量子物理计算方法选讲实验报告

李昊恩　2021010312

Density Matrix Renormalization Group(DMRG), Task 3

目录

1 引言

1.1 模型

本实验求解的模型是一个开边界条件（OBC）下的 1D Ising Cluster 自旋模型。该模型的哈密顿量为：

$$H = -\sum_j (Z_{j-1}Y_jZ_{j+1} + Z_jZ_{j+1} + X_j), \tag{1}$$

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。对于 3 体项，$(j-1, j, j+1) \in \{(0,1,2), (1,2,3), \cdots, (N-2, N-1, N)\}$；对于 2 体项，$(j, j+1) = \in \{(0,1), (1,2), \cdots, (N-1, N)\}$；对于单体项，$j \in \{0, \cdots, N\}$.

在本次实验中，我们将采用变分矩阵乘积态算法（variational matrix product state, vMPS）进行计算，并将计算结果与精确对角化结果进行对比。

1.2 vMPS 方法概述

对于具有开放边界条件（OBC）的体系来说，其关于可观测量 O（例如哈密顿量 H）期望值（例如能量）

$$\langle O\rangle = \frac{\langle\psi|O|\psi\rangle}{\langle\psi|\psi\rangle}, \tag{2}$$

该表达式可以用以下张量网络图描述。这里哈密顿量写成开边界的 MPO 形式（两侧的张量为 **L** 和 **R**，在第二部分会加以说明）：

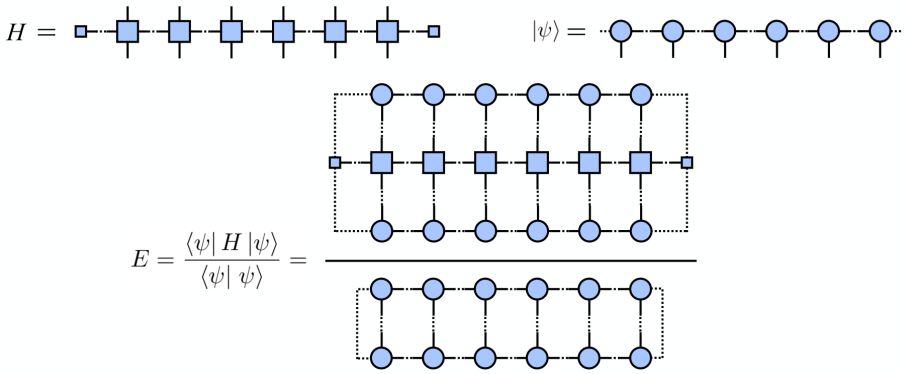

图 1: 期望值计算的张量网络图

如果我们考虑的是具有正则形式的 MPS，那么根据每个局域张量的等距性质（如 $U^\dagger U = \mathrm{Id}$ 以及 $VV^\dagger = \mathrm{Id}$）可知，上式中的分母必然为 1，如2所示的张量缩并图所示。

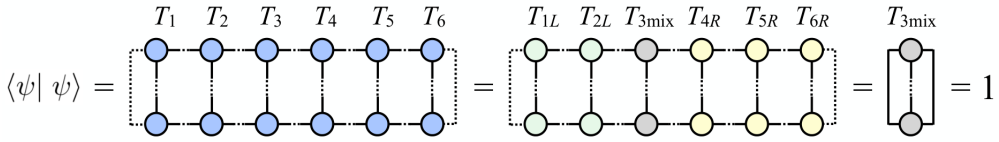

图 2: 正则 MPS 的 $\langle\psi|\psi\rangle = 1$

所以，正则 MPS 的能量期望值就是 $E = \langle\psi|H|\psi\rangle$。要求解体系的基态，实际上可以转化为以下的优化问题:

$$E = \min_{\||\psi_{\mathrm{MPS}}\rangle\|=1} \langle\psi|H|\psi\rangle\,, \quad |\psi\rangle = \mathrm{argmin}_{\||\psi_{\mathrm{MPS}}\rangle\|=1} \langle\psi|H|\psi\rangle\,. \tag{3}$$

由此，我们可以随机地初始化一个 MPS，然后变分地对 MPS 进行优化，使得 $E[|\psi\rangle] = \langle\psi|H|\psi\rangle$ 收敛到最小值，这一求解基态的过程就称为 vMPS。在本次实验中，我们主要考虑的是两种优化方式，其中一种是每次只优化一个位点上的张量，另一种则是每次同时优化相邻两个位点的张量。

我们首先回顾如何将一个任意的 MPS 化为正则形式。在 MPS 的正则形式中，有一个张量是所谓的“中心张量”（也称为混合张量，记号为 T_{mix}），该张量左侧的张量是左正则的，右侧的张量是右正则的。我们从左侧出发，对每个张量 T_i 的矩阵化（即暂时将局部张量的左虚拟指

标和物理指标看成同一个指标，右虚拟指标看成另一个指标，此时该张量变成了一个具有 2 个指标的 2-order-tensor，也就是矩阵）进行 QR 分解（其中 Q 是酉矩阵，R 是上三角矩阵，如图3左上侧所示），然后将酉矩阵重新重组为 3 阶张量的形式，作为新的局域张量（此时它自然变成是左正则的），另一个二阶张量 R 则与右侧的张量 T_{i+1} 在左虚拟指标上缩并，变成一个新的 3 阶局域张量，由此重复该过程，直到把中心张量左侧的所有张量都化成左正则形式。与之完全对称地，我们可以从右侧出发重复同样的过程，只不过这次考虑的是 LQ 分解（Q 是酉矩阵，L 是下三角矩阵，如图3右上侧所示），相应地把中心张量右侧的所有张量转化为右正则形式。我们将以上正则化过程用图3表示。

