Monte Carlo simulations
The Ising Model and Phase transition
FYS3150: Computational Physics

Sigurd Sandvoll Sundberg[*]
Department of Geosciences, University of Oslo

(Dated: November 20, 2020)

I. INTRODUCTION

II. THEORY

The Ising model is a binary system, where the objects at each lattice site can take one of two values, either spin up or spin down. The energy of the Ising model, with no external magnetic field can be expressed as

$$E = -J \sum_{\langle k,e \rangle} s_k s_e, \tag{1}$$

where we sum over the nearest neighbors interactions at each lattice site and s_i can take the values ± 1, which represents either spin up or spin down, respectively.

We will in this article use natural units such as $k_B = J = 1$, where k_B is the Boltzmann constant.

A. Canonical Ensemble

In this article we will be working with an canonical ensemble, where the energy follows an expectation value at a given temperature. For us to calculate the expectation values such as the mean energy $\langle E \rangle$, we will be using a probability distribution, namely the Boltzmann distribution

$$P_i(\beta) = \frac{e^{-\beta E_i}}{Z} \tag{2}$$

where $\beta = 1/k_B T$ being the inverse temperature, k_B is the Boltzmann constant, E_i is the energy at microstate i and Z is the partition function. We will in this article use natural units, that is $k_B = J = 1$.

The partition function Z is given as

$$Z = \sum_{i=1}^{M} e^{-\beta E_i}, \tag{3}$$

where we sum over all microstates M.

The expectation value of the energy can be calculated from the probability distribution P_i as

$$\langle E \rangle = \sum_{i=1}^{M} E_i P_i(\beta) = \frac{1}{Z} \sum_{i=1}^{M} E_i e^{-\beta E_i} \tag{4}$$

The corresponding variance is defined as

$$\begin{aligned} \sigma_E^2 &= \langle E^2 \rangle - \langle E \rangle^2 \\ &= \frac{1}{Z} \sum_{i=1}^{M} E_i^2 e^{-\beta E_i} - \left(\frac{1}{Z} \sum_{i=1}^{M} E_i^2 e^{-\beta E_i} \right)^2 \end{aligned} \tag{5}$$

If we divide by $k_B T^2$, we obtain the specific heat

$$C_v = \frac{1}{k_B T^2} \left(\langle E^2 \rangle - \langle E \rangle^2 \right). \tag{6}$$

In the same way we can evaluate the mean magnetization through

$$\langle \mathcal{M} \rangle = \sum_{i}^{M} \mathcal{M} P_i(\beta) = \frac{1}{Z} \sum_{i}^{M} \mathcal{M} e^{-\beta E_i} \tag{7}$$

and the corresponding variance

$$\begin{aligned} \sigma_{\mathcal{M}}^2 &= \langle \mathcal{M}^2 \rangle - \langle \mathcal{M} \rangle^2 \\ &= \frac{1}{Z} \sum_{i=1}^{M} \mathcal{M}_i^2 e^{-\beta E_i} - \left(\frac{1}{Z} \sum_{i=1}^{M} \mathcal{M}_i^2 e^{-\beta E_i} \right)^2. \end{aligned} \tag{8}$$

This quantity defines the susceptibility $\mathcal{X}$

$$\mathcal{X} = \frac{1}{k_B T} \left(\langle \mathcal{M}^2 \rangle - \langle \mathcal{M} \rangle^2 \right). \tag{9}$$

B. Phase Transition

For our 2-dimensional Ising model, we from from a finite magnetization $\langle \mathcal{M} \rangle \neq 0$ to a paramagnetic phase with $\langle \mathcal{M} \rangle = 0$ at a critical temperature T_C. At this critical temperature the mean magnetization approaches zero with an infinite slope. Quantities like the heat capacity C_v and the susceptibility $\mathcal{X}$ are discontinuous or diverge in the thermodynamic limit[1], this happens as the lattice size L approaches infinity. As we can not simulate a infinite lattice, we will have to deal with approximating the critical temperature at which phase transition occurs.

When studying a finite lattice, the variance in energy and magnetization no longer diverge or are discontinuous, however they scale as $\sim 1/\sqrt{M}$, where M is given as $M = 2^{L^2}$ microstates, for an LxL lattice. This means that we will not be able to observe a diverging behavior, but we will should be able to see as we increase the lattice size, that the peaks of C_v and $\mathcal{X}$ should get sharper.

