Report to FOPRA32: Tensor Network Simulation

Ziao Liu

June 25, 2022

Abstract

In this experiment, I mainly finished two tasks. The first one is that, with the MPS representation of states, I use TEBD algorithm to compute the phase diagram of the ground state of the quantum Ising model. The other one is that,I apply the time evolution and compute the dynamic structure factor.

1 Introduction

When computing the entanglement entropy and entanglement spectra in many-body system, we could take the bipartition of the Hilbert space into consideration, i.e. $\mathcal{H} = \mathcal{H}_A \otimes \mathcal{H}_B$. Then we could write the Schmidt decomposition of the state $|\Psi\rangle \in \mathcal{H}$:

$$|\Psi\rangle = \sum_{\alpha} \Lambda_\alpha \, |\alpha\rangle_A \otimes |\alpha\rangle_B$$

Using this scenario, we could write a N-qubit state of an 1-D Ising model as

$$|\psi\rangle = \sum_{j_1, j_2, \cdots j_N} \psi_{j_1 j_2 \cdots j_N} \, |j_1, j_2, \cdots j_N\rangle$$

Then we could represent this state as matrix-product state (MPS):

$$\begin{aligned} |\psi\rangle &= \sum_{j_1, j_2, \cdots j_N} \sum_{\alpha_1, \alpha_2, \cdots \alpha_N} M^{[1]j_1}_{\alpha_1 \alpha_2} M^{[2]j_2}_{\alpha_2 \alpha_3} \cdots M^{[N]j_N}_{\alpha_N \alpha_N + 1} \, |j_1, j_2, \cdots j_N\rangle \\ &\equiv \sum_{j_1, j_2, \cdots j_N} M^{[1]j_1} M^{[2]j_2} \ldots M^{[N]j_N} \, |j_1, j_2, \cdots j_N\rangle \end{aligned}$$

Here the matrix M could be represent by

$$M^{[n]j_n} = \left(\phi^{[n]}_{j_n} \right)$$

We could replace

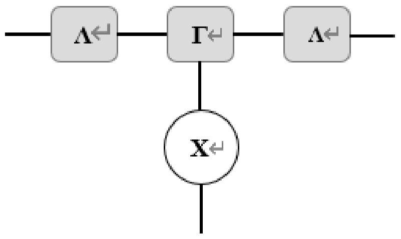
$$M^{[n]j_n} \to \widetilde{M}^{[n]j_n} := M^{[n]j_n} X^{-1}, M^{[n+1]j_{n+1}} \to \widetilde{M}^{[n]j_n} := X M^{[n+1]j_{n+1}}$$

Without loss of generallity, we could apply Sigle Value Dicomposition(SVD) to the MPS, and the result could be written as

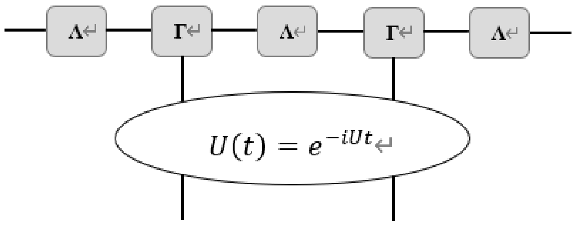
$$|\psi\rangle = \sum_{j_1,\cdots,j_N} \Lambda^{[1]}\Gamma^{[1]j_1}\Lambda^{[2]}\Gamma^{[2]j_2}\Lambda^{[3]}\cdots\Lambda^{[N]}\Gamma^{[N]j_N}\Lambda^{[N+1]}\,|j_1,\cdots,j_N\rangle$$

Here, $\Gamma^{[n]j_n}$is the matrix composing the core of a site, and $\Lambda^{[n]}$ is the matrix stored the infomation of Schmidt value for Schmidt Decomposition in the diagonoal matrix.

For this MPS, we could apply all unitary operators,such as X(Y,Z)-gate or time evolving operator, on every site or every bond.



(a) X-gate acts on sigle site



(b) U(t) acts on bond

Figure 1: The MPS form would help us in computing the result when applying operators to the state.

With this tensor network, we could compute several other Physics attributes easily, for example, the correlation between sites with the center point, the entanglement entropy of each bond in time evolving,and the dynamic structure factor.

2 Implementation of the TEBD algorithm

2.1 spin-up MPS

To get the all spin-up initial MPS, we could just import the $a_mps.py$ file and apply the code in that file to initiate MPS.

```
def init_spinup_MPS(L):
    """Return a product state with all spins up as an MPS"""
    B = np.zeros([1, 2, 1], np.float)
    B[0, 0, 0] = 1.
    S = np.ones([1], np.float)
    Bs = [B.copy() for i in range(L)]
    Ss = [S.copy() for i in range(L)]
    return MPS(Bs, Ss)

def q_ab(m, op):
    print(m.site_expectation_value(op))
    return

m = a_mps.init_spinup_MPS(L)
print('for spin up initial MPS')
print('the site expectation values are')
print('op=sz')
q_ab(m, sz)
print('op=sx')
q_ab(m, sx)
```

If we apply the Z-gate, the expectation values for each sites are all $\langle\sigma_Z\rangle = \langle\uparrow|\sigma_Z|\uparrow\rangle = 1$, and if X-gate is applied, the expectation values are all $\langle\sigma_X\rangle = \langle\uparrow|\sigma_X|\uparrow\rangle = 0$.

