VQE 求解哈密顿量

陈祥蔚

2023 年 12 月 8 日

1 实验原理

哈密顿量：用于描述物理或化学系统的能量的量，可以被表示成泡利字符串的线性组合的形式。

泡利字符串：由几种方阵的张量，方阵可以被理解成量子电路中作用在量子比特上的门。

量子电路：作用在量子比特上的电路。作用通过门体现。

作用过程：对于每一个量子比特，它有一个初态，经过一个门的作用，就会变成另外一个状态。

状态：经典电路中只有 0 和 1，但是量子电路中为两个向量，这两个向量可以放在一个单位球（布洛赫球）上来理解，经典电路中的 0 和 1 是两个极点，其他状态在球面上。每一个作用（门）就是把一个向量从球上的一个位置转换到另外一个位置。

原理：

从物理的角度来理解，哈密顿量是一种通过叠加不同粒子的状态，描述一个系统的能量的物理量，它包含的东西是基于系统中不同粒子不同状态进行各种运算所得到的。

根据我个人的物理知识，微观粒子接受能量的方式不是连续的，而是离散的，这也就导致每个系统有不同的能级，系统处在每个能级都会有不同的能量。而在每个能级时，系统中的每个粒子又都有自己的运行轨道，即一个状态，用于数学描述每个粒子的状态的工具是波函数。不同粒子的状态经过叠加之后就是一个系统的波函数。

从数学的角度来理解，哈密顿量就是在 n 个自由度上作用泡利矩阵的张量的线性组合，因此其最后被表示出来的就是一个 $2^n * 2^n$ 的矩阵，而大小为 n 的方阵在复数域上都有 n 个特征值和特征向量。方阵作用在这些特征向量所张成的子空间时只改变其长短（特征值倍）而不对其旋转。而量子电路中的门对于量子比特的作用就相当于在转换量子比特的状态，通过添加不同的门让不同的比特纠缠在一起，就可以模拟出不同的系统状态。

因此从这个角度就很好理解变分量子本征求解器在干什么。从现实的角度出发，一个系统因为只能接受离散的能量，所以只能处于特定的状态，而这些能量就是特征值，状态的波函数就相当于特征向量。

因此求解哈密顿量的本征值，就相当于找到哈密顿量在复数域上的一个特征值分解，分解后中间的矩阵的对角线上的特征值就是能级，而对应的特征向量组成的矩阵就是系统处于不同能级的波函数。

用量子电路来做这件事情，就是要通过电路对于比特的作用，模拟出波函数，而后通过模拟出来的波函数和要求解的哈密顿量的作用，观察哈密顿量是否对于模拟出的波函数进行了伸缩变换，如果没有的话根据变分原理确定的优化算法重新设置参数。等待总得损失函数收敛了就可以得出各个特征值即基态能量了。

2 实验步骤

包含以下步骤：

1、利用泡利字符串构造哈密顿量。

2、设计电路，这里使用可以利用 81 个参数模拟出任意量子态 uni3。

3、编写网络。

4、记录并输出结果，生成基态能量图。

关于电路：为了实现对于该哈密顿量的求解，我首先用了教程中提供的，在 Circuit 类中预设好的 uni3 电路，该电路可模拟全部的量子状态。其次，还使用了在 *A Practical Guide to Quantum Machine Learning and Quantum Optimization* 一书中（271）提供的一个电路，但是效果并不理想，有一点偏差，但是也放上来。