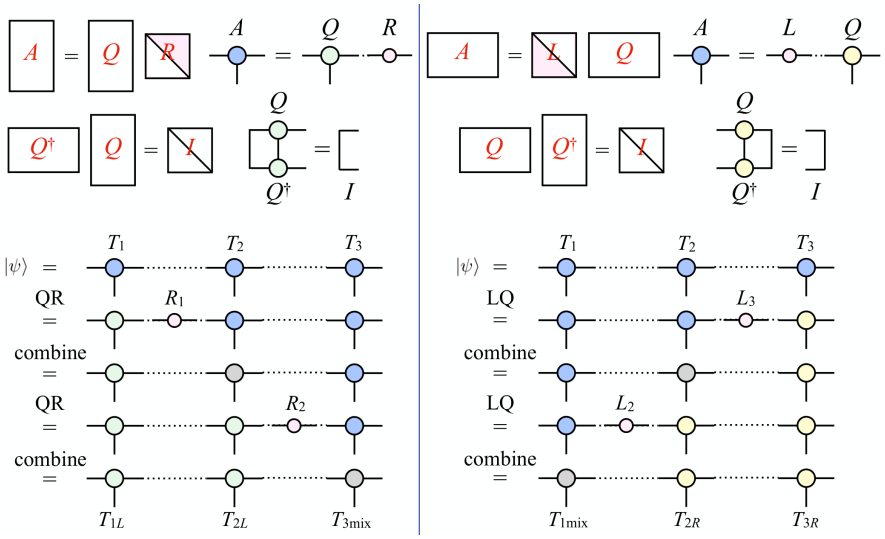

图 3: 正则化的张量网络图表示

其次，我们讨论随后的变分优化过程。对于每次优化一个位点的算法（1-site-varMPS），我们首先以待优化位点为中心张量，将 MPS 转化为正则形式，随后，求解将其他位点的张量固定不变的条件下的最优化问题3，事实上这等价于求解一个特征值问题:

$$H_{\text{eff}}\mathbf{x} = E\mathbf{x}, \tag{4}$$

这里 H_{eff} 为 “有效哈密顿量” 或称为 “环境哈密顿量”，它是将求解期望值 $\langle\psi|H|\psi\rangle$ 所用到的张量缩并图，去掉两个中心张量所剩余的 “环境部分”，它是一个 6 阶张量，相应地，两个 T_{mix} 都是 3 阶张量. 我们将两个 T_{mix} 的指标合并，看成两个向量 $\mathbf{x}$ 和 $\mathbf{x}^\dagger$，此时环境部分相应地变成矩阵 H_{eff}. 事实上，我们就是求解以下的优化问题：

$$\min_{\mathbf{x}} \mathbf{x}^\dagger H_{\text{eff}}\mathbf{x}, \quad \mathbf{x}_{\text{opt}} = \text{argmin}_{\mathbf{x}}\mathbf{x}^\dagger H_{\text{eff}}\mathbf{x}. \tag{5}$$

用张量网络图表示如图4所示。

在实际算法中，我们从左到右依次优化每个位点，再从右往左依次优化每个位点。每一次从左到右再从右回到左的过程通常称为一次 “扫描”，如图5和6所示，其中绿色张量表示是左正则的，黄色张量表示是右正则的，灰色张量表示是待优化的中心（混合）张量。

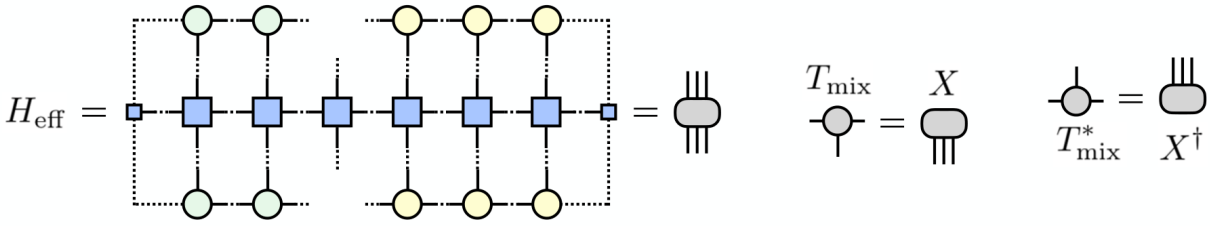

图 4: 单位点 MPS 变分优化

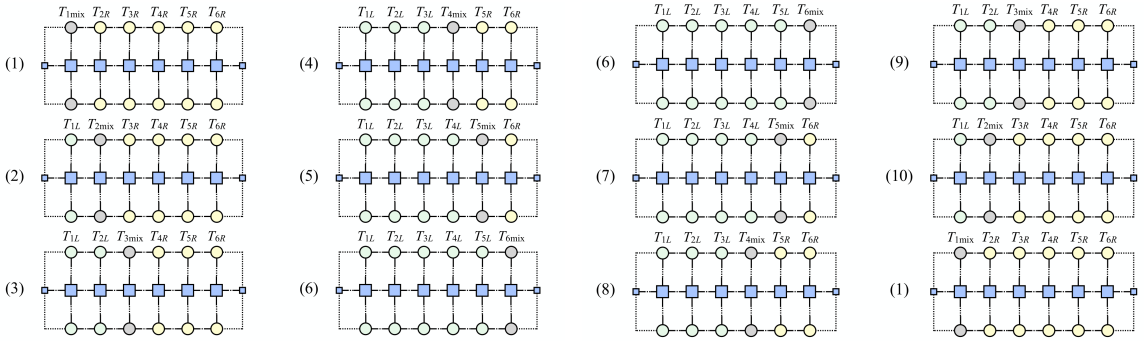

图 5: 从左到右扫描　　图 6: 从右到左扫描

除了单位点优化之外，还有 2-site-varMPS。与 1-site-varMPS 不同的是，2-site-varMPS 单次同时优化两个位点。相应地，这里的有效哈密顿量是一个 8 阶张量，待优化的 T_{mix} 则是一个具有 2 个虚拟指标和 2 个物理指标的 4 阶张量。同样，讲 T_{mix} 变形为一个向量（1 阶张量）后，H_{effect} 相应地变成矩阵，同样求解特征值问题4，如图7所示。