[*] https://github.com/SigurdSundberg/FYS3150

The critical temperature for a finite lattice follows the following relation

$$T_C(L) = aL^{-1/\nu} + T_C(\infty), \tag{10}$$

where $T_C(L)$ is the critical temperature found for a finite lattice size LxL. From equation (10) we will try to estimate $T_C(\infty)$ using $\nu = 1$ by studying the linear relation between the critical temperature and lattice size.

C. Analytical Results

In 1944 Lars Onsager was the first to solve the Ising model analytically for a general lattice size L[3]. He found that the critical temperature of the system is

$$T_C = \frac{2J}{k_B ln(1+\sqrt{2})} \simeq 2.269\ldots \tag{11}$$

We will use the ideas from phase transition to try to estimate T_C, using larger lattice size.

However first we need a benchmark for our code, where we will use a 2x2 system with periodic boundary conditions to calculate numerical expectation values. The analytical expectation values and the partition function can be calculated through equations (3,4, 7), and the value displayed in Table I. Likewise we can find values of E^2 and M^2.

We find that

$$Z = 12 + 4\cosh(8J\beta) \qquad \beta = (k_BT)^{-1} \tag{12}$$

$$\langle E\rangle = -\frac{32J}{Z}\sinh(8J\beta) \tag{13}$$

$$\langle E^2\rangle = \frac{256}{Z}\cosh(8J\beta) \tag{14}$$

$$\langle M\rangle = 0 \tag{15}$$

$$\langle M^2\rangle = \frac{32}{Z}\left(e^{8J\beta}+1\right) \tag{16}$$

$$\langle |M|\rangle = \frac{8}{Z}\left(e^{8J\beta}+2\right) \tag{17}$$

From equation (6), the heat capacity of the system is

$$C_v = \frac{256}{Zk_BT^2}\left(\cosh(8J\beta) - \frac{4}{Z}\sinh^2(8J\beta)\right) \tag{18}$$

and we can find the susceptibility from equation (9), which can be expressed as

$$\mathcal{X} = \frac{32\beta}{Z}\left(e^{8J\beta}+1\right). \tag{19}$$

III. ALGORITHMS

We will be studying the 2-dimensional Ising model, through the use of Monte Carlo methods. Monte Carlo methods are class of computational algorithms, which use repeated random sampling to obtain numerical solutions. The can stretch from easy and trivial to implement to more advanced models where one utilizes different properties of a system to create more efficient algorithms.

TABLE I: Display over the possible spin configurations for the 2x2 lattice. As well as listing the energy and magnetization of system.

$N_\uparrow$	Degeneracy	E	M	Configurations
4	1	-8J	4	$\begin{bmatrix}\uparrow & \uparrow\\ \uparrow & \uparrow\end{bmatrix}$
3	4	0	2	$\begin{bmatrix}\downarrow & \uparrow\\ \uparrow & \uparrow\end{bmatrix}$ $\begin{bmatrix}\uparrow & \downarrow\\ \uparrow & \uparrow\end{bmatrix}$ $\begin{bmatrix}\uparrow & \uparrow\\ \downarrow & \uparrow\end{bmatrix}$ $\begin{bmatrix}\uparrow & \uparrow\\ \uparrow & \downarrow\end{bmatrix}$
2	4	0	0	$\begin{bmatrix}\downarrow & \downarrow\\ \uparrow & \uparrow\end{bmatrix}$ $\begin{bmatrix}\uparrow & \uparrow\\ \downarrow & \downarrow\end{bmatrix}$ $\begin{bmatrix}\downarrow & \uparrow\\ \downarrow & \uparrow\end{bmatrix}$ $\begin{bmatrix}\uparrow & \downarrow\\ \uparrow & \downarrow\end{bmatrix}$
2	2	8J	0	$\begin{bmatrix}\downarrow & \uparrow\\ \uparrow & \downarrow\end{bmatrix}$ $\begin{bmatrix}\uparrow & \downarrow\\ \downarrow & \uparrow\end{bmatrix}$
1	4	0	-2	$\begin{bmatrix}\uparrow & \downarrow\\ \downarrow & \downarrow\end{bmatrix}$ $\begin{bmatrix}\downarrow & \uparrow\\ \downarrow & \downarrow\end{bmatrix}$ $\begin{bmatrix}\downarrow & \downarrow\\ \uparrow & \downarrow\end{bmatrix}$ $\begin{bmatrix}\downarrow & \downarrow\\ \downarrow & \uparrow\end{bmatrix}$
0	1	-8J	-4	$\begin{bmatrix}\downarrow & \downarrow\\ \downarrow & \downarrow\end{bmatrix}$

