Computation Physics II 5640, Spring 2017

Josh Pond

February 2017

1 Master equation and Metropolis dynamics for classical Ising ring

The state of a 1-D classical Ising ring can be defined by an array of binary variable, $\{\sigma_i\}$, where we also assume periodic boundary conditions, $\sigma_{N+1} = \sigma_1$. The probability of a ring state transitioning to another state, differing by a single spin flip, is given by the Metropolis algorithm:

$$P\left(\{\sigma_i'\} \to \{\sigma_i\}\right) = \frac{1}{N} min \left\{ 1, e^{\frac{-(E\{\sigma_i\} - E\{\sigma_i'\})}{T}} \right\} \tag{1}$$

Where $E\{\sigma\}$ is given by the Hamiltonian of the system, which is defined as:

$$\hat{H} = -J \sum_{i=1}^{N} \sigma_i \sigma_{i+1} \tag{2}$$

From this we can construct a $2^N \times 2^N$ transition matrix $\mathbb{P}_{\{\sigma_i'\},\{\sigma_i\}} = P\left(\{\sigma_i'\} \to \{\sigma_i\}\right)$, an example of which, in the $N = 3$ case, is given by Table 1, where $r = e^{\frac{-(E\{\sigma_i\} - E\{\sigma_i'\})}{T}}$, the states of which can be seen graphically in figure 1.

	0	1	2	3	4	5	6	7
0	1 - r	1/3	1/3	1/3	0	0	0	0
1	r/3	0	0	0	1/3	1/3	0	0
2	r/3	0	0	0	1/3	0	1/3	0
3	r/3	0	0	0	0	1/3	1/3	0
4	0	1/3	1/3	0	0	0	0	r/3
5	0	1/3	0	1/3	0	0	0	r/3
6	0	0	1/3	1/3	0	0	0	r/3
7	0	0	0	0	1/3	1/3	1/3	1 - r

Table 1: Transition Matrix of N=3 Ising ring, value is probability of transitioning from column state to row state. Note, columns add to 1.

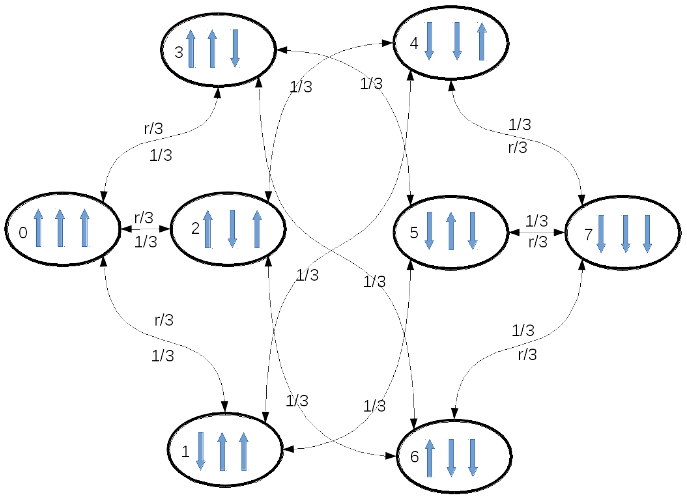

Figure 1: Transition map of $N = 3$ case. Probabilities above transition lines are for transitions to the right, below are to the left. Probabilities on the lines are for both directions.

The eigenvectors, $\mathbb{U}_\alpha$, and eigenvalues, λ_α, of $\mathbb{P}$ can be used to obtain the Markovian dynamics of the system. This can be seen by taking $\mathbb{U}_0$, the eigenvector corresponding to the largest eigenvalue, λ_0, which is always equal to one. $\mathbb{U}_0$ is the probability distribution of the steady state of the system when normalized by the sum of the vector. We can show this by taking the zero element, $\mathbb{U}_0\{0\}$, corresponding to the fully polarized state and plotting its normalized value over the analytical solution, given by equation 3, which can be seen in figure 2.

$$\begin{gathered}\frac{\mathbb{U}_0\{0\}(T)}{\sum_i \mathbb{U}_0\{i\}(T)} = \frac{exp(-NJ/T)}{Z_N} \\ Z_N = (2\cosh\beta J)^N + (2\sinh\beta J)^N\end{gathered} \tag{3}$$

Another behavior that can be determined in this way is the relaxation time, τ, of the system. τ is how many steps it takes the system to reach the steady state, and it's given by the second largest eigenvalue, λ_1, in such $\tau(T) = \frac{-1}{ln\lambda_1(T)}$. The temperature dependence of which can be seen for several values of N in figure 3.

