量子物理计算方法选讲实验报告

李昊恩　2021010312

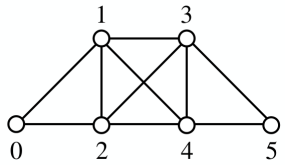
Exact Diagonalization, Task 1

目录

1 引言

考虑图1所示的具有 6 个位点的梯形拓展 Hubbard 模型，其哈密顿量为：

$$H = -t \sum_{\{i,j\},\sigma} (c^\dagger_{i,\sigma} c_{j,\sigma} + c^\dagger_{j,\sigma} c_{i,\sigma}) + U \sum_i n_{i\uparrow} n_{i\downarrow} - V \sum_{\{\{i,j\}\}} (n_{i\uparrow} + n_{i\downarrow})(n_{j\uparrow} + n_{j\downarrow}) - \mu \sum_{i,\sigma} n_{i,\sigma}, \quad (1)$$



$\{i,j\} \in (0,2),(1,3),(2,4),(4,5),(1,2),(3,4)$

$\{\{i,j\}\} \in (0,1),(2,3),(1,4),(3,5)$

图 1: 拓展 Fermi-Hubbard 模型

为了尽可能地减小我们处理的 Hilbert 空间的维数，我们在本实验中考虑体系的 U(1) 对称性。我们注意到上述定义中的 H 与粒子数算子是交换的，其中粒子数算子的定义为：

$$n_\sigma = \sum_i n_{i,\sigma}, \quad \text{其中} \quad \sigma \in \{\uparrow,\downarrow\}, \quad n_{i,\sigma} := c^\dagger_{i,\sigma} c_{i,\sigma}. \quad (2)$$

该算子能够计算一个量子态中自旋为上的费米子和自旋为下的费米子的总和。

回忆量子力学中的如下事实：对于具有一定对称性的系统（也就是一个与 H 交换的算子 R），哈密顿量 H 不会将 R 的不同特征空间 “混合” 在一起，因此，我们可以将这个较大的 Hilbert 空间分解为 R 的不同特征空间的直和：

$$\mathcal{H} = \bigoplus_{\lambda \in \sigma(R)} \mathcal{V}_R(\lambda). \tag{3}$$

其中 $\mathcal{H}$ 是量子态所在的 Hilbert 空间（全空间），$\sigma(R)$ 是 R 的谱集，$\mathcal{V}_R(\lambda)$ 是对应于 R 的特征值 λ 的特征空间。此时，每个 $\mathcal{V}_R(\lambda)$ 都是 H-不变子空间，也就是说，H 可以按照该直和表示为分块对角矩阵的形式，可以分别处理，这样大大缩小了问题的规模。

在本实验的 U(1) 对称性中，根据 $n_\sigma(\sigma \in \{\uparrow, \downarrow\})$ 的定义，我们可以将大的 Hilbert 空间分解为一些以给定的自旋上、下粒子数 $(N_\uparrow, N_\downarrow)$ 标记的子空间的之和，然后分别研究每个子空间。在本实验的 Fermi-Hubbard 模型中，根据哈密顿量的形式可知，(N, S_z) 在 U(1) 对称性下被保持，这里 $N = N_\uparrow + N_\downarrow$，$S_z = \frac{N_\uparrow - N_\downarrow}{2}$（分别为粒子数和总自旋 z 分量），由此可知 $(N_\uparrow, N_\downarrow)$ 和 (N, S_z) 是一一对应的。根据费米子的 Pauli 不相容原理，在以上具有 6 个位点的格点模型中，$N_\uparrow$ 和 $N_\downarrow$ 是从 0 到 6 变化的整数，因此总共有 $7 \times 7 = 49$ 个不同的 $(N_\uparrow, N_\downarrow)$ 对，因此共有 49 个不同的子空间，每个子空间的维数是 $\binom{N}{N_\uparrow} \times \binom{N}{N_\downarrow}$。

