2025 年计算物理大作业

题目: 计算 PXP 模型的性质

姓名: 李梓瑞　学号:22377056

2025 年 5 月 18 日

目录

1 所选题目

12 题 (自己设计), 计算 PXP 模型的性质

2 对题目的分析

2.1 物理问题

在一般的平衡态统计物理中, 我们一般从准遍历假设出发考虑系统的热力学性质. 然而在现代凝聚态物理中, 我们直接从特定的量子多体模型出发可以得到遍历性破缺的态甚至是整个系统, 前者包括量子多体疤痕态 (quantum many-body scars), 后者包括多体局域化 (MBL) 和时间晶体 [1]. 与之相对的, 在量子多体层面的准遍历性假设称为本征态热化假说 (ETH). 特别的, 对于满足 ETH 的系统, 纠缠熵符合体积律, 能级分布符合 Possion 分布; 对于满足 MBL 的系统, 纠缠熵符合面积律, 能级分布符合 Wigner-Dyson 分布. 诸如 Anderson localization 这样的单粒子效应和 Yang-Baxter 方程等可积系统并不在我们的讨论中, 尽管他们的性质很相似.

考虑弱遍历性破缺, 如量子多体疤痕, 其部分本征态偏离热平衡行为, 导致动力学强烈依赖初始条件, 同时大部分态仍满足遍历性, 其能级分布介于 Possion 分布和 Wigner-Dyson 分布之间,quench 后纠缠熵的行为也不同于 MBL 的对数扩散增长. 本文将以封闭系统的 PXP 格点模型为例, 从数值上研究量子多体疤痕的性质 [2].

2.2 物理模型

本文物理上可以采用 Fibonacci Chain 来模拟 PXP 模型, 比如使用一维 Rydberg 原子来保证 hard core 约束.

2.3 数学模型

$$H = \sum_{i=1}^{L} P_i X_{i+1} P_{i+2} \tag{1}$$

其中 X 代表泡利 x 矩阵, 即自旋翻转;$P = (1 - Z)/2$ 代表投影算符, 即保证格点上为 spin-down 态, 这在物理上就是禁止相邻格点同时激发. 该模型具有两个对称性, 包括粒子-空穴对

称 $P=\prod_i Z_i$ 和空间反演对称. 本文中考虑周期性边界条件 (PBC), 因此还有空间平移对称性.

可以对上述原始模型(1)做向前散射近似 (FSA) 得到有效紧束缚模型, 这在计算上体现为 Lanszos 算法近似

$$H_{FSA}=\sum_{n=0}^{L}\beta_n(|n\rangle\langle n+1|+h.c.) \tag{2}$$

其中

$$\beta_n=\langle n+1|H^+|n\rangle\,,\quad H^+=\sum_{even}P_{i-1}\sigma_i^+P_{i+1}+\sum_{odd}P_{i-1}\sigma_i^-P_{i+1} \tag{3}$$

$$|0\rangle=|Z_2\rangle\,,\quad |n\rangle=(H^+)\,|0\rangle\,/\|(H^+)\,|0\rangle\,\| \tag{4}$$

$|Z_2\rangle$ 表示 1 维空间上有 Z_2 对称性的态. 该有效哈密顿量可以直接给出能带边界.

2.4 离散模型

本文考虑一维格点模型, 用 spin-1/2 模拟基态 (spin-down) 和激发态 (spin-up), 则约束条件为可以看成高主量子数等效费米子禁止相邻格点同时为激发态, 希尔伯特空间维度随斐波那契数列增长.

格点模型中电荷密度波态 (CDW) 表示为

$$|Z_2\rangle=|0101\cdots\rangle\,,\quad |Z_3\rangle\,,\quad \ldots \tag{5}$$

快速热化的全极化态表示为

$$|0\rangle=|0000\cdots\rangle \tag{6}$$

3 研究方法与计算原理

采用 python 程序和 quspin 包做精确对角化模拟, 直接对角化哈密顿量对应的整个矩阵. 其中构造哈密顿量的部分借助 numba 包使用 C 语言重写, 以保证计算速度, 减少内存占用. 如果想计算更大的格点还可以限制对称性后对子块做对角化, 详细程序见 quspin 官方文档.

