2024 量旋杯解题方案说明

罗马花椰菜[*]

(Dated: 2024 年 7 月 7 日)

本文分别使用 QUBO (Quadratic unconstrained binary optimization [Jun24]) + VQE (variational quantum eigensolver [FPGA22]) 和 VQLS (Variational Quantum Linear Solver [BPLC+23]) 算法来求解线性方程组，在这里我们更倾向于使用 QUBO 的方案，且还可以使用 QAOA 来替代 VQE。本文除了给出公式推导，还会展示代码实验结果 (有噪和无噪环境)，同时会对两个方法进行简单的对比。

Contents

I. INTRODUCTION

Problem 1 现卖牛 2 头羊 5 只，买猪 13 头，余下 1000 钱；卖牛 3 头、猪 3 头，买羊 9 只，钱正好足够；卖羊 6 只、猪 8 头，买牛 5 头，还差 600 钱。问：牛、羊、猪各多少钱？答：牛价格 1200，羊价格 500，猪价格 300。请设计量子算法来求解该线性方程组问题。

[*]Electronic address: f.g.m.leonardo@gmail.com

Problem 2 为了方便求解，我们使用线性代数来表示上述问题：

$$\mathrm{Ax} = \mathrm{b}, \tag{1}$$

$$\mathrm{A} = \begin{bmatrix} 2 & 5 & -13 \\ 1 & -3 & 1 \\ -5 & 6 & 8 \end{bmatrix}, \quad \mathrm{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1200 \\ 500 \\ 300 \end{bmatrix}, \quad \mathrm{b} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 1000 \\ 0 \\ -600 \end{bmatrix}.$$

同时，我们在这里事先给出，解题采用的两套方案的优缺点，方便快速浏览，在这里我们更倾向于使用 QUBO 的方案：

方案	优势	劣势
QUBO	不对 A、b 进行预处理，理论推导即可获得哈密顿量；答案精准；量子门数量少	需要较多的 qubits，取决于解的数量和采用的编码方式
VQLS	需要的量子比特数少	A、b 要进行预处理转换，$\|b\rangle$ 要进行门分解，训练耗时，精度不稳定，误差大

II. SOLUTION: QUBO

A. Quadratic unconstrained binary optimization

首先，为了能使用 VQE 算法求解线性方程组问题，我们需要将线性方程组转换成一个 QUBO 问题，同时通过映射将 QUBO 变成哈密顿量。通过阅读论文 QUBO formulations for a system of linear equations[Jun24]，我们有如下关系式：

$$\mathrm{Ax} = \mathrm{b} \Rightarrow \|\mathrm{Ax} - \mathrm{b}\|_2^2 = 0, \tag{2}$$

由此我们进行二范数展开，有

$$\begin{aligned} &\|\mathrm{Ax} - \mathrm{b}\|_2^2 = \mathrm{x}^T \mathrm{A}^T \mathrm{Ax} - 2\mathrm{b}^T \mathrm{Ax} + \mathrm{b}^T \mathrm{b} \\ &= \sum_{k=0}^{n-1} \left\{ \left(\sum_{i=0}^{n-1} a_{k,i} x_i \right)^2 - 2b_k \sum_{i=0}^{n-1} a_{k,i} x_i + b_k^2 \right\} \\ &= \sum_{k=0}^{n-1} \left\{ \sum_{i=0}^{n-1} (a_{k,i} x_i)^2 + 2 \sum_{0 \le i < j}^{n-1} a_{k,i} a_{k,j} x_i x_j - 2b_k \sum_{i=0}^{n-1} a_{k,i} x_i + b_k^2 \right\}. \end{aligned} \tag{3}$$

由于，b_k^2 是常数项，我们可以将其忽略，也就是说，我们将一个解线性方程组问题变成了一个优化问题，具体形式如下：

$$\min_{\mathrm{x}} \sum_{k=0}^{n-1} \left\{ \sum_{i=0}^{n-1} (a_{k,i} x_i)^2 + 2 \sum_{0 \le i < j}^{n-1} a_{k,i} a_{k,j} x_i x_j - 2b_k \sum_{i=0}^{n-1} a_{k,i} x_i + b_k^2 \right\} \tag{4}$$

另外，解 x_0, x_1, x_2 均为十进制正整数，我们可以使用二进制来表示，即

$$x_i = \sum_{l=0}^{m-1} 2^l q_{i,l}, \quad q_{i,l} \in \{0, 1\}, \quad q_{i,l}^2 = q_{i,l}. \tag{5}$$

将公式 (5) 带入到方程 (3) 中展开，依次得到下面的公式：

$$\begin{aligned}\sum_{k=0}^{n-1}\sum_{i=0}^{n-1}(a_{k,i}x_i)^2 &= \sum_{k=0}^{n-1}\sum_{i=0}^{n-1}(a_{k,i}\sum_{l=0}^{m-1}2^l q_{i,l})^2 \\ &= \sum_{k=0}^{n-1}\sum_{i=0}^{n-1}\sum_{l=0}^{m-1}2^{2l}a_{k,i}^2 q_{i,l}^2 + \sum_{k=0}^{n-1}\sum_{i=0}^{n-1}\sum_{l_1\neq l_2}^{m-1}2^{l_1+l_2}a_{k,i}^2 q_{i,l_1}q_{i,l_2} \\ &= \sum_{k=0}^{n-1}\sum_{i=0}^{n-1}\sum_{l=0}^{m-1}2^{2l}a_{k,i}^2 q_{i,l} + \sum_{k=0}^{n-1}\sum_{i=0}^{n-1}\sum_{l_1\neq l_2}^{m-1}2^{l_1+l_2}a_{k,i}^2 q_{i,l_1}q_{i,l_2},\end{aligned} \tag{6}$$