为了写下 H 的矩阵表示，我们需要为我们选用的基底定义一个总的排序。一个可行的方法是将费米子的占据位点转化为 $0-1$ 串的形式，然后通过找出它们对应的整数进行排序。对于本实验，我们采用一个通常被称为 **Lin Table** 的方法。具体来说，对于 $(N_\uparrow, N_\downarrow)$ 的子空间，我们知道 $\uparrow$ 费米子的指标（记为 i）的维数是 $\binom{N}{N_\uparrow}$，而 $\downarrow$ 费米子的指标（记为 j）则是 $\binom{N}{N_\downarrow}$。根据 Lin Table 的规定，我们可以为 (i, j) 建立总的指标 $k = \binom{N}{N_\downarrow} \cdot i + j$。此时，总指标的最大值则是 $\binom{N}{N_\uparrow} = \binom{N}{N_\downarrow} \cdot \left[\binom{N}{N_\uparrow} - 1\right] + [\binom{N}{N_\downarrow} - 1] = \binom{N}{N_\downarrow} \cdot \binom{N}{N_\uparrow} - 1$，与 $(N_\uparrow, N_\downarrow)$ 子空间的维数是一致的。

根据以上讨论，我们就可以写出 H 在各子空间的投影的矩阵表示。分别对子空间哈密顿量进行精确对角化，即可求出每个子空间的特征值、特征向量，将它们放在一起就可以求出整个系统的相应性质。该实验的具体实现留在第二部分讨论。

2 代码实现

首先，为了方便，我们定义一些对 $0-1$ 串的操作函数，如下：

```
#flip nth bit
def FlipBit(i, n):
    return i^(1<<n)
#read nth bit(from right to left)
def ReadBit(i, n):
    return (i&(1<<n))>>n
#count how many 1 bits
```

```
def PopCntBit(i):
    return bin(i).count("1")
#list of integers to list of bit patterns
def int_to_bin (list_of_int):
    list_of_bin = [bin(k) for k in list_of_int]
    return list_of_bin
```

为了定义每个子空间，我们通过以下代码找出所有具有特定 “1” 的数量的 6 位 $0-1$ 串：

```
#create list of integer numbers with certain number of "1" in their binary
    representations
int_val_with_particle_num = [[] for i in range(7)]
for n in range(2**6):
    for k in range(7):
        if PopCntBit(n) == k:
            int_val_with_particle_num[k].append(n)
```

2.1 矩阵的构造

观察哈密顿量的形式可知，只有第一项 $H_t = -t\sum_{\{i,j\},\sigma}(c^\dagger_{i,\sigma}c_{j,\sigma} + c^\dagger_{j,\sigma}c_{i,\sigma})$ 会提供非对角项，而后面几项只提供对角项。下面我们逐项进行分析。

对于 H_t 中的每一项 $-t(c^\dagger_{i,\sigma}c_{j,\sigma} + c^\dagger_{j,\sigma}c_{i,\sigma})$，只有当 i 位置和 j 位置的具有 σ 的自旋的占据数不同（即在其中一个位置有 σ 自旋的粒子占据，而另一个位置没有 σ 自旋的粒子占据）时，该算子作用在这样的态上才不得零。根据费米子产生湮灭算子的计算规则，该算符作用在这样的态上之后会将 σ-自旋粒子在 i 和 j 两个位点处的占据数交换，前面再根据交换反对称性补充一个相位因子。例如，对于 $c^\dagger_{i,\uparrow}c_{j,\uparrow} + c^\dagger_{j,\uparrow}c_{i,\uparrow}$，它作用在 $\underbrace{|\cdots 0(j\ \text{位置})\cdots 1(i\ \text{位置})\cdots\rangle}_{\uparrow\ \text{自旋}} \otimes \underbrace{|*\rangle}_{\downarrow\ \text{自旋}}$ 上的结果为：

$$(c^\dagger_{i,\uparrow}c_{j,\uparrow} + c^\dagger_{j,\uparrow}c_{i,\uparrow})\underbrace{|\cdots 0(j\ \text{位置})\cdots 1(i\ \text{位置})\cdots\rangle}_{\uparrow\ \text{自旋}} \otimes \underbrace{|*\rangle}_{\downarrow\ \text{自旋}} \tag{4}$$

