A Monte Carlo Metropolis computation of the extended Mon-Jasnow algorithm

Fredrik Knapskog, *in collaboration with*, Herman Lileng Ottesen[a]

[a]*Institute for physics, Norwegian University of Science and Technology, N-7491 Trondheim, Norway.*

Abstract

The extended Mon-Jasnow algorithm for determining the free line energy tension between a torus lattice and a Klein bottle lattice was computed by Monte Carlo with the Metropolis method and the Ising model. The constant τ_0 was found to be within the range of 1,6 to 2,0.

1. Intoduction

When computing the average for a variable, the brute force method is often used. However, sometimes the number of configurations to average over becomes too large to handle. For example, a relatively small lattice of 100 sites with 2 configuration each results in $\approx 1{,}3 \cdot 10^{30}$ configurations in total. Other methods has to be used instead for solving such problems. Monte Carlo with the Metropolis algorithm averages just over the most likely configurations instead, which may give a rather representative result. Here this is done for computing the line tension as Mon and Jasnow did in 1984 [1], but for a torus lattice against a Klein bottle lattice instead. How will this differ from the computations they did?

2. Theory

The Ising model deals with spins on a square lattice interacting with only their four nearest neighbour spins. The model assumes the spins can only occupy two spin states, $\sigma = 1$ and $\sigma = -1$. The Hamiltonian, H, for a such lattice is the sum of all the spin interactions

$$H = \frac{J}{2} \sum_{\langle m,n \rangle} \sigma_m \sigma_n, \tag{1}$$

where J is the coupling strength, and m and n are the node addresses for a node and its nearest neighbours respectively. The factor of a half is to make up for double counting. The coupling strength is here set to 1 for simplicity. Line tension, τ, is defined as the free energy associated with the boundaries between domains of opposite spins. The Mon-Jasnow algorithm is a way of determining τ by looking at the difference in free energy, F, between two lattices with fixed edges. One lattice with two columns of fixed spins up

(++), and one of each (+-), both with periodic boundary conditions in the y-direction as a tube. The line tension free energy thus is

$$N_y\tau = F_{+-} - F_{++}, \tag{2}$$

with N_y as number of rows. The partition function then yields

$$Z = \exp(-\frac{F}{k_BT}) = \sum_{conf.} \exp(-\frac{H}{k_BT}), \tag{3}$$

where T denotes the temperature and Boltzmann constant, k_B, is set to 1. Equation (2) and (3) hence leads to

$$\tau = -\frac{T}{N_y} \ln \frac{Z_{+-}}{Z_{++}}. \tag{4}$$

Mon and Jasnow [1] observed the ratio between the partition functions could be written on the form

$$\frac{Z_{+-}}{Z_{++}} = \sum_{conf.} \frac{\exp(-\frac{H_{+-}-H_{++}}{T}) \exp(-\frac{H_{++}}{T})}{Z_{++}} = \langle \exp(-\frac{H_{+-} - H_{++}}{T}) \rangle_{++}. \tag{5}$$

For the extended Mon-Jasnow algorithm the fixed lattice boundary conditions in x-direction are exchanged for periodic boundary conditions. The lattice with positive edges is replaced by a torus, and the other replaced by a Klein bottle. The Klein bottle is just like the torus, except that when crossing its edges the next column is flipped. That is, the indices come in decreasing order and its coupling constant turns negative. For the torus there won't be any domain walls for low temperatures. For the Klein bottle however there will be one, even though there's just one domain present. The line tension free energy then becomes

$$\tau = \frac{T}{N_y} \ln \langle \exp(-\frac{H_k - H_t}{T} \rangle_t. \tag{6}$$

