量子物理计算方法选讲实验报告

李昊恩　2021010312

infinite time-evolving block decimation(iTEBD), Task 4

目录

1 引言

1.1 模型

本实验求解的模型是一个无限长（或周期边界条件）下的 1D Ising Cluster 自旋模型。该模型的哈密顿量为：

$$H = -\sum_j (gZ_{j-1}Y_jZ_{j+1} + JZ_jZ_{j+1} + hX_j), \tag{1}$$

这里 $X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$，$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$，$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ 是 Pauli 算子。

在本次实验中，我们将采用 3-site iTEBD（无限长时间演化块消减算法）来求解该体系的基态能量、磁化以及纠缠熵、纠缠谱。

1.2 iTEBD 方法概述

时间演化块消减方法（TEBD）(G.Vidal, "*Efficient Simulation of One-Dimensional Quantum Many-Body Systems,*" **PRL 2004**, 93, 040502) 首先被用于计算矩阵乘积态的时间演化。该

方法也可以被推广于计算 1D 无限长平移不变系统的虚时演化，进而计算基态能量。这一方法通常称为无限时间演化块消减算法 (infinite TEBD, iTEBD, G.Vidal. "*Classical Simulation of Infinite-Size Quantum Lattice Systems in One Spatial Dimension,*" **PRL 2007**, 98, 070201)。

在本次任务中,我们考虑实现 3 张量平移不变的矩阵乘积态。设哈密顿量为 $H=\sum_j h_{j,j+1,j+2}$，单个切片的虚时间演化算符记为：

$$U(\tau)=e^{-\tau H}, \tag{2}$$

根据不对称单元张量的分布，可以将哈密顿量中的耦合分成三类：

$$H=\sum_{j\equiv 0 \mod 3} h_{j,j+1,j+2}+\sum_{j\equiv 1 \mod 3} h_{j,j+1,j+2}+\sum_{j\equiv 2 \mod 3} h_{j,j+1,j+2}, \tag{3}$$

分别对应着序号模 3 余数为 $0,1,2$ 的张量与其右侧相邻的两个张量之间的耦合.

我们以 $\delta\tau$ 作为虚时演化的单元，根据 Trotter-Suzuki 分解可得：

$$U(\tau)\approx \prod_{j\equiv 0 \mod 3} \exp(-\delta\tau h_{j,j+1,j+2}) \prod_{j\equiv 1 \mod 3} \exp(-\delta\tau h_{j,j+1,j+2}) \prod_{j\equiv 2 \mod 3} \exp(-\delta\tau h_{j,j+1,j+2}) \tag{4}$$

由此，我们可以构建 "砖墙结构" 的时间演化算符，如图1所示。因为砖墙结构自身是 3 格点平移不变的，因此，我们至少需要使用 3 张量平移不变的矩阵乘积态来实现（图中是一个 2 张量平移不变的情形）。

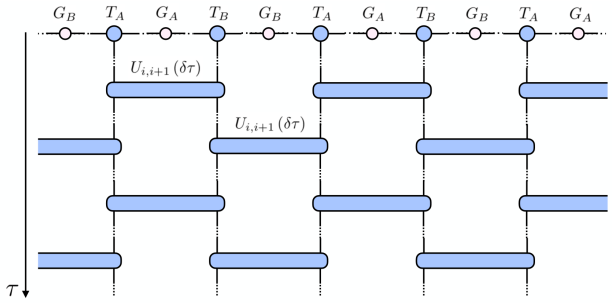

图 1: 砖墙结构的时间演化算符，以 2 位点情形为例

下面我们阐述虚时间演化的基本原理。对于哈密顿量 H，有谱分解：

$$H=\sum_i E_i\left|\psi_i\right\rangle\!\left\langle\psi_i\right|, \tag{5}$$

这里 E_i 是第 i 个能量特征值，我们不妨设 $E_0<E_1<\cdots<0$。于是，对于初始态 $|\phi_0\rangle$，如果将时间演化算符 $U(\tau)$ 作用足够长时间：

$$\lim_{\tau\to\infty} U(\tau)\left|\phi_0\right\rangle=\lim_{\tau\to\infty}\sum_i e^{-\tau E_i}\left|\psi_i\right\rangle\left\langle\psi_i|\phi_0\right\rangle=\lim_{\tau\to\infty} e^{-\tau E_0}\sum_i\left\langle\psi_i|\phi_0\right\rangle e^{\tau(E_0-E_i)}, \tag{6}$$