$$= c^\dagger_{j,\uparrow}c_{i,\uparrow}\underbrace{|\cdots 0(j\ \text{位置})\cdots 1(i\ \text{位置})\cdots\rangle}_{\uparrow\ \text{自旋}} \otimes \underbrace{|*\rangle}_{\downarrow\ \text{自旋}} \tag{5}$$

$$= (-1)^{n_1}c^\dagger_{j,\uparrow}\left|\underbrace{\cdots 0\cdots}_{n_1\ \text{个 “1”}}0\cdots\right\rangle \otimes |*\rangle \tag{6}$$

$$= (-1)^{n_1+n_2}\left|\overbrace{\cdots}^{n_2\ \text{个 “1”}}1\cdots 0\cdots\right\rangle \otimes |*\rangle \tag{7}$$

在本实验中，我们默认 $\downarrow$ 自旋的占据数放在右边，$\uparrow$ 自旋的占据数放在左边，且位置的编号从右往左进行（最右边表示 0 位点，最左边表示 5 位点）。在这个例子中，可以发现 $c^\dagger_{i,\uparrow}c_{j,\uparrow} + c^\dagger_{j,\uparrow}c_{i,\uparrow}$ 作

用后，↑ 自旋的占据态由 $|\cdots 0\cdots 1\cdots\rangle$ 变为 $|\cdots 1\cdots 0\rangle$（记为 $|\mathrm{Up}_0\rangle$ 变为 $|\mathrm{Up}_1\rangle$；而 ↓ 对应的占据态（记为 $|\mathrm{Down}_0\rangle$）则不发生变化，由此可知 $\langle \mathrm{Up}_1|\otimes\langle \mathrm{Down}_0|\left[-t(c^\dagger_{i,\uparrow}c_{j,\uparrow}+c^\dagger_{j,\uparrow}c_{i,\uparrow})\right]|\mathrm{Up}_0\rangle\otimes|\mathrm{Down}_0\rangle = (-1)^{n_1+n_2}(-t)$，$-t(c^\dagger_{i,\uparrow}c_{j,\uparrow}+c^\dagger_{j,\uparrow}c_{i,\uparrow})$ 是一个只有上述矩阵元为 $(-1)^{n_1+n_2}(-t)$，其他矩阵元都是 0 的稀疏矩阵。

对于 i 位点没有占据，j 位点有占据，以及算子为 ↓ 自旋时，推导类似，此处略去。

对于第二项 $H_U = U\sum_i n_{i\uparrow}n_{i\downarrow}$，分析可知，只有当 i 位置的 ↑、↓ 自旋占据数均为 1 时，$n_{i\uparrow}n_{i\downarrow}$ 作用在该态时才不得零。此时，H_U 在矩阵对角线上该态对应位置贡献一个对角元 U。

对于第三项 $H_V = -V\sum_{\{\{i,j\}\}}(n_{i\uparrow}+n_{i\downarrow})(n_{j\uparrow}+n_{j\downarrow})$，我们注意到，只有当 i 位置和 j 位置都满足：↑ 和 ↓ 至少有一个占据数为 1 时，$(n_{i\uparrow}+n_{i\downarrow})(n_{j\uparrow}+n_{j\downarrow})$ 作用在该态上才不得 0。另外，当 $(i,\uparrow),(i,\downarrow),(j,\uparrow),(j,\downarrow)$ 中有且仅有 2 个为 1 时，提供的对角元为：

$$\langle\cdots 1\cdots 0\cdots|\otimes\langle\cdots 0\cdots 1\cdots|\left[-V(n_{i\uparrow}+n_{i\downarrow})(n_{j\uparrow}+n_{j\downarrow})\right]|\cdots 1\cdots 0\cdots\rangle\otimes|\cdots 0\cdots 1\cdots\rangle \tag{8}$$

$$=\langle\cdots 1\cdots 0\cdots|\otimes\langle\cdots 0\cdots 1\cdots|\left[-V(n_{i\uparrow}+n_{i\downarrow})\right]|\cdots 1\cdots 0\cdots\rangle\otimes|\cdots 0\cdots 1\cdots\rangle \tag{9}$$

$$=-V\langle\cdots 1\cdots 0\cdots|\cdots 1\cdots 0\cdots\rangle\langle\cdots 0\cdots 1\cdots|\cdots 0\cdots 1\cdots\rangle=-V \tag{10}$$

