Computational Physics II 5640, Spring 2017

Josh Pond

February 2017

1 Exact diagonalization of quantum Ising ring

The 1-D quantum Ising ring differs from the 1-D classical Ising ring by adding uncertainty to the direction of the spinner. Therefore the 1-D quantum Ising ring can be mapped to a 2-D classical Ising ring, with spin in the x direction being one and z the other, and with sufficiently small numbers of spinners, N, the system can be exactly solved by exact diagonalization.

To start, the Hamiltonian of the system is defined by:

$$\hat{H} = -\sum_{i=1}^{N} \hat{\sigma}_i^z \hat{\sigma}_{i+1}^z + \frac{h}{J} \sum_{i=1}^{N} \hat{\sigma}_i^x \tag{1}$$

Where $\hat{\sigma}^x$ and $\hat{\sigma}^z$ are the first and third Pauli matrices, respectively, and $\frac{h}{J}$ is a relative parameter of the system. We also assume periodic boundary conditions, $\sigma_{N+1} = \sigma_1$.

We construct our $2^N \times 2^N$ Hamiltonian by taking all possible states and calculating $\hat{H}$ at each matrix sight. We assume $\hat{H}$ to be 0 for any two states that differ by more than one spinner flip. We then diagonalize the $\hat{H}$ matrix using Python's numpy.linalg package, which is based on LAPACK. The eigenvalues, $\{\epsilon\}$, can then be used to calculate the energy density E, and the heat capacity C,

$$\begin{aligned} E &= \frac{\langle \hat{H} \rangle}{N} = \frac{-1}{N} \frac{\partial lnZ}{\partial \beta} \\ C &= \frac{\langle \hat{H}^2 \rangle - \langle \hat{H} \rangle^2}{NT^2} = \frac{1}{T} \frac{\partial^2 lnZ}{\partial \beta^2} \end{aligned} \tag{2}$$

where T is temperature, β is $\frac{1}{T}$, and Z is the partition function, given by:

$$Z = \sum_{m=1}^{2^N} e^{-\beta \epsilon_m} \tag{3}$$

Plots of E and C as functions of temperature for several values of $\frac{h}{J}$, and $N = 10$ can be found in figure 1.

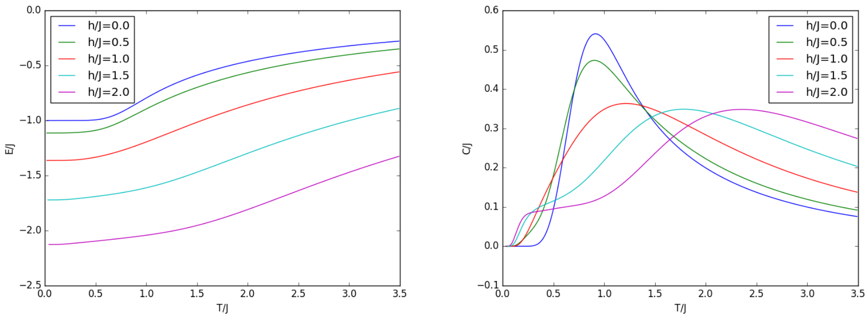

Figure 1: Plots of E and C as functions of temperature, T. $N = 10$.

1.1 Appendix: Simplification of E and C for plotting

The equation for E can be simplified by:

$$\begin{aligned} E &= \frac{-1}{N}\frac{\partial lnZ}{\partial\beta} \\ &= \frac{-1}{N}\frac{\partial}{\partial\beta}ln\left(\sum_m e^{-\beta\epsilon_m}\right) \\ &= \frac{-1}{N}\frac{1}{\sum_m e^{-\beta\epsilon_m}}\sum_m -\epsilon_m e^{-\beta\epsilon_m} \\ &= \frac{1}{N}\frac{\sum_m \epsilon_m e^{-\beta\epsilon_m}}{\sum_m e^{-\beta\epsilon_m}} \end{aligned} \tag{4}$$

