By repeatedly substituting each new value back into the equation, we generate a sequence of integers that can be used as pseudo-random numbers. In our program, we use the following parameters for the linear congruential generator

- $a = 1664525$,
- $c = 1013904223$,

- $m = 2^{32}$,

parameters taken from the Wikipedia entry on linear congruential generators, which cites the values recommended in Numerical Recipes. The linear congruential generator produces a sequence of apparently random integers by repeatedly iterating the same recurrence relation. It is one of the most widely known and historically important methods for generating pseudo-random numbers. In this way, we construct the linear congruential generator (LCG) to produce uniformly distributed random numbers in the interval $[0, 1)$, using the parameters given above. Using this generator, we create 10 000 Gaussian random variables (using the straightforward method described in Newman's book) and then construct a histogram of the results. We compare this histogram with the standard Gaussian distribution (mean 0, standard deviation 1), and the result obtained is shown below

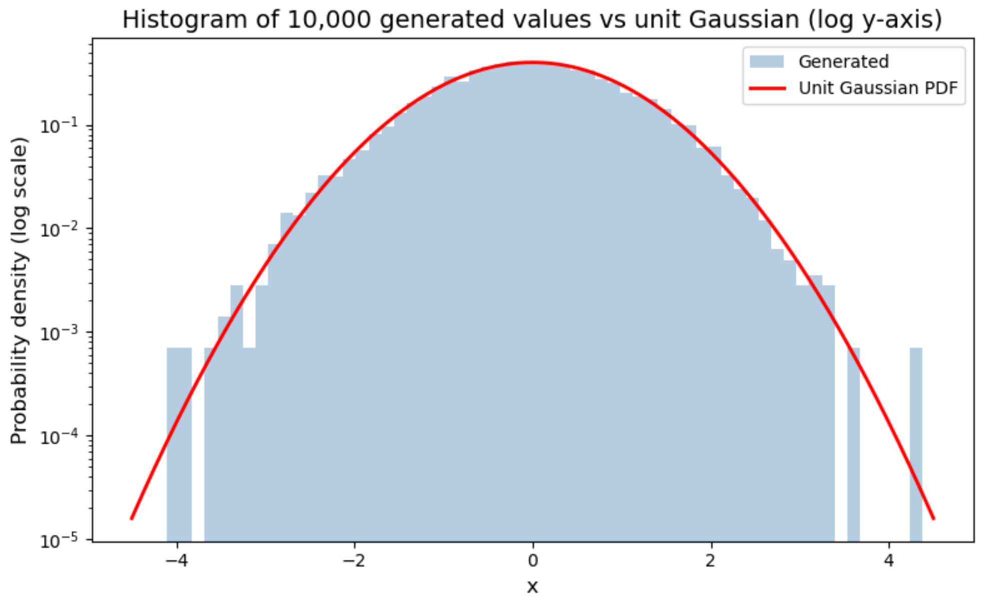


Figure 13: Comparison between the histogram of 10 000 values generated by the LCG constructed above and the standard Gaussian distribution, demonstrating that our code performs correctly.

From this plot, we immediately observe good agreement between the histogram and the standard Gaussian probability density function. The use of a logarithmic scale on the y-axis makes it easier to verify that the frequency of rare events is correctly reproduced by our code.

A further consistency check consists in computing the discrete Fourier transform of the data and evaluating the power spectrum of the set of 10 000 Gaussian random numbers, in particular verifying that the spectrum exhibits the correct scaling with the wavenumber k by plotting $\log P$ as a function of $\log k$. Hence, by following exactly the same procedure used in Homework 3, we obtain