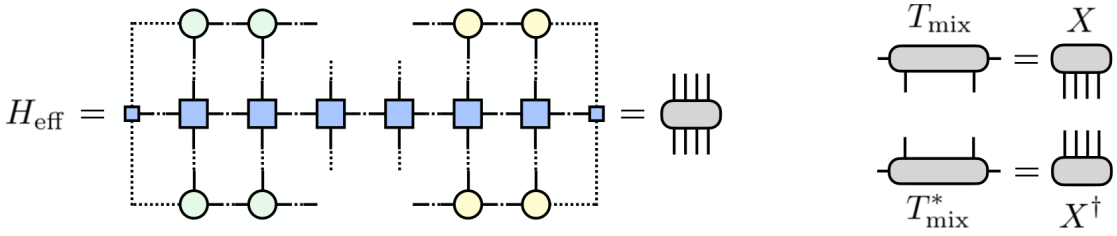

图 7: 优化两个位点的 varMPS

在 2-site 优化的扫描过程中，我们要先将 T_{mix} 通过奇异值分解分裂成酉矩阵 $U, V^\dagger$ 和对角矩阵 S 的乘积：

$$T_{\text{mix}} = USV^\dagger, \tag{6}$$

然后，将 U 对应的 3 阶张量（注意它因此是左正则的）作为新的 T_i，将 S 与 $V^\dagger$ 缩并成一个新的 3 阶局域张量，作为新的 T_{i+1}。随后，将 T_{i+1} 与 T_{i+2} 缩并为新的 T_{mix}，进行下一步的优化。注意，在这样的移动过程中，MPS 仍然保持着正则形式。从右向左扫描的情况完全对称，只不过这时是将左奇异向量矩阵 U 和 S 缩并。这样的移动过程如图8所示。

2-site-vMPS 的每一次优化过程中，相当于变分优化的参数空间相比 1-site-vMPS“更大”，所

图 8: 2-site-varMPS 的扫描过程

以可以预期，2-site-vMPS 的收敛精度应更好。但是，由于每次优化时求解的特征值问题维数相应地会更高，会更加耗时。

1.3 构建 MPO

对于本次实验中的 IsingCluster 模型：

$$H = -\sum_j (Z_{j-1}Y_jZ_{j+1} + Z_jZ_{j+1} + X_j), \tag{7}$$

我们可以用有限状态自动机（finite state automata）快速地写出其 MPO。自动机如图9所示：

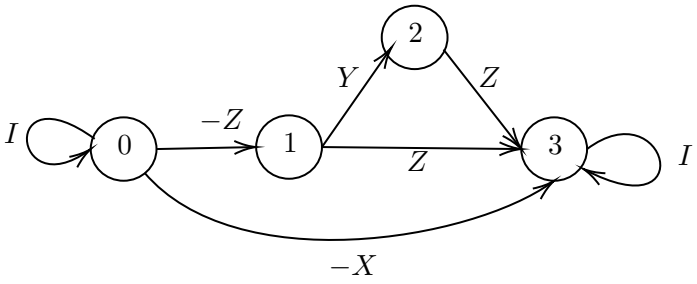

图 9: 有限状态自动机图

根据自动机可以快速地写出 MPO 的表达式为：

$$M = \begin{pmatrix} \mathrm{Id} & -Z & 0 & -X \\ 0 & 0 & Y & Z \\ 0 & 0 & 0 & Z \\ 0 & 0 & 0 & \mathrm{Id} \end{pmatrix}, \quad \mathrm{MPO} = \mathbf{L}^T M \cdot M \cdot \cdots \cdot M\mathbf{R} \tag{8}$$

这里 $\mathbf{L} = (1, 0, \cdots, 0)^T$，$\mathbf{R} = (0, 0, \cdots, 1)^T$.

2 代码实现

2.1 1-site 和 2-site vMPS

我们首先随机地初始化一个 MPS，并用 LQ 分解将其化成右正则形式。我们需要指定三个参数，`Ns` 指的是自旋的数量，`Dp` 是每个物理指标的维数，此处为费米子自旋，因此 `Dp`=2，`Ds`

指的是虚拟指标的最大值。然后，根据第 i 个虚拟指标的维数不会超过 D_p^i 和 $D_p^{N_s-i}$ 中的较小者，可以尽可能地节省所使用的虚拟指标维数。我们调用子程序 `Sub.Mps_LQP` 来完成 LQ 分解：

```
#Initialize the right-canonical mps, using LQ decomposition
def InitMps(Ns,Dp,Ds):
    T = [None]*Ns
    for i in range(Ns):
        Dl = min(Dp**i,Dp**(Ns-i),Ds)
        Dr = min(Dp**(i+1),Dp**(Ns-1-i),Ds)
        T[i] = np.random.rand(Dl,Dp,Dr)

    U = np.eye(np.shape(T[-1])[-1])
    for i in range(Ns-1,0,-1):
        U,T[i] = Sub.Mps_LQP(T[i],U)

    return T
```

