2D Ising model simulated with Metropolis algorithm

Zahra Akbari

Theory and Methods

The Ising model consists of a 2D square lattice of spins that can be either up or down, and interact only with their nearest neighbors. The energy of a spin configuration is given by:

$$E(s) = -J\sum_{i,j} s_i s_j - h\sum_i s_i$$

In this problem, Metropolis Mont Carlo algorithm is used to simulate the 2D Ising model. without any external magnetic field. the boundary condition is periodic. The following quantities are calculated using the following formulas:

$$\begin{aligned} E &= -\sum_{i,j} S_i S_j \\ M &= |\sum_i s_i| \\ C_v &= \frac{\beta^2 var(< E >)}{N^2} \\ \chi &= \frac{\beta var(< M >)}{N^2} \end{aligned}$$

where N is the grid size. β is also considered 1 so that J is the variable in this problem.

Results

The following results are obtained by simulating the 2d ising model with three grid sizes: 10,15,20. The function for implementing the metropolice algorithm is runned for 100000 time steps (using IsingMetroPolice function). At last Calculator function is used to run the metropolice function for 100 runs, then the desired variables are obtained by meaning over the 100 results. At last we graph the variables over different Js:

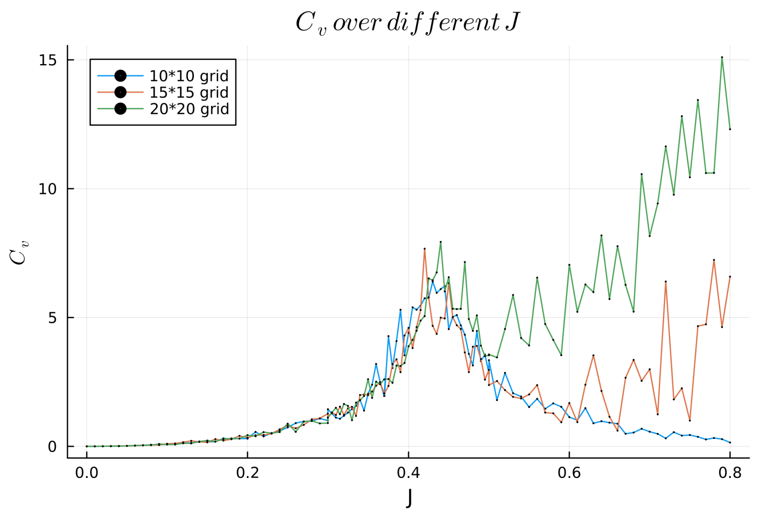


Figure 1: C_v with 100000 metropolice time steps

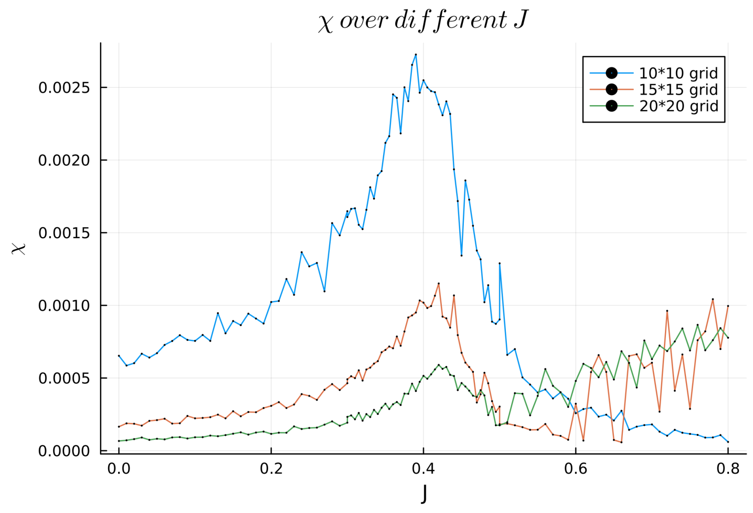


Figure 2: χ with 100000 metropolice time steps

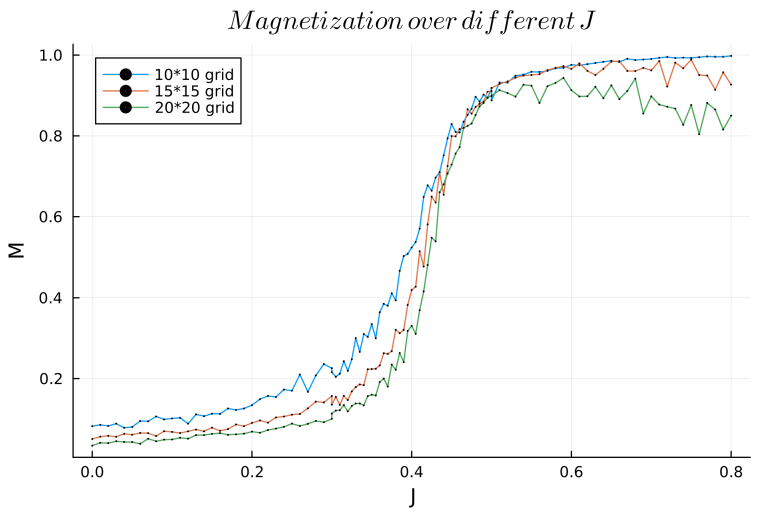


Figure 3: M with 100000 metropolice time steps

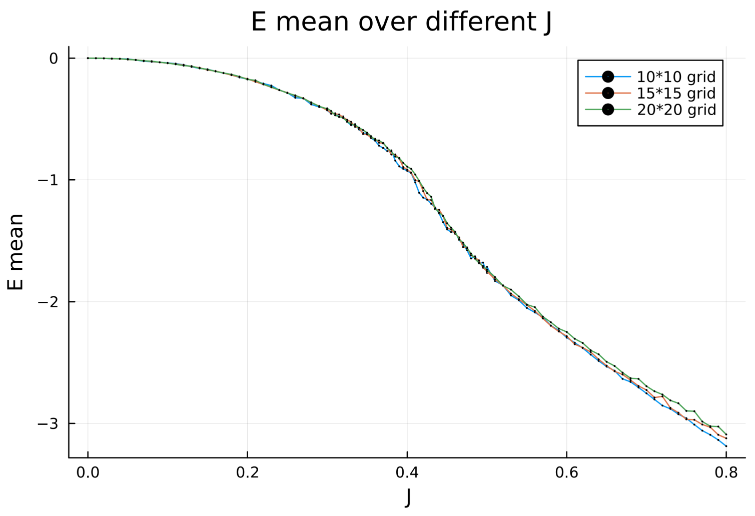


Figure 4: E with 100000 metropolice time steps

For getting better results for finding the critical exponents, the used metropolice time step didn't seem sufficent. Thus the simulation was repeated for 20*20 grid with 1 million metropolice time steps. As this incresed the run time, multithreading was needed. Using @threads macro in julia the code took about 130 minutes to compile.