$$\begin{aligned}\sum_{k=0}^{n-1}\sum_{0\leq i<j}^{n-1}2a_{k,i}a_{k,j}x_i x_j &= \sum_{k=0}^{n-1}\sum_{0\leq i<j}^{n-1}2a_{k,i}a_{k,j}(\sum_{l_1=0}^{m-1}2^{l_1}q_{i,l_1})(\sum_{l_2=0}^{m-1}2^{l_2}q_{j,l_2}) \\ &= \sum_{k=0}^{n-1}\sum_{0\leq i<j}^{n-1}\sum_{l_1=0}^{m-1}\sum_{l_2=0}^{m-1}2^{l_1+l_2+1}a_{k,i}a_{k,j}q_{i,l_1}q_{j,l_2},\end{aligned} \tag{7}$$

$$-\sum_{k=0}^{n-1}\sum_{i=0}^{n-1}2b_k a_{k,i}x_i = -\sum_{k=0}^{n-1}\sum_{i=0}^{n-1}\sum_{l=0}^{m-1}2^{l+1}b_k q_{i,l} \tag{8}$$

由上述推导 (6)(7)(8)，我们将一个一般性的优化问题 (4)，变成了 0-1 整数规划，这便是我们需要的 QUBO 形式。下面给出，将变量 $q_{i,l}$ 映射成 pauli 算子的表达式：

$$q_{i,l} \to \frac{\mathbb{I}-\mathbb{Z}_{i,l}}{2}, \tag{9}$$

其中，$\mathbb{I}$ 表示 $2^n\times 2^n$ 的单位矩阵，n 是编码所有解变量 x_i 为二进制形式所需要的变量 $q_{i,l}$ 的总数量，也是需要的总的 qubits 数量。$\mathbb{Z}_{i,l}$ 表示在第 $i\cdot m+l$ qubit 上的是 pauli $\mathbb{Z}$ 算子，其它位置上的均为 2×2 的单位矩阵 $\mathbb{I}$，也可称为单位算子，m 是编码一个解变量 x_i 需要的 $q_{i,l}$ 的数量。则公式(6)(7)(8) 的 pauli 算子形式可以写成：

$$\sum_{k=0}^{n-1}\sum_{i=0}^{n-1}\sum_{l=0}^{m-1}2^{2l}a_{k,i}^2 q_{i,l} = \sum_{k=0}^{n-1}\sum_{i=0}^{n-1}\sum_{l=0}^{m-1}2^{2l-1}a_{k,i}^2(\mathbb{I}-\mathbb{Z}_{i,l}), \tag{10}$$

$$\sum_{k=0}^{n-1}\sum_{i=0}^{n-1}\sum_{l_1\neq l_2}^{m-1}2^{l_1+l_2}a_{k,i}^2 q_{i,l_1}q_{i,l_2} = \sum_{k=0}^{n-1}\sum_{i=0}^{n-1}\sum_{l_1\neq l_2}^{m-1}2^{l_1+l_2-2}a_{k,i}^2(\mathbb{I}-\mathbb{Z}_{i,l_1}-\mathbb{Z}_{i,l_2}+\mathbb{Z}_{i,l_1}\mathbb{Z}_{i,l_2}), \tag{11}$$

$$\sum_{k=0}^{n-1}\sum_{0\leq i<j}^{n-1}\sum_{l_1=0}^{m-1}\sum_{l_2=0}^{m-1}2^{l_1+l_2+1}a_{k,i}a_{k,j}q_{i,l_1}q_{j,l_2} = \sum_{k=0}^{n-1}\sum_{0\leq i<j}^{n-1}\sum_{l_1=0}^{m-1}\sum_{l_2=0}^{m-1}2^{l_1+l_2-1}a_{k,i}a_{k,j}(\mathbb{I}-\mathbb{Z}_{i,l_1}-\mathbb{Z}_{j,l_2}+\mathbb{Z}_{i,l_1}\mathbb{Z}_{j,l_2}), \tag{12}$$

$$-\sum_{k=0}^{n-1}\sum_{i=0}^{n-1}\sum_{l=0}^{m-1}2^{l+1}b_k q_{i,l} = \sum_{k=0}^{n-1}\sum_{i=0}^{n-1}\sum_{l=0}^{m-1}-2^l b_k(\mathbb{I}-\mathbb{Z}_{i,l}) = \sum_{k=0}^{n-1}\sum_{i=0}^{n-1}\sum_{l=0}^{m-1}2^l b_k(-\mathbb{I}+\mathbb{Z}_{i,l}), \tag{13}$$

有了 pauli 公式(10)(11)(12)(13)，我们可以通过构造 python 函数来生成该问题的哈密顿量，且仅使用字符串来记录需要的 pauli 算子及其位置，不需要转换成矩阵进行存储，具体见 hamiltonian_until.py 中的函数 linear_equation_hamiltonian 。也可在 1_QUBO_introduction.ipynb 文件中，使用特征值分解函数求解哈密顿量的解，以此来验证生成的哈密顿量是我们需要的。