随后，我们初始化“环境”。在该程序中用两个列表 `HL` 和 `HR` 进行存储。左、右环境张量的示意图和缩并形式如图10所示，

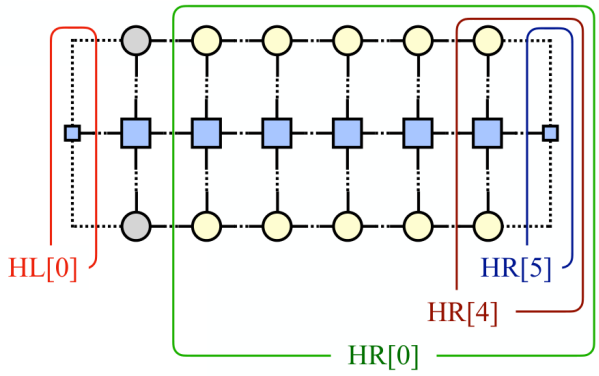

图 10: 左、右环境张量

注意到，我们首先是从左到右进行扫描。在扫描过程中，我们可以通过将优化后的局域张量向左缩并来不断地更新左环境，因此在初始化时，我们只需要定义 `HL[0]`，根据 MPO 的形式可知，`HL[0]` 应当就是 **L**。而在扫描过程中，我们要用到每个 `HR[0]`，因此需要从最右侧开始，逐个将右环境、局域张量 T_i 和 MPOM 进行缩并，递归地求出每个 `HR[i]`。代码实现如下：

```
#Initialize the right-environmental tensors by contracting the mpos and component
    tensors T
def InitH(Mpo,T):
    Ns = len(T)
    Dmpo = np.shape(Mpo)[0]

    HL = [None]*Ns
```

```
    HR = [None]*Ns

    HL[0] = np.zeros((1,Dmpo,1))
    HL[0][0,0,0] = 1.0
    HR[-1] = np.zeros((1,Dmpo,1))
    HR[-1][0,-1,0] = 1.0

    for i in range(Ns-1,0,-1):
        HR[i-1] = Sub.NCon([HR[i],T[i],Mpo,np.conj(T[i])
    ],[[1,3,5],[-1,2,1],[-2,2,3,4],[-3,4,5]])

    return HL,HR
```

对于每个 T_i，我们可以用前一部分讲述的特征值分解方法进行优化。在示例代码中，我们给出了直接、迭代两种可选的方法。代码实现为：

```
#Optimizting T site using eigenvalue decomposition, returning the optimal T and the
    energy on this site
def OptTSite(Mpo,HL,HR,T,Method=0):
    DT = np.shape(T)
    Dl = np.prod(DT)

    if Method == 0:
        A = Sub.NCon([HL,Mpo,HR],[[-1,1,-4],[1,-5,2,-2],[-6,2,-3]])
        A = Sub.Group(A,[[0,1,2],[3,4,5]])+0j
        Eig,V = LAs.eigsh(A,k=1,which='SA')
        T = np.reshape(V,DT)

    if Method == 1:
        def UpdateV(V):
            V = np.reshape(V,DT)
            V = Sub.NCon([HL,V,Mpo,HR],[[-1,3,1],[1,2,4],[3,2,5,-2],[4,5,-3]])
            V = np.reshape(V,[Dl])
            return V

        V0 = np.reshape(T,[Dl])

        MV = LAs.LinearOperator((Dl,Dl),matvec=UpdateV)
        Eig,V = LAs.eigsh(MV,k=1,which='SA',v0=V0)
        T = np.reshape(V,DT)
        Eig = np.real(Eig)

    return T,Eig
```

在扫描过程中，完成一个位点的优化后，立刻通过 LQ 或 QR 分解将此位点变成左正则或者右正则形式，然后将局域张量和 MPO 算符分别收敛到左（或右）环境张量中，完成环境张量的更新。代码实现如下：

```
#sweep back and forth to successively optimize each T until the energy convergence
    is reached
def OptT(Mpo,HL,HR,T):
    Ns = len(T)
    Eng0 = np.zeros(Ns)
    Eng1 = np.zeros(Ns)

    for r in range(100):

        for i in range(Ns-1):
            T[i],Eng1[i] = OptTSite(Mpo,HL[i],HR[i],T[i],Method=1)
            #print(i,Eng1[i])
            T[i],U = Sub.Mps_QR0P(T[i])

            #updating HL[i+1] using HL[i] and the optimized T[i]
            HL[i+1] = Sub.NCon([HL[i],np.conj(T[i]),Mpo,T[i
    ]],[[1,3,5],[1,2,-1],[3,4,-2,2],[5,4,-3]])

            #-U-T[i+1]-  -->  -T[i+1]- (new)
            T[i+1] = np.tensordot(U,T[i+1],(1,0))

        for i in range(Ns-1,0,-1):
            T[i],Eng1[i] = OptTSite(Mpo,HL[i],HR[i],T[i],Method=1)
            #print(i,Eng1[i])
            U,T[i] = Sub.Mps_LQ0P(T[i])

            #updating HR[i-1] using HR[i] and the optimized T[i]
            HR[i-1] = Sub.NCon([HR[i],T[i],Mpo,np.conj(T[i])
    ],[[1,3,5],[-1,2,1],[-2,2,3,4],[-3,4,5]])

            #-T[i-1]-U-  --> -T[i-1]- (new)
            T[i-1] = np.tensordot(T[i-1],U,(2,0))

        if abs(Eng1[1]-Eng0[1]) < 1.0e-7:
            break
        Eng0 = copy.copy(Eng1)

    # print the output energy (per-site & average)
    print("(1) Energy per Site:{}".format(Eng1/float(Ns)))
```

```
    print("(1) Energy average:{}".format(np.mean(Eng1/float(Ns))))
    return T
```