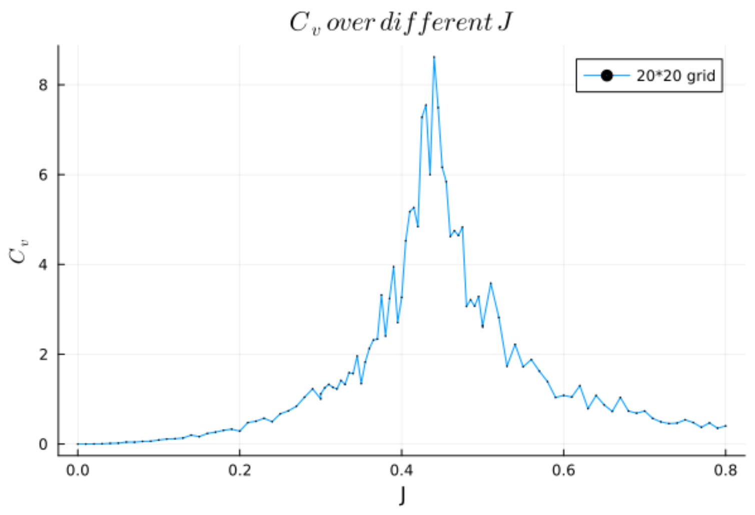


Figure 5: C_v with 1000000 metropolice time steps

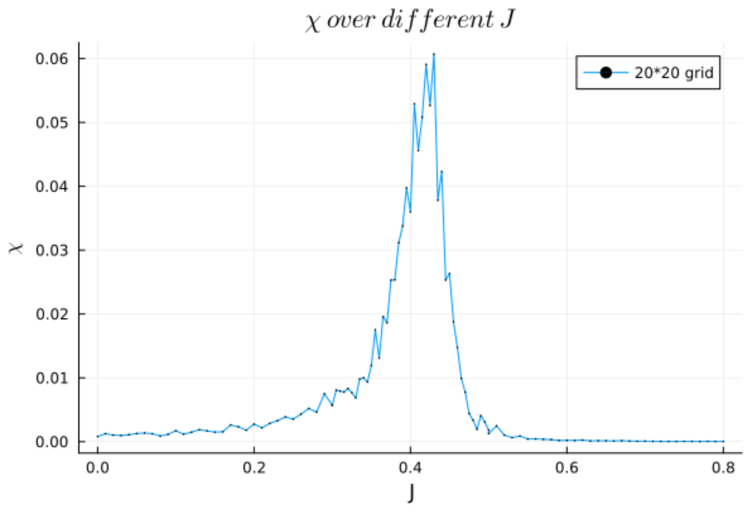


Figure 6: χ with 1000000 metropolice time steps

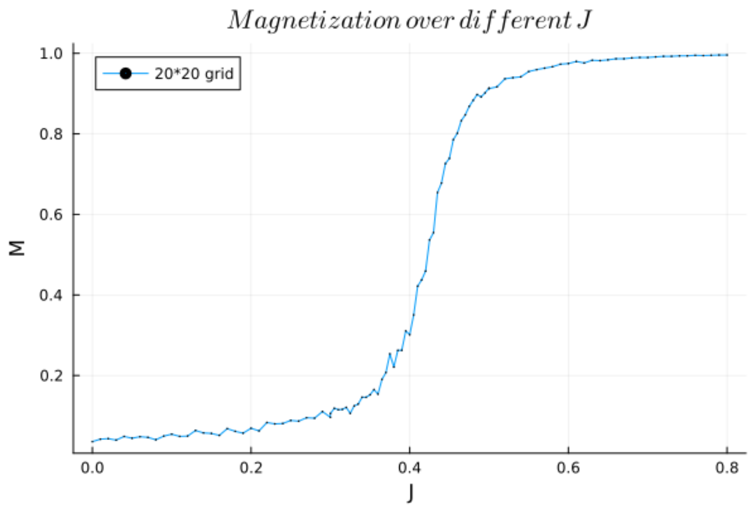


Figure 7: M with 1000000 metropolice time steps

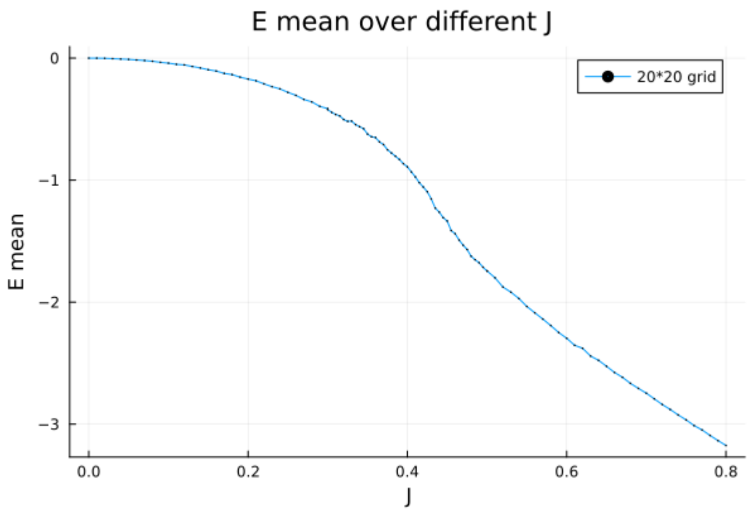


Figure 8: E with 1000000 metropolice time steps

Testing the IsingMetroPolice functions with a sample 100*100 grid is illustrated below. the gif is also available.

