1.5.3 变分量子本征求解器（VQE）求解 $2^n \times 2^n$ 厄米矩阵的最小本征值

变分量子本征求解器 (Variational Quantum Eigensolver, VQE)[1, 2] 是第一个被提出的变分量子算法。该算法的提出是为了求解给定哈密顿量的基态和相应的本征值[1]。在 VQE 算法提出以前，求解给定哈密顿量基态的量子算法一般基于绝热态制备和量子相位估计 [3, 4]，在实际应用中，这两种算法所需要的电路深度都超出了目前 NISQ 时代的量子硬件水平。因此，Alberto Peruzzo 在 2014 年提出变分量子本征求解器来在当前的量子计算机上求解分子体系哈密顿量的基态能量 [1]。

与 QPE 算法相比，VQE 算法需要更少的量子门个数和更短的相干时间。它以多项式级的线路重复次数换取所需的量子比特相干时间的减少。因此，它更适合于 NISQ 时代。在近期的量子计算机的应用上，有着较大的应用前景和优势。

VQE 的基本思想是通过制备参数化的量子线路 $U(\vec{\theta})$ 来构造一个基态的试探函数[2] $|\psi(\vec{\theta})\rangle$，使用经典优化器不断调节试探函数的参数，使哈密顿量的期望值 $E(\vec{\theta})$ 最小化，得到的最小能量，即所求的基态能量 E_0。

$$|\psi(\vec{\theta})\rangle = U(\vec{\theta})|0\rangle \tag{1.1}$$

$$E(\vec{\theta}) = \langle\psi(\vec{\theta})|\hat{H}|\psi(\vec{\theta})\rangle \geq E_0 \tag{1.2}$$

事实上，VQE 的量子子程序等价于基于参数 $\hat{\theta}$ 的集合来制备一个量子态，并在适当的基上进行一系列的测量。在这个算法中参数化量子态的制备是比较困难的，所以参数量子线路的选择对算法的性能的影响较大。

由于多体物理中粒子哈密顿量的矩阵描述是厄米矩阵，哈密顿量的基态能量直接对应厄米矩阵的最小本征值。在这一节中，为了便于读者理解 VQE 算法，我们将这一算法从物理背景中提取出来，简化为 VQE 求解 $2^n \times 2^n$ 厄米矩阵的最小本征值。

1.5.3.1 泡利基展开厄米矩阵

我们在哈密顿量模拟一节中提到了，任何一个 $2^n \times 2^n$ 厄米算符都可以用泡利算符的线性组合来表示。比如任意一个 2×2 的厄米矩阵可以写为如下的形式：

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & b \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ c & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & d \end{bmatrix}$$

其中，

$$\begin{bmatrix} a & 0 \\ 0 & 0 \end{bmatrix} = \frac{a}{2}(I+Z), \begin{bmatrix} 0 & b \\ 0 & 0 \end{bmatrix} = \frac{b}{2}(X+iY), \begin{bmatrix} 0 & 0 \\ c & 0 \end{bmatrix} = \frac{c}{2}(X-iY), \begin{bmatrix} 0 & 0 \\ 0 & d \end{bmatrix} = \frac{d}{2}(I-Z)$$