两种方法的代码源文件在文件夹中，附录中中也可看到，这里给出电路图。

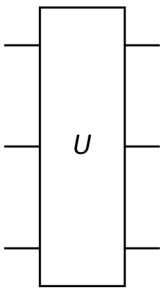

图 1: circuit with uni3.png

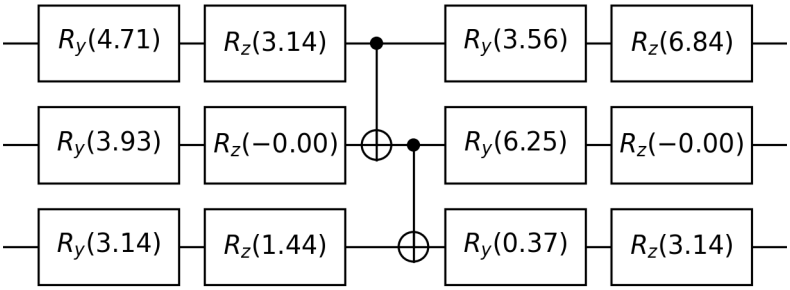

图 2: circuit.png

3 实验结果

Listing 1: result with uni3

```
iter: 300 loss: -27.3082
The estimated ground state energy is:  [1.7660027]
The theoretical ground state energy is:  -1.7691807
The estimated 1st excited state energy is: [1.7679551]
The theoretical 1st excited state energy is: -1.7691806554794312
The estimated 2nd excited state energy is: [1.526334]
The theoretical 2nd excited state energy is: -1.5264338254928589
The estimated 3rd excited state energy is: [1.5259106]
The theoretical 3rd excited state energy is: -1.5264338254928589
The estimated 4th excited state energy is: [-1.5261815]
The theoretical 4th excited state energy is: 1.5264338254928589
The estimated 5th excited state energy is: [-1.5260484]
The theoretical 5th excited state energy is: 1.5264338254928589
The estimated 6th excited state energy is: [-1.7661927]
The theoretical 6th excited state energy is: 1.7691806554794312
The estimated 7th excited state energy is: [-1.7677778]
The theoretical 7th excited state energy is: 1.7691806554794312
```

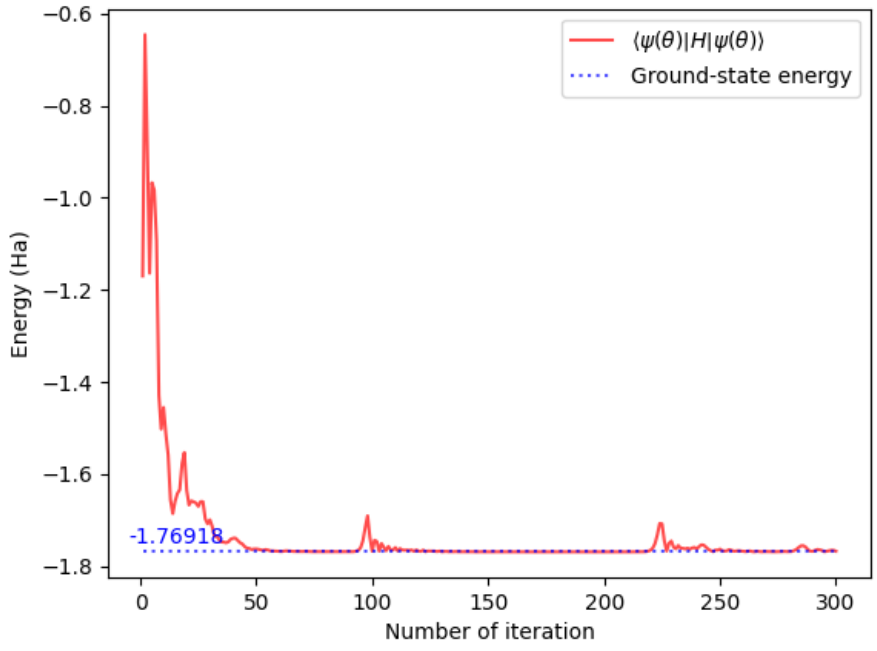

图 3: ground energy with uni3.png

Listing 2: result

```
iter: 300 loss: -13.7184
The estimated ground state energy is:  [1.6402833]
The theoretical ground state energy is:  -1.7691807
The estimated 1st excited state energy is: [1.6349993]
The theoretical 1st excited state energy is: -1.7691806554794312
The estimated 2nd excited state energy is: [-1.4826725]
The theoretical 2nd excited state energy is: -1.5264338254928589
The estimated 3rd excited state energy is: [-1.4905293]
The theoretical 3rd excited state energy is: -1.5264338254928589
The estimated 4th excited state energy is: [1.4905294]
The theoretical 4th excited state energy is: 1.5264338254928589
The estimated 5th excited state energy is: [1.4826726]
The theoretical 5th excited state energy is: 1.5264338254928589
The estimated 6th excited state energy is: [-1.634999]
The theoretical 6th excited state energy is: 1.7691806554794312
The estimated 7th excited state energy is: [-1.6402835]
The theoretical 7th excited state energy is: 1.7691806554794312
```

