# 量子物理计算方法选讲实验报告

### 李昊恩 2021010312

#### Tensor Network, Task 1

## 目录

1	引言		1
	1.1	模型	1
	1.2	投影纠缠对态	3
2 实验部分		部分	4
	2.1	代码实现	4
	2.2	结果	8

# 1 引言

本次实验的目的是利用 PEPS(投影纠缠对态, Projection Entangled Pair State)构建环面 Kitaev's toric code 模型的四重简并基态,并将张量网络进行缩并,验证它们的确是系统的基态.

#### 1.1 模型

本次实验研究的模型是二维 Kitaev's toric code,这是一个最简单的具有内禀拓扑序的自旋 模型,它具有解析解,因此其各种性质被研究得比较透彻,在拓扑量子计算、量子自旋液体等领 域具有重要的理论意义.

该模型考虑一个具有周期边界条件的二维格点(也就是一个环面格点),格点的每条边上具 有一个自旋,如图1所示:

该体系的哈密顿量为:

$$H = -\sum_{\text{Vertex}} \prod_{j \in \text{Vertex}} Z_j - \sum_{\text{Plaquette}} \prod_{j \in \text{Plaquette}} X_j, \tag{1}$$

如图2所示,近邻相互作用存在于一个方格四条边上的四个自旋(蓝色线),以及一个顶点所 连接的四个自旋(绿色线).



图 1: 二维 toric code 模型示意图



图 2: toric code 中的算符示意图

注意到每个顶点项  $\prod_{j \in Vertex} Z_j$ 和每个格点项  $\prod_{j \in Plaquette} X_j$ 要么不涉及任何共同自旋,要 么涉及两个共同自旋,再注意到  $X_i Z_j = (-1)^{\delta_{ij}} Z_j X_i$ 可知每个顶点项和格点项都是**交换的**,因 而具有共同的本征态.而每个顶点项和格点项只有 -1,1两个特征值,因此基态是"每个顶点项 和格点项都取特征值 +1 的态"(这时能量最小).对于顶点项,由于  $Z |\uparrow\rangle = |\uparrow\rangle$ ,  $Z |\downarrow\rangle = - |\downarrow\rangle$ , 因此基态的必要条件是,每个顶点处所连接的个自旋(图3中的红色线条)是偶数.另一方面,考 虑格点项,其作用是将一个格点的四条边自旋反转,如图3所示,因此可以想象,对某个态作用 所有格点项之后,不会改变 torus 红色线条构成的闭合回路的数目.因此,可以想象体系的基态 应该是所有红色线条组成的闭合回路的等权叠加态.



图 3: 基态是所有闭合回路的等权叠加态

该体系具有的拓扑不变量为横绕和纵绕 torus 的环的个数的奇偶性,因此其基态应是如图4所示的四种模式.在下一部分我们将说明如何用 PEPS 将四种基态构造出来.



图 4: 四种基态示意图

#### 1.2 投影纠缠对态

投影纠缠对态是二维张量网络的一种重要例子,其在解析解和数值上都有着重要的作用.特别地,在 Kitaev's toric code 模型中,利用投影纠缠对态可以便捷地将基态显式表示.如图5所示,Kitaev's toric code 的一个基态是如下图所示的 vertex tensor 和 bond tensor 缩并得到的结果.具体来说,每个顶点上都放置有 4 个虚拟自由度的张量,而每条边上则放置一个有 2 个虚拟自由度、1 个物理自由度的张量.



图 5: toric code 态的 PEPS 表示

对于每个 vertex 张量,只有四个虚拟指标中 "1" 的个数为偶数的张量元是 1,而其他张量 元都是 0;对于每个 bond 张量,则是只有当 2 个虚拟指标和 2 个物理指标都取 "0"或者全取 "1"的张量元为 1,其他张量元为 0,如图6所示.

$$0 \xrightarrow{0}{0} 0 = 1 \xrightarrow{0}{1}{1} = 0 \xrightarrow{1}{1}{0} = 0 \xrightarrow{1}{1}{1} = 1 \xrightarrow{1}{0}{0} = 1 \xrightarrow{0}{1}{0} = 0 \xrightarrow{0}{1}{1}{1} = 1 \xrightarrow{1}{1}{1}{1} = 1$$
$$0 \xrightarrow{0}{-0}{-0}{0} = 1 \xrightarrow{0}{-0}{1}{1}{1} = 0 \xrightarrow{0}{1}{1}{1}{1}{1} = 1$$

图 6: 基态的非零张量元

得到其中一个基态之后,如果还要再拿到具有其他拓扑不变量的简并基态,我们就需要运用 该体系的 Z<sub>2</sub> 对称性,具体来说,如图7所示,我们可以在 torus 上添加纵向、横向具有不同奇偶 性闭合的 *Z*-string 来得到其他的基态.