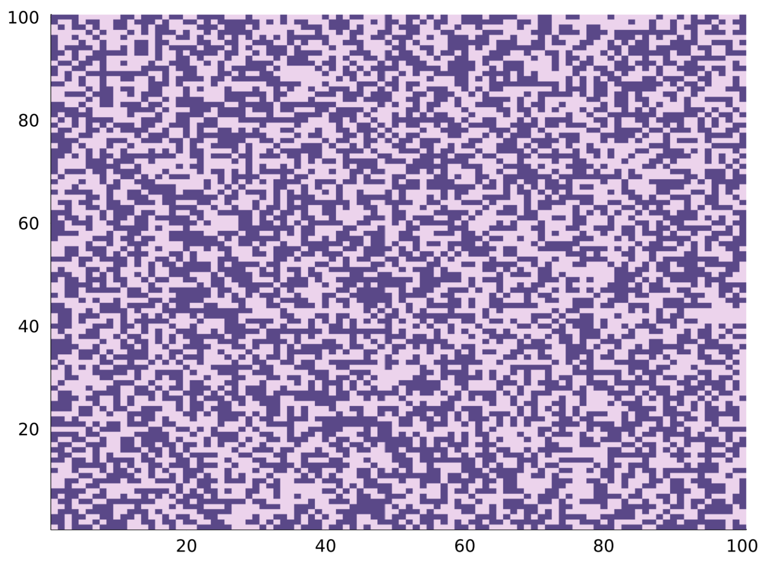


Figure 9: Random grid generated

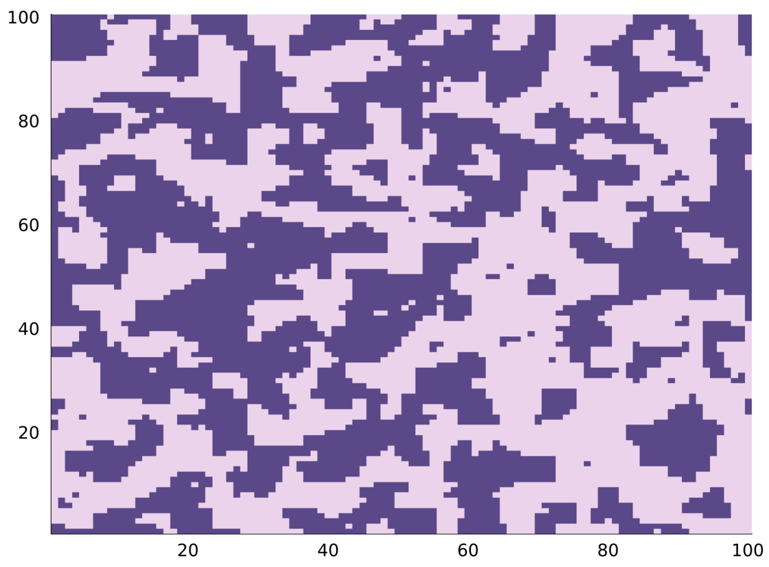


Figure 10: The grid after 50000 metropolice steps in T = 1

1 Statistics

In another file (named data) the saved data is used for finding the critical exponents. We fit a line to logarithm of the variables over log(j) for finding the following exponents:

Relation	Critical Exponent		$T_c(\infty)$
$C_V \sim c_0 \ln \lvert T - T_c \rvert$	c_0	-0.565	2.29
$\chi \sim \lvert T - T_c \rvert^{-\gamma}$	γ	2.05	2.35
$\langle \lvert m \rvert \rangle \sim \lvert T - T_c \rvert^{\beta}$	β	0.515	-

It should be noted that the line is fitted only to the critical interval just before the critical temperatre:

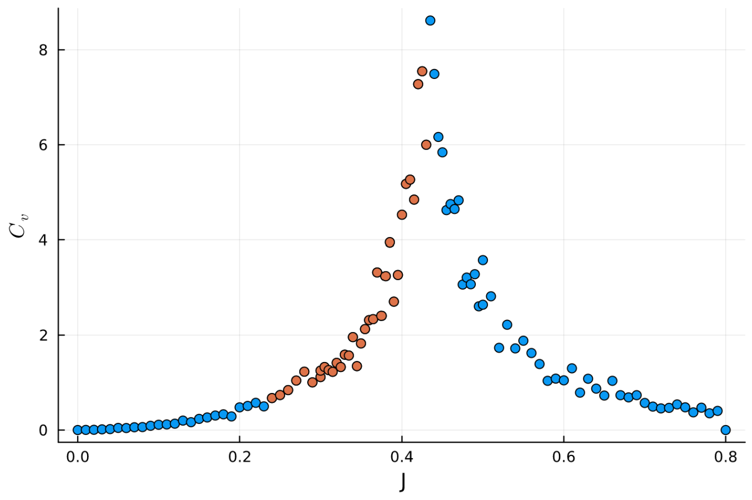


Figure 11: The interval where critical exponents are extracted from.

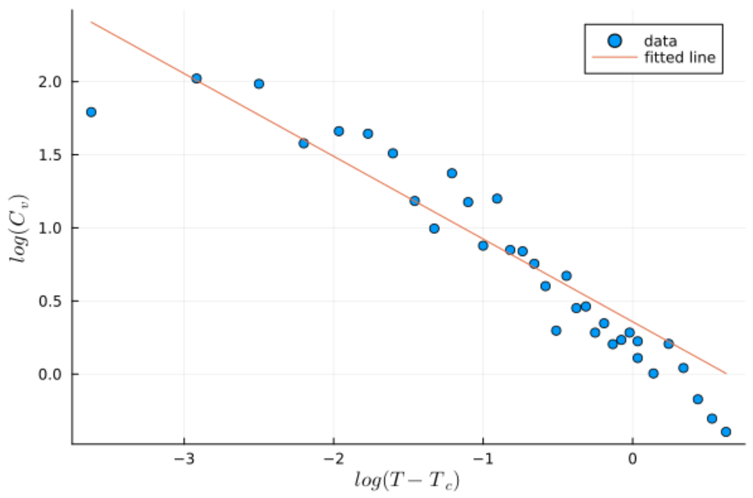


Figure 12: An example: log(Cv) is fitted over $log(T - T_c)$ in critical interval for finding critical exponents.

An attemp was also made into finding critical exponents for correlation length. The following code is the method used. Though the resulting data doesn't seem working. The following graphs are AutoCorrelation for half of a grid:

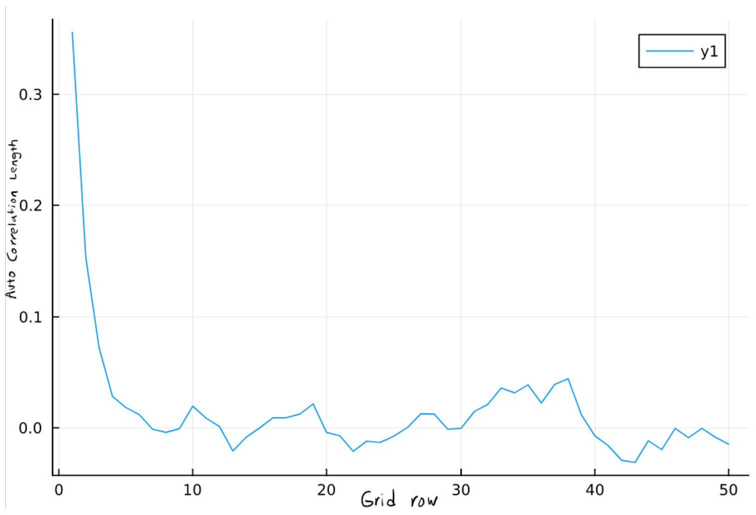


Figure 13: Autocorrelation length going through half of a grid with 256 rows.