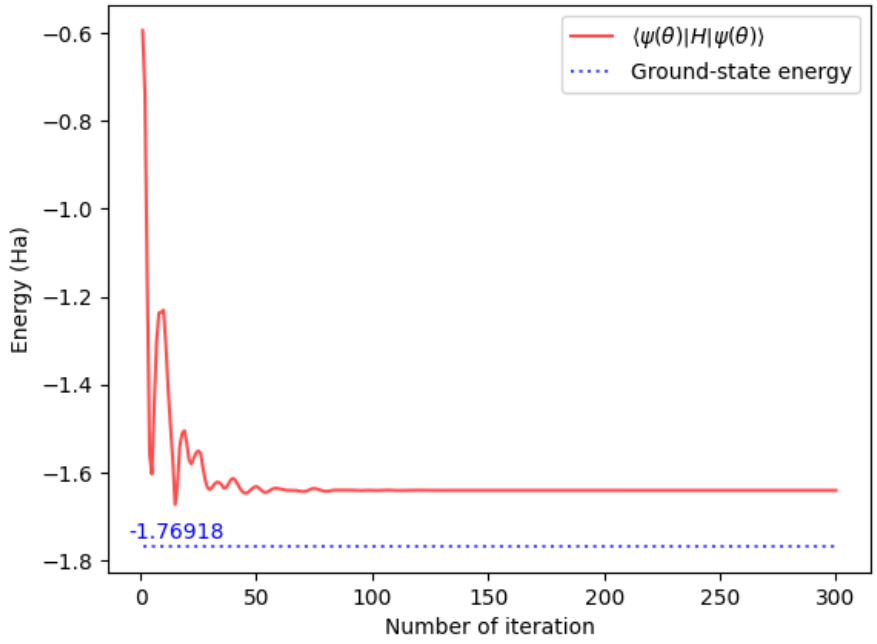

图 4: ground energ.png

4 问题总结

1、库中的一些包中的函数和最新的 numpy 不匹配。
creation.py 中 125 行中出现的 np.object，在现在的 numpy 包中已经没有 object 属性。
framework.py 中 1104 行中出现的 np.bool，在现在的 numpy 包中已经没有 bool 属性。
解决方案是改掉包里的函数即可。

2、教程里面说，优化器除了 Adam 之外还有另外两个选择，但事实上并不好用。分别使用 SGD 优化器和 RMSProp 优化器的具体结果放在附录 C 中。

3、在使用第二种电路的时候，计算出的基态能量总是差了一点，但是不知道是哪里出问题了，可能还是原理中有些没有搞清楚的地方。我考虑过电路的精简构造，因为 uni3 是一个可以模拟各个量子态的电路，里面有 81 个参数，如果想要减少电路的层数，就要考虑尽可能地从电路构造上删掉那些不对最终的酉矩阵有贡献的量。

$$\begin{bmatrix} \bar{X}_1 & \bar{X}_2 \\ \bar{Y}_1 & \bar{Y}_2 \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} \begin{bmatrix} X_1 & X_2 \\ Y_1 & Y_2 \end{bmatrix}$$

$$\begin{bmatrix} \bar{X}_1 AX_1 + \bar{Y}_1 BY_1 & \bar{X}_1 AX_2 + \bar{Y}_1 BY_2 \\ \bar{X}_2 AX_1 + \bar{Y}_2 BY_1 & \bar{X}_2 AX_2 + \bar{Y}_2 BY_2 \end{bmatrix}$$

$$\begin{cases} \bar{X}_1 AX_2 + \bar{Y}_1 BY_2 = 0 \\ \bar{X}_2 AX_1 + \bar{Y}_2 BY_1 = 0 \end{cases} \tag{1}$$

其中，$\begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}$ 是给定哈密顿量的矩阵形式，可以通过打印发现这个矩阵是一个 $8*8$ 的分块矩阵，这个分块矩阵左下角和右上角的 $4*4$ 矩阵全部都是零矩阵。而 $\begin{bmatrix} X_1 & X_2 \\ Y_1 & Y_2 \end{bmatrix}$ 则是电路的酉矩阵。我在考虑是否可以通过由此得到的方程组 (1) 来在求解之前就获得一些酉矩阵的性质，但是没有什么结果。