对于 2 位点的优化来说，我们分别定义两个函数 `OpT2SiteL()` 和 `OpT2SiteR()`，分别用于从左到右的扫描和从右到左的扫描中。在从左到右的扫描中，我们对 `T1` 和 `T2` 两个张量缩并得到的两位点 T_{mix} 进行 SVD 分解后，将 U 对应的张量作为新的 `T1`，将 $SV^{\dagger}$ 对应的张量作为新的 `T2`，对于从右到左的扫描则恰好对称。下面给出 `OpT2SiteL()` 的代码，`OpT2SiteR()` 在此文档中略去。

```
#optimize each 2-site from left to right
def OptT2SiteL(Mpo, HL, HR, T1, T2):
    DT1 = np.shape(T1)
    DT2 = np.shape(T2)

    # -T1-T2- -> -T1T2-
    #  |   |       ||
    Contract12 = Sub.NCon([T1,T2],[[-1,-2,1],[1,-3,-4]])

    # get the shape of -T1T2-
    #                  ||
    D = np.shape(Contract12)

    # get the effective Hamiltonian by contracting environmental tensors and the
 mpos
    A = Sub.NCon([HL,Mpo,Mpo,HR],[[-1,1,-5],[1,-6,2,-2],[2,-7,3,-3],[-8,3,-4]])
    A = Sub.Group(A, [[0,1,2,3],[4,5,6,7]])

    # apply eigenvalue decomposition to A and reshape V to - V -
    #                                                      ||
    Eig,V = LAs.eigsh(A, k=1, which='SA')
    V = np.reshape(V, D)

    # reshape V to a matrix and apply SVD
    V = Sub.Group(V, [[0,1],[2,3]])
    UL, Sing, VR = LA.svd(V,full_matrices=False)

    # left singular vector matrix as new T1, then contract schmidt matrix and
right singular vector matrix as new T2
    # = V =  -->  = U - S - V = --> -U-S-V-  -->  -[U-S]-V-  --> -T1-T2- (new)
    #                               |   |          |    |        |   |

    #truncate bond dimension
    max_vdim = min(Sing.shape[0],Ds)
```

```
        Sing_trunc = np.diag(Sing[:max_vdim])
        UL = UL[:,:max_vdim].reshape(V.shape[0],max_vdim)
        VR = VR[:max_vdim,:].reshape(max_vdim,V.shape[1])
        T1 = np.reshape(UL, DT1[:2]+(int(np.prod(UL.shape)/np.prod(DT1[:2])),))
        T2 = np.reshape(Sing_trunc @ VR, (int(np.prod((Sing_trunc @ VR).shape)/np.
    prod(DT2[1:])),)+DT2[1:])

        return T1, T2, Eig
```

在扫描过程中，我们每次利用 HL[i] 和 HR[i+1] 更新 T[i] 和 T[i+1] 两个张量，需要注意的是，我们（以从左到右为例）每次只需要更新 HL[i+1] 用于下一步的计算，而并不需要更新 HL[i+2]（在下一步计算中并不会用到），从右到左扫描的情况类似。代码实现如下：

```
#sweep back and forth to successively optimize each T until the energy convergence
    is reached
def OptT2(Mpo, HL, HR, T):
    Ns = len(T)
    Eng0 = np.zeros(Ns)
    Eng1 = np.zeros(Ns)

    for r in range(100):
        for i in range(0,Ns-1):
            T[i],T[i+1],Eig = OptT2SiteL(Mpo,HL[i],HR[i+1],T[i],T[i+1])
            Eng1[i] = Eig

            #update HL[i+1] using HL[i] and T[i]. Notice that there is no need to
    update HL[i+2]
            HL[i+1] = Sub.NCon([HL[i],np.conj(T[i]),Mpo,T[i
    ]],[[1,3,5],[1,2,-1],[3,4,-2,2],[5,4,-3]])

        for i in range(Ns-2,-1,-1):
            T[i],T[i+1],Eig = OptT2SiteR(Mpo,HL[i],HR[i+1],T[i],T[i+1])
            Eng1[i+1] = Eig

            #update HR[i] using HR[i+1] and T[i+1]. Notice that there is no need to
    update HR[i-1]
            HR[i] = Sub.NCon([HR[i+1],T[i+1],Mpo,np.conj(T[i+1])
    ],[[1,3,5],[-1,2,1],[-2,2,3,4],[-3,4,5]])

        #print(Eng1)
        if abs(Eng1[1]-Eng0[1]) < 1.0e-7:
            break
        Eng0 = copy.copy(Eng1)
```

```
26
27      # print the output energy (per-site & average)
28      print("(1) Energy per Site:{}".format(Eng1/float(Ns)))
29      print("(1) Energy average:{}".format(np.mean(Eng1/float(Ns))))
30      return T
```

最后我们给出基于 MPS 的磁化计算的代码。以 Z 磁化为例，实际上就是计算 $\langle\psi|Z_i|\psi\rangle$。回忆前面叙述的可观测量的期望值计算的张量缩并图。在这里，可观测量的算子都只涉及其中一个自旋位点。因此，可以预见，使用正则形式的 MPS 可以大大减小计算量。具体来说，如11所示，如果要计算 $\langle\psi|Z_3|\psi\rangle$，我们可以先以 3 位点为中心张量将 MPS 化成正则形式，则最终实际上只需要计算 $\mathbf{T}^\dagger(\mathrm{Id}\otimes Z_3\otimes\mathrm{Id})\mathbf{T}$，其中 $\mathbf{T}$ 指的是 T_{mix} 的向量化，这样计算量就大大减小。