[1] VQE 用于多电子体系总能量的求解时，本征方程为定态薛定谔方程，算符就是系统的哈密顿算符，最小本征态即为系统的基态，最小本征值即为系统的基态能量。

[2] 又称试验态

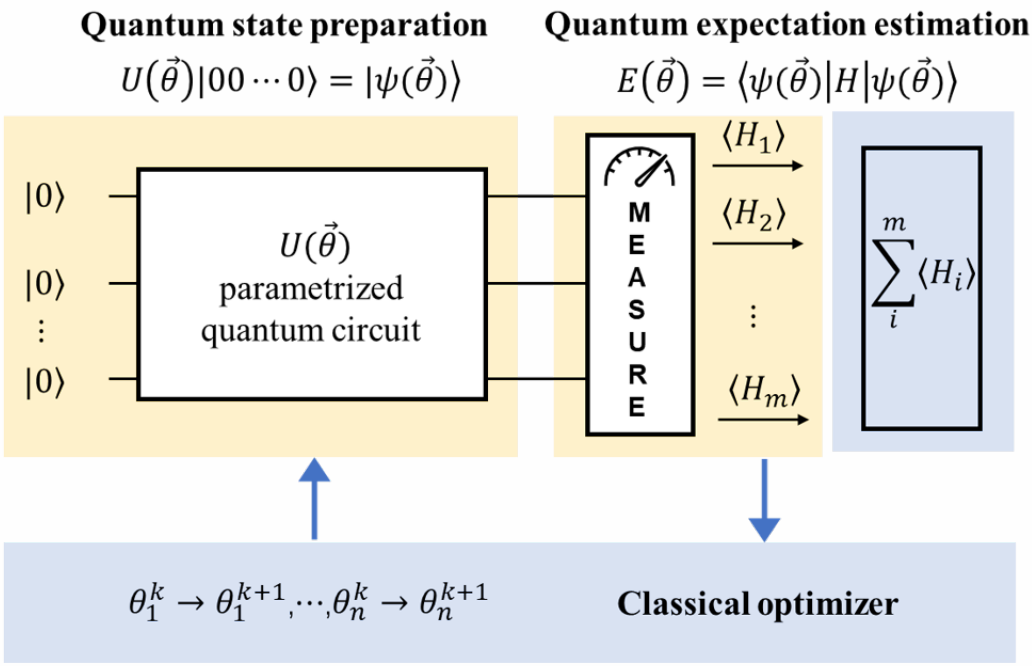

图 1.1: VQE 整体流程图

整理后可得，

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \frac{b+c}{2}X + \frac{b-c}{2}iY + \frac{a+d}{2}I + \frac{a-d}{2}Z$$

按照这种方式，任意一个 $2^n \times 2^n$ 的厄米矩阵可以由泡利矩阵 (X, Y, Z, I) 的线性组合表示。

使用 pyQPanda 实现泡利基展开厄米矩阵

pyQPanda 中提供这种分解的接口 matrix_decompose_hamiltonian，输入 $2^n \times 2^n$ 的厄米矩阵，即可得到输出的泡利算符类。下面我们展示通过这个接口分解矩阵

$$\begin{bmatrix} 2 & 1 & 4 & 2 \\ 1 & 3 & 2 & 6 \\ 4 & 2 & 2 & 1 \\ 2 & 6 & 1 & 3 \end{bmatrix}$$

示例代码与结果如下：

Code Listing 1.1: using matrix_decompose_hamiltonian API with pyQPanda

```
import pyqpanda as pq
import numpy as np

if __name__ == "__main__":

    mat = np.array([[2,1,4,2],[1,3,2,6],[4,2,2,1],[2,6,1,3]])
    f = pq.matrix_decompose_hamiltonian(mat)
    print(f)
```

```
9
10
11  {
12  "" : 2.500000,
13  "X0" : 1.000000,
14  "X1" : 5.000000,
15  "X0 X1" : 2.000000,
16  "Z0 X1" : -1.000000,
17  "Z0" : -0.500000
18  }
```

把厄米矩阵表示为泡利算符直积的和之后，项数是呈指数增加的，但是这些算符不需要放在线路上演化，而是直接作用在最后的测量中，通过量子期望估计来得到试验态的本征值[3]。

1.5.3.2 试验态的制备

为了获得与本征态相近的试验态，我们需要通过含参数的量子线路来制备试验态。并且理论上，假设的试验态与最终的本征态越接近，越有利于后面得到正确的最小本征值。在 VQA 中我们也介绍过，这个含参数的量子线路称作 ansatz。应用在 VQE 上 ansatz 主要分为两大类，一类是化学启发的 ansatz，如酉正耦合簇 (unitary coupled-cluster, UCC)，另一类是基于量子计算机硬件特性构造的 ansatz，即 Hardware-Efficient ansatz。这里我们主要介绍后者。