2.2 spin-right MPS

We could modify the code above to initiate all-spin-right MPS.

```
def init_spinright_MPS(L):
    B = np.zeros([1, 2, 1], np.float)
    B[0, 0, 0] = 1./np.sqrt(2.)
    B[0, 1, 0] = 1./np.sqrt(2.)
    S = np.ones([1], np.float)
    Bs = [B.copy() for i in range(L)]
    Ss = [S.copy() for i in range(L)]
    return MPS(Bs, Ss)

m1 = init_spinright_MPS(L)
print('for spin right initial MPS')
print('the site expectation values are')
print('op=sz')
q_ab(m1, sz)
print('op=sx')
q_ab(m1, sx)
```

Obviously, the expectation values of X-gate are $\langle\sigma_X\rangle = \langle\rightarrow|\sigma_X|\rightarrow\rangle = 1$, while those of Z-gate are $\langle\sigma_Z\rangle = \langle\rightarrow|\sigma_Z|\rightarrow\rangle = 0$.

2.3 Energy for two product states

According to the code provided by teaching team of this experiment course, the class TFIModel contains an initial function, a Hamiltonian bond operator initiation function, and a function calculating energy.With running this code block, we could get the energy values for different state setup and different Hamiltonian setup:

```
J=1
for g in [0.5,1.,1.5]:
    ising = TFIModel(L, J, g)
    print('when g={}'.format(g))
    print('spinup energy is {}'.format(ising.energy(m)))
    print('spinright energy is {}'.format(ising.energy(m1)))
```

The corresponding output is:

```
when g=0.5
spinup energy is -7.0
spinright energy is -12.999999999999996
when g=1.0
spinup energy is -14.0
spinright energy is -12.999999999999996
when g=1.5
spinup energy is -21.0
spinright energy is -12.999999999999996
```

2.4 Longitudinal field

To append the longitudinal field term to the Hamltonian, we should add the chain of the same operator as the neighboring interaction and times this chain with longitudinal factor.Therefore, the Hamiltonian we used to create the ground state should be

$$H = -J\sum_n \sigma_n^x\sigma_{n+1}^x - g\sum_n \sigma_n^z - h\sum_n \sigma_n^x$$

Then what we need is just modifying the function in the file b_model.py as belowing:

```
def init_H_bonds(self):
    """Initialize `H_bonds` hamiltonian.
        Called by __init__()."""
    sx, sz, id = self.sigmax, self.sigmaz, self.id
    d = self.d
    H_list = []
    for i in range(self.L - 1):
        gL = gR = 0.5 * self.g
        hL = hR = 0.5 * self.h
        if i == 0: # first bond
            gL = self.g
            hL = self.h
        if i + 1 == self.L - 1: # last bond
            gR = self.g
            hR = self.h
        H_bond = -self.J * np.kron(sx, sx)
                  - gL * np.kron(sz, id)
                  - gR * np.kron(id, sz)
                  - hL * np.kron(sx, id)
                  - hR * np.kron(id, sx)
        # H_bond has legs ``i, j, i*, j*``
        H_list.append(np.reshape(H_bond, [d, d, d, d]))
    self.H_bonds = H_list
```

2.5 Ground state with TEBD

Calling the file c_tebd.py directly, we could obtain the result

```
finite TEBD, (imaginary time evolution)
L=14, J=1.0, g=0.10
dt = 0.10000: E = -13.0400250378130
dt = 0.01000: E = -13.0400278909074
dt = 0.00100: E = -13.0400281716624
dt = 0.00010: E = -13.0400281997531
dt = 0.00001: E = -13.0400282025618
final bond dimensions:  [2, 4, 8, 10, 10,
                10, 10, 10, 10, 10, 8, 4, 2]
```

With changing different dt scales step by step, we could calculate the ground state effiently, otherwise it would take longer time in computing. Initially, the bond dimensions should satisfy $dim(R_n) = 2^{L-(n-1)}$. However, as we set a standard for ignoring neglectable superfluous bond indices, the final bond dimensions are small enough to handle with, and the error is acceptable.