另外，公式(5)说明问题的解可以使用 n bits 二进制串来表示。在这里，我们使用一个小技巧将向量 x 和 b 缩放，从而可以使用更少的 n bits 数来表示变量，即

$$\mathrm{x}^{'} = \frac{\mathrm{x}}{100} = \begin{bmatrix} \sum_{l=0}^{3} 2^l q_{0,l} \\ \sum_{l=0}^{3} 2^l q_{1,l} \\ \sum_{l=0}^{3} 2^l q_{2,l} \end{bmatrix} = \begin{bmatrix} 0\cdot 2^0 + 0\cdot 2^1 + 1\cdot 2^2 + 1\cdot 2^3 \\ 1\cdot 2^0 + 0\cdot 2^1 + 1\cdot 2^2 + 0\cdot 2^3 \\ 1\cdot 2^0 + 1\cdot 2^1 + 0\cdot 2^2 + 0\cdot 2^3 \end{bmatrix} = \begin{bmatrix} 12 \\ 5 \\ 3 \end{bmatrix}, \tag{14}$$

$$\mathrm{x}^{''} = \begin{bmatrix} q_{0,0} \\ q_{0,1} \\ q_{0,2} \\ q_{0,3} \\ q_{1,0} \\ q_{1,1} \\ q_{1,2} \\ q_{1,3} \\ q_{2,0} \\ q_{2,1} \\ q_{2,2} \\ q_{2,3} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathrm{b}^{'} = \frac{\mathrm{b}}{100} = \begin{bmatrix} 10 \\ 0 \\ -6 \end{bmatrix}, \tag{15}$$

其中 $\mathrm{x}^{''}$ 是将 $\mathrm{x}^{'}$ 中的二进制变量展开，变成长度为 12 的向量。这同样代表着，我们需要 12 qubits 来分别编码这 12 个二进制变量，也就是说，每个解变量需要 4 个二进制变量来编码。

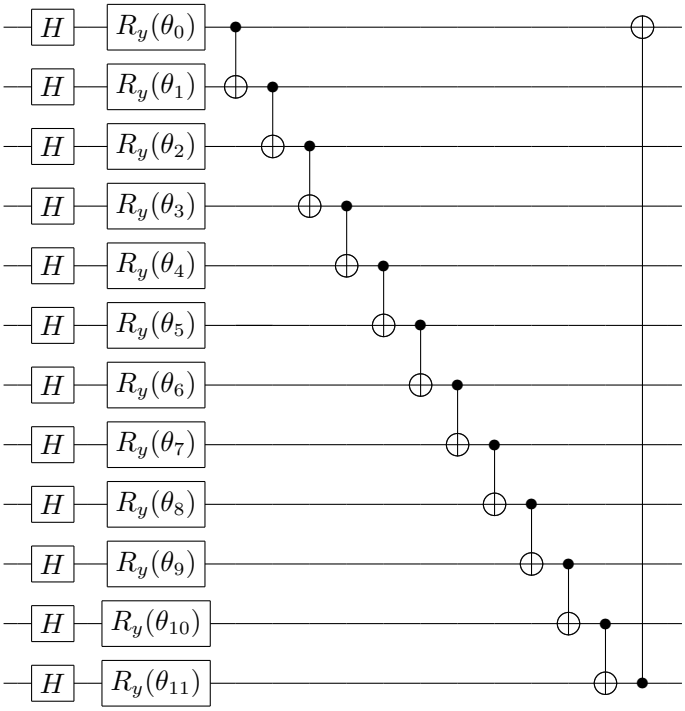
需要注意的是，本题我们的解是整数，才可以使用公式 (5) 的方式来进行二进制编码，如果解中包含小数或者负数，我们需要更复杂的编码方式，如

$$\begin{aligned} x_i &\approx \sum_{l=-m}^{m} 2^l q_{i,l}, \\ x_i &\approx \sum_{l=-m}^{m} 2^l q_{i,l}^{+} - \sum_{l=-m}^{m} 2^l q_{i,l}^{-} \end{aligned} \tag{16}$$

当然我们也可以考虑使用计算机中的 float32 的二进制表示，来表示我们的解变量，但需要的 qubits 数量会迅速上升，从而导致生成的哈密顿量的需要的 qubits 数很快会超过所能接受的 qubits 数，从而无法进行模拟计算或真机计算。

B. Variational quantum eigensolver

在拥有了线性方程组的哈密顿量 $\mathcal{H}$ 之后，我们采用 VQE 算法来求解该哈密顿量的基态，从而恢复出正确的解向量。以下是我们，在该实验中采用的变分量子电路 $U(\theta)$，仅一层电路结构，总共有 36 个量子门，其中单比特门 24 个，双比特门 12 个。



(17)

设该变分电路生成的量子态为 $|\psi(\theta)\rangle = U(\theta)\,|0\rangle^{\otimes 12}$，在 VQE 中选择的损失函数为：

$$\min_{\theta} \mathcal{L}(\theta) = \langle\psi(\theta)|\,\mathcal{H}\,|\psi(\theta\rangle = \langle 0|^{\otimes 12}\,U^{\dagger}(\theta)\mathcal{H}U(\theta)\,|0\rangle^{\otimes 12}\,, \tag{18}$$

$$\theta = (\theta_1, \theta_2, \ldots, \ldots, \theta_{11}), \tag{19}$$

$$\begin{aligned}\theta_{opt} = (&4.71237303, 4.7124589, 1.57078406, 4.71237222,\\ &4.71235171, 1.5708085, 1.57080377, 1.57074532,\\ &1.57081405, 4.71238723, 1.57089928, 4.71242189),\end{aligned} \tag{20}$$