Hardware-Efficient ansatz 是变分量子算法中常用的通用 ansatz，当我们对需要解决的问题知之甚少时，可以使用一些基础量子门来制备参数化量子线路。Hardware-Efficient ansatz 直接将 $|00\cdots0\rangle$ 演化成纠缠态，不经过初态的构建。这种 ansatz 的量子线路的结构一般包括许多重复、密集的模块，每个模块由特定类型的含参数的量子门构成，这些量子门在目前含噪声的中型量子器件 (NISQ) 上更容易实现，因为其更能满足现有量子计算机的特点——较短的量子比特相干时间与受限的量子门结构。这里我们使用一个制备 Hardware-Efficient ansatz 量子线路的模板，该量子线路中包括每个量子比特上的单比特旋转门 (RZ RX RZ)，与相邻两比特的纠缠门 (受控 RY 门)。以四量子比特为例，量子线路图如下所示：

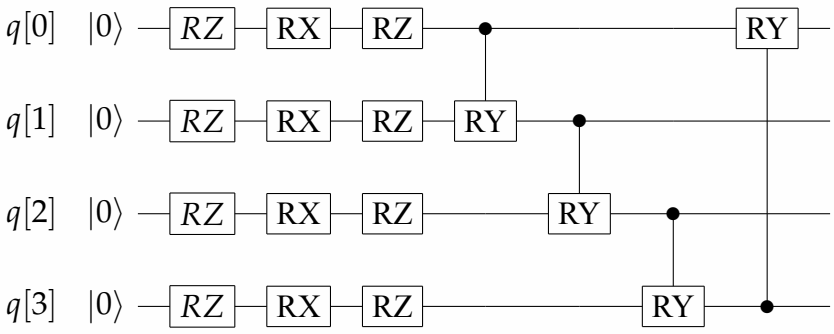

图 1.2: Hardware-Efficient ansatz

[3]此处我们为了演示 vqe 算法流程需要将矩阵表示为泡利基的形式，应用在量子化学的哈密顿量算符并不通过这种方式分解。

使用 pyQPanda 实现 Hardware-Efficient ansatz 的构建

Code Listing 1.2: build Hardware-Efficient ansatz with pyQPanda

```
import pyqpanda as pq

def prepare_HE_ansatz(qlist, para):
    '''
    prepare Hardware-Efficient ansatz, return a QCircuit

    Args:
        qlist(QVec): qubit list
        para(list[float64]): initial parameter. The number of parameters is four times qubit number.
    Return:
        quantum circuit(QCircuit)
    '''
    circuit = pq.QCircuit()
    qn = len(qlist)
    for i in range(qn):
        circuit.insert(pq.RZ(qlist[i], para[4*i]))
        circuit.insert(pq.RX(qlist[i], para[4*i+1]))
        circuit.insert(pq.RZ(qlist[i], para[4*i+2]))

    for j in range(qn-1):
        ry_control = pq.RY(qlist[j+1], para[4*j+3]).control(qlist[j])
        circuit.insert(ry_control)

    ry_last = pq.RY(qlist[0], para[4*qn-1]).control(qlist[qn-1])
    circuit.insert(ry_last)
    return circuit
```

1.5.3.3 量子期望估计

在上一节中，我们演示了使用 Hardware-Efficient ansatz 制备出试验态 $\left|\psi(\vec{\theta})\right\rangle$，在这一步后，还需要通过测量操作来得到试验态在不同泡利基上的期望，这一步叫做量子期望估计。在 VQE 算法中，每个子项期望的测量是在量子处理器上进行的，而经典处理器则负责对各个期望进行求和。

$$E = \overbrace{\sum \underbrace{h_{\alpha}^{i}\langle\psi|\sigma_{\alpha}^{i}|\psi\rangle}_{\text{QPU}} + \sum \underbrace{h_{\alpha\beta}^{ij}\langle\psi|\sigma_{\alpha}^{i}\otimes\sigma_{\beta}^{j}|\psi\rangle}_{\text{QPU}} + \ldots}^{\text{CPU}} \tag{1.3}$$

