Long range Ising model

Santiago Sanz Wuhl

May 26, 2025

1 Introduction

The Ising model **?** is able to explain the spontaneous magnetization of spin systems by only considering interactions of nearest neighbours. The Metropolis **?** algorithm is used in order to more efficiently probe the 2^d dimensional phase space.

However, as it is well-known, the 1-dimensional Nearest Neighbour Ising Model presents no phase transition, whereas the consideration of long range interactions in the Long Range Ising Model (LRIM) the critical temperature at which the system becomes ferromagnetic becomes non-zero **?**.

Computer simulations of the LRIM are however very computationally expensive, as it requires an $\mathcal{O}(N)$ at each time step t. In order to alleviate this computational cost, we look at the bond dilution approximation. In this approximation, each interaction of strength $r_{ij}^{-(d+\sigma)}$, is modelled as an interaction of strength 1 (up to changes of units), but with a probability of having an interaction given by $r_{ij}^{-(d+\sigma).}$ Here d is the dimension of the space, and σ is a real parameter, which controls the range of the interactions.

In this work we study the dynamics of this bond diluted approximation, and whether this allows us to replicate the behavior of the LRIM.

2 Preliminaries

We consider a system made up of N particles, governed by the Hamiltonian

$$H = -\sum_{1\leq j<i}^{N} J_{ij} s_i s_j, \tag{1}$$

with J_{ij} the elements of a 2-dimensional symmetric matrix, dictating the interaction strength between the particle at the site i and the particle at the site j, and $s_i = \pm 1$ the spin of the particle at the site i. Any two particles interact via a power law

$$J_{ij} = \frac{J_0}{r_{ij}^{d+\sigma}}, \tag{2}$$

with $r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|$, d the spatial dimensions, J_0 a positive constant and σ a free parameter characterizing the range of the interaction.

2.1 The Ising model

A fundamental quantity in statistical mechanics is the partition function

$$Z = \sum_s \exp(-\beta E_s),$$

where the sum is performed over states, $\beta = \frac{1}{k_B T}$ is the inverse temperature, k_B the Boltzmann coefficient and E_s the energy of the state s. By use of the partition function, one is able to calculate observables e.g. the average energy of the system

$$\begin{aligned} \langle E \rangle = \sum_s E_s P_s &= \frac{1}{Z} \sum_s E_s \exp\left(-\beta E_s\right) \\ &= -\frac{1}{Z} \frac{\partial}{\partial \beta} Z \\ &= -\frac{\partial \ln Z}{\partial \beta}, \end{aligned} \tag{3}$$

where P_s is the probability of finding the system in the state s, with energy E_s.

The averaging over states is however very computationally expensive, as it is to be summed over the $2dN$ dimensional phase space, with d the dimensions of the configuration space. One instead calculates thermal averages by use of an algorithm by **?**, where the thermal average of an observable A is calculated by generating a sequence of M configurations $\{s_1, \ldots, s_M\}$ and calculating

$$\langle A \rangle = \frac{1}{M} \sum_{m=0}^{M} A_m. \tag{4}$$

A configuration s_j is generated with a probability π_{ij} from a configuration s_i. The algorithm proposed by **?** relies on the sampling of the state s_j with an *a priori* probability α_{ij}, and it is accepted with a probability P_{ij}. While α_{ij} is set by the system, the acceptance probability is

$$P_{ij} = \begin{cases} \exp[-\beta(E_j - E_i)] & E_j > E_i \\ 1 & E_j \leq E_i. \end{cases} \tag{5}$$

This way, a new state s_j is always accepted if it lowers the total energy of the system, but also accepts states of total higher energy with a probability that decreases with the temperature.

The simplest case that allows one to study the formation of spontaneous magnetization of ferromagnetic systems is the Nearest Neighbour Ising Model (NNIM), where only interaction between nearest neighbors is considered. This is modeled by the Hamiltonian