Random sampling with Monte Carlo method gives an asymptotic approximation error of $\sigma/\sqrt{N}$. Where σ is the variance of the system and N is the number of samples[1]. Compared to simpler methods for approximating numerical results, the Monte Carlo method performs poorly when the sample size is small. Thus it could be infeasible for simpler problems to apply Monte Carlo methods to simple systems. However the Ising model has not a simple system to simulate and fits the Monte Carlo method perfectly.

A. Markov Chains

A Markov process (or chain) generates random states by use of random walks, that depends on a chosen probability distribution. A move from state i to a new state j can be described by a *transition probability* $P(i \to j)$, which is the probability that the system will go from state i to state j.

If we let a Markov chain run for long enough under certain conditions, the system will reach its most likely state, regardless of initial state. In our case the most likely state is the equilibrium state of the system, with the probability distribution $w_j(t_k) = e^{\beta E_j}$. The time development of our probability distribution is given by

$$w_j(t_{k+1}) = P(i \to j)w_i(t_k) = P_{ij}w_i(t_k). \tag{20}$$

In vector-matrix representation, we have

$$\vec{w}(t_{k+1}) = \mathbf{P}\vec{w}(t_k) \tag{21}$$

where the transition probability P_{ij} is represented on matrix form. The system is said to be in its most likely state when $||w(t_{k+1}) - w(t_k)|| \to 0$.

Following are some properties that the Markov process must obey.

1. Time Independence

The transition probability must be time-independent, such that for a given time t_v, the probability for transitioning from a state $i \to j$ is the same as at time t_u.

The second part of the time independence is that $P(i \to j)$ should only be dependent on the states i and j, that is the system is said to be "without history", independent of the time evolution of the system.

2. State Change

A Markov chain must put the system in some state j given a state i. So the system follows the following equation

$$\sum_j P(i \to j) = 1, \tag{22}$$

this includes a state change from $i \to i$, such that $P(i \to i) \neq 0$.

3. Ergodicity

A last principle for the Markov chain is that: *It is possible to reach any state j, from any state i given a long enough Markov chain.* This condition is called ergodicity. It is important because every state i in the Boltzmann distribution has a non-zero probability.

B. Metropolis Algorithm

In most cases, including ours the transition probability $P_{ij} = P(i \to j)$ is not known. A Markov chain is such a process where the transition probability is generally unknown. This is where the Metropolis algorithm comes to the rescue.

We can model the transition probability as the product of two probabilities. One probability of accepting the proposed move from state i to state j and the other probability for making the transition to the state j being in the state i[1]. We will label these probabilities as $A(i \to j)$ and $T(i \to j)$, respectively. Our transition probability then reads

$$P(i \to j) = T(i \to j)A(i \to j). \tag{23}$$

We want to derive the probabilities A and T, such that $w_j(t_{k+1}) = w_j t_k$ when $t \to \infty$, starting from any initial distribution.

As we are modeling our system as a Markov chain, we require that as as $t \to \infty$ we have

$$\sum_i w_i T(i \to j)A(i \to j) = \sum_i w_i T(j \to i)A(j \to i) \tag{24}$$

$$= \sum_i w_i P(j \to i) \tag{25}$$

using that $\sum_i P(j \to i) = 1$, we have

$$w_j = \sum_i w_i T(i \to j)A(i \to j) = \sum_i w_i P(i \to j) \tag{26}$$

which is the standard Markov chain equation when a steady state has been reached.

1. Detailed Balance

In general the condition that the probabilities should be equal, is normally not enough to ensure we reach the most likely state. We could end up in a cyclic solution where we go back and forth between to solutions. To counter this we will introduce detailed balance[2]

$$P(i \to j)w_i = P(j \to i)w_j. \tag{27}$$

At equilibrium detailed balance gives us

$$\frac{P(i \to j)}{P(j \to i)} = \frac{w_j}{w_i}, \tag{28}$$

rewriting in terms of T and A we have

$$\frac{T(i \to j)A(i \to j)}{T(j \to i)A(j \to i)} = \frac{w_j}{w_i}. \tag{29}$$

This condition ensures that it is our probability distribution that is reach when the system reaches equilibrium[1].