For temperatures close to the critical temperature,

$$T_c = \frac{2}{\ln(1+\sqrt{2})} \approx 2,2691, \tag{7}$$

the line tension free energy may, according to Lars Onsager [2], be given on the form

$$\tau = \tau_0 t^\mu \Sigma(N_y^{1/\nu} t), \tag{8}$$

for the temperatures below T_c. The constant $\tau_0 = 3,99...$ for a thermodynamical system, $\mu = \nu = 1$ and $\Sigma(z)$ is a scaling function which approaches 1 as $z \to \infty$ and $\Sigma(z) \sim z^{-\mu}$ when $z \to 0^+$. Lastly t is the reduced temperature

$$t = \frac{T_c - T}{T_c}. \tag{9}$$

3. Method

In order to find τ in equation (6), a Monte Carlo simulation with the Metropolis algorithm was used. As the amount of possible configurations in the lattice grows exponentially with sites, $N^2 = N_x^2 = N_y^2$, it becomes impossibles with today's technology to compute all the configurations. The Metropolis algorithm averages over the most likely configurations, without actually knowing the probability distribution. The partition function, Z and the Boltzmann factor helps finding the most probable states in the Metropolis algorithm,

$$\langle A \rangle = \frac{1}{Z} \sum_{conf.} A e^{\frac{-H}{k_B T}} \approx \frac{1}{m} \sum_{conf.} A. \tag{10}$$

The Metropolis algorithm is followed from the lecture notes in "TFY4235: Computational Physics" [3]:

- Start with an initial random lattice $x = x_0$.
- Draw a new state, $\tilde{x}$, according to the trial probability $t(\tilde{x}|x_n)$. The trial probability is a uniformly random chosen spin site on x_n flipped upside down.
- Accept the new state with probability $a(\tilde{x}|x_n)$ satisfying the detailed balance:
  - if accepted, set: $x_{n+1} \leftarrow \tilde{x}$
  - if rejected, set: $x_{n+1} \leftarrow x_n$

  The probability condition $a(\tilde{x}|x_n)$ is based on the energy difference from before and after switching the chosen spin site on the torus lattice:
  - $W = \exp(-\frac{H_t(\tilde{x}) - H_t(x_n)}{T})$
  - $r \in [0, 1]$ from a uniform distribution.
  - accept if $W > r$
  - reject otherwise.
- Repeat steps 2-3 until equilibrium is reached and enough data have been generated. Discard the non-equilibrium steps for the averaging.

For the function, W, the difference, $H_t(\tilde{x}) - H_t(x_n)$, is equal to twice the energy of the flipped spin site chosen in $\tilde{x}$. That is, $2\sigma_m \sum_n \sigma_n$, and it has only 5 different outcomes. Thus it is faster to store W as an array and extract the values from there instead of computing them, especially since this is the most repeated calculation. The 5 outcomes are $\exp(-J[-8, -4, 0, 4, 8]/T)$, and can be reached by the index $2 - \frac{1}{2}\sigma_m \sum_n \sigma_n$.

When it comes to sampling $\exp(-\frac{H_k - H_t}{T})$, the Klein bottle boundary condition can be positioned unconditionally at any row or column. Thus a $N_x \times N_y$ lattice can produce $N_x + N_y$ different values for $\exp(-\frac{H_k - H_t}{T})$. The difference $H_k - H_t$ is found from the column, or row, on each side of the boundary condition. H_t is computed by dotting the two columns with each other and multiplying by -1. Similar for H_k, but flipping one of the columns or rows, and not multiplying by -1 as its coupling constant is negative. For

the original Mon-Jasnow lattices, the difference was found by multiplying the sum of the next rightmost column by 2, as this was the only column making a difference.

To decide which lattice spin sites to be evaluated by $a(\tilde{x}|x_n)$, an array containing all the coordinates was initialised and then shuffled by numpy.random.shuffle. Furthermore the list is truncated after a uniformly random fraction of the list. When all the coordinates in the list are tested a sample is taken. Before the sampling starts this is repeated a few hundred times to get closer to equilibrium. Then for a constant number of samples, the standard deviation is taken. If this is the lowest standard deviation yet, the average is valued on those samples. The total average is also returned at the end. When deciding r, numpy.random.uniform was used.

For the scaling function the same approximation is used as in [1]

$$\Sigma(z) \cong 1 + Bz^{-\mu}, \tag{11}$$

with $B = 0,7323$. They state the model should work for $(N^{\frac{1}{\nu}}t)^{\mu} < 0,16$.