The equation for C can be simplified by:

$$\begin{aligned} C &= \frac{1}{T^2}\frac{\partial^2 lnZ}{\partial\beta^2} \\ &= \frac{1}{T^2}\frac{\partial}{\partial\beta}E \\ &= \frac{1}{T^2}\frac{\partial}{\partial\beta}\frac{1}{N}\frac{\sum_m \epsilon_m e^{-\beta\epsilon_m}}{\sum_m e^{-\beta\epsilon_m}} \\ &= \frac{1}{N}\frac{1}{T^2}\left(\frac{\sum_m \epsilon_m e^{-\beta\epsilon_m}}{\left(\sum_m e^{-\beta\epsilon_m}\right)^2}\sum_m -\epsilon_m e^{-\beta\epsilon_m} + \frac{\sum_m \epsilon_m^2 e^{-\beta\epsilon_m}}{\sum_m e^{-\beta\epsilon_m}}\right) \\ &= \frac{1}{N}\frac{1}{T^2}\left(\frac{-\left(\sum_m \epsilon_m e^{-\beta\epsilon_m}\right)^2}{\left(\sum_m e^{-\beta\epsilon_m}\right)^2} + \frac{\sum_m \epsilon_m^2 e^{-\beta\epsilon_m}}{\sum_m e^{-\beta\epsilon_m}}\right) \end{aligned} \tag{5}$$

Equations 4 and 5 were used to make the plots in figure 1.

1.2 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)
            if not o in out:
                out.append(o)
    return out #returns list of all possible states of N spins

def get_diag_H(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_H(state1,state2,hoJ):
    flips = 0
    out = 0.0
    for i in range(len(state1)):
        if state1[i] == state2[i]:
            flips +=1
    if flips > 1:
        return 0.0
    else:
        return hoJ

def get_H_hat(N,hoJ):
    ret = np.zeros(shape=(2**N,2**N))
    states = get_states(N)
```

```python
    for i in range(2**N):
        for j in range(2**N):
            if i == j:
                ret[i,j] = get_diag_H(states[i])
            else:
                ret[i,j] =\
                get_off_diag_H(states[i],states[j],hoJ)
    return ret

def E(e_vals,T,N):
    topTerm = 0.0
    botTerm = 0.0
    for em in e_vals:
        topTerm += em*np.e**(-em/T)
        botTerm += np.e**(-em/T)
    return (1.0/float(N))*(topTerm/botTerm)

def C(e_vals,T,N):
    topTerm1 = 0.0
    topTerm2 = 0.0
    botTerm = 0.0
    for em in e_vals:
        topTerm1 += em*np.e**(-em/T)
        topTerm2 += em*em*np.e**(-em/T)
        botTerm += np.e**(-em/T)
    return (1.0/N)*(1.0/(T*T))*\
    (-((topTerm1**2.0)/(botTerm**2.0)) +\
    (topTerm2/botTerm))

if __name__ == '__main__':
    N=10
    x = np.arange(0.01,3.5,0.01)
    y = np.zeros(shape=(2,5,len(x)))
    hoJ = np.arange(0.0,2.5,.5)
    for j in range(len(hoJ)):
        print hoJ[j]
        H_hat = get_H_hat(N,hoJ[j])
        w,v = np.linalg.eig(H_hat)
        for T in range(len(x)):
            y[0,j,T] = E(w,x[T],N)
            y[1,j,T] = C(w,x[T],N)
    for i in range(5):
        plt.plot(x,y[0,i,:],label="h/J="+str(i*0.5))
        plt.legend(loc=2)
        plt.xlabel("T/J")
        plt.ylabel("E/J")
```

```python
plt.show()

for i in range(5):
    plt.plot(x,y[1,i,:], label="h/J="+str(i*0.5))
    plt.legend(loc=1)
    plt.xlabel("T/J")
    plt.ylabel("C/J")
plt.show()
```