4、在实验过程中，我还想过要使用哈密顿量的构造一章中教程提供的方法，在这里面，教程中使用了ExpectVal()类来求损失函数，但是我在使用之后遇到了一些暂时没有办法解决的问题。
简单来说，在该类的调用过程中，会涉及到一个向前传播函数forward()，这个函数会调用state_vector.unitary_transformation_without_swapback()，而这个函数中包含这样一个部分：

```
seq_for_acted = qubit_idx + \
[x for x in qubit_sequence if x not in qubit_idx]
perm_map = pq.intrinsic._perm_of_list(qubit_sequence, seq_for_acted)...
```

可以看到，这段代码将qubit_idx和qubit_sequence中qubit_idx不包含的部分结合起来返回，然而，在这个调用这个函数的过程中，传输给qubit_idx的参数是pauli_site，而传输给qubit_sequence的参数是original_sequence，一个是泡利位点，一个是被操作的量子比特的原始序列。
一方面这个操作的本意是记录下来待作用序列和原始序列的对应关系，但是这里传递的参数让它表面上看来好像没什么意义，另外一方面这样也会造成参数传递的过程中溢出，从而使得下一步的维度不匹配无法继续运算。

5 附录 A

Listing 3: code with uni3

```
import numpy
from numpy import savez

import os
import matplotlib.pyplot as plt

import paddle
import paddle_quantum
from paddle import matmul
from paddle_quantum.ansatz import Circuit
from paddle_quantum.qinfo import pauli_str_to_matrix
from paddle_quantum.linalg import dagger
from paddle_quantum.state import zero_state

N = 3  # 量子比特数/量子神经网络的宽度
SEED = 50  # 固定随机种子

# 定义哈密顿量的项
pauli_str = [(1, 'I0, X1, X2'),
(0.5, 'I0, X1, Z2'),
(-0.2, 'Z0, X1, X2'),
(1.2, 'Z0, Z1, I2')]

# 生成 Hamilton 量的矩阵信息
complex_dtype = paddle_quantum.get_dtype()
H = pauli_str_to_matrix(pauli_str, N).astype(complex_dtype)


def U_theta(num_qubits: int) -> Circuit:
"""
U_theta
"""
# 按照量子比特数量/网络宽度初始化量子神经网络
cir = Circuit(num_qubits)

cir.universal_three_qubits([0, 1, 2])

# 返回量子神经网络的电路
return cir


class Net(paddle.nn.Layer):
```

```
43                 def __init__(self, num_qubits: int):
44                 super(Net, self).__init__()
45
46                 # 构造量子神经网络
47                 self.cir = U_theta(num_qubits)
48
49                 # 定义损失函数和前向传播机制
50                 def forward(self, H):
51
52                 # 计算损失函数
53                 U = self.cir.unitary_matrix()
54                 loss_struct = paddle.real(matmul(matmul(dagger(U), H), U))
55
56                 # 输入计算基去计算每个子期望值，相当于取 U^dagger*H*U 的对角元
57                 loss_components = []
58                 for i in range(len(loss_struct)):
59                 loss_components.append(loss_struct[i][i])
60
61                 # 最终加权求和后的损失函数
62                 loss = 0
63                 for i in range(len(loss_components)):
64                 weight = i
65                 # weight = 8 - i
66                 loss += weight * loss_components[i]
67                 print('the unitary matrix is ', U)
68                 return loss, loss_components, self.cir
69
70
71                 ITR = 300  # 设置训练的总迭代次数
72                 LR = 0.3  # 设置学习速率
73
74                 paddle.seed(SEED)
75
76                 # 我们需要将 numpy.ndarray 转换成 PaddlePaddle 支持的 Tensor
77                 hamiltonian = paddle.to_tensor(H)
78                 # print(H)
79                 # 确定网络的参数维度
80                 net = Net(N)
81
82                 # Adam 优化器来获得相对好的收敛，
83                 # 当然你可以改成 SGD 或者是 RMS prop.
84                 opt = paddle.optimizer.Adam(learning_rate=LR, parameters=net.parameters())
85                 # opt = paddle.optimizer.SGD(learning_rate=LR, parameters=net.parameters())
86                 # opt = paddle.optimizer.RMSProp(learning_rate=LR, parameters=net.parameters
                       ())
87
```

```
88                  # 定义初始态
89                  init_state = zero_state(N)
90
91                  # 记录优化结果
92                  summary_iter, summary_loss = [], []
93                  summary_loss_components = []
94
95                  # 优化循环
96                  for itr in range(1, ITR + 1):
97
98                  # 前向传播计算损失函数并返回估计的能谱
99                  loss, loss_components, cir = net(hamiltonian)
100
101                 # 在动态图机制下，反向传播极小化损失函数