$$H = -J_0 \sum_{\langle ij \rangle}^{N} s_i s_j. \tag{6}$$

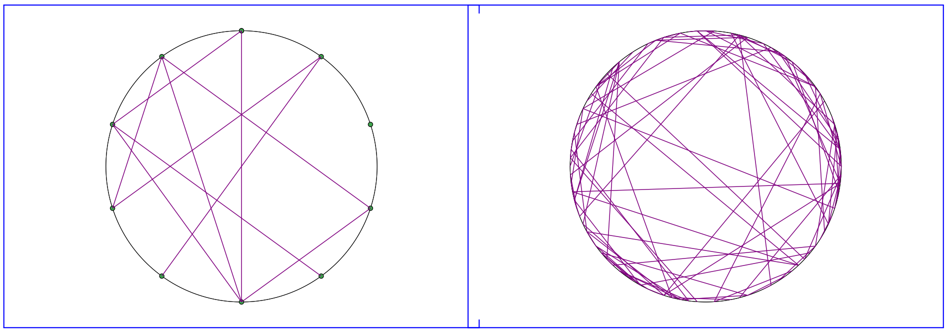

Figure 1: Partially connected graphs of different N and N_l, but same $\sigma = 5$. In the left panel, $N = N_l = 10$, and in the right panel $N = N_l = 100$.

Here $\langle ij \rangle$ represents summation only over nearest neighbours, and J_0 is a real constant. In terms of the Hamiltonian (**??**),

$$J_{ij} = \begin{cases} J_0 & i \,\&\, j \text{ neighbors} \\ 0 & \text{else.} \end{cases} \tag{7}$$

The Long Range Ising Model (LRIM), described by the Hamiltonian (**??**) was studied by **?**, where the magnetization squared m^2 is studied as an order variable.

For the Ising model, at each iteration m, a new state s_{m+1} is proposed by flipping the spin at a random site with a probability $\alpha_{ij} = N^{-1}$. In the NNIM, the change in energy of the total system depends only on the spins at the nearest neighbors of the site i, thus of complexity $\mathcal{O}(1)$. By calculating the change in energy, the new configuration s_{m+1} is accepted with a probability P_{ij} given by (**??**).

However, the NNIM presents no phase transition **?** for 1-dimensional spin chains. In order to study the fully connected spin chain with long range interactions (**??**), one needs to compute $\mathcal{O}(N)$ at each time step t of the Metropolis algorithm. In the interest of reducing this computation time, a mode dilution approximation is proposed **Citation?**.

2.2 Mode dilution

The mode dilution approximation studies the Hamiltonian given by (**??**), by setting all the interaction strengths to a constant J, and only allowing N_l pairs of sites to interact[1], any two pair of sites i, j is connected with a probability (without normalising) of $r_{ij}^{-(d+\sigma)}$. Figure **??** displays two examples of partially connected graphs with the same $\sigma = 5$, but $N = N_l = 10$ for the left panel, and $N = N_l = 100$ for the right panel. Both of these graphs present the same coordination number

$$z = 2\frac{N_l}{N},$$

i.e. the average number of bonds (both in- and outgoing) for each node.

[1]Thus, J_{ij} will have $2N_l$ non-zero elements.

2.3 Set up

We consider a 1-dimensional spin-chain with periodic boundary conditions, in the sense that spins at the boundaries act as if they were nearest neighbours. In this way, the distance between any two nodes i, j is the minimum distance along the circle that joins them, $r_{ij} = \min(|i-j|, |i-j| - N)$.

The units are chosen so that J_0 sets the units of energy, and temperature is measured in units of $\frac{J_0}{k_B}$. We set $J_0 = k_B = 1$ for convenience.

3 Algorithms

3.1 Generation of the J_{ij}

To simulate the dilute bond approximation, we need an algorithm with which to generate the matrix J_{ij} with non-uniformly distributed non-zero entries. an alias algorithm proposed by **?** is used. This method allows one to sample integers from 1 to α with a non-uniform probability distribution in $\mathcal{O}(1)$ computation complexity with $\mathcal{O}(N)$ memory complexity.

The connection set algorithm We will only be interested in the indices i, j for which J_{ij} is non-zero. Furthermore, since J_{ij} is symmetric, we characterize connections by the (unordered) set data structure $\{i, j\}$. The drawn connections $\{i, j\}$ are stored in another set `connectionSet`. Since sets are unordered, repeated connections do not change `connectionSet`, therefore the algorithm proceeds while `length(connectionSet) < N_l`. The algorithm to draw random connections is straight-forward: Draw an *uniformly* distributed integer from 1 to N to choose the first particle. The second site j is then chosen by generating a random integer $m \in [1, N-1]$, where the probability P_m of sampling m is $P_m = \min(m, \frac{N}{2}, N-m)^{-(1+\sigma)}$.