4. Results and discussion

The line tension in the original and extended Mon-Jasnow algorithm are shown in Figure 1 as a function of temperature. First of all, the extended version gives a better average as the lattice holds $N_x + N_y$ different values for Z, so the curve becomes a lot smoother. The extended one starts declining earlier than the original one, so the original is steeper than the extended. This can be a result of the two columns responsible for energy difference between the two lattices both vary in the extended algorithm, while only one of them in the original one. Another aspect is what Mon and Jasnow pointed out. The energy in the +- lattice is not necessarily in its most likely state as it differs so much from the ++ lattice.

Figure 4 in [1] is recreated quite well in Figure 2. The asymptotic line is lower for the extended Mon-Jasnow algorithm than for the original one as the original algorithm declines later and thus with a steeper slope. Both of them reach the asymptotic line at approximately $N = 8$ as Mon and Jasnow did. According to equation (8), the asymptotic line should be at $y = \tau_0$. Here $y \approx 1,9$ which is a lot lower than for the theoretical one, but as the system is a limited small lattice it is unreasonable to compare those values. Figure 3 shows how larger lattices decline later and have steeper slopes. For the largest N the phase transition really becomes apparent. For $N = 128$ not all the iterations found the global minimum, but the low sampling did not affect the other values as much.

For the lower temperatures the deviation method where the mean with lowest deviation was chosen. This worked well as the time for the system to reach equilibrium varies more for lower temperature. This is caused by the system falling into a local minimum with domain walls instead of the global minimum we're looking for. As the temperature rises, the increase in fluctuations breaks up these domains and the global minimum is reached faster. When the temperatures approaches the critical temperature however, the fluctuations leads to an increase in more likely states as the the probability for finding an up spin or down spin approaches equal. To get a good enough estimate more samples are needed, so the mean of all the values is used instead in this regime. When enough samples is collected is hard to tell before the simulation. What is certain at least, is that the needed amount of samples increases with N and T.

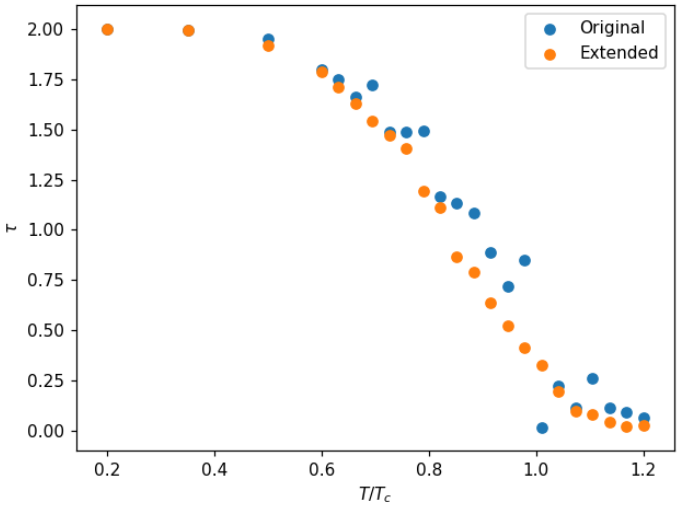

Figure 1: The line tension plotted as a function of temperature for $N = 12$. The orange dots are from the original Mon-Jasnow algorithm, and the blue ones from the extended one.

The logarithm of τ is plotted in Figure 4 for the logarithm of N, at the critical temperature. The analytical line is the logarithm of equation (8). As the argument for the scaling function approaches zero, the reduced temperatures even out. The logarithm thus takes the form

$$\log(\tau) = \log(\tau_0) - \frac{\mu}{\nu}\log(N). \tag{12}$$

The numerically computed values follows a straight line with the same slope as the analytical line, but is shifted away from it. The computations for lower N appears to be more stable than the ones for higher N, which emphasises that higher N needs more data to be sampled. The line followed by the numerical values is the same line as the asymptotic line as in Figure 2, as they are the same computation just visualised differently. As the asymptotic line isn't located at the theoretical τ_0, the numerically computed values here won't be on the analytical line either.

The master curve is defined as τ/t^μ, and is in Figure 5 plotted for $1/(N^{\frac{\mu}{\nu}}t^\mu)$. In the low temperature regime the scaling function goes as 1, so τ/t should go as the constant τ_0. The lines are constant for each lattice size N, but not as for the temperatures. The lines seems to converge towards $T = 0$, where τ_0 can be extracted as 2,0. In the high temperature regime the scaling function goes as $1/(Nt)$, so with $y = \tau/t$ and x $= 1/(tN)$ equation (8) yields $y = \tau_0 x$. Figure 6 shows the master curve for this regime, and all the dots fall on a single line with a slope equal to 1,6. The lines in Figure 2 and 4 both fall within this interval, so τ_0 for this system should be located between 1,6 and 2,0.