102                 loss.backward()
103                 opt.minimize(loss)
104                 opt.clear_grad()
105
106                 # 更新优化结果
107                 summary_loss.append(loss.numpy())
108                 summary_loss_components.append(min(loss_components).numpy())
109                 summary_iter.append(itr)
110
111                 # 打印训练结果
112                 if itr % 10 == 0:
113                 print('iter:', itr, 'loss:', '%.4f' % loss.numpy()[0])
114
115                 def output_ordinalvalue(num):
116                 r"""
117                 Convert to ordinal value
118
119                 Args:
120                 num (int): input number
121
122                 Return:
123                 (str): output ordinal value
124                 """
125                 if num == 1:
126                 return str(num) + "st"
127                 elif num == 2:
128                 return str(num) + "nd"
129                 elif num == 3:
130                 return str(num) + "rd"
131                 else:
132                 return str(num) + 'th'
133
```

```
134
135                 for i in range(len(loss_components)):
136                 if i == 0:
137                 print('The estimated ground state energy is: ', loss_components[i].numpy())
138                 print('The theoretical ground state energy is: ', numpy.linalg.eigh(H)[0][i
                        ])
139                 else:
140                 print('The estimated {} excited state energy is: {}'.format(
141                 output_ordinalvalue(i), loss_components[i].numpy())
142                 )
143                 print('The theoretical {} excited state energy is: {}'.format(
144                 output_ordinalvalue(i), numpy.linalg.eigh(H)[0][i])
145                 )
146
147                 # 储存训练结果到 output 文件夹
148                 os.makedirs("output_with_uni3", exist_ok=True)
149                 savez("./output/summary_data_with_uni3", iter = summary_iter, energy=
                        summary_loss_components)
150
151                 cir.plot(
152                 save_path=r"D:\pythonstarter\baiduquantumnotbroken\circuit_with_uni3.png",
                        # 保存图像的路径
153                 dpi=300,  # 分辨率
154                 show=False,  # 是否显示图像
155                 output=True,  # 是否返回 Figure 实例
156                 scale=1.0,  # 缩放系数
157                 tex=False  # 是否使用 LaTeX 字体
158                 )
159
160                 result = numpy.load('./output/summary_data_with_uni3.npz')
161
162                 eig_val, eig_state = numpy.linalg.eig(H)
163                 min_eig_H = numpy.min(eig_val.real)
164                 min_loss = numpy.ones([len(result['iter'])]) * min_eig_H
165
166                 plt.figure(2)
167                 func1, = plt.plot(result['iter'], result['energy'],
168                 alpha=0.7, marker='', linestyle="-", color='r')
169                 func_min, = plt.plot(result['iter'], min_loss,
170                 alpha=0.7, marker='', linestyle=":", color='b')
171                 plt.xlabel('Number of iteration')
172                 plt.ylabel('Energy (Ha)')
173
174                 plt.legend(handles=[func1, func_min],
175                 labels=[
176                 r'$\left\langle {\psi \left( {\theta } \right)} '
```

```
            r'\right|H\left| {\psi \left( {\theta } \right)} \right\rangle $',
            'Ground-state energy',
            ], loc='best')
            #标记基态能量大小
            plt.text(-5, -1.75, f'{min_eig_H:.5f}', fontsize=10, color='b')
            plt.savefig(r"D:\pythonstarter\baiduquantumnotbroken\ground_energy_with_uni3
                .png")
```