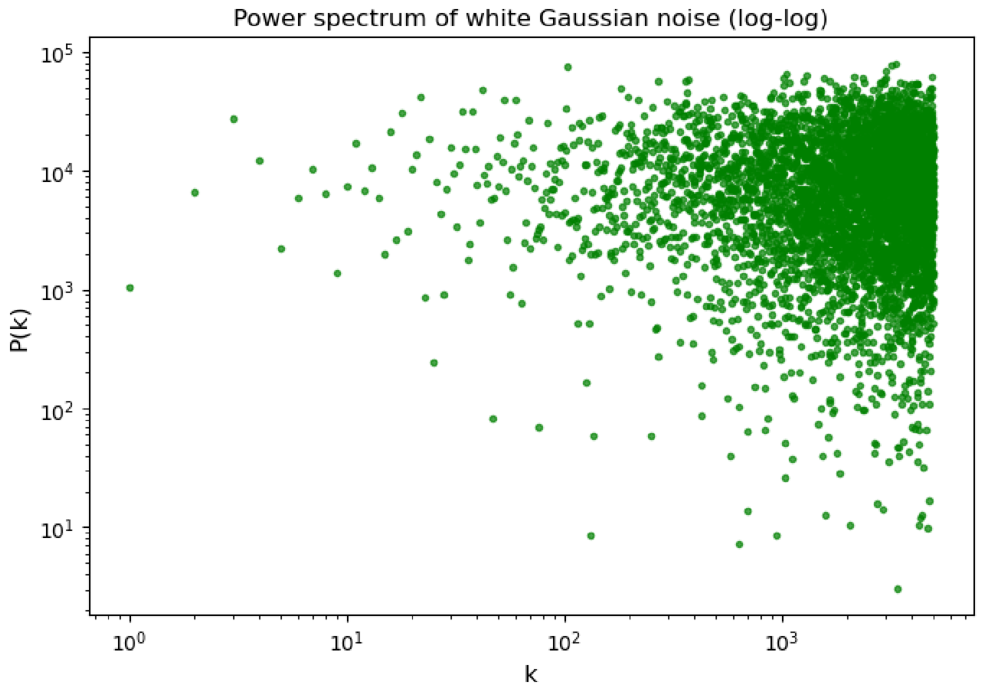


Figure 14: Power spectrum of our list of 10 000 random Gaussian numbers.

From this plot, we immediately see that the power spectrum has no peaks but instead fluctuates around an horizontal line. Since there is no dependence on the frequency k, we can conclude that the signal is white noise, characterized by an essentially flat spectrum, and this is exactly what we expect for the power spectrum of Gaussian-distributed, uncorrelated variables, so the result is fully consistent with theory.

We now use this sequence of Gaussian random numbers to construct a random walk, where the i-th value of the walk is given by the $(i-1)$-th value plus the i-th Gaussian random number. In particular, by plotting the random walk as a function of the iteration index i, we obtain

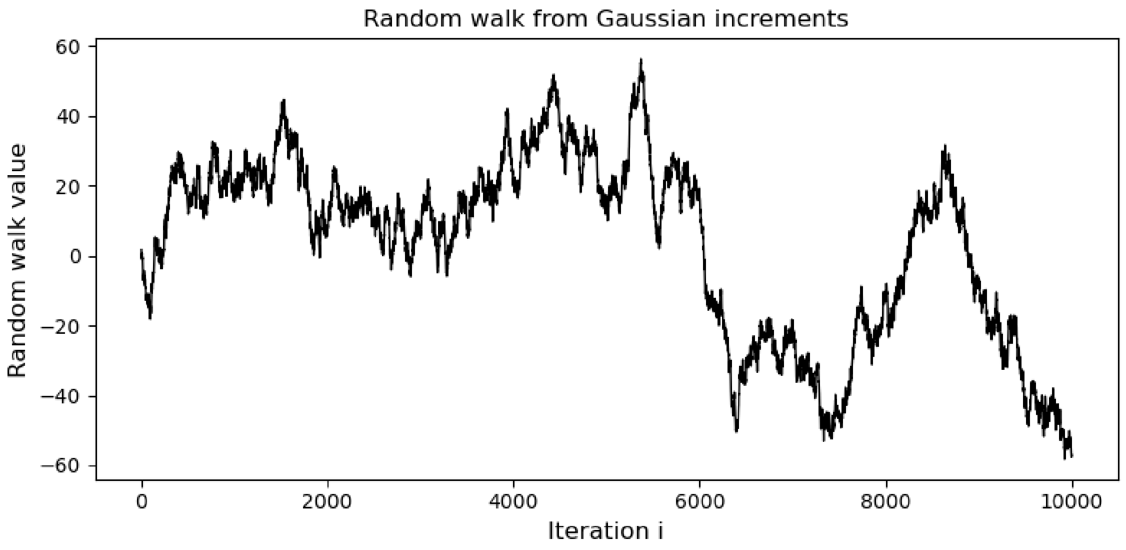


Figure 15: Random walk from Gaussian increments.