其中 σ 是泡利算符，$\alpha,\beta \in (X,Y,Z,I)$，而 i,j 则表示算符子项所作用的子空间，h 是实数。

假设某一个厄米算符为 $\hat{\mathrm{H}}$ 且它最终可以展开成这种形式：

$$\hat{\mathrm{H}}_P = \hat{\mathrm{H}}_1 + \hat{\mathrm{H}}_2 + \hat{\mathrm{H}}_3 = I^{\otimes 2} + \sigma_z^0 \otimes \sigma_z^1 + \sigma_x^0 \otimes \sigma_y^1 \tag{1.4}$$

在该式中，所有子项系数 h 均是 1。并假设所制备出的试验态为这种形式：

$$|\psi\rangle = a|00\rangle + b|01\rangle + c|10\rangle + d|11\rangle \tag{1.5}$$

其中，a^2、b^2、c^2、d^2 分别是指测量试验态时，坍塌到 $|00\rangle$、$|01\rangle$、$|10\rangle$、$|11\rangle$ 的概率 P_S，将厄米算符的各个子项 $\hat{\mathrm{H}}_1$、$\hat{\mathrm{H}}_2$、$\hat{\mathrm{H}}_3$ 分别作用于试验态上，可以依次得到期望 E(1)、E(2)、E(3)。

$$\begin{aligned} \mathrm{E}(1) &= \langle \psi | \hat{\mathrm{H}}_1 | \psi \rangle \\ \mathrm{E}(2) &= \langle \psi | \hat{\mathrm{H}}_2 | \psi \rangle \\ \mathrm{E}(3) &= \langle \psi | \hat{\mathrm{H}}_3 | \psi \rangle \end{aligned} \tag{1.6}$$

下面我们将以 E(1)、E(2)、E(3) 为例，详细介绍 VQE 算法是如何构造线路来测量各项期望进而计算出本征值 E 的。

对于期望 E(1)，系数 h 就是期望，无须构造线路测量。

$$\mathrm{E}(1) = \langle \psi | \mathrm{I}^{\otimes 2} | \psi \rangle = h = 1 \tag{1.7}$$

对于期望 E(2)，其厄米算符子项为

$$\sigma_z^0 \otimes \sigma_z^1 \tag{1.8}$$

由于测量操作是在 σ_z 上（以 σ_z 的特征向量为基向量所构成的子空间) 进行的，所以只需要在 0 号量子比特和 1 号量子比特上加上测量门即可，然后将测量结果传递给经典处理器求和，如下图所示：具体测量过程是这样的，由于两个 σ_z 矩阵的张量积形成的一个矩阵是对角矩阵:

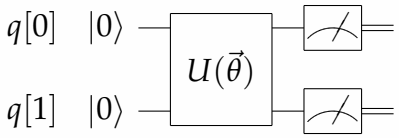

图 1.3: 期望 E(2) 的量子线路

$$\sigma_z^0 \otimes \sigma_z^1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.9}$$

根据线性代数知识可知其特征值就是对角线上的元素 1、-1、-1、1，与特征值相对应的特征向量正好就是基底 S：$|00\rangle$、$|01\rangle$、$|10\rangle$、$|11\rangle$，如果仔细观察的话，可以发现，当态为 $|00\rangle$、$|11\rangle$ 时，其中 1 的个数分别为 0 和 2，均为偶数，特征值均为 +1，而态为 $|01\rangle$、$|10\rangle$ 时，其中 1 的个数均为 1，为奇数，特征值均为 -1。

事实上，对于任何作为基底的量子态的特征值都有这样“奇负偶正”的规律。这样，一将测量门加到相应的量子线路，试验态就会以一定概率坍塌到不同的态 s，接着确定态 s 中 1 的个数 N_s 就行了，然后通过下式就可以计算出期望 $\mathrm{E}(i)$。

$$\mathrm{E}(i) = h^{ij..}_{\alpha\beta..} \sum_{|00..\rangle}^{|11..\rangle} (-1)^{N_s} P_s \tag{1.10}$$

即，如果坍塌后的态中有奇数个 1 ，其概率 P 取负值; 如果态中有偶数个 1，其概率 P 取正值，然后累加起来，再乘以系数 h 就得到了期望。

$$\mathrm{E}(2) = \langle \psi \left| \sigma_z^0 \otimes \sigma_z^1 \right| \psi \rangle = a^2 - b^2 - c^2 + d^2 \tag{1.11}$$