4 程序框图

大致" 程序框图": 构造 Hilbert 空间的基 $\rightarrow$ 构造哈密顿量矩阵 $\rightarrow$ 对角化矩阵/对特定初态做演化

5 计算程序

```
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['Microsoft␣Yahei']
plt.rcParams['axes.unicode_minus'] = False

from quspin.operators import hamiltonian, exp_op
from quspin.tools.measurements import obs_vs_time
from quspin.tools.lanczos import lanczos_full
from quspin.basis.user import user_basis
from quspin.basis.user import (
    pre_check_state_sig_32,
    op_sig_32,
)
from numba import carray, cfunc
from numba import uint32, int32
import numpy as np
from scipy.optimize import curve_fit

N = 16 # site number

# operator: X, Y, Z
@cfunc(op_sig_32, locals=dict(s=int32, b=uint32))
def op(op_struct_ptr, op_str, ind, N, args):
    op_struct = carray(op_struct_ptr, 1)[0]
    err = 0
    ind = N - ind - 1
    s = (((op_struct.state >> ind) & 1) << 1) - 1
    b = 1 << ind
    #
    if op_str == 120:
```

```
            op_struct.state ^= b
        elif op_str == 121:
            op_struct.state ^= b
            op_struct.matrix_ele *= 1.0j * s
        elif op_str == 122:
            op_struct.matrix_ele *= s
        else:
            op_struct.matrix_ele = 0
            err = -1
        #
        return err
    #
    op_args = np.array([], dtype=np.uint32)

    # hard core check
    @cfunc(
        pre_check_state_sig_32,
        locals=dict(s_shift_left=uint32, s_shift_right=uint32),
    )
    def pre_check_state(s, N, args):
        mask = 0xFFFFFFFF >> (32 - N)
        s_shift_left = ((s << 1) & mask) | ((s >> (N - 1)) & mask)
        s_shift_right = ((s >> 1) & mask) | ((s << (N - 1)) & mask)
        #
        return (((s_shift_right | s_shift_left) & s)) == 0
    #
    pre_check_state_args = None

    # construct basis for Hilbert space
    maps = dict()
    op_dicts = dict(op=op, op_args=op_args)
```

```
    pre_check_state = (
        pre_check_state,
        pre_check_state_args,
    )
    basis = user_basis(
        np.uint32,
        N,
        op_dicts,
        allowed_ops=set("xyz"),
        sps=2,
        pre_check_state=pre_check_state,
        Ns_block_est=300000,
        **maps,
    )
    print(basis)

    # construct Hamiltonian
    h_list = [[1.0, i] for i in range(N)]
    static = [
        ["x", h_list],
    ]
    no_checks = dict(check_symm=False, check_pcon=False, check_herm=False)
    H = hamiltonian(static, [], basis=basis, dtype=np.float64, **no_checks)

    # construct initial state
    Ns = basis.Ns
    def create_state(Ns, dw_str, basis):
        i_0 = basis.index(dw_str)
        psi = np.zeros(Ns)
        psi[i_0] = 1.0
        return psi
```

```
92      assert N == 16 # following codes only for N=16 (although easy to extend)
93      psi_0 = create_state(Ns, "0000000000000000", basis) # |0>
94      psi_z2 = create_state(Ns, "0101010101010101", basis) # |Z_2>
95      psi_z3 = create_state(Ns, "0010010010010010", basis)
96      psi_z4 = create_state(Ns, "0001000100010001", basis)
97
98      # evolve states
99      start = 0.0
100     stop = 25.0
101     step = 0.1
102     num = round((stop-start)//step)
103     U = exp_op(H, a=-1j, start=start, stop=stop, num=num)
104     psi_0_t = U.dot(psi_0)
105     psi_z2_t = U.dot(psi_z2)
106     psi_z3_t = U.dot(psi_z3)
107     psi_z4_t = U.dot(psi_z4)
108
109     # get entropy (half chain)
110     t_array = np.linspace(start, stop, num)
111     def entropy(psi_t, H, basis, t_array):
112         obs_t = obs_vs_time(psi_t, t_array, dict(E=H), return_state=True)
113         Sent_t = basis.ent_entropy(obs_t["psi_t"], sub_sys_A=range(N//2))["
                Sent_A"]
114         return Sent_t