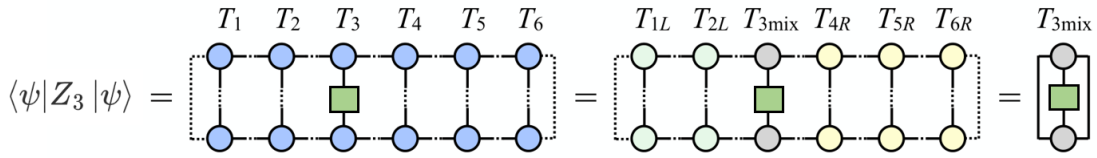

图 11: 磁化的计算

因此，我们的思路是定义函数 `Get_magnet`，接受 MPST 和要求解的单体算符 `op` 作为参数。随后，我们将 MPS 初始化成右正则形式，然后逐个位点计算图11所示的缩并，并通过 LQ 分解更新局域张量 `T[i]` 和环境张量 `HL[i+1]`，使其变成以下一个待计算位点为中心的正则形式。扫描完成后，我们就得到每个点的磁化。代码实现如下：

```
1  def Get_magnet(T,op):
2  Ns = len(T)
3
4  # create an empty list to record the <O> on each site
5  per_site = []
6  S = copy.copy(T)
7
8  # use LQ decomposition to get the right-canonical form of T
9  U = np.eye(np.shape(S[-1])[-1])
10 for i in range(Ns-1,0,-1):
11     U,S[i] = Sub.Mps_LQP(S[i],U)
12 HL[0] = np.eye(1)
13 HR[-1] = np.eye(1)
14 for i in range(Ns-1,0,-1):
15     HR[i-1] = Sub.NCon([HR[i],S[i],np.conj(S[i])],[[1,2],[-1,3,1],[-2,3,2]])
16
17 #sweep from left to right to calulate <O> on each site, use QR decomposition to keep
       the left-environment of T left-canonical,
```

```
#the right-enviroment of T right-canonical
for i in range(Ns):
    magnet = Sub.NCon([S[i],op,np.conj(S[i])],[[1,2,4],[2,3],[1,3,4]])
    per_site.append(magnet.item())
    if i < Ns-1:
        S[i],U = Sub.Mps_QR0P(S[i])
        HL[i+1] = Sub.NCon([HL[i],np.conj(S[i]),S[i]],[[1,2],[1,3,-1],[2,3,-2]])
        S[i+1] = np.tensordot(U,S[i+1],(1,0))
    else:
        continue

return per_site
```

2.2 精确对角化

对于精确对角化，其代码实现相对比较直接。与上次作业相同，我们定义多体算子函数，在特定位点放置单自旋算子，其他位点放置单位算子，返回这些算符的张量积，就得到一个多体算子。

```
#define many_body_operator
def many_body_operator(idx, oprts, size = Ns):
    "Tensor product of `orts` acting on indexes `idx`. Fills rest with Id."
    matrices = [eye if k not in idx else oprts[idx.index(k)] for k in range(size)]
    prod = matrices[0]
    for k in range(1, size):
        prod = np.kron(prod, matrices[k])
    return prod
```

对多体算符求和即可得到哈密顿量，对角化可得基态能量和对应的特征向量：

```
Hamil = np.zeros([2**Ns,2**Ns])+0j
for t in range(Ns - 2):
    Hamil += -1*many_body_operator([t, t+1, t+2], [pZ,pY,pZ])
for b in range(Ns - 1):
    Hamil += -1*many_body_operator([b, b+1], [pZ,pZ])
for s in range(Ns):
    Hamil += -1*many_body_operator([s], [pX])

# Diagonalize the Hamiltonian
evals, evecs = np.linalg.eigh(Hamil)

# Find the lowest eigenvalue
lowest_evals = np.sort(evals)[:1][0]/Ns
```

```
14
15 gs_vec = evecs[:,np.argsort(evals)[0]]
16
17 print("(1) Energy per Site:{}".format(lowest_evals))
```

对特征向量计算 $\langle\psi|O|\psi\rangle$ 即可得到磁化（O 可以用前面定义过的多体算符函数来生成）：

```
1 magnet_x_list = []
2 for i in range(Ns):
3     magnet_op = many_body_operator([i],[pX])
4     eval_magnet = np.conjugate(gs_vec) @ magnet_op @ gs_vec
5     magnet_x_list.append(eval_magnet)
6 print("(2) Sigma_X per Site:{}".format(magnet_x_list))
7 print("(2) Sigma_X average:{}".format(np.mean(magnet_x_list)))
8
9 magnet_z_list = []
10 for i in range(Ns):
11     magnet_op = many_body_operator([i],[pZ])
12     eval_magnet = np.conjugate(gs_vec) @ magnet_op @ gs_vec
13     magnet_z_list.append(eval_magnet)
14 print("(3) Sigma_Z per Site:{}".format(magnet_z_list))
15 print("(3) Sigma_Z average:{}".format(np.mean(magnet_z_list)))
```