3 Ground state Phase diagram

3.1 Correlations function

When computing the correlation $\langle\psi|X_iY_j|\psi\rangle$, we could regard the sites out the range[i,j] trivial,and then we could ignore them and just compute the operation within the range[i,j]. The function could be constructed as below:

```
def correlation(op_x, op_y, psi, i, j):
    theta_i = psi.get_theta1(i) # vL, i, vR
    op_theta_i = np.tensordot(op_x, theta_i, axes=[1, 1])
        # i, i*; vL, i, vR -> i, vL, vR

    result = np.tensordot(theta_i.conj(), op_theta_i,
                                     [[0,1],[1,0]])
           # vL*, i*, vR*; i, vL, vR ->vR*, vR

    for index in range(i+1,j):
        bj = psi.Bs[index]  # vL, i, vR
        cell = np.tensordot(bj.conj(), bj, [[1],[1]])
        result = np.tensordot(result, cell, [[0,1],[0,2]])
# vR*, vR
    Bj = psi.Bs[j]  #vL, i, vR
    op_Bj = np.tensordot(op_y, Bj, axes=[1, 1]) #i, vL, vR
    final_cell = np.tensordot(Bj.conj(), op_Bj,
                         [[1,2],[0,2]])  # vL*, vL
    #print(result.shape, final_cell.shape)
    result = np.tensordot(result, final_cell, [[0,1],[0,1]])
    #print(result)
    return result
```

3.2 Run TEBD for L=30 ising model

In the code to solve this question, we define all needed variable first, and then traverse all site position.

```
def plot_cor(i, g, L):
    j_list = np.arange(i+1, L) - i
    _, psi, _=c_tebd.example_TEBD_gs_finite(L, 1, g)
    site_exp = psi.site_expectation_value(sx)
    result = []
    c_j = []
    for j in j_list:
        result.append(correlation(sx, sx, psi, i, i+j))
        c_j.append(result-site_exp[i]*site_exp[i+j])
    plt.plot(j_list, result, label='g={}'.format(g))
    return result, c_j


g_list = [0.3,0.5,0.8,0.9,1.,1.1,1.2,1.5]
L = 30
phase_transition = {}
c_j = {}
```

```
for g in g_list:
    p,c=plot_cor(int(L/4), g, L)
    phase_transition[g] = p
    c_j[g] = c
plt.legend()
plt.title('Correlation v.s. dist')
plt.xlabel('distance')
plt.ylabel('correlation')
plt.savefig('correlation.png')
plt.clf()
```

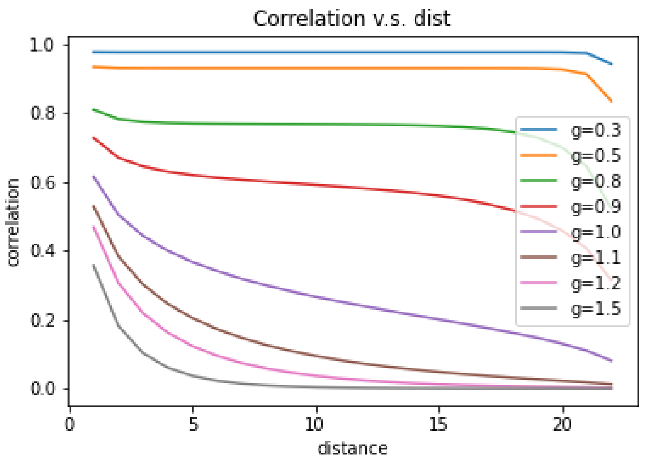


Figure 2: Every cureve indicate how different g-value settings infect the correlation between different site-site distance.

From the output picture, we could find that, when g-values are samller than 1, obviously, platforms appear on each curve until the distance beyond $3L/4$. While the g-values are greater than 1, the correlations decrease rapidly at beginning and remain in low position.

3.3 Phase transition

Considering the boundary effects at the tail of the finite 1D Ising model, we cut the tail and select the correlation at $3L/4$ site, with distance $L/2$, as the phase corresponding to different g-value settings.

```
phase = []
for g in g_list:
    phase.append(phase_transition[g][int(L/2)])
plt.plot(g_list, phase)
plt.xlabel('g')
plt.ylabel('magnetization␣sq')
plt.title('phase_transistion')
plt.savefig('phase.png')
plt.clf()
```

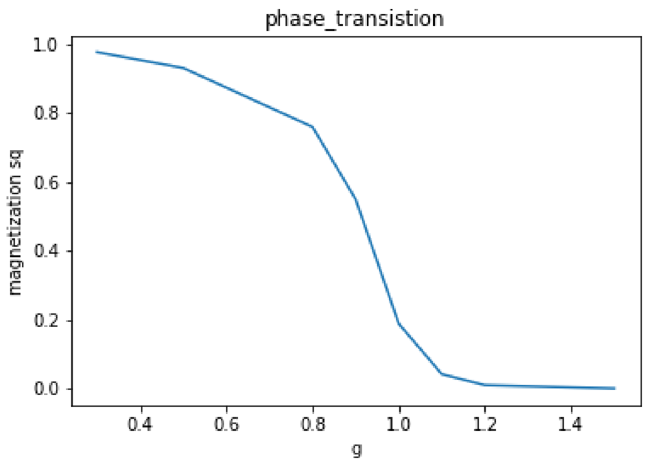


Figure 3: This pictture shows how the phase change with different g-values.