同样可以验证，当以上四者有且仅有 3 个、4 个为 1 时，提供的对角元分别为 $-2V$ 和 $-4V$。

对于第四项 $H_\mu = -\mu\sum_{i,\sigma}n_{i,\sigma}$，事实上可以验证 $H_\mu = -6\mu\mathrm{Id}$。或者也可以如下分析：当 $(i,\uparrow)$ 和 $(i,\downarrow)$ 有且仅有一个为 1 时，提供的对角元为 $-\mu$；当两者均为 1 时，提供的对角元为 -2μ。两种思路都可以构造出 H_μ 对应的对角矩阵。

综上所述，可以用第4部分代码来构造矩阵，这里先对 $N_\uparrow=2$，$N_\downarrow=4$ 的子空间进行计算，找出 6 个最小特征值，同时验证代码的正确性，然后再对所有子空间重复以上过程，从而找出整个系统的 20 个最小特征值。

2.2 基态自旋密度的计算

我们希望对体系的真实基态 $|\psi_0\rangle$ 计算其自旋密度，也就是对位点 $i\in\{0,1,2,3,4,5\}$ 和 $\sigma\in\{\uparrow,\downarrow\}$，分别计算粒子数算子 $n_{i,\sigma}$ 的期望值：

$$\langle n_{i,\sigma}\rangle = \frac{\langle\psi_0|n_{i,\sigma}|\psi_0\rangle}{\langle\psi_0|\psi_0\rangle}. \tag{11}$$

在本次实验中，我们输出的体系的特征向量是以长度为 D 的一维数组的形式表示的，具体来说，设某个子空间 $(N_\uparrow,N_\downarrow)$ 的基底为 $\{|j\rangle\}_{j=0}^{D-1}$（其中 $D=\binom{N}{N_\uparrow}\cdot\binom{N}{N_\downarrow}$ 指的是该子空间的维数），对这组基下的表示矩阵进行对角化，就得到由 D 维数组表示的若干特征向量。设其中一个特征向量的表示为 $(\phi_0,\cdots,\phi_{D-1})$，则该特征向量实际为：

$$|\phi\rangle=\sum_j\phi_j|j\rangle. \tag{12}$$

因此，要想真实基态 $|\psi_0\rangle=((\psi_0)_0,(\psi_0)_1,\cdots(\psi_0)_{D-1})$ 关于算子 $n_{i,\sigma}$ 的期望值，需要先找出它属于哪个子空间。这可以通过如下代码实现：

```
#we next calculate the density of up spin w.r.t. the true ground state of the system
indices = np.argsort(whole_system_evals)
true_ground_state = whole_system_evecs[indices[0]]
sector = sectors[indices[0]]
```

找出其位于哪个子空间后，就可以构造该子空间中 $n_{i,\sigma}$ 的矩阵表示（注意到这些子空间显然都是 $n_{i,\sigma}$ 的不变子空间，因而我们只要考虑 $n_{i,\sigma}$ 在该子空间下的矩阵即可），进而按照下面的式子计算期望值：

$$\langle n_{i,\sigma}\rangle = \frac{\langle\psi_0|n_{i,\sigma}|\psi_0\rangle}{\langle\psi_0|\psi_0\rangle} = \frac{\sum_j |(\psi_0)_j|^2 \langle j|n_{i,\sigma}|j\rangle}{\sum_j |(\psi_0)_j|^2} \tag{13}$$

对于 $\langle j|n_{i,\sigma}|j\rangle$，只有当 $|j\rangle$ 的 (i,σ) 位点占据数为 1 时，该矩阵元得 1，否则得 0，据此即可计算期望值。