当 VQE 学到最优解 θ_{opt} 时，损失函数 $\mathcal{L}(\theta_{opt})$ 即是哈密顿量 $\mathcal{H}$ 的基态能量，变分电路对应的量子态 $|\psi(\theta_{opt})\rangle$ 即是基态。此时我们只需要进行多次测量 (1024 shots)，得到概率最大的基向量，将其态矢量的形式 (eg.$|001110101100\rangle$) 写出来，再按照公式 (5) 的方式，即可获得线性方程组的解，具体过程可参看 1_QUBO_introduction.ipynb 或 II D 。

C. Distributed VQE

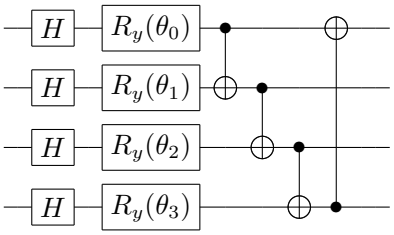
假设我们有 m_1 个解变量，每个解变量使用 (5) 编码成 m_2 二进制变量，则我们总共需要 $m_1 \cdot m_2$ 个 qubits 来构造哈密顿量，从而使得哈密顿的矩阵大小为 $(2^{m_1 \cdot m_2} \times 2^{m_1 \cdot m_2})$。如果要求解的问题规模足够大的话，是很难在计算机上模拟运行或在真机上运算的。

参考论文 Divide-and-conquer quantum algorithm for hybrid de novo genome assembly of short and long reads [FLH+24] 提出的 Distributed VQE 方案，将哈密顿量拆分成多个子系统，每个子系统的 qubits 数比未拆分前小很多。例如，我们在前面构造了一个 12 qubits 的哈密顿量，可以拆分成 3 个子系统，每个子系统由 4 个 qubits 的哈密顿量来表示。这原理是 QUBO 构造的 Hamiltonian 其矩阵是一个稀疏矩阵，仅有对角项，且其解是一个基向量，是可以由多个子系统的基向量张量积的来的。下面给出基于本文哈密顿量的简单推导，假设如下：

$$\mathcal{H} = \sum_t c_t H_t^A \otimes H_t^B \otimes H_t^C, \tag{21}$$

$$|\psi\rangle = U_A(\theta_A)\,|0\rangle^{\otimes 4} \otimes U_B(\theta_B)\,|0\rangle^{\otimes 4} \otimes U_C(\theta_C)\,|0\rangle^{\otimes 4}, \tag{22}$$

每个子系统量子态使用的相同的变分电路构造，即 U_A、U_B、U_C，仅是参数不同，如下：



(23)

TBA

D. Result

在这里，我们依次展示代码文件中，不同 ipynb 文件的运算结果，并进行说明。

1. Introduction QUBO

首先，查看文件 1_QUBO_introduction.ipynb，在这里演示了如何生成一个线性方程组的哈密顿量，只需要使用函数 linear_equation_hamiltonian ，注意生成的是根据公式(6)(7)(8)构造的哈密顿量。

图 1 展示了问题的 2 的哈密顿量，及其基态和基态向量。可以看到居中图显示，哈密顿量对角元上序号为 940 的数值，即是该哈密顿量的基态能量，对应的单位向量即是基态。写成态矢量的形式，应为 $|001110101100\rangle$ (与 (15) 一致)，同时注意 940 的二进制写法，也是 001110101100。每隔 4 qubits 则代表一个解，即 0011、1010、1100 分别代表解 x_1、x_2、x_3，例如将 0011 代入到二进制公式 (5) 有 $0\times 2^0 + 0\times 2^1 + 1\times 2^2 + 1\times 2^3 = 12$(与向量 (14) 和 (15) 一致) ，再乘上缩放系数 100 等于 1200，这正是该问题的正确答案。右图，展示将 4 qubits

图 1: 左图：线性方程组的哈密顿量；居中：哈密顿量的基态及其基态能量；右图：基态转换成线性方程组的解

二进制信息转换成十进制解的过程，我们需要将 0011 逆序排列为 1100，依次类推，在进行转换。这是因为在经典计算机中，二进制 bits 从右到左分别表示 $2^0, 2^1, 2^2, \ldots, 2^n$，而在我们公式 (5) 中，我们设置的是从左往右。

其次，我们查看第二个文件 2_QUBO_VQE_ideal_1.ipynb，在这里我们采用 VQE 算法，迭代训练计算线性方程组哈密顿量的基态能量及其基态。训练过程是无噪的，且未使用测量的方式获得损失值，是理想的训练环境。使用的电路为 Cir.17，初始参数由种子数 2048 生成，和迭代终止后的参数分别如下：

$$\begin{aligned}\theta_{init} = (&3.70464424, 0.85958082, 4.15965008, 6.20230976,\\ &5.63890652, 3.63767986, 0.40771267, 0.68193054,\\ &2.81191313, 2.12668968, 2.02620423, 5.69293231),\\ \theta_{end} = (&4.71227239, 1.57090722, 4.71230282, 4.71239447,\\ &4.71235273, 4.71240151, -1.5708641, 1.57063951,\\ &4.71220418, 4.71198229, 1.57046012, 7.85373219),\end{aligned} \tag{24}$$