With our probability distribution given by the Boltzmann distribution (2), the partition function (3) is infeasible to calculate and brute force Monte Carlo calculation will not results in the microstates which are important at equilibrium. To select the contributions that are important to the equilibrium, we need to use the detailed balance which is just given by a ration of probabilities. This makes us able to never have to calculate the partition function Z. For Boltzmann distribution the detailed balance results in

$$\frac{w_j}{w_i} = e^{-\beta(E_j - E_i)}. \tag{30}$$

For the Ising model, a change in state from $i \to j$, would mean one of two things. Either a spin-flip or no change at all. This selection of states has to results in the Boltzmann distribution otherwise, our choice of microstates is wrong. Leading to us getting wrong results when sampling the system.

2. The Metropolis Algorithm Specialized

The simplest form of the Metropolis algorithm is assuming that the transition probability $T(i \to j)$ is symmetric, implying that $T(i \to j) = T(j \to i)$. Using this we have from equation (29) and equation (30)

$$\frac{A(i \to j)}{A(j \to i)} = e^{-\beta(E_j - E_i)}. \tag{31}$$

Suppose $E_j > E_i$. Since it is only the ration that needs to obey the equality, we can make the algorithm more efficient by setting the largest of the two acceptance ratios to 1. Then we have

$$\frac{A(i \to j)}{A(j \to i)} = e^{-\beta(E_j - E_i)} < 1, \tag{32}$$

where we impose $A(j \to i) = 1$. When $E_j < E_i$, the energy of the system is lowered and we accept the state change regardless. The acceptance probability can be summarized as

$$A(i \to j) = \begin{cases} e^{-\beta(E_j - E_i)}, & E_j - E_i > 0 \\ 1, & else. \end{cases} \tag{33}$$

Or in a different way, we can write the following `if(`Δ`E < 0 or r<`$e^{-\beta(E_j-E_i)}$`)` {`We accept the proposed swap`}. This ensures that we will reach the most likely state, but allowing energy states are less likely to be accepted. Making the acceptance ratio, able to follow the principle of ergodicity. This is done by choosing a random number `r` from a uniform distribution between 0 and 1.

C. Monte Carlo - Metropolis Algorithm

If we combine the above principles our final algorithm for performing the Monte Carlo simulations can read as follows

Algorithm 1: Monte Carlo simulation with Metropolis sampling

```
Pick T and $L_i$
Compute $E_i$ and $M_i$
for i = 0; i < MC cycles; i++ do
    for j = 0; j < Total Spins; j++ do
        Sample a random index of the spin matrix
        Compute $\Delta E$
        if $\Delta E < 0$ or $r < e^{-\beta \Delta E}$ then
            Accept the flip
            Update E and M
        end
        Update the mean values for E and M
    end
end
Normalize all computed values
```

IV. METHOD

A. Lattice Representation

In order for us to represent the lattice in a efficient way, we will be using a matrix representation of the lattice. This enables us to easily apply our boundary conditions as well as enables us to with ease to increase the efficiency of our code by improving upon our algorithm. Both of which we will discuss shortly.

B. Periodic Boundary Conditions

For boundary conditions we have chosen to use periodic boundary conditions, and as the lattice grows towards the thermodynamic limit, the choice of boundary conditions does not play a role. As approximating infinity with one additional element does not change the outcome. We are essentially going to approximate a large system, by making it slightly larger.

When applying periodic boundary conditions we need a way to index ghosts points, that is indexing points outside the scope of our lattice representation. The simple way of doing so would be to implement `if`-statements to check whether we are indexing a ghost point or not. This is however slow and hard to optimize.

A second possibility for handling the ghost points would be to expand our matrix to a (L+2)x(L+2) matrix, where we have the ability to index the ghosts points directly without the use of `if`-statements. However this runs into its own limitations, namely we would need to update the ghosts points whenever we change its corresponding value. In a 1-dimensional case this would lead to us having to update index 0 if we made a change to index n, for an array from 0 to n+1. This can be implemented using either `if`-statements or using properties of modulus operator.