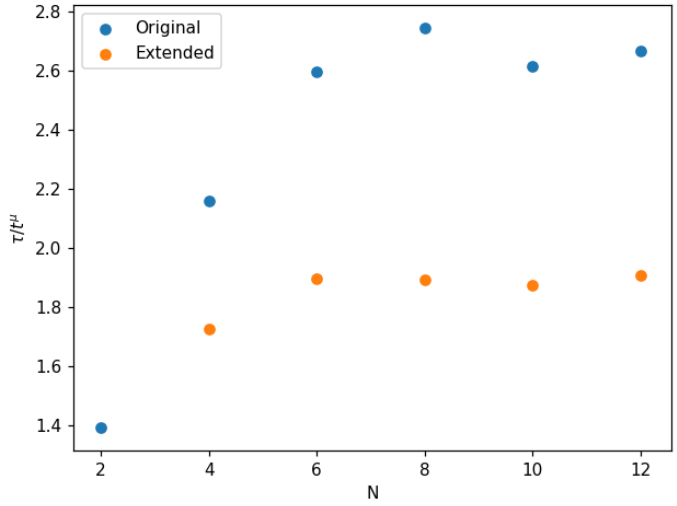

Figure 2: The line tension multiplied by lattice size plotted as a function of lattice size. The blue dots are from the original Mon-Jasnow algorithm, and the orange ones from the extended one.

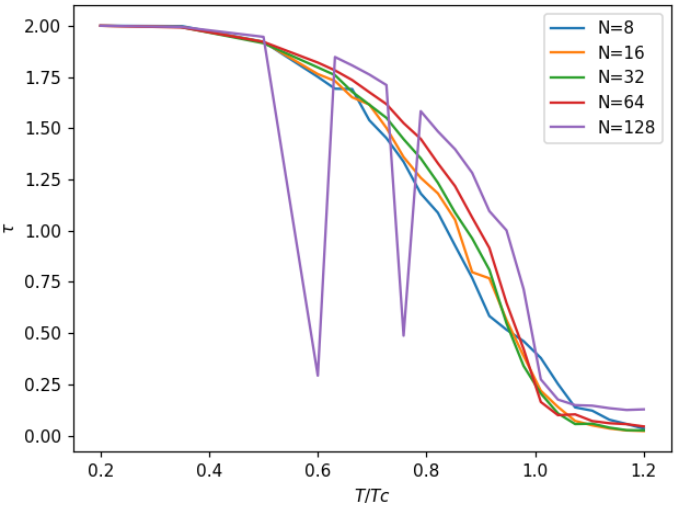

Figure 3: The line tension plotted against temperature for different lattice sizes.

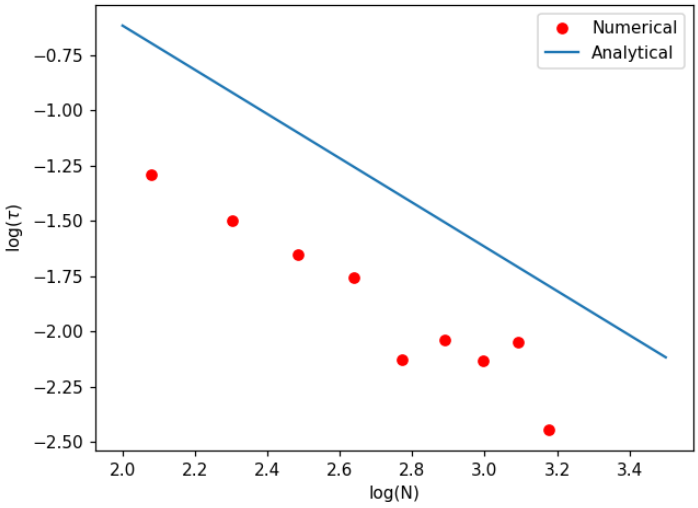

Figure 4: The logarithm of τ plotted for the logarithm of N. The dots are computed values, and the line is the analytical value from equation (12).