图 (3) 和图 (2) 展示了 VQE 的训练过程和相关结果，我们可以看到 VQE 最终收敛到一个局部最小值 -26.0，若进行测量操作，训练后的电路将以将近 100% 的概率得到态矢量 $|011111100010\rangle$，而正确答案 $|001110101100\rangle$ 对应的测量概率为 0。

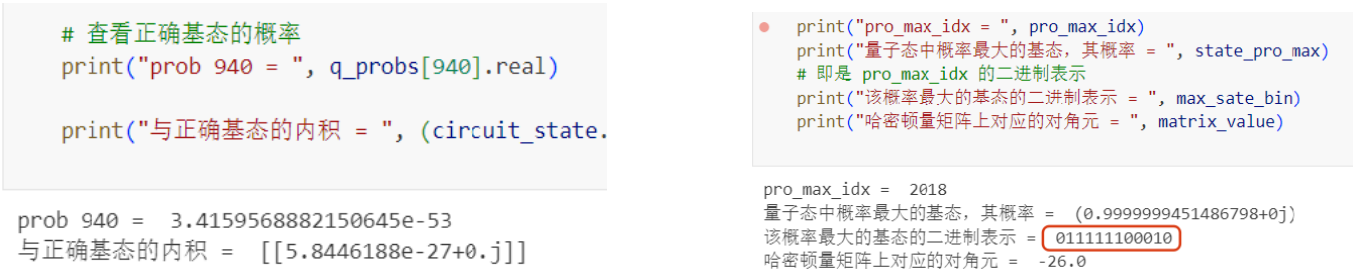


图 2: 基于随机初始参数的 VQE 训练后的相关结果

启示

我们可以通过查看测量概率中序号 940 的数值，来查看 $|001110101100\rangle$ 理论上的测量概率，同时注意到本问题的哈密顿量有 $2^{12} = 4096$ 个局部最小值，想学到真正的全局最小值，是比较困难的，需要对初始参数有较好的猜测。

接着，我们同时查看 3_circuit_params.ipynb 和 4_QUBO_VQE_ideal_2.ipynb。由于随机选取一组参数参数，可能会陷入局部最优解，那么我们便先获取能生成基态 $|001110101100\rangle$ 的变分电路的最优参数，这在 3_circuit_params.ipynb 中实现，主要采用了如下损失函数来衡量变分量子态和真实量子态之间的差异，比较简单，学到的最优参数为 (20)：

$$\min_{\theta} \mathcal{C}(\theta) = 1 - \langle 0|^{\otimes 12} U^{\dagger}(\theta) |001110101100\rangle , \tag{25}$$

在 4_QUBO_VQE_ideal_2.ipynb 中将学到的最优参数 θ_{opt} 做一个随机的扰动，再进行 VQE 训练，在这里我们构造了两个量子设备，其中一个采用非测量的方式计算损失值，另一个采用测量的方式计算损失值 (1024