From the output picture, we could find that the curve drops a great range when g-values change between 0.8 and 1.

3.4 Correlation length (optional)

To solve this question, we could just apply the logarithm plot of correlaton value on distance.

```
for g in g_list:
    x = np.arange(len(c_j[g]))+1
    y = np.log(c_j[g][-1])
    z = np.polyfit(x, y, 1)
    plt.plot(x, y, label='g={}'.format(g))
    print('when␣g={0},possible␣zeta␣is␣{1}'.format(g,z[0]))
plt.legend()
plt.xlabel('distance')
plt.ylabel('log␣of␣correlation')
plt.title('log_corr␣v.s.␣dist')
plt.savefig('log_corr_dist.png')
plt.clf()
```

The corresponding output should be:

```
when g=0.3,possible xi is -0.0004506260745895987
when g=0.5,possible xi is -0.001581576795561635
when g=0.8,possible xi is -0.009164707277142364
when g=0.9,possible xi is -0.023789176208651296
when g=1.0,possible xi is -0.0762684653245938
when g=1.1,possible xi is -0.158071267029386
when g=1.2,possible xi is -0.24150833153183635
when g=1.5,possible xi is -0.46235622620625516
```

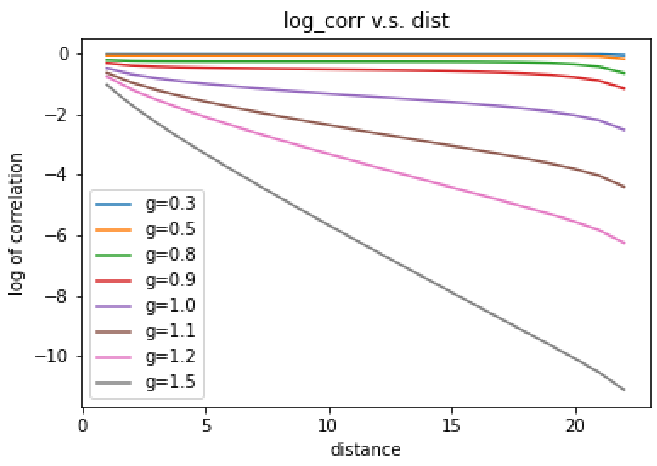


Figure 4: This pictture shows how the relation $C(j) \propto exp(-|j - L/4|/\xi)$.

From both output and the figure, we could conclude that, when g-value is smaller than 1, $1/\xi$ is extreme small. While g-value is set to greater than 1, the curve becomes more and more oblique.

4 Dynamical spinstructure factor

4.1 Correlation and Entropy with TEBD

Here we need time evoving for real time steps, then we could construct a function by modifying the code in c_tebd.py to initiate the unitary operators for each bonds of the MPS.

```
def calc_U_bonds_realtime(model, dt):
    """Given a model, calculate
          ``U_bonds[i] = expm(-dt*model.H_bonds[i])``.
    Each local operator has legs
         (i out, (i+1) out, i in, (i+1) in),
         in short ``i j i* j*``.
    Note that no imaginary 'i' is included,
         thus real `dt` means imaginary time evolution!
    """
    H_bonds = model.H_bonds
    d = H_bonds[0].shape[0]
    U_bonds = []
    for H in H_bonds:
        H = np.reshape(H, [d * d, d * d])
        U = expm(-dt * H * 1j)
        U_bonds.append(np.reshape(U, [d, d, d, d]))
    return U_bonds
```

For convinience, we could also construct a function to generate the left part and the right part of a particular site.

```
def get_Lp_Rp(psi, psi_0):
    L = psi.L
    Lp = []
    Rp = []
    Lp.append(np.tensordot(psi_0.get_theta1(0).conj(),
          psi.get_theta1(0), [[0,1],[0,1]])) #vR*, vR

    for i in range(1,L-1):
        cell = np.tensordot(psi_0.Bs[i].conj(),
                psi.Bs[i], [[1],[1]]) #vL*, vR*, vL, vR

        Lp.append(np.tensordot(Lp[-1],
                cell, [[0,1],[0,2]]))

    Rp.append(np.tensordot(psi_0.Bs[-1].conj(),
          psi.Bs[-1], [[1,2],[1,2]]))     #vL*, vL
    for i in range(2,L):
        cell = np.tensordot(psi_0.Bs[L-i].conj(),
                psi.Bs[L-i], [[1],[1]]) #vL*, vR*, vL, vR
        Rp.insert(0, np.tensordot(cell, Rp[0], [[1,3],[0,1]]))
#vL*, vL
    return Lp, Rp
```

Then we need a function for computing the correlation with time evolving process, i.e.

$$C(j,t) = \langle\psi_0|e^{iHt}\sigma_j^y e^{-iHt}\sigma_{L/2}^y|\psi_0\rangle$$