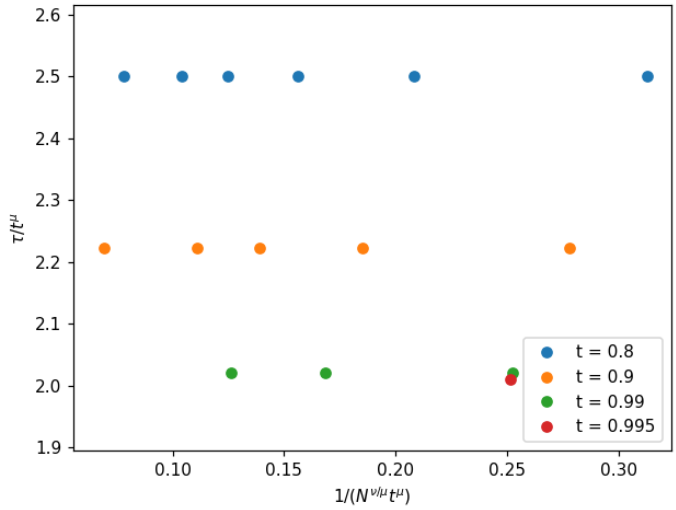

Figure 5: The master curves with fixed temperatures in the low temperature regime.

5. Conclusion

Mon and Jasnow [1] determined τ_0 to be 3,0, which is higher than the interval for the extended algorithm of $\tau_0 \approx 2$. This could be a result of the states in the +- lattice

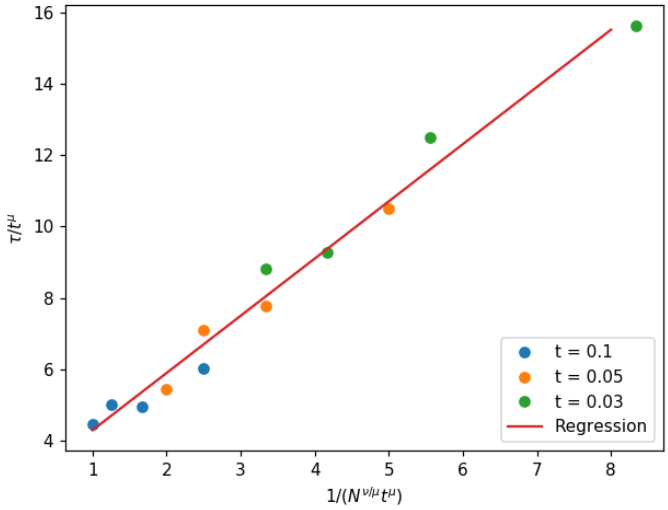

Figure 6: The master curves with fixed temperatures in the high temperature regime.

not being in its equilibrium when the ++ lattice is, especially for small lattices where the fixed boundary conditions contribute more. As both of the columns contributing to energy difference for the Klein bottle and torus varies, this could be also be a more representative result. For better approximation the lattice sizes should be larger, and thus more samples should be taken.

References

[1] K. K. Mon and David Jasnow. Direct calculation of interfacial tension for lattice models by the monte carlo method. *Phys. Rev. A*, 30:670–673, Jul 1984.