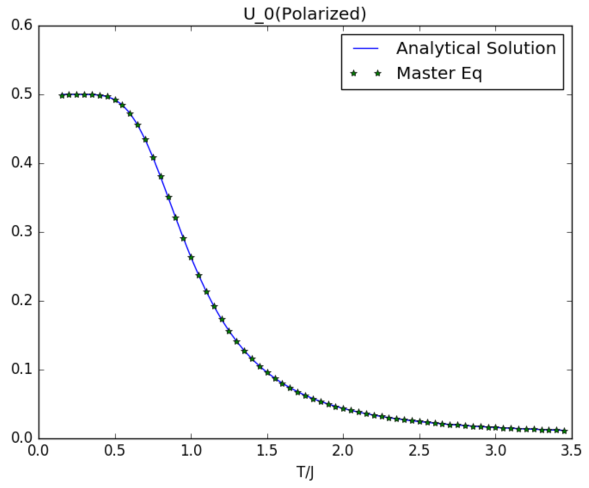

Figure 2: Plot Master Equation over the Analytical Solution, for $N = 10$ case.

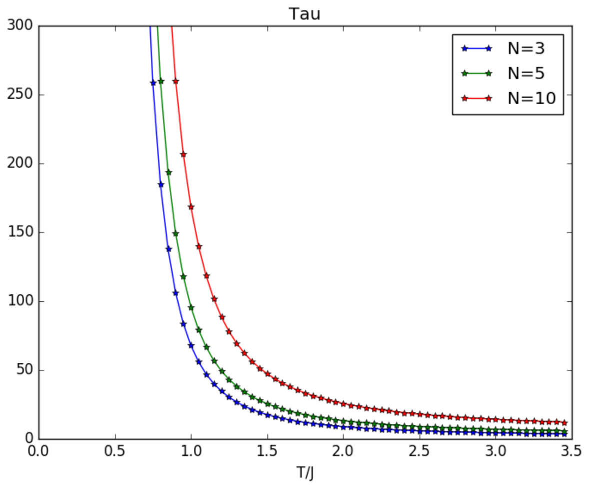

Figure 3: Plot of the temperature dependence of τ for several N.

2 Appendix: Source Code

```
#! /usr/bin/python
import numpy as np
import itertools
from matplotlib import pyplot as plt

def get_states(N):
    out = []
    for i in itertools.combinations_with_replacement([0,1],N):
        for j in itertools.permutations(i,N):
            o = list(j)
            o.reverse()
            if not o in out:
                out.append(o)
    return out #returns list of all possible states of N spins

def get_E(state):
    ran = range(len(state))
    out = 0.0
    for i in ran:
        if i != ran[-1]:
            if state[i] == state[i+1]:
                out += -1.0
            else:
                out += 1.0
        else:
            if state[i] == state[0]:
                out += -1.0
            else:
                out += 1.0
    return out

def get_off_diag_P(state1,state2,N,T):
    flips=0
    for i in range(len(state1)):
        if not state1[i] == state2[i]:
            flips +=1
    if not flips == 1:
        return 0.0
    else:
        return min([1.0,np.e**(-(get_E(state1) - get_E(state2))/T)])/float(N)

def get_pi_hat(states,N,T):
    ret = np.zeros(shape=(2**N,2**N))
    for i in range(2**N):
        for j in range(2**N):
            if not i == j:
                ret[i,j] = get_off_diag_P(states[i],states[j],N,T)
    for k in range(2**N):
        if ret[:,k].sum() != 1.0:
            ret[k,k] = 1.0 - ret[:,k].sum()
    return ret

def analytical_Sol(T,N):
    Z  =((2.0*np.cosh(1.0/T))**N + (2.0*np.sinh(1.0/T))**N)
    return (np.e**(float(N)/T))/Z

if __name__ == '__main__':
    for N in [10]:
        states = get_states(N)
        T=np.arange(0.15,3.5,0.05)
        Y = []
        y = []
        for t in T:
            pi_hat = get_pi_hat(states,N,t)
            w,v = np.linalg.eig(pi_hat)
            w = list(w)
            o = v[:,w.index(max(w))]
            Y.append(o[-1]/o.sum())
        y = [analytical_Sol(t,N) for t in T]
        plt.plot(T,y,label="Analytical Solution")
        plt.plot(T,Y,linestyle='',marker='*',label="Master Eq")
    plt.title("U_0(Polarized)")
    plt.xlabel("T/J")
    plt.legend()
    plt.show()
    for N in [3,5,10]:
        states = get_states(N)
        T=np.arange(0.15,3.5,0.05)
        Y = []
        for t in T:
            pi_hat = get_pi_hat(states,N,t)
            w,v = np.linalg.eig(pi_hat)
            w = list(w)
            w.sort()
            o = v[:,w.index(max(w))]
            Y.append(-1.0/(np.log(w[-2])))
        plt.plot(T,Y,marker='*',label="N="+str(N))
    plt.title("Tau")
    plt.xlabel("T/J")
    plt.ylim(0.0,300)
    plt.legend()
    plt.show()
```