3 结果

调整 $(t,U,V,\mu) = (1.0, 8.0, 0.4, 4.0)$，运行代码，输出如下：

```
=========Task 1:output=========
(1)_U(1)_sector_up2_down4:[-27.30624878 -27.00069745 -26.92490668 -26.77252547
    -26.66863714 -26.54837612]
(2)_lowest_20_eigenvalues:[-27.52865033 -27.30624878 -27.30624878 -27.30624878
    -27.00069745 -27.00069745 -27.00069745 -26.92490668 -26.92490668 -26.92490668
    -26.81123368 -26.77252547 -26.77252547 -26.77252547 -26.77252547 -26.77252547
    -26.67702648 -26.66863714 -26.66863714 -26.66863714]
(3)_Density_Up_Spin:[0.49979075 0.50139373 0.49881552 0.50139373 0.49881552
    0.49979075]
(3)_Density_Down_Spin:[0.49979075 0.50139373 0.49881552 0.50139373 0.49881552
    0.49979075]
```

观察最低的 20 的特征向量的分布情况，可以看到出现了一些简并的现象，这与 Fermi-Hubbard 模型的哈密顿量对称性以及几何对称性有关。

另外，观察不同自旋和不同位置的密度分布可以发现，该分布关于位置和自旋都有一定的对称性，这也是该模型对称性的体现。

4 附录：构造矩阵的代码实现

```
#we first consider the 2_up 4_down sector of this system
n_up = 2
n_down = 4
Ht,HU,HV,Hmu,HRt,HLt,HDU,HDV,HDmu = [],[],[],[],[],[],[],[],[]
up_arrow_bit_pattern = int_val_with_particle_num[n_up].copy()
down_arrow_bit_pattern = int_val_with_particle_num[n_down].copy()
```

```
up_arrow_dim = len(up_arrow_bit_pattern)
down_arrow_dim = len(down_arrow_bit_pattern)

Nl = up_arrow_dim * down_arrow_dim

for i0 in up_arrow_bit_pattern:
    for j0 in down_arrow_bit_pattern:

        #-t (off-diagonal) terms
        for iht in range(len(Hoplist_t)):
            Pos0 = Hoplist_t[iht][0]
            Pos1 = Hoplist_t[iht][1]
            if ReadBit(i0,Pos0) == 0 and ReadBit(i0,Pos1)==1:
                i1 = FlipBit(i0,Pos0)
                i1 = FlipBit(i1,Pos1)
                phase_factor = (-1)**(PopCntBit(PickBit(i0,Pos0+1,5-Pos0))+PopCntBit
    (PickBit(i0,Pos1+1,5-Pos1))-1)
                HLt.append(up_arrow_bit_pattern.index(i1)*down_arrow_dim +
    down_arrow_bit_pattern.index(j0))
                HRt.append(up_arrow_bit_pattern.index(i0)*down_arrow_dim +
    down_arrow_bit_pattern.index(j0))
                Ht.append(-t*phase_factor)
            if ReadBit(i0,Pos0) == 1 and ReadBit(i0,Pos1)==0:
                i1 = FlipBit(i0,Pos0)
                i1 = FlipBit(i1,Pos1)
                phase_factor =(-1)**(PopCntBit(PickBit(i0,Pos1+1,5-Pos1)) +PopCntBit
    (PickBit(i0,Pos0+1,5-Pos0)))
                HLt.append(up_arrow_bit_pattern.index(i1)*down_arrow_dim +
    down_arrow_bit_pattern.index(j0))
                HRt.append(up_arrow_bit_pattern.index(i0)*down_arrow_dim +
    down_arrow_bit_pattern.index(j0))
                Ht.append(-t*phase_factor)
            if ReadBit(j0,Pos0) == 0 and ReadBit(j0,Pos1)==1:
                j1 = FlipBit(j0,Pos0)
                j1 = FlipBit(j1,Pos1)
                phase_factor = (-1)**(PopCntBit(PickBit(j0,Pos0+1,5-Pos0))+PopCntBit
    (PickBit(j0,Pos1+1,5-Pos1))-1+2*PopCntBit(i0))
                HLt.append(up_arrow_bit_pattern.index(i0)*down_arrow_dim +
    down_arrow_bit_pattern.index(j1))
                HRt.append(up_arrow_bit_pattern.index(i0)*down_arrow_dim +
    down_arrow_bit_pattern.index(j0))