图 7: 不同的基态

这样,我们相当于用局域张量的性质还原了整个体系的拓扑性质,这样间接的表示要归功于 具有 Z<sub>2</sub>-对称性的 PEPS.

#### 2 实验部分

本实验选用 2×2 环面格点上的 toric code 模型,也就是具有 8 个自旋的体系.根据前面的 分析,只有当所有的 v 算符和 p 算符都取到其 +1 本征态时,toric code 态具有最低能量,因此基态能量应当为 –8,而且应该是 4 重简并.我们将用精确对角化的方法从数值上验证这一点. 另外,我们将 4 种环面 PEPS 进行缩并,验证它们的确是 4 重简并的基态.

#### 2.1 代码实现

首先定义四种算符:

```
1 Id = np.eye(2)
```

```
2 pX = np.array([[0,1.],[1.,0]])
```

3 pY = np.array([[0,-1.j],[1.j,0]])

```
4 pZ = np.array([[1.,0],[0,-1.]])
```

然后,对体系的哈密顿量进行精确对角化.注意在代码中对自旋采用的编号如图8所示.

```
1 L = 2
2 N_2d = 2*L**2
3 s = np.arange(N_2d)
4
5 ### Exact Diagonalization of Toric code model ###
6 print("### Exact Diagonalization of Toric code model ###")
7
8 #construct adjacent lists
9 plaqlist = [(0,2,3,4),(1,3,5,2),(4,6,7,0),(5,7,1,6)]
```

```
10 vertlist = [(0,2,1,6), (0,3,1,7), (2,4,6,5), (3,5,7,4)]
11
12 #define many-body operators
13 def many_body_operator(idx, oprts, size = N_2d):
14
      "Tensor product of `oprts` acting on indexes `idx`. Fills rest with Id."
      matrices = [Id if k not in idx else oprts[idx.index(k)] for k in range(size)]
16
      prod = matrices[0]
      for k in range(1, size):
17
          prod = np.kron(prod, matrices[k])
18
      return prod
19
20
21 Hamil = np.zeros([2**(N_2d), 2**(N_2d)])+0j
22
23 for i in range(L**2):
24
      Hamil += -1 * many_body_operator(plaqlist[i],[pX,pX,pX])
      Hamil += -1 * many_body_operator(vertlist[i],[pZ,pZ,pZ])
25
26
27 evals, evecs = np.linalg.eigh(Hamil)
28 lowest_16_evals = np.sort(evals)[:16]
29 index = np.argsort(evals)
30 gsv = evecs[index[0]]
31 print("The lowest 16 eigenvalues of the 2*2 toric code model is {}".format(
     lowest_16_evals))
```



#### 图 8: 自旋的编号

然后,我们构建 PEPS 并对 PEPS 进行缩并.为了得到另外三个基态,我们在 torus 上增加 Z-string,重新进行缩并.缩并时代码中采用的指标规定如图9所示.