. Simultaneously, we could call the function in a_mps.py to calculate the entanglement entropy for given MPS. Therefore, we could construct three functions for this task.

```
def get_correlation(psi_origin, L, psi, chi_max, eps):
    cor_list = []
    Lp, Rp = get_Lp_Rp(psi, psi_origin)

    theta_j0 = np.tensordot(sy, psi.get_theta1(0),
                [[1],[1]]) # i, vL, vR
    first_cell = np.tensordot(psi_origin.get_theta1(0).conj(),
                theta_j0, [[0,1], [1,0]]) #vR*, vR
    cor_list.append(np.tensordot(first_cell, Rp[0],
                [[0,1],[0,1]]))

    for j in range(1, L-1):
        Bj =  np.tensordot(sy, psi.Bs[j], [[1],[1]]) # i, vL, vR
        cell_j = np.tensordot(psi_origin.Bs[j].conj(), Bj,
                    [[1], [0]]) #vL*, vR*, vL, vR
        cell_mit_r = np.tensordot(cell_j, Rp[j],
                    [[1,3], [0,1]])     #vL*, vL
        cor_list.append(np.tensordot(cell_mit_r, Lp[j-1],
                    [[0,1],[0,1]]))

    final_bj = np.tensordot(sy, psi.Bs[-1], [[1],[1]]) # i, vL, vR
    final_cell = np.tensordot(psi_origin.Bs[-1].conj(),
                final_bj, [[1,2],[0,2]]) #vL*, vL
    cor_list.append(np.tensordot(Lp[-1], final_cell,
                [[0,1],[0,1]]))
    return np.array(cor_list)

def TEBD_realtime(psi, model, t, chi_max, eps, dt, psi_origin, E):
    U_bonds = calc_U_bonds_realtime(model, dt)
    steps = int(t/dt)
    ent_list = []
    cor_list = []
    for s in range(steps):
        c_tebd.run_TEBD(psi, U_bonds, 1, chi_max, eps)
        ent_list.append(psi.entanglement_entropy())
        cor_list.append(get_correlation(psi_origin, psi.L, psi,
                chi_max, eps)*np.exp(E*s*dt*1j))
    return ent_list, cor_list

def time_evolution_correlation(L, g, h, t, dt):
    chi_max=100
    eps=1.e-10
    E, psi, model =c_tebd.example_TEBD_gs_finite_2(L, 1,
               g, h)
    psi_origin = psi.copy()
    psi.Bs[int(L/2)] = np.transpose(np.tensordot(sy,
                psi.Bs[int(L/2)],[[1],[1]]), (1,0,2))# vL, i, vR
    ent_list , cor_list = TEBD_realtime(psi, model, t,
                chi_max, eps, dt, psi_origin, E)
    return np.array(ent_list), np.array(cor_list)
```

4.2 Method to compute the Dynamic Structure Factor

As the time evolving process occur in finite time sequence, when computing the time space Fourier transformation, we need to multiple the Gaussian window function to the result elimilating the edge effects of finite time sequence.

$$S(k,\omega) = \sum_{j=0}^{L-1} \sum_{t_n=0}^{N} e^{i\omega\Delta t\cdot t_n - ik(j-L/2)} C(\Delta t\cdot t_n, j)\cdot G(t_n)$$

where we have

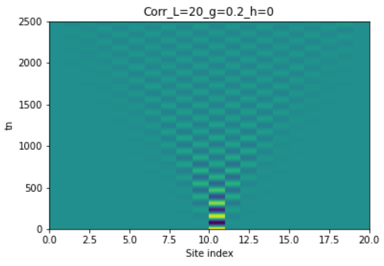
$$G(t_n) = e^{-1/2\cdot(t_n/\sigma N)^2}$$

As the assistant code provided by teaching team with using fourier transformation method in numpy library worked irregularly, I used the formuler above to compute the Dynamic Structure Factor manually.

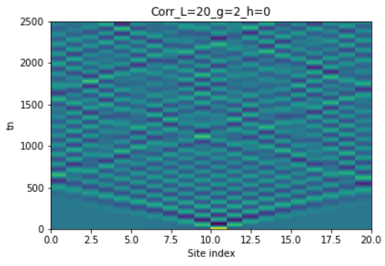
```
def get_dsf_manul(cor_arr, dt, L):
    k_list = np.arange(-3.1, 3.1, 0.2)
    w_list = np.arange(-1, 10.5, 0.2)
    xi = int(L/2)
    sigma = 0.4
    N=cor_arr.shape[0]
    skw = []
    for w in w_list:
        sw = []
        for k in k_list:
            s = 0
            for j in range(L):
                for tn in range(N):
                    s+=np.exp(1j*(w*dt*tn-k*(j-xi)))
                        *cor_arr[tn, j]
                        *np.exp(-0.5*(tn/sigma/N)**2)
            sw.append(s)
        skw.append(sw)
    return skw
```

4.3 Numerical Procedure

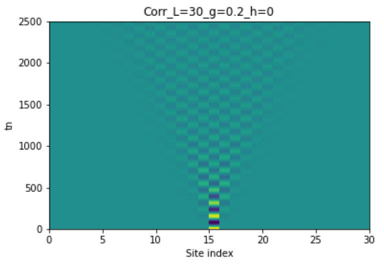
In this part, we compute the correlation, entanglement entropy, and dynamic spin structure factor for different setting of MPS. For time evolving, the total time length is $t = 25J$, and the time step for evolving is $dt = 0.01$. Firstly, we plot the correlation of diffent settings and compare the results.