                Ht.append(-t*phase_factor)
```

```
            if ReadBit(j0,Pos0) == 1 and ReadBit(j0,Pos1)==0:
                j1 = FlipBit(j0,Pos0)
                j1 = FlipBit(j1,Pos1)
                phase_factor = (-1)**(PopCntBit(PickBit(j0,Pos1+1,5-Pos1)) +
PopCntBit(PickBit(j0,Pos0+1,5-Pos0))+2*PopCntBit(i0))
                HLt.append(up_arrow_bit_pattern.index(i0)*down_arrow_dim +
down_arrow_bit_pattern.index(j1))
                HRt.append(up_arrow_bit_pattern.index(i0)*down_arrow_dim +
down_arrow_bit_pattern.index(j0))
                Ht.append(-t*phase_factor)

        #U terms
        for iU in range(6):
            if ReadBit(i0,iU) == ReadBit(j0,iU) == 1:
                HDU.append(up_arrow_bit_pattern.index(i0)*down_arrow_dim +
down_arrow_bit_pattern.index(j0))
                HU.append(U)

        #-V terms
        for ihV in range(len(Hoplist_V)):
            Pos0V = Hoplist_V[ihV][0]
            Pos1V = Hoplist_V[ihV][1]
            if ReadBit(i0,Pos0V) + ReadBit(i0,Pos1V) + ReadBit(j0,Pos0V) + ReadBit(
j0,Pos1V) == 4:
                HDV.append(up_arrow_bit_pattern.index(i0)*down_arrow_dim +
down_arrow_bit_pattern.index(j0))
                HV.append(-4*V)
            if ReadBit(i0,Pos0V) + ReadBit(i0,Pos1V) + ReadBit(j0,Pos0V) + ReadBit(
j0,Pos1V) == 3:
                HDV.append(up_arrow_bit_pattern.index(i0)*down_arrow_dim +
down_arrow_bit_pattern.index(j0))
                HV.append(-2*V)
            if ReadBit(i0,Pos0V)+ReadBit(i0,Pos1V)+ReadBit(j0,Pos0V)+ReadBit(j0,
Pos1V) == 2 and ReadBit(i0,Pos0V)!= ReadBit(j0,Pos0V) :
                HDV.append(up_arrow_bit_pattern.index(i0)*down_arrow_dim +
down_arrow_bit_pattern.index(j0))
                HV.append(-1*V)

        #-mu terms
        for imu in range(6):
            if ReadBit(i0,imu)+ReadBit(j0,imu) == 1:
```

```
                HDmu.append(up_arrow_bit_pattern.index(i0)*down_arrow_dim +
    down_arrow_bit_pattern.index(j0))
                Hmu.append(-1*mu)
            if ReadBit(i0,imu)+ReadBit(j0,imu) == 2:
                HDmu.append(up_arrow_bit_pattern.index(i0)*down_arrow_dim +
    down_arrow_bit_pattern.index(j0))
                Hmu.append(-2*mu)

#diagonal matrix
Ham_off_d = sparse.coo_matrix((Ht,(HLt,HRt)),shape=(Nl,Nl)).tocsc()

#off-diagonal matrices
Ham_d_U = sparse.coo_matrix((HU,(HDU,HDU)),shape=(Nl,Nl)).tocsc()
Ham_d_V = sparse.coo_matrix((HV,(HDV,HDV)),shape=(Nl,Nl)).tocsc()
Ham_d_mu = sparse.coo_matrix((Hmu,(HDmu,HDmu)),shape=(Nl,Nl)).tocsc()

#subspace Hamiltonian matrix
Hamr = Ham_off_d + Ham_d_U + Ham_d_V + Ham_d_mu

H = Hamr.toarray()

#Diagonalize the Hamiltonian
evals, evecs = np.linalg.eig(H)
# Find the 6 lowest-lying eigenvalues
lowest_evals = np.sort(evals)[:6]

print("(1)_U(1)_sector_up2_down4:{}".format(lowest_evals))
```

如果设定 $(t, U, V, \mu) = (1.0, 4.0, 0.1, 2.0)$，则输出为：

```
(1)_U(1)_sector_up2_down4:[-15.2600026  -14.95060662 -14.70152294 -14.51119205
    -14.36481718 -14.07944009]
```

与讲义中的 Sample output 结果一致。