The sampled connection is thus $\{n, \mod (n+m, N)\}$, where $\{\}$ indicates the data structure `set`. The use of sets is not only motivated by convenience of the code, but they also present $\mathcal{O}(1)$ lookup time in `julia`, as it was tested by a user **?**.

The `julia` code used to generate the non-zero J_{ij} is found in Appendix **??**, and examples of diluted spin models with $N = N_l$ and $\sigma = 5$ are displayed in Figure **??**, with $N = 10$ on the left panel and $N = 100$ on the right panel.

3.2 Cluster counting

We will also be interested in the identification of clusters, this is, all simply connected subsets of nodes, either directly or indirectly. This is done by a Depth First Search (DFS) algorithm, which is explained below. In order to apply this algorithm, the connections set must be given a new format so that the DFS algorithm can be easier applied. An array `connectionArray` of N entries is created, where each of its entries is an empty array. The array `connectionArray[i]` contains the indices of the nodes to which the node `i` is connected. An example of a `connecionArray` corresponding to a fully connected graph with $N = 4$ is

```
connectionArray = [[2,3,4], [1,3,4], [1,2,4], [1,2,3].
```

The DFS algorithm is in the worst case $\mathcal{O}(N + N_l)$, with N the number of vertices of the graph and N_l the number of edges (of each cluster). Two sets are defined

1. `clusters`: The identified clusters

2. `visited`: The visitied nodes

Without loss of generality, we arbitrarily start at the first node `n=1`. Initialize the array `stack` containing only said node, and an empty set `cluster`. Then,

1. While the stack is non-empty: pop the stack to the variable `curr = pop(stack)`,

2. If `curr` is not in `visited`, add it to `cluster`, and to the `visited` set.

3. Retrieve the neighbors of `curr` from `connectionArray[curr]`, and add them to the stack.

4. Repeat step 1.

Reaching an empty stack means there are not any more non-visited nodes in the current cluster, and therefore `cluster` is returned and added to the set `clusters`. This procedure is repeated for every node, if it is not in `visited`.

The corresponding `julia` code can be found in **??**.

3.3 Ising model

Application of the Metropolis algorighm with the current set up is simple. Generate a random `spinArray` of length N containing only 1 or -1. Then,

1. A site `i` is chosen at random,,

2. Calculate the energy change `E` corresponding to flipping the spin `spinArray[i]`,

3. Flip the spin at site i with the probability P_{ij} given by Equation (**??**) and with the neighboring spins contained in `connectionArray[i]`,

4. Repeat step 1.

Care should be taken in the generation of the J_{ij}, as having a too low coordination number avoids the system from being almost fully connected, and prevents phase transitions.

4 Results

Figure **??** displays the change in number of clusters for different configurations, for a constant N, as a function of the number of bonds N_l. Different curves correspond to different σ, as indicated by the legend. The standard deviation of the sample mean is reported.

Figure **??** displays the size distribution of the clusters for different N_l, for fixed N. The three panels display the distributions for different values of σ. For visualization purposes, the curves are normalised, since there are less available

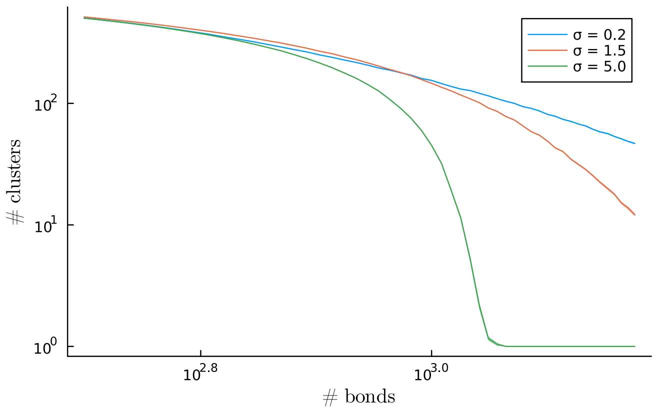

Figure 2: Average number of clusters and variation of the mean as a function of number of bonds, for $N = 50$ and varying σ, as indicated by the legend.

clusters of length N, than there are clusters of length 1 (isolated sites), which leading to underrepresentation.

The order parameter of choice is the magnetization $m = \frac{1}{N}\sum_i s_i$ of the spin chain. Figure **??** shows the magnetization of the spin chain, as the Metropolis algorithm advances, with $N = N_l = 4096, \sigma = 0.5$. The top panel shows the magnetization for 50 different starting spin seeds, and the bottom panel presents the averaging of ^{2} across said spin seeds in blue. The curve $y = 10^{-5.5}x^{1.7}$ is given in red, for reference.