115     Sent_0 = entropy(psi_0_t, H, basis, t_array)
116     Sent_z2 = entropy(psi_z2_t, H, basis, t_array)
117     Sent_z3 = entropy(psi_z3_t, H, basis, t_array)
118     Sent_z4 = entropy(psi_z4_t, H, basis, t_array)
119     def f(x, k, b):
120         y = k*x + b
121         return y
```

```
122     para, cov = curve_fit(f, t_array, Sent_z2)
123     delta_Sent_z2 = Sent_z2 - f(t_array, *para)
124
125     # correlation <Z_i*Z_{i+1}>
126     Z_list = [[1.0, i, (i + 1) % N] for i in range(N)]
127     static_z2 = [["zz", Z_list]]
128     no_checks = dict(check_symm=False, check_pcon=False, check_herm=False)
129     correlation = hamiltonian(static_z2, [], basis=basis, dtype=np.float64,
            **no_checks)
130     obs_t = obs_vs_time(psi_z2_t, t_array, dict(E=H, O=correlation),
            return_state=True)
131     zz_t = obs_t["O"]/N
132
133     # overlap with eigen/FSA
134     (E_eigen, eigenstate) = H.eigh()
135     E_SA, eig_SA = H.eigsh(k=1, which='SA')
136     measurement1 = np.log(np.abs(np.dot(eigenstate, psi_z2))**2)
137     (E, V, Q_T) = lanczos_full(H, psi_z2, N)
138     eig_lanczos = np.transpose(np.dot(np.linalg.pinv(np.conj(Q_T)), V))
139     measurement_1_FSA = np.log(np.abs(np.dot(eig_lanczos, psi_z2))**2)
140     measurement2 = N*np.abs(np.dot(Q_T, eig_SA.ravel()))**2
141     measurement_2_FSA = N*np.abs(np.dot(Q_T, eig_lanczos[np.argmin(E)]))**2
142     measurement3 = N*np.abs(np.dot(Q_T, eigenstate[np.argsort(E_eigen)[3]]))
            **2
143     measurement_3_FSA = N*np.abs(np.dot(Q_T, eig_lanczos[np.argsort(E)[3]]))
            **2
144
145     # plot entropy for |0>, |Z_2>, |Z_3>, |Z_4>
146     plt.figure()
147     plt.xlabel(r'时间$t$/s')
148     plt.ylabel(r'半链纠缠熵$S$')
```

```
149     plt.plot(t_array, Sent_0, label=r"$|0\rangle$")
150     plt.plot(t_array, Sent_z2, label=r"$|Z_2\rangle$")
151     plt.plot(t_array, Sent_z3, label=r"$|Z_3\rangle$")
152     plt.plot(t_array, Sent_z4, label=r"$|Z_4\rangle$")
153     plt.grid(True)
154     plt.legend()
155     plt.savefig("entropy.png", dpi=500)
156 
157     # plot entropy and correlation
158     fig1, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(8, 6))
159     plt.xlabel(r'时间$t$/s')
160     ax1.set_ylabel(r'半链纠缠熵的振荡$\Delta␣S$')
161     ax1.plot(t_array, delta_Sent_z2, label="delta_Sent_z2")
162     ax1.legend()
163     ax1.grid(True)
164     ax2.set_ylabel(r'关联$\langle␣Z_iZ_{i+1}\rangle$')
165     ax2.plot(t_array, zz_t, label="zz_t")
166     ax2.legend()
167     ax2.grid(True)
168     plt.tight_layout()
169     plt.savefig("z2_t.png", dpi=500)
170 
171     # plot bond energy
172     plt.figure()
173     plt.ylim((-11,0))
174     plt.xlabel(r'能量特征值$E$')
175     plt.ylabel(r'$\ln{|\langle␣Z_2|\psi\rangle␣|^2}$')
176     plt.scatter(E_eigen, measurement1, label="ed")
177     plt.scatter(E, measurement_1_FSA, label="FSA")
178     plt.legend()
179     plt.savefig("measurement1.png", dpi=500)
```

```
180
181     # plot overlap
182     fig2, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(8, 6))
183     plt.xlabel(r'格点位置$n$')
184     ax1.set_ylabel(r'$|\langle␣n|\psi\rangle|^2␣L$')
185     ax1.plot(np.arange(0, N), measurement2, label="ed")
186     ax1.plot(np.arange(0, N), measurement_2_FSA, label="FSA")
187     ax1.legend()
188     ax1.grid(True)
189     ax2.set_ylabel(r'$|\langle␣n|\psi\rangle|^2␣L$')
190     ax2.plot(np.arange(0, N), measurement3, label="ed")
191     ax2.plot(np.arange(0, N), measurement_3_FSA, label="FSA")
192     ax2.legend()
193     ax2.grid(True)
194     plt.tight_layout()
195     plt.savefig("measurement2&3.png", dpi=500)
```