对于期望 $\mathrm{E}(3)$ 项，其厄米算符子项为

$$\sigma_x^0 \otimes \sigma_y^1 \tag{1.12}$$

此时，不能直接测量。这是因为对于试验态中的每一个基底（如 $|01\rangle$）, 它们均是单位矩阵和 σ_Z 的特征向量，但不是 σ_x 和 σ_y 的特征向量。根据线性代数知识，需要分别对 σ_x 和 σ_y 进行换基操作，也就是让试验态再演化一次，而：

$$\begin{aligned} \sigma_x &= H \times \sigma_z \times H \\ \sigma_y &= RX\left(-\frac{\pi}{2}\right) \times \sigma_z \times RX\left(\frac{\pi}{2}\right) \end{aligned} \tag{1.13}$$

所以在测量前，需要在 q[0] 号量子比特上添上 H 门、在 q[1] 号量子比特上添上 $RX\left(-\frac{\pi}{2}\right)$ 门，如下图所示：

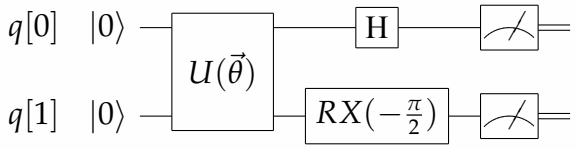

图 1.4: 期望 $\mathrm{E}(3)$ 的量子线路

此时，试验态 $|\psi\rangle$ 演化为 $|\psi'\rangle$。

$$|\psi'\rangle = A|00\rangle + B|01\rangle + C|10\rangle + D|11\rangle \tag{1.14}$$

接着再利用“奇负偶正”这一规律进行测量，不难得到：

$$\mathrm{E}(3) = \left\langle \psi \left| \sigma_x^0 \otimes \sigma_y^1 \right| \psi \right\rangle = \langle \psi' \left| \sigma_z^0 \otimes \sigma_z^1 \right| \psi' \rangle = A^2 - B^2 - C^2 + D^2 \tag{1.15}$$

在 CPU 上对这三个期望求和，就得到了总期望 E，即最终的本征值。

使用 pyQPanda 实现量子期望估计

在实现量子期望估计这个函数前，我们先定义会用到两个函数，第一个是测量前的换基操作，正如上文中式1.13提及的。

Code Listing 1.3: define transform_base function with pyQPanda

```
import pyqpanda as pq
import numpy as np

def transform_base(qlist,component):
    '''
    rotate x,y axis to z-axis for one component of matrix

    Args:
        qlist(QVec): qubit list
        component(List[Tuple[Dict[int,str],float]): paulioperator and coefficient. e.g.({0: 'Y',
                                                    1: 'Z', 2: 'X'}, 1.0)
    Return:
        quantum circuit(QCircuit)
    '''
    circuit = pq.QCircuit()
    for i, j in component[0].items():
        if j=='X':
            circuit.insert(pq.H(qlist[i]))
        elif j=='Y':
            circuit.insert(pq.RX(qlist[i],-np.pi/2))
        elif j=='Z':
            pass
        else:
            assert False
    return circuit
```

第二个函数是奇偶校验，即对坍塌后的态中 1 的个数进行计算。若是奇数个 1 返回 True，偶数个 1 返回 False，在下个 get_expectation 接口中进行“奇负偶正”的计算。这个函数中第一步要将坍塌后的态进行顺序倒置，因为 pyQPanda 中量子比特默认低位在右，而 python 切片索引默认是从左开始数起。接着遍历 paulidict 中的量子位，即 component 中泡利字典的 key。默认的测量是对所有量子比特进行测量，但我们只关心 component 中泡利算符所作用的量子位。