2.3 结果

我们对 $N_s = 10$，$D_s \in \{4, 6\}$ 运行程序，输出结果如图12和13。

```
==================== 2-site ====================
(1) Energy per Site:[-1.38599736 -1.38599736 -1.38599736 -1.38603311 -1.38606836 -1.38607924
 -1.38606836 -1.38603311 -1.38599736 -1.38599736]
(1) Energy average:-1.3860268968219571
(2) Sigma_X per Site:[(0.7525714589619878-3.944304526105059e-31j), (0.59567709638435 55+0j), (0.5390506399235852+0j), (0.5149310665537654+0
j), (0.5052385162871 78+0j), (0.5054975732394492+0j), (0.515590234476309+0j), (0.539546107288295+0j), (0.5953733619659177+0j), (0.75201397194
67184+0j)]
(2) Sigma_X average:(0.5815490027027592-3.944304526105059e-32j)
(3) Sigma_Z per Site:[(1.0154674928708474e-07+8.213375100636016e-32j), (1.1582186731917687e-07+0j), (1.1825383156027769e-07+0j), (1.20074517
16256057e-07+0j), (1.252446583666078e-07+0j), (1.2584502839807 5e-07+0j), (1.2621016121094897e-07+0j), (1.4757941352305792e-07+0j), (1.979224
4870808773e-07+0j), (1.948257927208985e-07+0j)]
(3) Sigma_Z average:(1.3733244682567758e-07+8.213375100636017e-33j)
==================== 1-site ====================
(1) Energy per Site:[-1.38601012 -1.38601012 -1.38601012 -1.38601012 -1.38601012 -1.38601012
 -1.38601012 -1.38601012 -1.38601012 -1.38601012]
(1) Energy average:-1.386010116644535
(2) Sigma_X per Site:[(0.7527459557391777+1.3010426069826053e-18j), (0.5958855547989952+0j), (0.5392849326964786+0j), (0.5147183906158799+0
j), (0.5044556514037393+0j), (0.5044556511014836+0j), (0.5147183914553927+0j), (0.5392849343434395+0j), (0.5958855568423375+0j), (0.75274595
71241834+0j)]
(2) Sigma_X average:(0.5814180976121107+1.3010426069826055e-19j)
(3) Sigma_Z per Site:[(1.0917650178354776e-07-6.5603621656617394e-18j), (1.2194972620616085e-07+0j), (1.2443879565671168e-07+0j), (1.2809734
883933999e-07+0j), (1.3119934144656398e-07+0j), (1.2916576197508078e-07+0j), (1.3189146186309841e-07+0j), (1.5363987804217771e-07+0j), (1.77
06715293996922e-07+0j), (1.6499097671429652e-07+0j)]
(3) Sigma_Z average:(1.3716169454669 47e-07-6.56036216566174e-19j)
==================== ED ====================
(1) Energy per Site:-1.3863578562570982
(2) Sigma_X per Site:[(0.7509204587204126+5.551115123125783e-17j), (0.5943390379602914+1.1102230246251565e-16j), (0.5387627133495049+5.55111
5123125783e-17j), (0.5154711808969572+5.551115123125783e-17j), (0.5059699004445618+0j), (0.505969900444562+0j), (0.5154711808969571-2.775557
5615628914e-17j), (0.5387627133495051-2.7755575615628914e-17j), (0.5943390379602906+0j), (0.7509204587204125+0j)]
(2) Sigma_X average:(0.5810926582743455+2.2204460492503132e-17j)
(3) Sigma_Z per Site:[(3.2009117578724045e-14+0j), (3.538142001602296e-14+0j), (3.655409308578328e-14+0j), (3.710920459809586e-14+0j), (3.74
145159298677 75e-14+0j), (3.7587988277465456e-14+0j), (3.715777685542321e-14+0j), (3.626959843572308e-14+0j), (3.58046925441613e-14+0j), (3.1
558089474970075e-14+0j)]
(3) Sigma_Z average:(3.568464967962371e-14+0j)
```

图 12: $D_s = 4$ 结果

```
=================== 2-site ====================
(1) Energy per Site:[-1.38634154 -1.38634154 -1.38634154 -1.38634321 -1.38634516 -1.38634614
 -1.38634516 -1.38634321 -1.38634154 -1.38634154]
(1) Energy average:-1.386343055803831
(2) Sigma_X per Site:[(0.7510064072981352+0j), (0.5943961871464177+0j), (0.5387317196499155+0j), (0.515362919189229+0j), (0.5058339931749828
+0j), (0.5058479176381621+0j), (0.5153858598176269+0j), (0.5387046435280056+0j), (0.5943075898145647+0j), (0.7509103090617908+0j)]
(2) Sigma_X average:(0.581048754631883+0j)
(3) Sigma_Z per Site:[(-2.913946861582417e-12-7.10600544191325e-32j), (-7.2689632091282874e-12+0j), (-1.2464917986676483e-11+0j), (-1.798133
864028273e-11+0j), (-1.4410417303878376e-11+0j), (-1.7850387834528192e-11+0j), (-6.2128857614141e-11+0j), (-3.4511005164716835e-11+0j), (1.5
09903313490213e-13+0j), (2.910677254774896e-11+0j)]
(3) Sigma_Z average:(-1.4027207173583634e-11-7.10600544191325e-33j)
=================== 1-site ====================
(1) Energy per Site:[-1.38634194 -1.38634194 -1.38634194 -1.38634194 -1.38634194 -1.38634194
 -1.38634194 -1.38634194 -1.38634194 -1.38634194]
(1) Energy average:-1.3863419421757408
(2) Sigma_X per Site:[(0.7510171252053368+0j), (0.5944162136739768+0j), (0.5387630761838462+0j), (0.515398273277475+0j), (0.5058468086038387+
0j), (0.5058468140658618+0j), (0.5153982737087383+0j), (0.5387630847023432+0j), (0.5944162399816947+0j), (0.7510171350993818+0j)]
(2) Sigma_X average:(0.581088304502524+0j)
(3) Sigma_Z per Site:[(3.004123799721 37e-08-3.8551146145814065e-18j), (3.1368121544694816e-08+0j), (5.307447370483942e-08+0j), (6.8056152047
19118e-08+0j), (4.487704785560709e-08+0j), (6.028864080187546e-08+0j), (8.772537207768494e-08+0j), (2.7542965963522903e-08+0j), (-1.52028739
59729857e-09+0j), (8.86952145062736e-09+0j)]
(3) Sigma_Z average:(4.103232460472839e-08-3.8551146145814065e-19j)
=================== ED ====================
(1) Energy per Site:-1.3863578562570982
(2) Sigma_X per Site:[(0.7509204587204126+5.551115123125783e-17j), (0.5943390379602914+1.1102230246251565e-16j), (0.5387627133495049+5.55111
5123125783e-17j), (0.5154711808969572+5.551115123125783e-17j), (0.5059699004445618+0j), (0.505969900444562+0j), (0.5154711808969571-2.775557
5615628914e-17j), (0.5387627133495051-2.7755575615628914e-17j), (0.5943390379602906+0j), (0.7509204587204125+0j)]
(2) Sigma_X average:(0.5810926582743455+2.2204460492503132e-17j)
(3) Sigma_Z per Site:[(3.2009117578724045e-14+0j), (3.538142001602296e-14+0j), (3.655409308578328e-14+0j), (3.710920459809586e-14+0j), (3.74
14515929867775e-14+0j), (3.7587988277465456e-14+0j), (3.715777685542321e-14+0j), (3.626959843572308e-14+0j), (3.58046925441613e-14+0j), (3.1
5580894749700 75e-14+0j)]
(3) Sigma_Z average:(3.568464967962371e-14+0j)
```