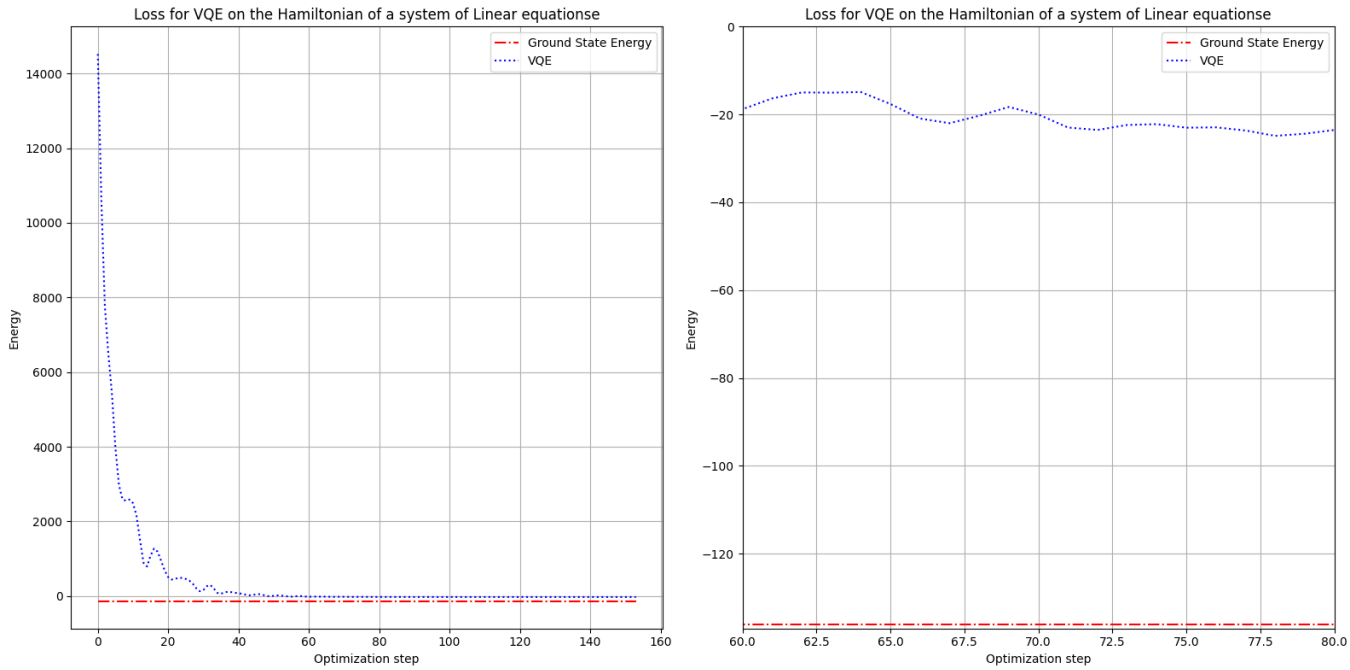


图 3: 基于随机初始参数的 VQE 的训练过程

shots)。初始参数和非测量模式下迭代终止后的参数:

$$\begin{aligned}\theta_{init} = (&4.86006426, 5.20937248, 1.58958732, 4.87048495,\\ &4.81141518, 1.82387583, 1.2070509, 1.07486244,\\ &1.22032293, 4.91082623, 1.66425184, 5.11233724),\\ \theta_{end} = (&4.69387585, 4.65868309, 1.5823514, 4.72660489,\\ &4.70326924, 1.52962019, 1.57952486, 1.54600057,\\ &1.6163221, 4.6989397, 1.46776989, 4.70481788),\end{aligned} \tag{26}$$

由图 (5)，我们可以看到在非测量模式下，损失值迅速收敛到全局最小值，而在测量模式下，损失值在随机震荡，无法收敛。图 (4) 展示了在上述两个设备中，正确基态 $|001110101100\rangle$ 的测量概率，均超过 90%，即通过解码均可以得到线性方程组正确的解。

启示

如果初始参数比较靠近最优解，无论采用测量或非测量模式，都会有很大的概率得到线性方程组的解。或者我们可以设置阈值条件，例如测量概率超过 5% 的基向量均解码转换成实际的数值 x_1', x_2', x_3'，再放入到实际的线性方程组中验证是否正确。也可以单纯的通过测量计算这些个基向量与哈密顿量的内积，选择内积最小的那个当作正确答案进行解码。

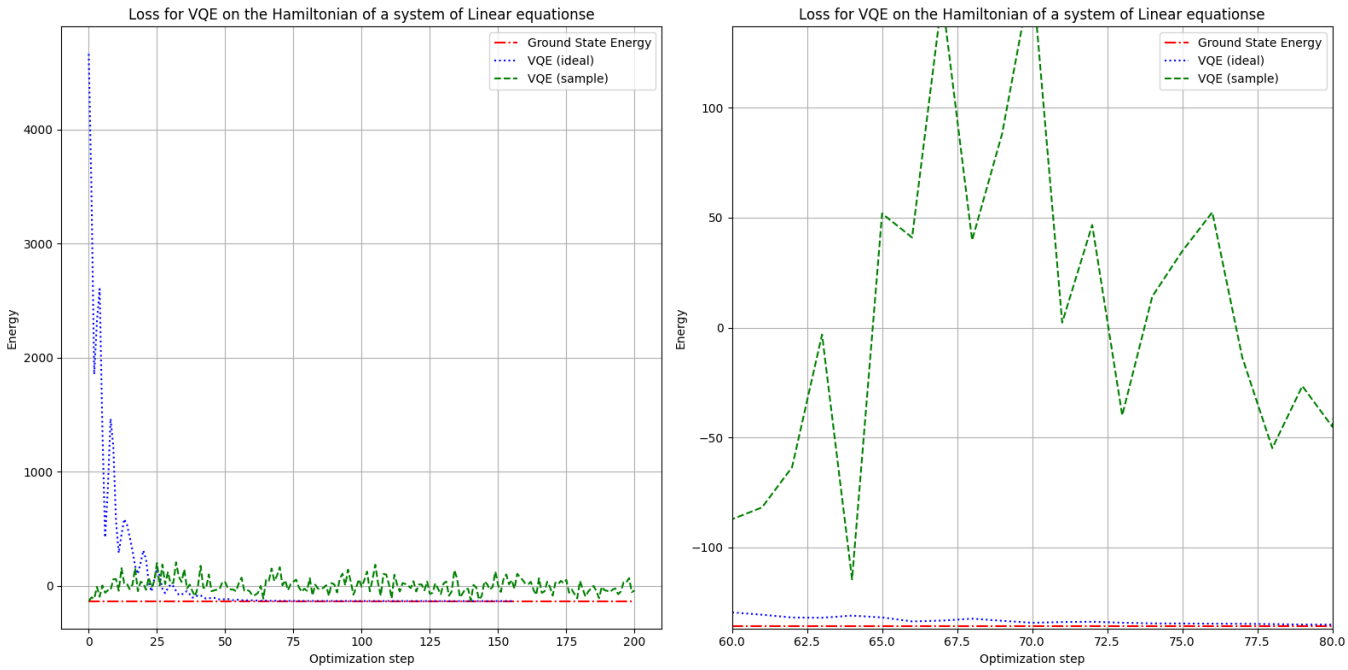


图 4: 测量与非测量模式下 VQE 的训练过程

```
# 变分电路学到的量子态
# 构造运行节点
qnode_state = qml.QNode(VQE_until.variational_circuit_state, dev_ideal)
circuit_state = qnode_state(ideal_params_history[-1], deep_layer)

# 参看正确答案对应序号的概率
print(np.argmax(circuit_state))
print(circuit_state[940]**2)
```

```
940
(0.9999999804460044+0j)
```

```
# 测量模式下，测量概率
qnode_prob = qml.QNode(VQE_until.variational_circuit_prob, dev_sample)
sample_prob = qnode_prob(sample_params_history[-1], deep_layer)

print(np.argmax(circuit_state))
print(sample_prob[940])
```

```
940
0.9970703125
```