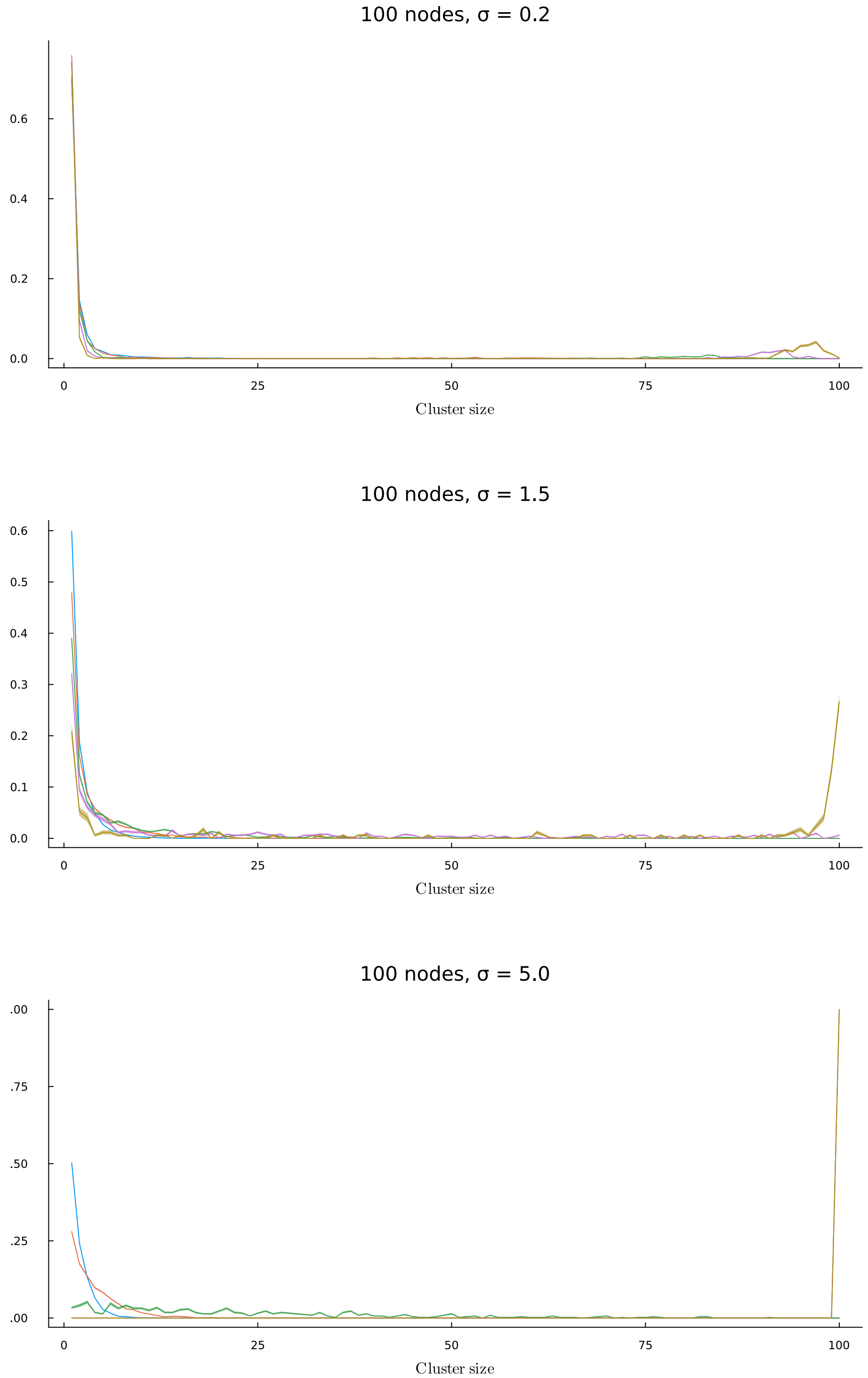

Figure 3: Normalized distributions of cluster sizes for fixed number of nodes $N = 50$ and varying σ values, as indicated at the top of each panel, for different number of bonds among the nodes, as indicated in the legend of the middle panel.