6 研究结果分析与讨论 (模拟结果)

结果如下面几张图.

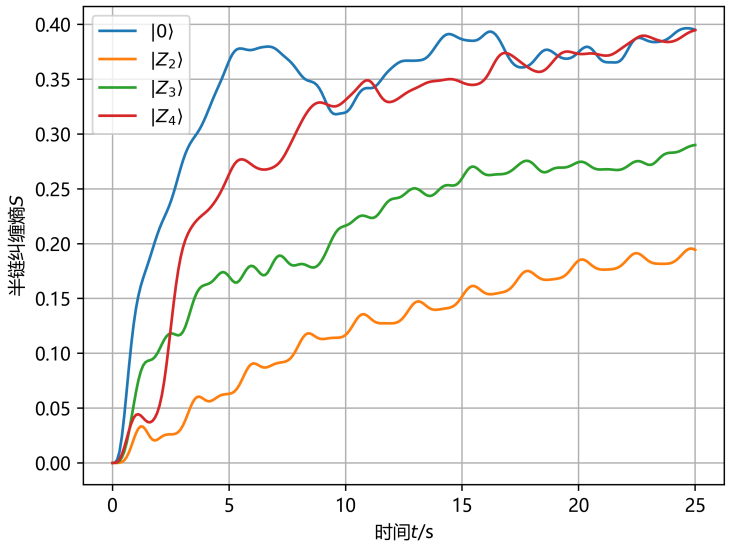

图 1: 纠缠熵曲线

从中可以看出热态 $|0\rangle$ 的纠缠熵增长最快,CDW 态 $|Z_2\rangle, |Z_3\rangle, |Z_4\rangle$ 都有振荡现象, 增长速度依次增大, 其中 $|Z_2\rangle$ 的振荡现象最明显, 说明将其作为典型疤痕态研究其他性质是合理的. 纠缠熵演化的振荡展现了态偏离热化的程度 [1].

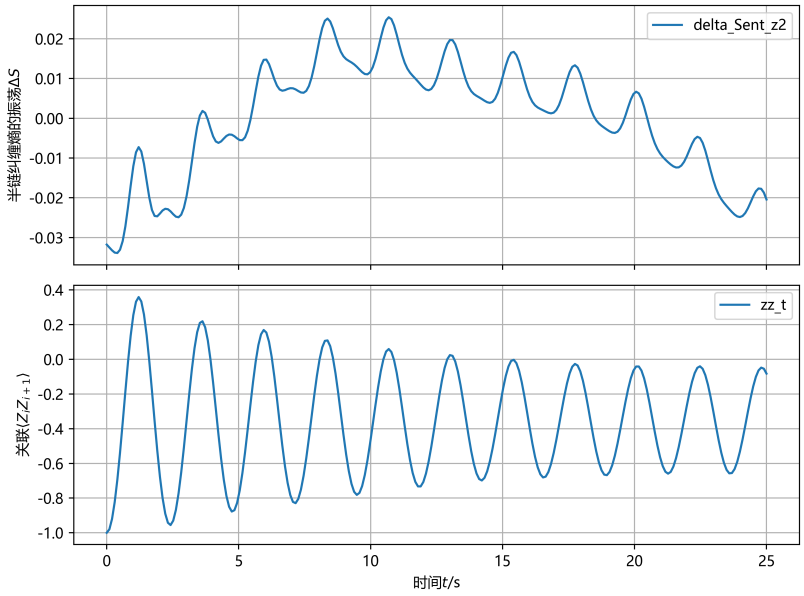

图 2: 纠缠熵与关联

疤痕态纠缠熵的振荡效应和关联的涨落基本同步, 这类似于 MBL 中由 quasi 运动积分支配的可观测量的行为.