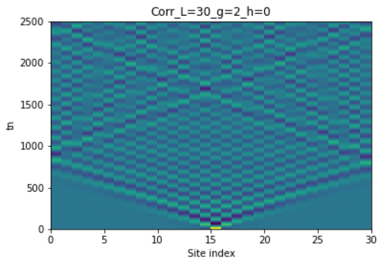
(a) L=20,g=0.2,h=0



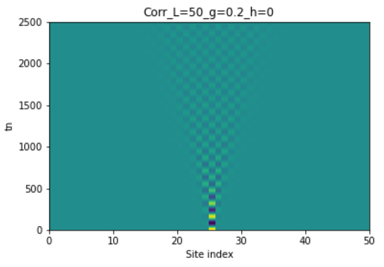
(b) L=20,g=2,h=0



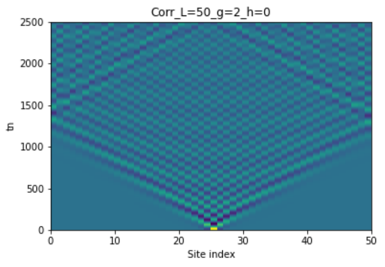
(c) L=30,g=0.2,h=0



(d) L=30,g=2,h=0



(e) L=50,g=0.2,h=0



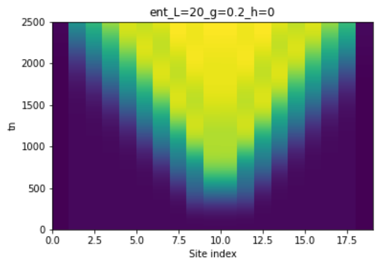
(f) L=50,g=2,h=0

Figure 5: The correlation figure for different MPS sizes, different Hamiltonian settings, when h-factor is set to 0.

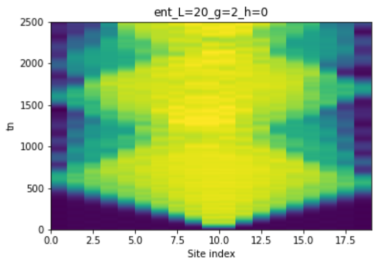
Viewing the pictures along rows, we could find that with larger g-value, the range of each sites correlated with the center site is getting larger, in other words, when g is smaller than 1, the MPS is in ordered phase and its ground state will not break the spin-flip symmetry. While we set g to larger than 1, the MPS is in a disordered phase.When g¡1, the correlation was restricted to the circular sector, i.e. with time increasing, the correlation range sprerad larger but the absolute correlation is being eliminated. If we set g over 1, the spread speed of the correlation between sites is creater, and simultaneously, we could observe the rhombus appearing periodically.

If we view the pictures along columns, we could find that if g is less than 1, the correlation range is roughly the same for different MPS size. However, if g is greater than 1, the period of the rhombus shown with time evolving is increasing. The reason is that, in this experiment, the system is not infinite in size, and when g-valule is large, the spin-flip will bounce at the edge of the system.

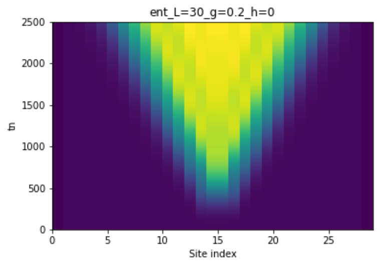
Then, we plot the entanglement entropy for different settings.