6 附录 B

Listing 4: code

```
import numpy as np
from numpy import savez

import os
import matplotlib.pyplot as plt

import paddle
import paddle_quantum
from paddle import matmul
from paddle_quantum.ansatz import Circuit
from paddle_quantum.qinfo import pauli_str_to_matrix
from paddle_quantum.linalg import dagger
from paddle_quantum.state import zero_state

N = 3  # 量子比特数/量子神经网络的宽度
SEED = 50  # 固定随机种子

# 定义哈密顿量的项
pauli_str = [(1, 'I0, X1, X2'),
(0.5, 'I0, X1, Z2'),
(-0.2, 'Z0, X1, X2'),
(1.2, 'Z0, Z1, I2')]

# 生成 Hamilton 量的矩阵信息
complex_dtype = paddle_quantum.get_dtype()
H = pauli_str_to_matrix(pauli_str, N).astype(complex_dtype)
theta = np.full([6], np.pi)

def U_theta(num_qubits: int) -> Circuit:
"""
U_theta
"""
# 按照量子比特数量/网络宽度初始化量子神经网络

cir = Circuit(num_qubits)

cir.ry([0, 1, 2])
cir.rz([0, 1, 2])
cir.cnot([0, 1])
cir.cnot([1, 2])
cir.ry([0, 1, 2])
cir.rz([0, 1, 2])
```

```
    # 返回量子神经网络的电路
    return cir


    class Net(paddle.nn.Layer):
    def __init__(self, num_qubits: int):
    super(Net, self).__init__()

    # 构造量子神经网络
    self.cir = U_theta(num_qubits)

    # 定义损失函数和前向传播机制
    def forward(self, H):

    # 计算损失函数
    U = self.cir.unitary_matrix()
    #print(U)
    loss_struct = paddle.real(matmul(matmul(dagger(U), H), U))

    # 输入计算基去计算每个子期望值，相当于取 U^dagger*H*U 的对角元
    loss_components = []
    for i in range(len(loss_struct)):
    loss_components.append(loss_struct[i][i])

    # 最终加权求和后的损失函数
    loss = 0
    for i in range(len(loss_components)):
    weight = i
    # weight = 8 - i
    loss += weight * loss_components[i]

    return loss, loss_components, self.cir


    ITR = 300  # 设置训练的总迭代次数
    LR = 0.3  # 设置学习速率

    paddle.seed(SEED)

    # 我们需要将 numpy.ndarray 转换成 PaddlePaddle 支持的 Tensor
    hamiltonian = paddle.to_tensor(H)
    # print(H)
    # 确定网络的参数维度
    net = Net(N)

    # Adam 优化器来获得相对好的收敛，
```

```
89                  # 当然你可以改成 SGD 或者是 RMS prop.
90                  opt = paddle.optimizer.Adam(learning_rate=LR, parameters=net.parameters())
91                  # opt = paddle.optimizer.SGD(learning_rate=LR, parameters=net.parameters())
92                  # opt = paddle.optimizer.RMSProp(learning_rate=LR, parameters=net.parameters
                        ())
93
94                  # 定义初始态
95                  init_state = zero_state(N)
96
97                  # 记录优化结果
98                  summary_iter, summary_loss = [], []
99                  summary_loss_components = []
100
101                 # 优化循环
102                 for itr in range(1, ITR + 1):
103
104                 # 前向传播计算损失函数并返回估计的能谱
105                 loss, loss_components, cir = net(hamiltonian)
106                 print(cir)
107                 # 在动态图机制下, 反向传播极小化损失函数
108                 loss.backward()
109                 opt.minimize(loss)
110                 opt.clear_grad()
111
112                 # 更新优化结果
113                 summary_loss.append(loss.numpy())
114                 summary_loss_components.append(min(loss_components).numpy())
115                 summary_iter.append(itr)
116
117                 # 打印训练结果
118                 if itr % 10 == 0:
119                 print('iter:', itr, 'loss:', '%.4f' % loss.numpy()[0])
120
121
122                 def output_ordinalvalue(num):
123                 r"""
124                 Convert to ordinal value
125
126                 Args:
127                 num (int): input number
128
129                 Return:
130                 (str): output ordinal value
131                 """
132                 if num == 1:
133                 return str(num) + "st"
```

```
134                 elif num == 2:
135                 return str(num) + "nd"
136                 elif num == 3:
137                 return str(num) + "rd"
138                 else:
139                 return str(num) + 'th'
140
141
142                 for i in range(len(loss_components)):
143                 if i == 0:
144                 print('The estimated ground state energy is: ', loss_components[i].numpy())
145                 print('The theoretical ground state energy is: ', np.linalg.eigh(H)[0][i])
146                 else:
147                 print('The estimated {} excited state energy is: {}'.format(
148                 output_ordinalvalue(i), loss_components[i].numpy())
149                 )
150                 print('The theoretical {} excited state energy is: {}'.format(
151                 output_ordinalvalue(i), np.linalg.eigh(H)[0][i])
152                 )
153
154                 # 储存训练结果到 output 文件夹
155                 os.makedirs("output", exist_ok=True)
156                 savez("./output/summary_data", iter = summary_iter, energy=
                        summary_loss_components)
157
158                 cir.plot(
159                 save_path=r"D:\pythonstarter\baiduquantumnotbroken\circuit.png",  # 保存图像
                        的路径
160                 dpi=300,  # 分辨率
161                 show=False,  # 是否显示图像
162                 output=True,  # 是否返回 Figure 实例
163                 scale=1.0,  # 缩放系数
164                 tex=False  # 是否使用 LaTeX 字体
165                 )
166
167                 result = np.load('./output/summary_data.npz')
168
169                 eig_val, eig_state = np.linalg.eig(H)
170                 min_eig_H = np.min(eig_val.real)
171                 min_loss = np.ones([len(result['iter'])]) * min_eig_H
172
173                 plt.figure(2)
174                 func1, = plt.plot(result['iter'], result['energy'],
175                 alpha=0.7, marker='', linestyle="-", color='r')
176                 func_min, = plt.plot(result['iter'], min_loss,
177                 alpha=0.7, marker='', linestyle=":", color='b')
```

```
178                 plt.xlabel('Number of iteration')
179                 plt.ylabel('Energy (Ha)')
180
181                 plt.legend(handles=[func1, func_min],
182                 labels=[
183                 r'$\left\langle {\psi \left( {\theta } \right)} '
184                 r'\right|H\left| {\psi \left( {\theta } \right)} \right\rangle $',
185                 'Ground-state energy',
186                 ], loc='best')
187                 #标记基态能量大小
188                 plt.text(-5, -1.75, f'{min_eig_H:.5f}', fontsize=10, color='b')
189                 plt.savefig(r"D:\pythonstarter\baiduquantumnotbroken\ground_energy.png")
```