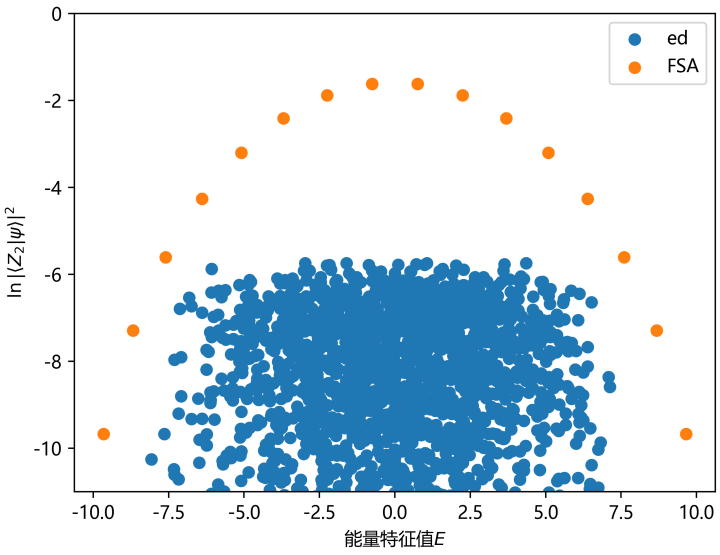

图 3: 能谱

FSA 成功给出能带, 与全能谱的结果吻合.

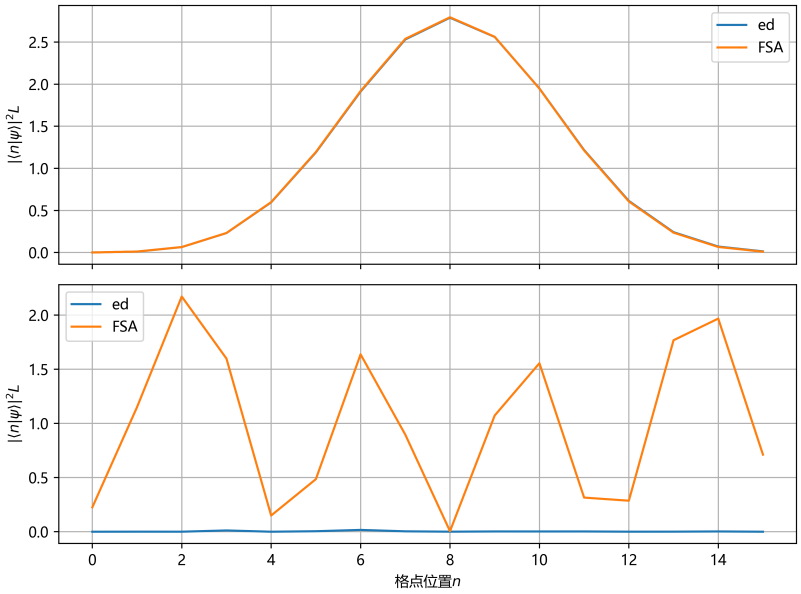

图 4: 波函数重叠: 上图是基态, 下图是激发态

FSA 对基态的近似效果要比激发态的效果好得多, 这也来源于 Lanzcos 算法只能准确计算最高和最低几个本征值. 不过,FSA 和精确解的定性行为基本一致.

7 总结

本文介绍了简单的遍历性破缺和以 PXP 模型为代表的弱遍历破缺理论, 并借助代码计算分析了 PXP 模型中的量子多体疤痕态的演化和能谱, 计算结果如图(1)(2)(3)(4). 结果给出了 PXP 模型中存在 CDW 态对应的疤痕态和其纠缠熵增长的振荡行为, 以及 FSA 给出的能带和全能谱, 也验证了 FSA 的合理性.

参考文献

[1] Dmitry A. Abanin, Ehud Altman, Immanuel Bloch, and Maksym Serbyn. Colloquium: Many-body localization, thermalization, and entanglement. *Rev. Mod. Phys.*, 91:021001, May 2019.

[2] C. J. Turner, A. A. Michailidis, D. A. Abanin, M. Serbyn, and Z. Papić. Weak ergodicity breaking from quantum many-body scars. *Nature Physics*, 14(7):745–749, Jul 2018.