根据图9中的缩并规则,代码实现如下:

```
1 ### Construct the 4-fold degenerated ground states using PEPS ###
2 print("### Construct the 4-fold degenerated ground states using PEPS ###")
3
4 v_nontrivial = [(0,0,0,0),(0,1,0,1),(1,0,1,0),(1,0,0,1),(1,1,0,0),(0,1,1,0)
    ,(0,0,1,1),(1,1,1,1)]
5 b_nontrivial = [(0,0,0),(1,1,1)]
```

```
7 vert = np.zeros([2,2,2,2])
8 bond = np.zeros([2,2,2])
10 for v_index in v_nontrivial:
11
      vert[v_index[0], v_index[1], v_index[2], v_index[3]] = 1
12 for b_index in b_nontrivial:
      bond[b_index[0], b_index[1], b_index[2]] = 1
13
14 # contract the tensors to obtain the toric code ground state
15 peps_tensor = np.einsum("abcd, efgh, ijkl, mnpq, dAj, lBb, cCe, kDm, hEn, qFf, gGa,
      pHi -> ABCDEFGH",\
16
                           vert, vert, vert, vert, bond, bond, bond, bond, bond, bond,
      bond, bond)
17
18 peps_vector = peps_tensor.reshape(2**N_2d)
19
20 # calculate the energy expectation value of the PEPS tensor and verify that it is
      indeed the ground state of the toric code model
21 energy = np.einsum("i,ij,j",peps_vector ,Hamil,peps_vector )/(np.sum(peps_vector**2)
22 print("Energy of the PEPS state with NO Z-string: {}".format(energy))
23
24 # adding closed Z-string around the torus horizontally
25 peps_tensor1 = np.einsum("abcd, efgh, ijkl, mnpq, dAj, lBb, rCe, sDm, hEn, qFf, gGa,
       pHi, cr, ks-> ABCDEFGH",\
26
                           vert, vert, vert, vert, bond, bond, bond, bond, bond, bond,
      bond, bond, pZ, pZ)
27
28 peps_vector1 = peps_tensor1.reshape(2**N_2d)
29
30 # calculate the energy expectation value of the PEPS tensor and verify that it is
      indeed the ground state of the toric code model
31 energy1 = np.einsum("i,ij,j",peps_vector1 ,Hamil,peps_vector1 )/(np.sum(peps_vector1
      **2))
32 print("Energy of the PEPS state with a Z-string AROUND the torus: {}".format(energy1
      ))
33
34 # adding closed Z-string across the torus vertically
35 peps_tensor2 = np.einsum("abcd, efgh, ijkl, mnpq, rAj, 1Bb, cCe, kDm, sEn, qFf, gGa,
       pHi, dr, hs -> ABCDEFGH",\
                           vert, vert, vert, vert, bond, bond, bond, bond, bond, bond,
36
      bond, bond, pZ, pZ)
37
```

```
2 实验部分
```

```
38 peps_vector2 = peps_tensor2.reshape(2**N_2d)
39
40 # calculate the energy expectation value of the PEPS tensor and verify that it is
      indeed the ground state of the toric code model
41 energy2 = np.einsum("i,ij,j",peps_vector2 ,Hamil,peps_vector2 )/(np.sum(peps_vector2
      **2))
42 print("Energy of the PEPS state with a Z-string ACROSS the torus: {}".format(energy2
      ))
43
44 # adding closed Z-string both around and across the torus horizontally and
      vertically
45 peps_tensor3 = np.einsum("abcd, efgh, ijkl, mnpq, uAj, lBb, rCe, sDm, vEn, qFf, gGa,
       pHi, cr, ks, du, hv -> ABCDEFGH", \
                          vert, vert, vert, bond, bond, bond, bond, bond, bond,
46
      bond, bond, pZ, pZ, pZ, pZ)
47
48 peps_vector3 = peps_tensor3.reshape(2**N_2d)
49
50 # calculate the energy expectation value of the PEPS tensor and verify that it is
      indeed the ground state of the toric code model
51 energy3 = np.einsum("i,ij,j",peps_vector3 ,Hamil,peps_vector3 )/(np.sum(peps_vector3
      **2))
52 print("Energy of the PEPS state with Z-strings both AROUND and ACROSS the torus: {}"
```

```
.format(energy3))
```



图 9: 左上角为 PEPS 张量缩并指标示意图,右上角为添加横向 Z-string 时的指标约定(只画 出有变化的部分,其他部分省略,下同),左下角为添加纵向 Z-string 时的指标约定,右下角为 添加纵、横 Z-string 时的指标

在代码实现中,我们计算了 PEPS 态的能量期望值  $\frac{\langle \psi | H | \psi \rangle}{\langle \psi | \psi \rangle}$ ,从而验证它们的确是基态.

7

2 实验部分

## 2.2 结果

运行以上代码的结果为:

```
1 ### Exact Diagonalization of Toric code model ###
2 The lowest 16 eigenvalues of the 2*2 toric code model is [-8. -8. -8. -8. -4. -4.
    -4. -4. -4. -4. -4. -4. -4. -4. -4.]
3 ### Construct the 4-fold degenerated ground states using PEPS ###
4 Energy of the PEPS state with NO Z-string: (-8+0j)
5 Energy of the PEPS state with a Z-string AROUND the torus: (-8+0j)
6 Energy of the PEPS state with a Z-string ACROSS the torus: (-8+0j)
7 Energy of the PEPS state with Z-strings both AROUND and ACROSS the torus: (-8+0j)
```

从精确对角化的结果来看,该模型的确具有 4 重简并基态,基态特征值为 –8. 另外,按照 图7的模式在环面上增加 Z-string 得到的四种态的能量期望值都是 –8,这说明它们的确都是基态.