注意到当 $i=1,2,\cdots$ 时，$E_0-E_i<0$，所以，当作用时间演化算符的时间足够长时，只要初始态和基态有非平凡的重叠，那么体系最终一定会收敛到基态。

在刚开始几步的迭代中，我们可以选用稍大的时间切片 δ 以加快收敛，当逐渐接近收敛时，可以将 δ 调小以尽可能地减小 Trotter 分解产生的误差。在本次实验中，我们将使用 $\delta = 0.1, 0.01, 0.001$ 分别迭代。

2 代码实现

2.1 3 格点张量的更新

我们首先实现对 3 格点张量 $G_\ell - T_1 - G_1 - T_2 - G_2 - T_3 - G_r$ 的更新。为此，我们首先将这些张量都缩并在一起，然后将这个新的 5 阶张量的 3 个物理指标和时间演化算符 $e^{-\delta\tau h}$ 进行缩并，得到一个新的 5 阶张量。然后，我们对这个 5 阶张量进行 SVD 裂分，并在最左边的虚拟指标上插入 G_ℓ 和 G_ℓ^{-1}，再将 G_ℓ^{-1} 和分解得到的奇异值矩阵 S_1 分别缩并到右侧的张量上，完成对 T_1 的更新。然后，对右侧的 4 阶张量再进行一次奇异值分解，随后在更新后的 T_1 右侧的虚拟指标上插入 S_1 和 S_1^{-1}，并立刻将 S_1^{-1} 缩并到右侧的张量上，完成对 T_2 的更新。最后，将最右侧的张量与 G_r^{-1} 缩并起来完成对 T_3 的更新。这个过程用图表示如2。