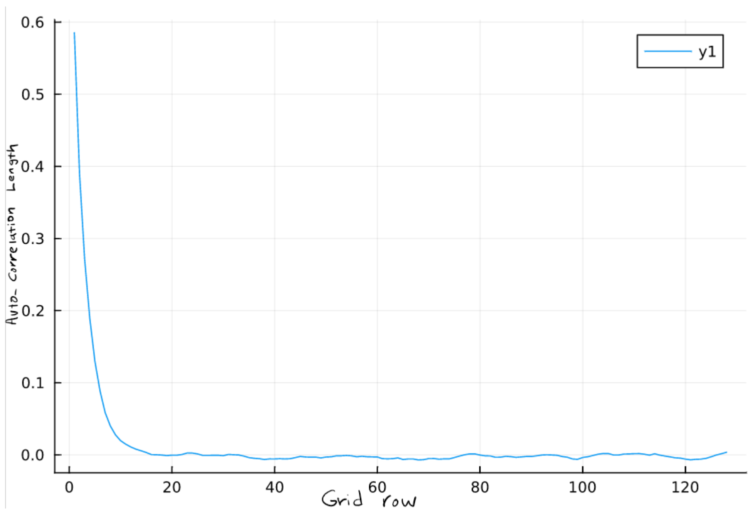
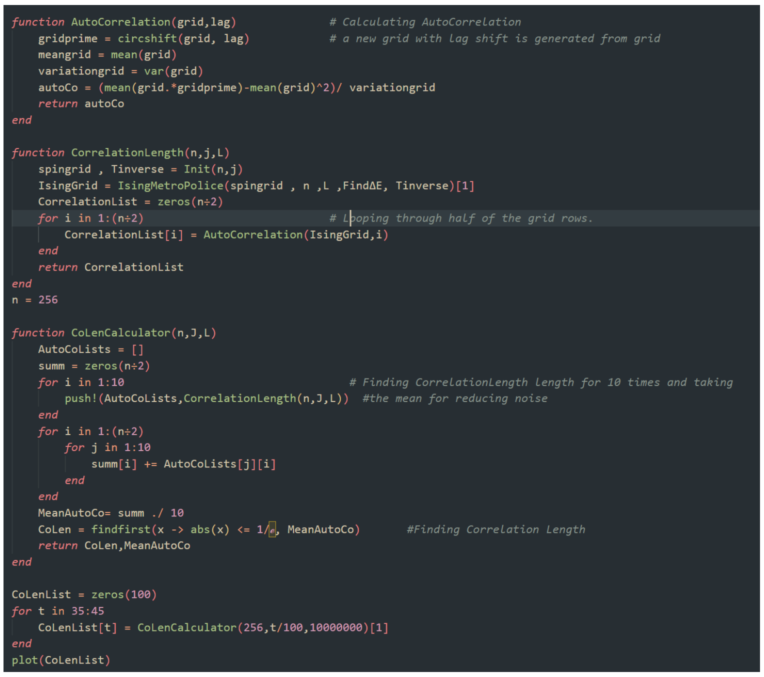


Figure 14: Autocorrelation length going through half of a grid with 100 rows



```
function AutoCorrelation(grid,lag)               # Calculating AutoCorrelation
    gridprime = circshift(grid, lag)             # a new grid with lag shift is generated from grid
    meangrid = mean(grid)
    variationgrid = var(grid)
    autoCo = (mean(grid.*gridprime)-mean(grid)^2)/ variationgrid
    return autoCo
end

function CorrelationLength(n,j,L)
    spingrid , Tinverse = Init(n,j)
    IsingGrid = IsingMetroPolice(spingrid , n ,L ,FindΔE, Tinverse)[1]
    CorrelationList = zeros(n÷2)
    for i in 1:(n÷2)                             # Looping through half of the grid rows.
        CorrelationList[i] = AutoCorrelation(IsingGrid,i)
    end
    return CorrelationList
end
n = 256

function CoLenCalculator(n,J,L)
    AutoCoLists = []
    summ = zeros(n÷2)
    for i in 1:10                                   # Finding CorrelationLength length for 10 times and taking
        push!(AutoCoLists,CorrelationLength(n,J,L))  #the mean for reducing noise
    end
    for i in 1:(n÷2)
        for j in 1:10
            summ[i] += AutoCoLists[j][i]
        end
    end
    MeanAutoCo= summ ./ 10
    CoLen = findfirst(x -> abs(x) <= 1/e, MeanAutoCo)       #Finding Correlation Length
    return CoLen,MeanAutoCo
end

CoLenList = zeros(100)
for t in 35:45
    CoLenList[t] = CoLenCalculator(256,t/100,10000000)[1]
end
plot(CoLenList)
```

Figure 15: The code intended to calculate correlation length