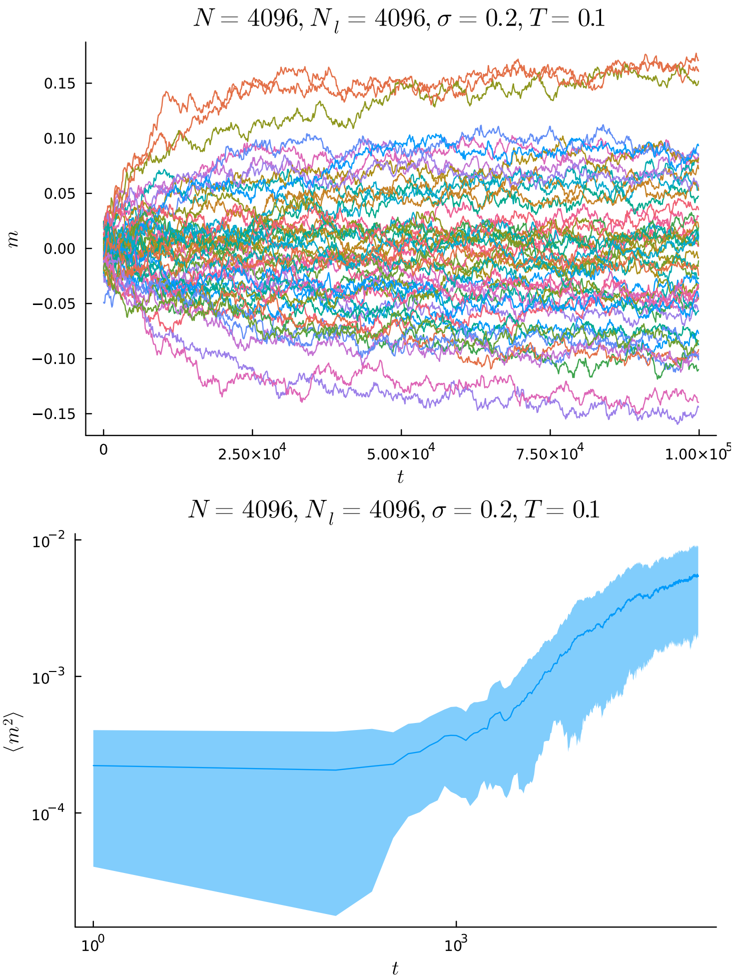

Figure 4: Magnetization μ of the 1-dimensional spin chain with periodic boundary conditions, as iterations of the Metropolis advance, for different spin seeds and the same J_{ij}. In the top panel, the magnetization of the spin chain for different starting seeds. In the bottom panel, the magnetization squared, averaged across seeds. The curve $y = 10^{-5.5}x^{1.7}$ is shown in red, for reference. Calculations here are done with $N = N_l = 4096, \sigma = 0.5$.

A Code snippets

A.1 Code for the generation of connection sets

```
using AliasTable

function generateConnectionsSet(N, N_l, sigma)

    distancesArray = 1:(N-1)
    chooseAT = AliasTable(ones(nPoints))
    distanceAliasTable = AliasTable(
                1 ./ distancesArray .^ (1+sigma)
                )

    # Stores connections
    connectionsSet = Set()

    # Stops if N_l is reached
    while len(connectionsSet) < N_l
        # Uniform distribution
        particle1Choice = rand(chooseAT, 1)[1]

        # Rolls integer m with probability P_m = m^(-(1+sigma))
        particle2Addition = rand(distanceAliasTable, 1)[1]
        particle2Choice = (particle2Addition + particle1Choice) % N

        # Stores the sampled connection. Since sets do not repeat elements
        # and are also unordered, if the connection already existed,
        # it will not be stored.
        push!(connectionsSet,
                Set([particle1Choice, particle2Choice]))
    end
    connectionsSet
end
```

A.2 Code for the clustering algorithms

```
function clusterIdentification(connectivityArray)
    nPoints = length(connectivityArray)
    visited = Set{Int}()
    clusters = []

    function dfs(node, cluster)
        stack = [ node ]
        while !isempty(stack)
            curr = pop!(stack)
            if !(curr in visited)
```

```
                push!(visited, curr)
                push!(cluster, curr)
                for neighbor in connectivityArray[curr]
                    if !(neighbor in visited)
                        push!(stack, neighbor)
                    end
                end
            end
        end
        cluster
    end

        for i in 1:nPoints
            if !(i in visited)
                cluster = Set{Int}()
                cluster = dfs(i, cluster)
                append!(clusters, [cluster])
            end
        end
        length(clusters)
end
```