7 附录 C

Listing 5: result with SGD

```
iter: 300 loss: 1.0785
The estimated ground state energy is:  [0.00114252]
The theoretical ground state energy is:  -1.7691807
The estimated 1st excited state energy is: [-0.67215115]
The theoretical 1st excited state energy is: -1.7691806554794312
The estimated 2nd excited state energy is: [0.48083964]
The theoretical 2nd excited state energy is: -1.5264338254928589
The estimated 3rd excited state energy is: [0.8348018]
The theoretical 3rd excited state energy is: -1.5264338254928589
The estimated 4th excited state energy is: [-0.46377513]
The theoretical 4th excited state energy is: 1.5264338254928589
The estimated 5th excited state energy is: [-0.65335745]
The theoretical 5th excited state energy is: 1.5264338254928589
The estimated 6th excited state energy is: [-0.09890926]
The theoretical 6th excited state energy is: 1.7691806554794312
The estimated 7th excited state energy is: [0.5714093]
The theoretical 7th excited state energy is: 1.7691806554794312
```

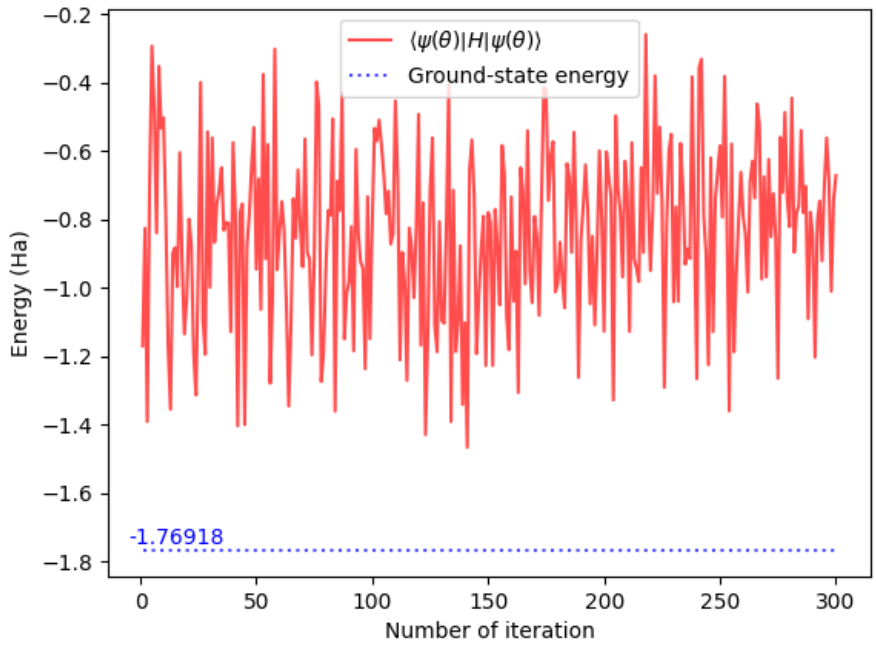

图 5: ground energ with SGD.png

Listing 6: result with RMSProp

```
iter: 300 loss: -0.6743
The estimated ground state energy is:  [-0.6664501]
The theoretical ground state energy is:  -1.7691807
The estimated 1st excited state energy is: [0.37210482]
The theoretical 1st excited state energy is: -1.7691806554794312
The estimated 2nd excited state energy is: [-0.18103871]
The theoretical 2nd excited state energy is: -1.5264338254928589
The estimated 3rd excited state energy is: [0.811299]