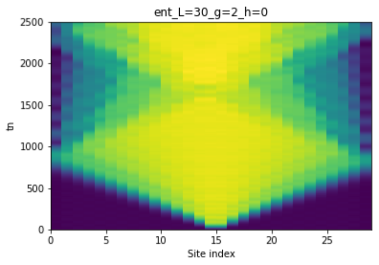
(a) L=20,g=0.2,h=0



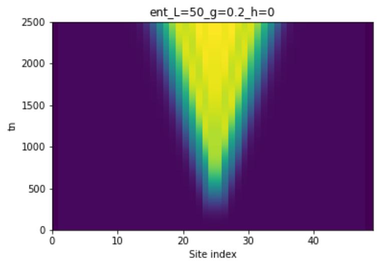
(b) L=20,g=2,h=0



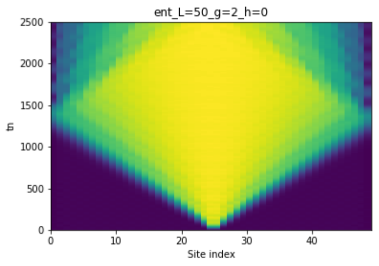
(c) L=30,g=0.2,h=0



(d) L=30,g=2,h=0



(e) L=50,g=0.2,h=0

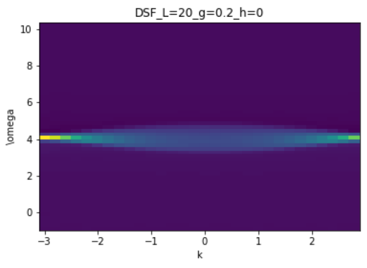


(f) L=50,g=2,h=0

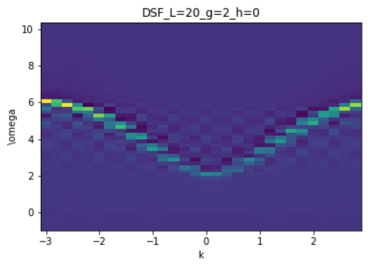
Figure 6: The entanglement entropy figure for different MPS sizes, different Hamiltonian settings, when h-factor is set to 0.

We could easily found that the performance of the entanglement entropy is similar to the correlation function.

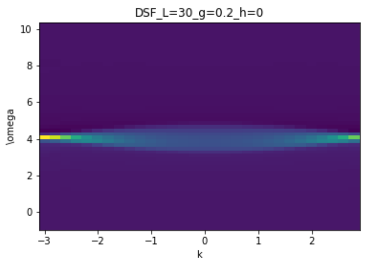
Finnaly, we plot the result of the Dynamic Structure Factor of different settings.



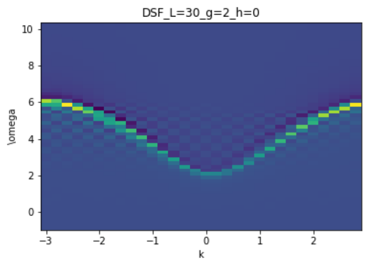
(a) L=20,g=0.2,h=0



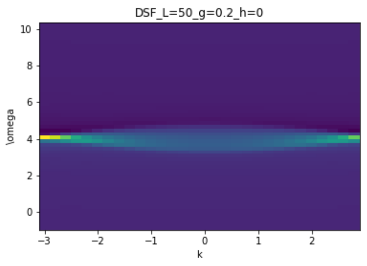
(b) L=20,g=2,h=0



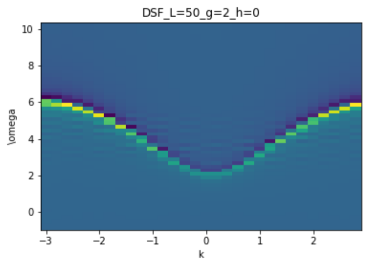
(c) L=30,g=0.2,h=0



(d) L=30,g=2,h=0



(e) L=50,g=0.2,h=0



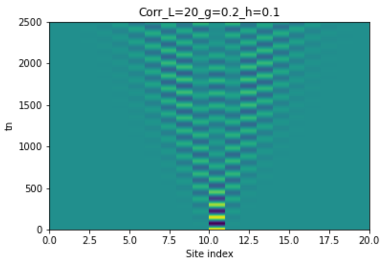
(f) L=50,g=2,h=0

Figure 7: The Dynamic Structure Factor figure for different MPS sizes, different Hamiltonian settings, when h-factor is set to 0.

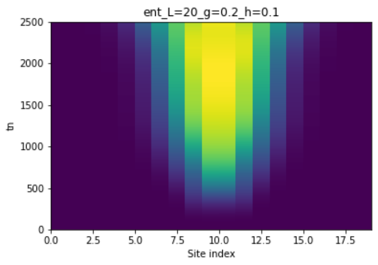
With lower g-value, the dominance of the system is the ferromagnetic fiel. Therefore, if the systems were ideal and infinite, the DSF firgures would perform a perfect straight line. While g-value is over the gap situation, i.e. g is larger than 1, the DSF firgures will show a curve with lowest angular frequency at the center point.

4.4 Effects of longitudinal field

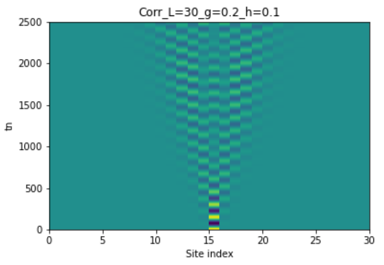
In this part we apply the code we composed above to create the ground states of Hamiltonian with longitudinal field.The longitudinal field, h-value, was set to 0.1, and then we could plot the correlation function and the entangelment entropy.