A third method, which is the one used in this project, is using the modulus operator directly on a LxL matrix. This is done by looking at the point of interest in the matrix and compute its neighboring indexes by the following equation

$$\text{neighbor} = (\text{current} + \text{L} + \text{position})\text{MOD}(\text{L}) \tag{34}$$

where current is the current matrix index, L is the dimensionality of the matrix and position is the index of the neighbor relative to current position. This equation handles both the ghost points and the normal points of the matrix and creates an efficient way for us to access all points of the matrix.

By implementing this as an inline function, we are able to, at compile time, to reduce the time used of our program.

TABLE II: The possible changes in energy ΔE for the possible different neighbor configurations. We see that each lattice site only can have five different values for ΔE, as they are only interacting with their nearest neighbors. The position of the neighboring spins does not change the change in energy, this is easy to show.

Initial state	Final state	Initial E[J]	Final E[J]	ΔE[J]
↑ ↑ ↑ ↑ ↑	↑ ↑ ↓ ↑ ↑	-4	4	8
↓ ↑ ↑ ↑ ↑	↓ ↑ ↓ ↑ ↑	-2	2	4
↓ ↓ ↑ ↑ ↑	↓ ↓ ↓ ↑ ↑	0	0	0
↓ ↓ ↑ ↓ ↑	↓ ↓ ↓ ↓ ↑	2	-2	-4
↓ ↓ ↑ ↓ ↓	↓ ↓ ↓ ↓ ↓	4	-4	-8

C. Exponential

In algorithm 1 we are required to calculate $e^{-\beta\Delta E}$ for every spin, that is if we let M = number of Monte Carlo cycles and L^2 denote total number of spins, we would have to calculate this value $M \cdot L^2$ times. This would become exponentially large as we tackle larger lattice sizes with large amount of Monte Carlo cycles. To help us having to calculate the exponential millions of times per run, we can use some properties of the 2-dimensional lattice.

One method of finding the change in energy after a lattice sweep is, sweeping through the lattice and calculate the energy. This is however not inefficient and not the chosen method. Instead we sweep through the lattice and only look at one spin at a time and updating the energy each time. This allows us to precalculate the values of the exponential as the change in energy at each lattice site only can take one of five values as shown in Table II. This enables ut for a given temperature T, to calculate all the possible values from the exponential prior to performing the Monte Carlo Metropolis algorithm.

D. Change in Energy and Magnetization

As we are sweeping through our lattice we need to calculate the change in energy ΔE and change in magnetization Δ M, for each flip we accept. As we saw from Table II the change in energy can only take one out of five values, since each spin site is only dependent on its four neighbors, as the other spins remain unchanged. Such that the change in energy ΔE can be found the following way

$$\Delta E = E^j - E^i \tag{35}$$

where E^j denotes the energy after the flip and E^i before the flip. We then have

$$\begin{aligned}\Delta E &= -J\sum_{\langle k,e\rangle} s_k^j s_e^j + J\sum_{\langle k,e\rangle} s_k^i s_e^i &(36)\\ &= -J\sum_{\langle k,e\rangle} s_k^j\left(s_e^j - s_e^i\right). &(37)\end{aligned}$$

We can simplify this further, by looking at $s_e^i = 1$, if we flip it then $s_e^j = -1$ This means that $s_e^j - s_e^i = -2$. Also if $s_e^i = -1$ then $s_e^j = 1$, thus $s_e^j - s_e^i = 2$. Therefore ΔE can be calculated using

$$\Delta E = 2Js_e^i\sum_{\langle k\rangle} s_k^i \tag{38}$$

The change in magnetization is given in the same way.

$$\begin{aligned}\Delta M &= M^j - M^i &(39)\\ &= \sum_k s_k^j - \sum_e s_e^i &(40)\\ &= s_k^j - s_k^i &(41)\\ &= 2s_k^j &(42)\end{aligned}$$

E. Initial State

To start of our representation of the lattice has no orientation. We would need to give it a starting orientation before performing the Monte Carlo simulations. We have two different ways of initializing the lattice. First, we can have all the spins point the same way, this gives a highly ordered lattice initially.

A second option is to use a random number generator to randomly assign whether a spin points upwards or downwards, independent of neighboring spins. This gives us a configuration of high disorder.

For small lattices this is computationally inexpensive, but as the lattice size grows initializing the lattice takes up more and more CPU cycles, if we need to initialize the lattice for each temperature. To reduce the number of CPU cycles we will look into how we can utilize small time steps and the equilibration time of the system to improve our calculations.