图 5: 测量与非测量模式下正确基态的测量概率

3. QUBO and VQE with noise

最后，我们查看文件 5_QUBO_VQE_noise.ipynb，该文件的实验与 II D 2 中的最大区别是，引入了 qiskit 模拟的真机噪声 fakemontreal，用于对比有噪和无噪环境下 VQE 的运行，此外使用的变分电路和初始参数与前面保持一致。

图 (6) 中，我们构造了 4 个量子设备 (测量或非测量模式下，有噪或无噪的量子设备)。左图，在非测量模式下，有噪设备计算出的损失值与无噪的相差 400 左右，理论上正确的基态 $|001110101100\rangle$ 的测量概率也比无

噪的小 0.02 左右。右图，在测量模式下，有噪设备测量出的损失值与无噪的差距进一步拉大，但理论上的测量概率却缩小了。而无噪设备，在测量或非测量模式下，损失值和测量概率差距不大。

```
# 基于非测量的节点的出值对比
noise_value = qnode_noise(params, deep_layer)
ideal_value = qnode_ideal(params, deep_layer)

noise_prob = qnode_noise_prob(params, deep_layer)
ideal_prob = qnode_ideal_prob(params, deep_layer)

print("noise_value = ", noise_value)
print("noise_prob 940 = ", noise_prob[940])

print("ideal_value = ", ideal_value)
print("ideal_prob 940 = ", ideal_prob[940])
```

```
noise_value =  4671.0498046875
noise_prob 940 =  0.7646484375
ideal_value =  5074.735045812266
ideal_prob 940 =  0.7853045632197279
```

```
# 基于测量的节点的出值对比
noise_sample_value = qnode_noise_sample(params, deep_layer)
ideal_sample_value = qnode_ideal_sample(params, deep_layer)

noise_sample_prob = qnode_noise_sample_prob(params, deep_layer)
ideal_sample_prob = qnode_ideal_sample_prob(params, deep_layer)

print("noise_sample_value = ", noise_sample_value)
print("noise_sample_prob 940 = ", noise_sample_prob[940])

print("ideal_sample_value = ", ideal_sample_value)
print("ideal_sample_prob 940 = ", ideal_sample_prob[940])
```

```
noise_sample_value =  6681.3671875
noise_sample_prob 940 =  0.77734375
ideal_sample_value =  5199.890625
ideal_sample_prob 940 =  0.7861328125
```

图 6: 基于有噪或无躁量子设备计算损失值

图 (7) 中，描述的是在非测量模式下，VQE 在无噪和有噪设备下的优化过程(II D 3)。左图，在无噪环境下，VQE 的损失值可以快速朝全局最小值点靠近，而在有噪环境下，损失值在反复震荡，没有收敛的迹象。右图则描述了，迭代终止后，有噪或无噪设备最终优化得到的变分电路关于基态 $|001110101100\rangle$ 的测量概率。可以看到无噪环境下的测量概率为 0.983，有噪环境则是 0.873，尽管有 0.11 的差距，但测量概率均很大，若是在真实量子设备上进行测量，我们依然可以从有噪设备上，解码恢复出线性方程组的解，这便是本方案的结果展示。

注意

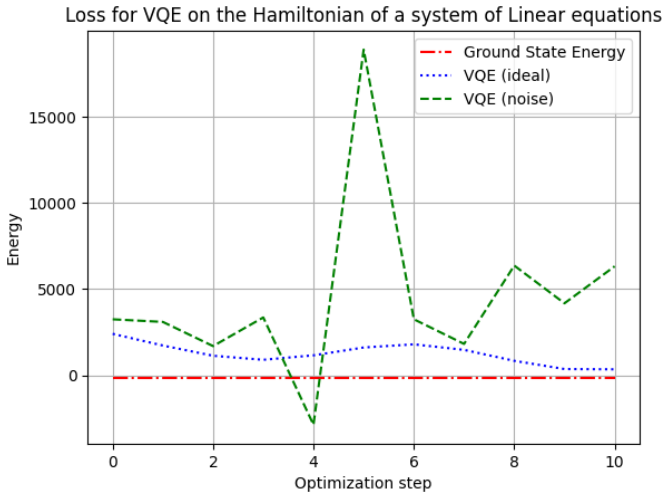
在非测量模式下，VQE 在有噪设备中，优化一次参数大约要运算 50 min，若是采用 1024 shots 的测量模式，需要的时间进一步增加，所以在这里，我们仅迭代优化 10 次看看运行效果。