图 13: $D_s = 6$ 结果

可以观察到：

- 对于 1-site 算法，当 $D_s = 4$ 时，其基态能量计算值在小数点后第 4 位相比精确对角化就出现了误差；$D_s = 6$ 时改善极少，仍然只有 5 位有效数字。而对于 2-site 算法，虽然相比 1-site 有一定改善（扩大了变分参数优化空间）。同时可以注意到，不论是 $D_s = 4$ 还是 $D_s = 6$，其能量、磁化计算都出现了不同程度上的左右不对称的现象（尤其是 2-site 算法），这说明体系的收敛精度并不好。

  为了改善结果，我们尝试使用 $D_s = 20$ 将变分的参数优化空间进一步增大，如图14，这一次，1-site 和 2-site 都有了 13 位以上有效数字。由此可知增大虚拟键维数以及是增加收敛精度的的有效方式；

- 在耗时上，我们能够明显看出精确对角化对于 $N_s = 10$ 的体系，其计算速度远比基于 vMPS 的 DMRG 慢。而对于 1-site 和 2-site，则是 2-site 略微慢于 1-site，这与我们的预期也是相符的。总之，我们从这个简单实例就可以看出，基于 vMPS 的 DMRG 算法已经显示出了相比精确对角化的优势，因此可以推知它应是一种很适合处理大规模（或强关联的）量子多体物理问题的数值方法。

```
==================== 2-site ====================
(1) Energy per Site:[-1.38635786 -1.38635786 -1.38635786 -1.38635786 -1.38635786 -1.38635786
 -1.38635786 -1.38635786 -1.38635786 -1.38635786]
(1) Energy average:-1.386357856256892
(2) Sigma_X per Site:[(0.7509204587249212+0j), (0.5943390379618406+0j), (0.5387627133479485+0j), (0.5154711808937797+0j), (0.505969900441348
3+0j), (0.5059699004428759+0j), (0.515471180894687+0j), (0.5387627133567512+0j), (0.5943390379596638+0j), (0.7509204587151079+0j)]
(2) Sigma_X average:(0.5810926582738924+0j)
(3) Sigma_Z per Site:[(2.736699755701011e-14+2.419654658675152e-32j), (3.019806626980426e-14+0j), (3.2862601528904634e-14+0j), (3.3140157285
06092e-14+0j), (3.302913498259841e-14+0j), (3.302913498259841e-14+0j), (3.197442310920451e-14+0j), (3.058664432842306e-14+0j), (3.1308289294
429414e-14+0j), (2.6145752229922437e-14+0j)]
(3) Sigma_Z average:(3.096412015679562e-14+2.419654658675152e-33j)
==================== 1-site ====================
(1) Energy per Site:[-1.38635786 -1.38635786 -1.38635786 -1.38635786 -1.38635786 -1.38635786
 -1.38635786 -1.38635786 -1.38635786 -1.38635786]
(1) Energy average:-1.386357856256847
(2) Sigma_X per Site:[(0.7509204587250672+0j), (0.5943390379616347+0j), (0.5387627133482363+0j), (0.5154711808939865+0j), (0.505969900441231
7+0j), (0.5059699004412398+0j), (0.5154711808939926+0j), (0.5387627133483128+0j), (0.5943390379617122+0j), (0.7509204587249799+0j)]
(2) Sigma_X average:(0.5810926582740394+0j)
(3) Sigma_Z per Site:[(9.764411501578252e-14-1.0302189002734178e-18j), (9.9253938401489e-14+0j), (1.0919043447188415e-13+0j), (1.25011112572
79263e-13+0j), (1.1735057370287905e-13+0j), (1.170175067954915e-13+0j), (1.563749130184533e-13+0j), (1.1368683772161603e-13+0j), (1.45716771
9820518e-13+0j), (-2.7755575615628914e-15+0j)]
(3) Sigma_Z average:(1.0784706461208772e-13-1.0302189002734179e-19j)
==================== ED ====================
(1) Energy per Site:-1.3863578562570982
(2) Sigma_X per Site:[(0.7509204587204126+5.551115123125783e-17j), (0.59433903796029 14+1.1102230246251565e-16j), (0.5387627133495049+5.55111
5123125783e-17j), (0.5154711808969572+5.551115123125783e-17j), (0.5059699004445618+0j), (0.505969900444562+0j), (0.5154711808969571-2.775557
5615628914e-17j), (0.5387627133495051-2.7755575615628914e-17j), (0.5943390379602906+0j), (0.7509204587204125+0j)]
(2) Sigma_X average:(0.5810926582743455+2.220446049250313 2e-17j)
(3) Sigma_Z per Site:[(3.2009117578724045e-14+0j), (3.538142001602296e-14+0j), (3.655409308578328e-14+0j), (3.710920459809586e-14+0j), (3.74
14515929867775e-14+0j), (3.7587988277465456e-14+0j), (3.71577685542321e-14+0j), (3.626959843572308e-14+0j), (3.58046925441613e-14+0j), (3.1
558089474970075e-14+0j)]
(3) Sigma_Z average:(3.568464967962371e-14+0j)
```

图 14: $D_s = 20$ 结果