F. Equilibration Time

At the start of each run, we have a lattice which is given one of two configurations. Either all spins pointing in the same direction, or all spins initial direction randomly chosen. This state is generally far away from the most likely state for a given temperature. As we are mainly interested in the expectation values of the system after reaching most likely state, we need to make sure that our system is at the most likely state before we start gathering data. There are different ways of determining when a system reaches its most likely state.

One method is to choose a lattice size and run the Monte Carlo simulations, gathering all data and study the data afterwards. By looking at the expectation values as a function of the Monte Carlo cycles, we can approximate how long the system needs to run before reach the most likely state. Afterwards we can let the system run for the number of cycles we found before calculating the expectation values, this is a simple solution, and has its downsides. A downside is that the number of cycles needed to reach the most likely state, may depend on the initial condition and the temperature at the point of calculation. Therefore we can not be sure we have reached the most likely state before sampling our system for expectation values. To counter this we can overestimate the number of cycles we need, but that defeats the purpose of this method.

A similar method which is easier to implement and does not require trial runs to find the approximation to the most likely state, is to simply slice away a portion of cycles and call it there. For low number of cycles, this also may not lead to the most likely state when we start to collect expectation values, however as the Monte Carlo method is greedy, we would normally use a number of cycles high enough to counter this problem outright. This is the standard approach for Monte Carlo methods and it is general practice to discard 10% to 15% of the runs before sampling the system, when it comes to the Ising model. As said however this does not guarantee that we are at an equilibrium position for the, as it depends on both the temperature and initial configuration.

Another way we could use the fact that for a system in equilibrium, will have little change in the energy between two different configurations. Such that we can sample and check a few cycles for whether the change in energy and magnetization is sufficiently small, however defining sufficiently small is hard. And as we are dealing with random number generators we can never be sure that the consecutive runs are actually at an equilibrium state. So this would either never reach equilibrium or use a large amount of CPU cycles to find equilibrium.

Lastly we can find the equilibrium from one time step to the next, with time step sufficiently small, by assuming that the equilibrium state for two lattices a small time step apart are the same.

In this project we have applied a mixture of the different methods. We will be discarding 10% of the Monte Carlo cycles after the initialization of the lattice, to make sure we are hopefully close to the equilibrium, and start sampling for expectation values for a given T. After we are done sampling for a T, we will assume that the lattice is at equilibrium for the next temperature, with us having a sufficiently small time step. That is us using the old configuration to approximate the new configuration without having to equilibrate the lattice once more. For this we will use a time step between $0.05J/k_B$ to $0.01J/k_B$, depending on lattice size.

G. Optimization

For the Monte Carlo methods, `c++` has been used as a programming language, and without any modifications, the language does not use multi-core processes. Meaning that we are running our programs on one core instead of utilizing the entire machine. As we are dealing with systems which requires many, many CPU cycles, we will use Message Parsing Interface(MPI), to parallelize the program. This allows us to perform multiple Monte Carlo simulations at a time, instead of one. We have here chosen to have each core, perform its own Monte Carlo simulation, effectively increasing the number of cycles we can perform for a given system. All the systems will be initialized by their own core and have no contact until all calculations are done. As an example, instead of running 10^6 cycles on one core, we are able to run $4 \cdot 10^6$ cycles, which is positive for the Monte Carlo method, as it is greedy.

Another choice could have been to split the cycles into smaller chunks and have each core process a part of the total number of cycles. This is less expensive in CPU time, but in turn results in fewer samplings of the system.

For this article all results with lattice sizes larger than 20x20, a parallelized version of algorithm 1 has been used, and ran on a computer with 8 1GHz cores.

V. RESULTS

A. 2x2 Lattice

B. Equilibration

C. Acceptance Ratio

D. Probability Distribution

E. Parallelization and Optimization

F. Phase Transition

G. Critical Temperature

VI. DISCUSSION

VII. CONCLUSION

[1] Morten Hjorth-Jensen. Computational physics, lecture notes fall 2015. *Department of Physics, University of Oslo*, 2015.

[2] Lars Onsager. Reciprocal relations in irreversible processes. i. *Phys. Rev.*, 37:405–426, Feb 1931.

[3] Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Phys. Rev.*, 65:117–149, Feb 1944.