This behavior is expected, since the random walk fluctuates around zero. As in the previous case, we then compute the power spectrum in log.log scale of the random walk, obtaining

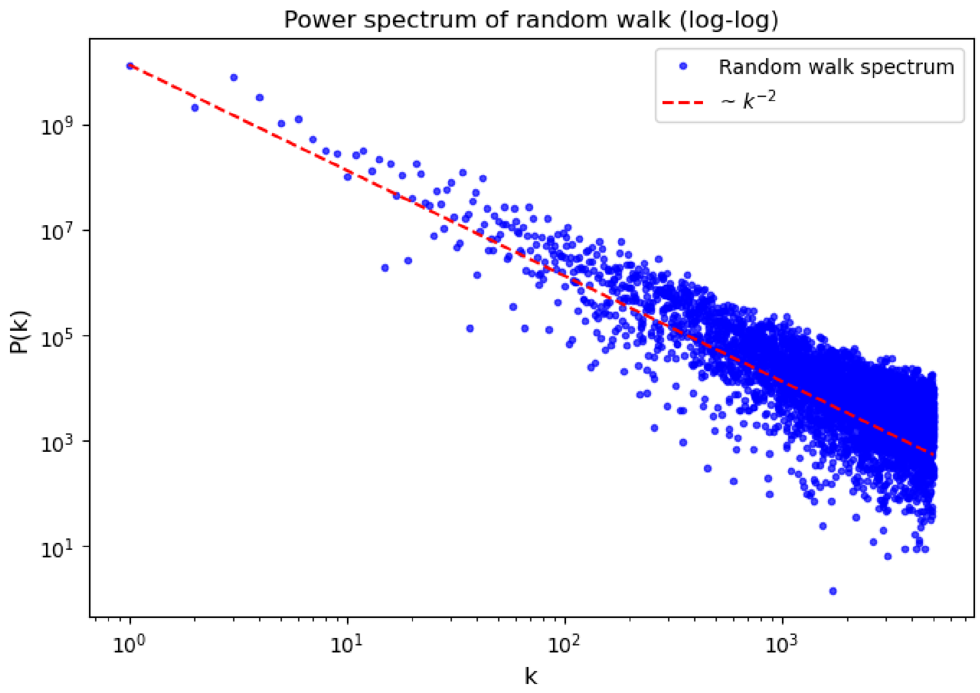


Figure 16: Power spectrum of random walk from Gaussian increments.

Here, once again, we observe the expected behavior: the power spectrum exhibits a dependence $P(k) \propto k^{-2}$ on the wavenumber k, which is characteristic of a random walk. From these results, we conclude that our LCG correctly generates uniformly distributed random numbers in the interval $[0, 1)$ and that our implementation of the Gaussian random number generator is accurate.

5 Conclusion

We have learned how to apply Markov Chain Monte Carlo methods in Sec. 2, where we studied the Ising model and investigate the emergence of spontaneous magnetization and its dependence on temperature. In Sec. 3, we addressed another well-known problem in condensed matter physics, namely the dimer covering problem, which we solved using the simulated annealing method. In particular, we studied the effect of varying the cooling schedule, whose time constant plays a crucial role in the quality of the final solution. Finally, in Sec. 4, we constructed a linear congruential random number generator to produce uniformly distributed random numbers in the interval $[0, 1)$. We then used this generator to produce Gaussian random variables and apply them to the study of a random walk. In this context, we analyzed the behavior of the corresponding power spectra using the techniques introduced in Homework 3.

6 Code

The code containing all the results can be found at the following Repository GitHub.