The theoretical 3rd excited state energy is: -1.5264338254928589
The estimated 4th excited state energy is: [0.67934734]
The theoretical 4th excited state energy is: 1.5264338254928589
The estimated 5th excited state energy is: [-0.90233004]
The theoretical 5th excited state energy is: 1.5264338254928589
The estimated 6th excited state energy is: [0.5334128]
The theoretical 6th excited state energy is: 1.7691806554794312
The estimated 7th excited state energy is: [-0.6463458]
The theoretical 7th excited state energy is: 1.7691806554794312
```

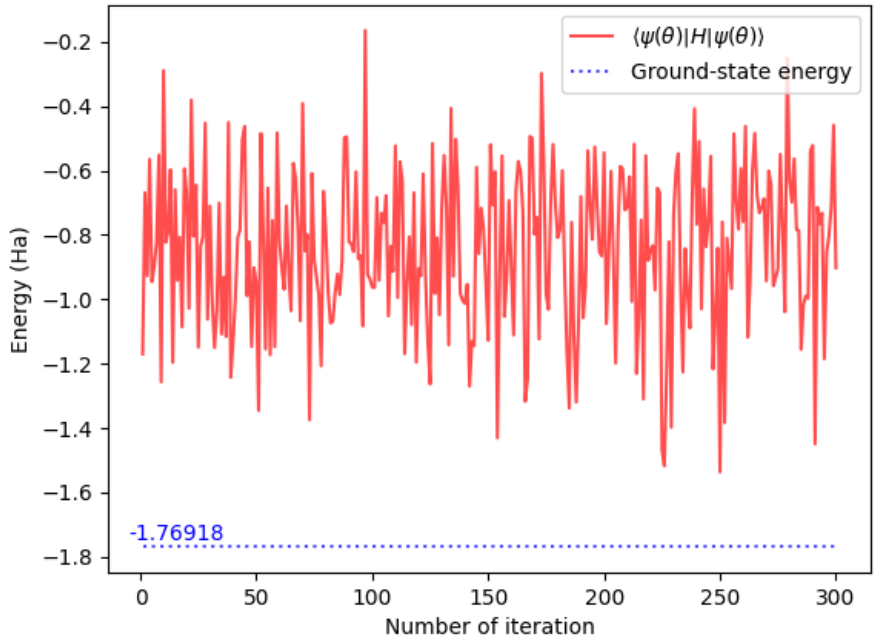

图 6: ground energ with RMSProp.png

8 附录 D

Listing 7: def forward() in ExpectVal()

```
if self.backend == Backend.StateVector:
output_state, seq_for_acted = state_vector.
    unitary_transformation_without_swapback(
state_data, [matrix], pauli_site, num_qubits, origin_seq)
perm_map = _perm_of_list(seq_for_acted, origin_seq)
output_state = _base_transpose(output_state, perm_map).reshape([-1, 2 **
    num_qubits, 1])
state_data_bra = paddle.conj(state_data.reshape([-1, 1, 2 ** num_qubits]))
batch_values = paddle.squeeze(paddle.real(paddle.matmul(state_data_bra,
    output_state)), axis=[-2, -1]) * \
self.coeffs[i]
expec_val_each_term.append(batch_values)
expec_val += batch_values
```

Listing 8: def seq_for_acted in state_vector.unitary_transformation_without_swapback()

```
seq_for_acted = qubit_idx + \
[x for x in qubit_sequence if x not in qubit_idx]
perm_map = pq.intrinsic._perm_of_list(qubit_sequence, seq_for_acted)...
```