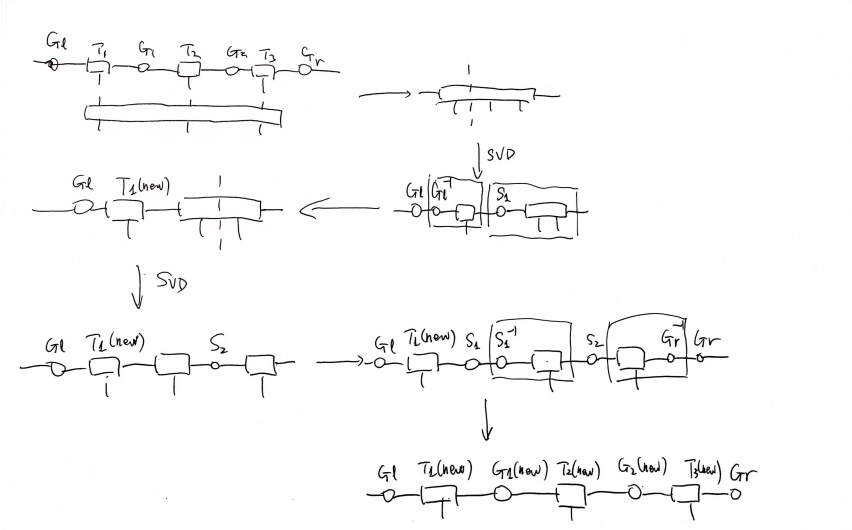

图 2: 抽象画作之：3 位点张量的更新

注意到，在这个更新过程中，G_ℓ 和 G_r 保持不变。另外，在实际的代码实现过程中，我们在每次 SVD 分解后，都根据虚拟指标维数来进行截断。同时，在每次分解完成后，我们将奇异值矩阵 G_1 和 G_2 归一化以避免各张量在指数函数作用多次后数值过大，使得迭代奇异值分解算法无法收敛。代码如下：

```
# evolution 2 bonds  - Gl - T1 - G1 - T2 - G2 - T3 - Gr - at a time in 3-site iTEBD
def Evo_2_bonds(Gl, T1, G1, T2, G2, T3, Gr, UH):
    A = Sub.NCon(
        [np.diag(Gl), T1, np.diag(G1), T2, np.diag(G2), T3, np.diag(Gr), UH],
```

```
        [[-1, 1], [1, 7, 2], [2, 3], [3, 8, 4], [4, 5], [5, 9, 6], [6, -5], [-2, -3, -4, 7, 8, 9]]
    )
    # record the shape of tensor A
    DA = np.shape(A)

    # get the matrization of tensor A, preparing for T1 and G1 updating
    matrix_A = Sub.Group(A, [[0,1],[2,3,4]])

    # update T1 and G1 using SVD decomp.
    U1, S1, V1 = np.linalg.svd(matrix_A, full_matrices=False)
    Dc1 = min(len(S1), Ds) #truncate SVD w.r.t. Ds
    U1 = U1[:, :Dc1]
    S1 = S1[:Dc1]
    V1 = V1[:Dc1, :]
    U1 = np.reshape(U1,[DA[0],DA[1],Dc1]) # reshape matrix U1 to component tensor shape (three legs)
    T1_new = np.tensordot(np.diag(1.0/G1),U1,(1,0))

    # update T2 and G2 using SVD decomp.
    U2, S2, V2 = np.linalg.svd(np.reshape(np.diag(S1)@V1, [Ds * DA[2], -1]), full_matrices=False)
    Dc2 = min(len(S2), Ds) #truncate SVD w.r.t. Ds
    U2 = U2[:, :Dc2]
    S2 = S2[:Dc2]
    V2 = V2[:Dc2, :]
    T2_new = np.tensordot(np.diag(1/S1), np.reshape(U2, [Dc1, DA[2], Dc2]), (1,0))

    # update T3
    T3_new = np.tensordot(np.reshape(V2, [Dc2, DA[3], DA[4]]), np.diag(1.0 /Gr), (2, 0))

    G1_new = S1
    S1 /= np.sqrt(np.sum(S1 ** 2))
    G2_new = S2
    S2 /= np.sqrt(np.sum(S2 ** 2))

    return T1_new, G1_new, T2_new, G2_new, T3_new
```

其中，哈密顿量的构建以及时间演化算符的计算通过以下代码实现：

```
def GetHam_IsingCluster(g,J,h):
Ham = - g * np.kron(pZ,np.kron(pY,pZ)) - J * np.kron(Id, np.kron(pZ, pZ)) - h * np.kron(Id, np.kron(pX, Id))
```

```
3
4 # reshape the 3-body Hamiltonian into a 6 - order tensor:
5 Ham = np.reshape(Ham,[Dp,Dp,Dp,Dp,Dp,Dp])
6 return Ham
7
8 def GetExpHam(Ham,Tau):
9 Dp = np.shape(Ham)[0]
10
11 if LA.norm(Ham) < 1.0e-12:
12     UH = np.reshape(np.eye(Dp**3),[Dp,Dp,Dp,Dp,Dp,Dp])
13 else:
14     # reshape hamiltonian tensor (6-order) into a matrix of size Dp**3, then apply
      eigenvalue decomposition
15     A = np.reshape(Ham,[Dp**3,Dp**3])
16
17     # Dc is the bond dimension of the diagonal matrix
18     V,S,Dc = Sub.SplitEigh(A,Dp**3)
19
20     # calculate e^{-\tau S}
21     W = np.diag(np.exp(-Tau*S))
22
23     # update e^A as new A
24     A = np.dot(np.dot(V,W),np.transpose(np.conj(V)))
25
26     # reshape UH = e^(-\tau H) (imaginary time evolution operator) into a 6-order
      tensor
27     UH = np.reshape(A,[Dp,Dp,Dp,Dp,Dp,Dp])
28
29 return UH
```

2.2 砖墙形时间演化算符的迭代过程

随后我们可以不断地、每次移动一个位点地（周期为 3）作用时间演化算符，对各位点的张量进行更新。这只需要不断地调用之前定义的 `Evo_2_bonds` 函数即可，其代码实现具体如下所示：

```
1 # iterative itebd procedure:
2 def Evo_3site(Ds,Ham,Tau_list,Iter,Prec):
3     Dp = np.shape(Ham)[0]
4     T,G = init_TG(Dp, Ds, 3)
5
6     r0 = 0
```

```
    for idt in range(len(Tau_list)):
        dt = Tau_list[idt]
        UH = GetExpHam(Ham,dt)

        G0 = np.ones(3)
        for r in range(Iter):
            for bond in range(3):
                T[bond], G[bond], T[(bond+1)%3], G[(bond+1)%3], T[(bond+2)%3] = Evo_2_bonds(
                    G[(bond-1)%3], T[bond], G[bond], T[(bond+1)%3], G[(bond+1)%3], T[(bond+2)%3], G[(bond+2)%3], UH
                )

            Err = 0.0
            for i in range(3):
                Err += np.abs(G[i][0]-G0[i])
            #if np.mod(r,100) == 1:
                #print(r+r0,Err)
            if Err < Prec[idt]:
                r0 += r
                break
            for i in range(3):
                G0[i] = G[i][0]
    print("Convergence is achieved!")
    return T,G
```