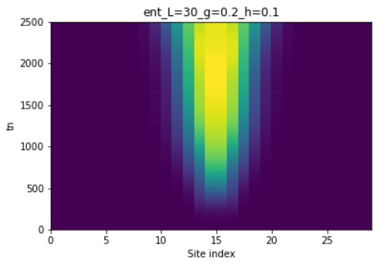
(a) correlation,L=20,g=0.2,h=0.1



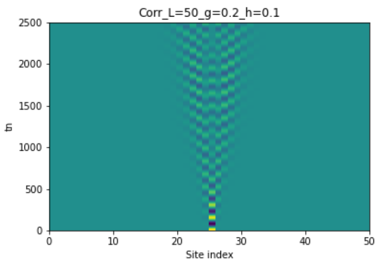
(b) entropy,L=20,g=0.2,h=0.1



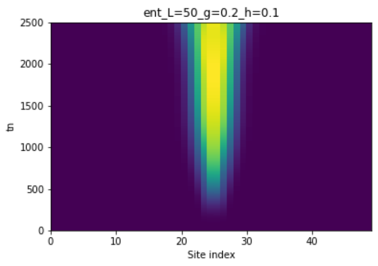
(c) correlation,L=30,g=0.2,h=0.1



(d) entropy,L=30,g=0.2,h=0.1



(e) correlation,L=50,g=0.2,h=0.1

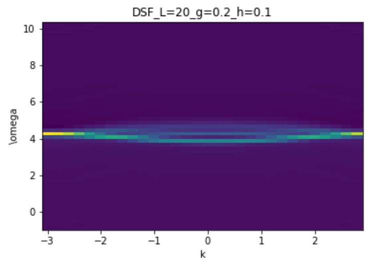


(f) entropy,L=50,g=0.2,h=0.1

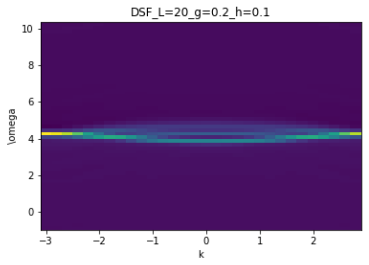
Figure 8: The correlation and entanglement entropy with setting longitudinal field to 0.1.

From these figures, we could found that the longitudinal field would constrain the correlation and entanglement entropy spreading along the 1D Ising model, despite the totle size.

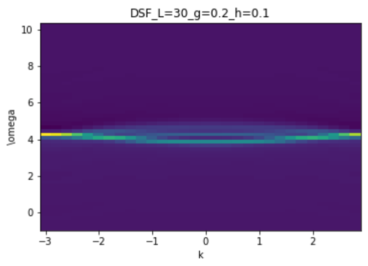
Then, we could view the result how the dynamic structure factor will be effected with the longitudinal field.



(a) DSF,L=20,g=0.2,h=0.1



(b) DSF,L=30,g=0.2,h=0.1



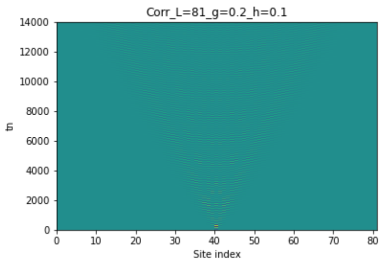
(c) DSF,L=50,g=0.2,h=0.1

Figure 9: The correlation and entanglement entropy with setting longitudinal field to 0.1.

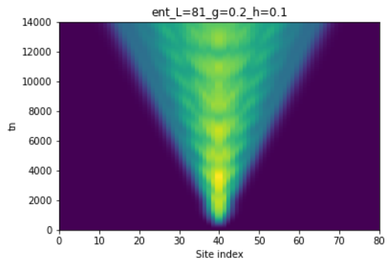
These figures show that with longitudinal field, the angular frequency of the system would shift for a small amount, as such field perform a preference on a specific spin direction. Simultaneously, in the area of the figure, we could observe slites distribution.

4.5 Appendix

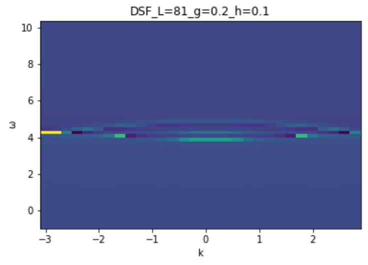
To test our code with larger system size and longer evolving time, we got the result:



(a) Corr,L=81,g=0.2,h=0.1



(b) ent,L=81,g=0.2,h=0.1



(c) DSF,L=81,g=0.2,h=0.1

Figure 10: The correlation, entanglement entropy, and the dynamic structure factor with L=81, evolving time=140J.

From the fig.10 (a), we could find out that with large time, the result of correlation function between sites and the center site is close to 0. For fig.10(b), the shape of the entanglement entropy is like the correlation function. In the circular secctor of the figure, we could find wave-like dark-light switching texture. Moreover, the slites texture of the DSF function figure is more clear than those with smaller size and less evolving time.