Code Listing 1.4: define parity_check function with pyQPanda

```
import pyqpanda as pq

def parity_check(state, paulidict):
    '''
    parity check, that is, to check how many "1" in each state after measurement.

    Args:
        state(str):the state after measurement
        paulidict(Dict):paulioperator dictionary. e.g.{0: 'Y', 1: 'Z', 2: 'X'}
    Return:
        bool:True for odd count, False for even count

    '''
    check=0
    state=state[::-1]
    for i in paulidict:
        if state[i]=='1':
            check+=1

    return check%2
```

接下来我们来定义量子期望估计的主体 get_expectation 函数。这个函数用来计算厄米算符的**子项**在试验态下的期望。这个接口需要传入的参数为虚拟机类型、量子比特数、厄米算符的一个子项、初始参数列表。程序大体流程是首先创建一个虚拟机，从虚拟机申请量子比特，接着创建量子线路，使用 Hardware-Efficient ansatz 来制备试验态，然后在测量前对体系哈密顿量子项进行换基操作，最后进行测量，并根据测量结果通过“奇负偶正”规则，计算当前厄米算符子项的期望。

Code Listing 1.5: define get_expectation function with pyQPanda

```
import pyqpanda as pq

def get_expectation(machine_type,qn,component,para):
    '''
    get expectation of one component of hermitian matrix.

    Args:
        machine_type(QMachineType): Quantum Machine Type
        qn(int):qubit number
        component(List[Tuple[Dict[int,str],float]]): paulioperator and coefficient, e.g.({0: 'Y', 1: 'Z', 2: 'X'}, 1.0)
        para(list[float64]): initial parameter. The number of parameters is four times qubit number.
    Return:
        expectation value of one component of hermitian matrix (float64)
    '''
    machine = pq.init_quantum_machine(machine_type)
    qlist=pq.qAlloc_many(qn)

    # build parametrized quantum circuit using Hardware-Efficient ansatz
    prog=pq.QProg()
    HE_circuit = prepare_HE_ansatz(qlist, para)
    prog.insert(HE_circuit)
    prog.insert(pq.BARRIER(qlist))

    # transform base for measurement
    if component[0]!='':
        prog.insert(transform_base(qlist,component))

    # measurement
    result = machine.prob_run_dict(prog,qlist,-1)

    # get expectation value of one component of hermitian matrix
    expectation=0
    for i in result:
        if parity_check(i, component[0]):
            expectation-=result[i]
        else:
            expectation+=result[i]
    return expectation*component[1]
```

在得到厄米算符的子项的期望后，我们要对所有子项进行求和，得到最终的本征值。这里我们定义 get_eigenvalue 函数来实现。

Code Listing 1.6: define get_eigenvalue function with pyQPanda

```
import pyqpanda as pq

def get_eigenvalue(machine_type,qn,matrix,para):
    '''
    get eigenvalue of a hermitian matrix.

    Args:
        machine_type(QMachineType):Quantum Machine Type
        qn(int):qubit number
        matrix(List[Tuple[Dict[int,str],float]]): matrix expressed by paulioperator e.g.:[({0: 'X', 2: 'Y'}, 2.0), ({1: 'Y', 2: 'Z', 3: 'X'}, 1.0)]
```

```
        para(list[float64]): initial parameter. The number of parameters is four times qubit
                                                                    number.
    Return:
        eigenvalue of a hermitian matrix(float64)
    '''
    expectation=0
    for component in matrix:
        expectation+=get_expectation(machine_type=machine_type,qn=qn,component=component,para=
                                                                    para)
    expectation=float(expectation.real)
    return expectation
```

至此，我们已经得到了厄米矩阵的本征值，下一步就是通过不断优化含参量子线路中的参数 $\vec{\theta}$ 来得到最小的本征值。这里我们通过经典优化器来对参数进行优化。

1.5.3.4 经典优化器参数优化

在这一节中我们将使用 scipy 工具包进行参数优化。由于 VQE 算法中对任意量子态的测量值的期望都不小于最小本征值，所以 get_eigenvalue 函数即为待优化的损失函数。

首先我们要定义待优化的函数 func(x)，接着使用 scipy.optimize.minimize 来进行最小值的求解，优化器方法这里指定 “SLSQP”，如下所示。