这里，接收的参数 Prec 指的是每一轮迭代的收敛判定条件。在本实验中由于收敛整体较快，我们统一采用 10^{-15} 作为收敛判定条件。

2.3 能量、磁化、纠缠熵和纠缠谱的计算

这些计算的实现都很容易，我们首先定义如下的函数，它接受一个涉及到 3 个不对称自旋位点的 3 体算子，返回该 3 体算子的期望值 $\langle\psi|O|\psi\rangle$。代码实现如下：

```
# calculate 3-body operator
def Cal_2_bonds(Op, Gl, T1, G1, T2, G2, T3, Gr):
    vec = Sub.NCon([np.diag(Gl), T1, np.diag(G1), T2, np.diag(G2), T3, np.diag(Gr)],
    [[-1, 1],[1, -2, 2], [2, 3], [3, -3, 4], [4, 5], [5, -4, 6], [6, -5]])
    expectation = Sub.NCon([vec, Op, np.conj(vec)],
    [[7, 1, 2, 3, 8], [4, 5, 6, 1, 2, 3], [7, 4, 5, 6, 8]])
    return expectation
```

该缩并过程的张量网络图表示为图3：

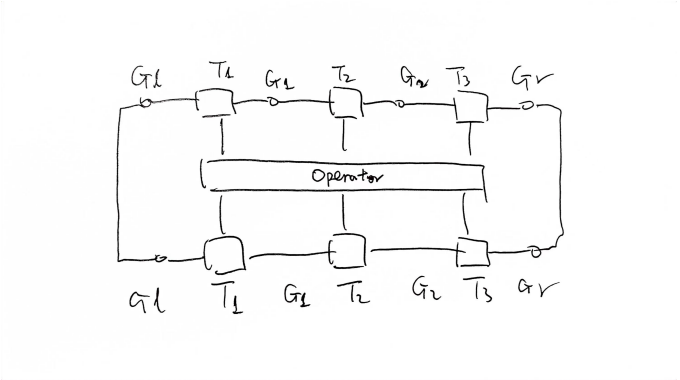

图 3: 计算 3 体算子期望值的张量网络图

将三体算子分别换成 3 位点哈密顿量 $h_{j,j+1,j+2}$、X 方向的磁化 $X \otimes \mathrm{Id} \otimes \mathrm{Id}$ 和 Z 方向的磁化 $X \otimes \mathrm{Id} \otimes \mathrm{Id}$，即可实现对各种可观测量的期望值计算，注意归一化：

$$\langle O \rangle = \frac{\langle \psi | O | \psi \rangle}{\langle \psi \rangle}, \tag{7}$$

代码实现如下：

```
# calculate energy after convergence is reached
def Cal_energy(T, G, Ham):
    D = np.shape(Ham)[0]

    # identity tensor for <psi|psi> calculation
    H00 = np.reshape(np.eye(D**3,D**3),[D,D,D,D,D,D])

    normalize = np.zeros(3)
    energy = np.zeros(3)

    for bond in range(3):

        normalize[bond] = np.real(\
        Cal_2_bonds(H00, G[(bond-1)%3], T[bond], G[bond], T[(bond+1)%3], G[(bond+1)%3], T[(bond+2)%3], G[(bond+2)%3]))

        energy[bond] = np.real(\
        Cal_2_bonds(Ham, G[(bond-1)%3], T[bond], G[bond], T[(bond+1)%3], G[(bond+1)%3], T[(bond+2)%3], G[(bond+2)%3]))

        energy[bond] /= normalize[bond]
        print(f'bond i = {bond}, energy: {energy[bond]}')

    energy = np.mean(energy)
    print(f'average energy: {energy}')
```

```
24
25      return energy
```