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

[3] I. Simonsen. Course slides tfy4235 : Computational physics. `http://web.phys.ntnu.no/~ingves/Teaching/TFY4235/Download/TFY4235_Slides_2020.pdf/`, 2020. (Downloaded: 05-06-2020).

```
#!/usr/bin/env python
# coding: utf-8

# In[2]:


import numpy as np
from matplotlib import pyplot as plt
from copy import copy
from numba import jit


# In[15]:


#Global variables
J = 1
Tc = 2/(np.log(1+np.sqrt(2)))

#Creating the initial lattice for the extended algorithm
def InitialLattice1(Nx, Ny):
    lattice = np.array(1-2*np.random.randint(0, 2, (Nx, Ny)), dtype = np.float64) #Random Nx b
    return lattice

#Creating the initial lattice for the original algorithm
def InitialLattice0(Nx, Ny):
    lattice = np.array(1-2*np.random.randint(0, 2, (Nx, Ny))) #Random Nx by Ny lattice
    plus = np.ones(Ny)
    lattice = np.insert(lattice, 0, plus, axis = 0) #Insert positive column left
    lattice = np.insert(lattice, Nx+1, plus, axis = 0) #Insert positive column right
    return lattice

#The Boltzmann factor for the torus lattice
@jit(nopython=True)
def W1(lattice, x, y, W, Nx, Ny): #Sums up neighbour attributes, periodic bc both directions
    deltaH = lattice[x,y]*(lattice[x-1,y]+lattice[(x+1)%(Nx-1),y]+lattice[x,(y+1)%Ny]+lattice[
    return  W[int(2+deltaH/2)] #Chose value from precomputed list

#The Boltzmann factor for the ++ lattice
@jit(nopython=True)
def W0(lattice, x, y, Ny, W): #Sums up neighbour attributes, periodic bc only in y-direction
    deltaH = lattice[x,y]*(lattice[x-1,y]+lattice[x+1,y]+lattice[x,(y+1)%Ny]+lattice[x,(y-1)%N
    return W[int(2+deltaH/2)]

#The partition function ratio for the extended algorithm
```

```
@jit(nopython=True)
def Z1(lattice, Nx, Ny, T):
    global J
    diff = 0
    for i in range(Nx): #Splits Klein bottle vertically
        diff += np.exp(-J*(np.sum(lattice[i]*lattice[(i+1)%Nx]) + np.sum(lattice[i]*lattice[(
    for i in range(Ny):#Splits Klein bottle horizontally
        diff += np.exp(-J*(np.sum(lattice[:,i]*lattice[:,(i+1)%Ny]) + np.sum(lattice[:,i]*latt
    return diff/(Nx+Ny) #Returns average

#The partition function ratio for the original algorithm
@jit(nopython=True)
def Z0(column, T):
    global J
    diff = 2*np.sum(column) #Considers only neighbour to rightmost column
    return np.exp(J*diff/T)

#Monte Carlo with Metropolis method
@jit(nopython=True)
def MonteCarlo(T, Nx, Ny, sweeps, lattice, W, start, subs):
    coords = np.array([(x, y) for y in range(Ny) for x in range(Nx)])
    Nc = len(coords)
    np.random.shuffle(coords) #Pick coords randomly from list of coords
    for i in range(start): #Gets closer to equilibrium
        for j in range(Nc):
            w = W1(lattice, coords[j,0], coords[j,1], W, Nx, Ny)
            if w > np.random.uniform(0.0, 1.0): #Tests for flip condition
                lattice[coords[j,0], coords[j,1]] *= -1 #Flips spin
    Z2 = 0
    Z3 = 0
    sigma1 = 100 #Start standard deviation
    for i in range(sweeps):
        Z = np.zeros(subs)
        for b in range(subs): #Subs is number of samples per deviation test (subsweeps)
            coords = np.array([(x, y) for y in range(Ny) for x in range(Nx)])
            np.random.shuffle(coords) #Pick coords randomly from list of coords
            coords = coords[:int(np.rint(Nc*np.random.uniform(0.0, 1.0)))]
            for j in range(Nc):
                w = W1(lattice, coords[j,0], coords[j,1], W, Nx, Ny)
                if w > np.random.uniform(0.0, 1.0): #Tests for flip condition
                    lattice[coords[j,0], coords[j,1]] *= -1 #Flips spin
            Z[b] = Z1(lattice, Nx, Ny, T) #Samples
        sigma = np.std(Z)
        if sigma < sigma1: #Deviation test, if smaller, update sigma and Z2
            Z2 = np.mean(Z)
            sigma1 = sigma
        Z3 += np.mean(Z)
```

```
    return Z2, Z3/sweeps #Returns average of sampled Z ratio

#Computes line tension
def main(T, Nx, Ny, sweeps, start, subs):
    global J
    W = np.exp(-1*np.array([-8, -4, 0, 4, 8])*J/T) #Computes possible Boltzmann factors
    lattice = InitialLattice1(Nx, Ny)
    return -T/Ny*np.log(MonteCarlo(T, Nx, Ny, sweeps, lattice, W, start, subs)) #Returns line
```