Code Listing 1.7: define opt function with pyQPanda

```
import scipy.optimize as opt

def func(x):
    return get_eigenvalue(machine_type,qn, Hf, x)

def opt_scipy():
    res = opt.minimize(func,init_para,
            method = "SLSQP",
            options = {"maxiter":100},
            tol=1e-9)
    print(res)
```

例 1. 下面我们将使用公式*1.4*中的厄米矩阵，对其进行最小本征值的求解。代码及结果如下：

Code Listing 1.8: using VQE to get lowest eigenvalue with pyQPanda

```
import pyqpanda as pq
import numpy as np

if __name__ == "__main__":

    d = pq.PauliOperator({"": 1,  "Z0 Z1": 1, "X0 Y1": 1})
    Hd = d.to_hamiltonian(True)

    machine_type = pq.QMachineType.CPU
    qn = 2
    init_para = np.random.rand(qn*4)
    opt_scipy()


 message: Optimization terminated successfully
 success: True
  status: 0
     fun: -0.9999999998069856
       x: [ 6.552e-01  4.712e+00 -3.142e+00 -3.142e+00  8.467e-01
            3.142e+00  5.441e-01  1.090e-05]
     nit: 13
     jac: [ 2.980e-08 -3.643e-06 -1.480e-05 -6.706e-07 -7.451e-09
           -9.105e-06  1.490e-08  1.803e-06]
```

```
    nfev: 120
    njev: 13
```

为了验证我们得到结果的正确性，下面使用 *numpy* 计算原矩阵的最小本征值。代码及结果如下。可见二者得到结果是一致的。

Code Listing 1.9: using numpy to get lowest eigenvalue for validation

```
import numpy as np

mat = np.array([[1,0,0,-1j],[0,0,1j,0],[0,-1j,0,0],[1j,0,0,1]])
np.min(np.linalg.eigvals(mat))


-0.9999999999999999+0j
```

例 2. 下面我们再举一个例子，求解矩阵 $\begin{bmatrix} 2 & 1 & 4 & 2 \\ 1 & 3 & 2 & 6 \\ 4 & 2 & 2 & 1 \\ 2 & 6 & 1 & 3 \end{bmatrix}$ 的最小本征值。代码及结果如下：

Code Listing 1.10: using VQE to get lowest eigenvalue with pyQPanda2

```
import pyqpanda as pq
import numpy as np

if __name__ == "__main__":

    mat = np.array([[2,1,4,2],[1,3,2,6],[4,2,2,1],[2,6,1,3]])
    f = pq.matrix_decompose_hamiltonian(mat)
    Hf = f.to_hamiltonian(True)

    machine_type = pq.QMachineType.CPU
    qn = 2
    init_para = np.random.rand(qn*4)
    opt_scipy()


 message: Optimization terminated successfully
 success: True
  status: 0
     fun: -3.6180339885637727
       x: [ 9.320e-01 -1.896e+00 -1.571e+00  2.865e-01  5.417e-01
            -1.760e+00  1.571e+00  2.714e-01]
     nit: 14
     jac: [-5.960e-08  9.537e-06  8.941e-07  1.487e-05  0.000e+00
            3.278e-05 -1.311e-06 -3.278e-07]
    nfev: 131
    njev: 14
```

同样，我们使用 *numpy* 计算矩阵的最小本征值。代码及结果如下。可见二者得到结果也是一致的。

Code Listing 1.11: using numpy to get lowest eigenvalue for validation2

```
import numpy as np

mat = np.array([[2,1,4,2],[1,3,2,6],[4,2,2,1],[2,6,1,3]])
np.min(np.linalg.eigvals(mat))


-3.618033988749894
```

练习 1.5.1. 尝试使用不同的 *ansatz* 来制备试验态，构建自定义的含参的量子线路并完成 *VQE* 流程。与 *Hardware-Efficient ansatz* 得到的结果哪个更准确？

练习 1.5.2. 在经典优化器参数优化一节中我们使用了基于梯度的 *SLSQP* 方法，尝试使用无梯度优化方法 *(Nelder-Mead*、*COBYLA* 等*)* 并简单叙述一下不同经典优化器对 *VQE* 算法的影响。