以及磁化：

```
1  # calculate x- and z- magnetization after convergence is reached
2  def Cal_mag(T, G):
3      S0,Sp,Sm,Sz,Sx,Sy = Sub.SpinOper(Dp)
4      D = Dp
5
6      H00 = np.reshape(np.eye(D**3,D**3),[D,D,D,D,D,D])
7      xmag_op = np.reshape(np.kron(pX, np.kron(Id, Id)),[D,D,D,D,D,D])
8      zmag_op = np.reshape(np.kron(pZ, np.kron(Id, Id)),[D,D,D,D,D,D])
9
10     xmag_val = np.zeros(3)
11     zmag_val = np.zeros(3)
12     normalize = np.zeros(3)
13
14     for bond in range(3):
15         normalize[bond] = np.real(\
16         Cal_2_bonds(H00, G[(bond-1)%3], T[bond], G[bond], T[(bond+1)%3], G[(bond+1)
       %3], T[(bond+2)%3], G[(bond+2)%3]))
17         xmag_val[bond] = np.real(\
18         Cal_2_bonds(xmag_op, G[(bond-1)%3], T[bond], G[bond], T[(bond+1)%3], G[(bond
       +1)%3], T[(bond+2)%3], G[(bond+2)%3]))
19         xmag_val[bond] /= normalize[bond]
20         zmag_val[bond] = np.real(\
21         Cal_2_bonds(zmag_op, G[(bond-1)%3], T[bond], G[bond], T[(bond+1)%3], G[(bond
       +1)%3], T[(bond+2)%3], G[(bond+2)%3]))
22         zmag_val[bond] /= normalize[bond]
23         print(f'bond i = {bond}, <\sigma_x>: {xmag_val[bond]}, <\sigma_z>: {zmag_val
       [bond]}')
24
25     xmag_val = np.mean(xmag_val)
26     zmag_val = np.mean(zmag_val)
27     print(f'average <\sigma_x>: {xmag_val}')
28     print(f'average <\sigma_z>: {zmag_val}')
29     return xmag_val, zmag_val
```

最后，我们利用 gauge tensor G_i 还可以计算出纠缠信息。根据纠缠谱的定义，i 位点的纠缠熵和纠缠谱分别是：

$$G_i = (\lambda_1, \cdots, \lambda_m), \quad S_i = -\sum_{j=1}^{m} \lambda_j^2 \log \lambda_j^2, \quad \mathrm{Sp}_j = -\log \lambda_j \tag{8}$$

2.4 结果

我们对 $N_s = 10$，$D_s \in \{4, 6\}$ 运行程序，输出结果如下：

```
(1) Magnetization
bond i = 0, <\sigma_x>: 0.4497740171328866, <\sigma_z>: 0.7391289425707998
bond i = 1, <\sigma_x>: 0.44911445420513546, <\sigma_z>: 0.7398519781418951
bond i = 2, <\sigma_x>: 0.44971885477024043, <\sigma_z>: 0.7391590144300675
average <\sigma_x>: 0.44953577536942085
average <\sigma_z>: 0.7393799783809207

(2) Energy
bond i = 0, energy: -1.491042002070756
bond i = 1, energy: -1.4918327863614669
bond i = 2, energy: -1.492590945298813
average energy: -1.4918219112436786

(3) Entanglement
bond i = 0, entanglement entopy: 0.15503835872018487, entanglement spectrum:
    [0.01764924 1.69404765 3.51996748 5.32201447 5.82359753 6.8896785 ]
bond i = 1, entanglement entopy: 0.15513577081522514, entanglement spectrum:
    [0.01766313 1.69367871 3.51898026 5.31943709 5.82176432 6.88861417]
bond i = 2, entanglement entopy: 0.15513577081547975, entanglement spectrum:
    [0.01766313 1.69367871 3.51898026 5.31943709 5.82176432 6.88861417]
average entanglement entropy: 0.15510330011696324, average entanglement spectrum:
    [0.017658498271772797, 1.693801692463181, 3.5193093316947617, 5.320296220290019,
     5.822375391993937, 6.888968950190111]
```

此处的基态的 Z 磁化可能为正值或负值（相当于自旋链上下翻转）。

需要注意的是，虚时间演化算符 $e^{-\tau h_{j,j+1,j+2}}$ 并非酉算符，因此，它会破坏 MPS 的正则性。但是同样地，当 τ 接近于 0 时，演化算符接近单位阵，因此在虚时间演化中，一般需要减小 τ，这样不但能够减小 Trotter-Suzuki 误差，而且能减少演化算符对 MPS 正则性的破坏。