启示

若是初始参数生成的量子态中，正确基态的概率较大，那么在有噪声环境下，VQE 可以持续优化使得正确基态的概率不断增大，但相比于无噪环境，优化效果较差。

以下是，有噪环境下，VQE 的初始参数和终止迭代后的参数值：

$$\begin{aligned}\theta_{init} = (&5.16163973, 4.287978761.58425936, 4.86709964,\\ &4.8774276, 1.25574156, 1.58577618, 1.92688623,\\ &1.15474026, 4.485776411.83660525, 4.79125114),\\ \theta_{end} = (&4.55298965, 4.49180278, 1.50765422, 4.88895401,\\ &4.65764962, 1.07404392, 1.68486474, 1.90871567,\\ &1.43820582, 4.7905775, 1.47480147, 4.704848178).\end{aligned} \tag{27}$$




```
# 返回测量概率
noise_prob = qnode_noise_prob(noise_params_history[-1], deep_layer)
ideal_prob = qnode_ideal_prob(ideal_params_history[-1], deep_layer)

print("noise_prob = ", noise_prob)
print("noise_prob 940 = ", noise_prob[940])

print("ideal_prob = ", ideal_prob)
print("ideal_prob 940 = ", ideal_prob[940])
```

```
noise_prob =  [0. 0. 0. ... 0. 0. 0.]
noise_prob 940 =  0.873046875
ideal_prob =  [4.85936268e-19 1.17962295e-32 2.03473596e-21 ... 1.62445298e-23
 1.47755661e-30 3.87952361e-21]
ideal_prob 940 =  0.9831552692199108
```

图 7: 基于有噪或无躁量子设备 VQE 的训练过程及其结果

III. SOLUTION: VQLS

A. Variational Quantum Linear Solver

针对问题 (2)，我们还采用了第二套方案来进行求解，即 Variational Quantum Linear Solver[BPLC+23]，该算法可求解的问题与我们的问题有所差别。

具体来说，存在如下 $2^n \times 2^n$ 的矩阵 $\mathcal{A}$，可由酉矩阵的线性组合构成，

$$\mathcal{A} = \sum_{l=0}^{L-1} c_l A_l, \tag{28}$$

其中 c_l 是复数，A_l 是酉矩阵，若是对 $\mathcal{A}$ 做 Pauli 分解，则 A_l 则是对应的 Pauli 项。此外，还存在一个归一化的复向量 $|b\rangle$ 可由酉矩阵 U_b 生成，

$$|b\rangle = U_b |0\rangle^{\otimes n}, \tag{29}$$

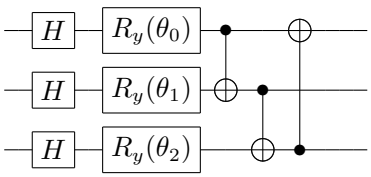
而该算法要构造一个量子态 $|x\rangle$，使得

$$|\Psi\rangle := \frac{\mathcal{A}|x\rangle}{\sqrt{\langle x|\mathcal{A}^\dagger\mathcal{A}|x\rangle}} \approx |b\rangle, \tag{30}$$

之所以如此是因为难以保证 $\mathcal{A}|X\rangle$ 依然是一个量子态，所以需要进行归一化。在这里我们设

$$|x\rangle = V(w)|0\rangle^{\otimes n}. \tag{31}$$

其中 $V(W)$ 是变分量子电路所表示的酉矩阵，在本题中，我们采用的变分量子电路：



(32)

VQLS 通常采用损失函数(33)，其中所有的期望值计算都可以用 Hadamard test 来估计，但所有的 $U_b^\dagger$、A_l、V 都需要以受控的方式实现，比较麻烦，

$$\begin{aligned}\mathcal{C}_G &= 1 - |\langle b|\Psi\rangle|^2 \\ &= 1 - \frac{\sum_{l,l'} c_l c_{l'}^* \langle 0|V^\dagger A_{l'}^\dagger U_b\,|0\rangle\,\langle 0|\,U_b^\dagger A_l V\,|0\rangle}{\sum_{l,l'} c_l c_{l'}^* \langle 0|V^\dagger A_{l'}^\dagger A_l V|0\rangle},\end{aligned} \tag{33}$$

在这里，我们与 pennylane 教程 VQLS 保持一致，使用局部的损失函数(34)，只有 A_l、$A_l^\dagger$、Z_j 需要由辅助比特来控制，

$$\begin{aligned}\mathcal{C}_L &= \frac{1}{2} - \frac{1}{2n}\frac{\sum_{j=0}^{n-1}\sum_{l,l'} c_l c_{l'}^* \langle 0|V^\dagger A_{l'}^\dagger U_b Z_j U_b^\dagger A_l V|0\rangle}{\sum_{l,l'} c_l c_{l'}^* \langle 0|V^\dagger A_{l'}^\dagger A_l V|0\rangle} \\ &= \frac{1}{2} - \frac{1}{2n}\frac{\sum_{j=0}^{n-1}\sum_{l,l'} c_l c_{l'}^* \mu_{l,l',j}}{\sum_{l,l'} c_l c_{l'}^* \mu_{l,l',-1}},\end{aligned} \tag{34}$$

$$\mu_{l,l',j} = \langle 0|V^\dagger A_{l'}^\dagger U_b Z_j U_b^\dagger A_l V|0\rangle, \tag{35}$$

其中，当 $j=-1$ 时，$Z_{-1}=\mathbb{I}$，于是整个 Hadamard test 电路图形表示如下，取自 (VQLS)，其中 $R_\Phi(-\frac{\pi}{2})$ 是相位门，为了估计 $\mu_{l,l',j}$ 的虚部，

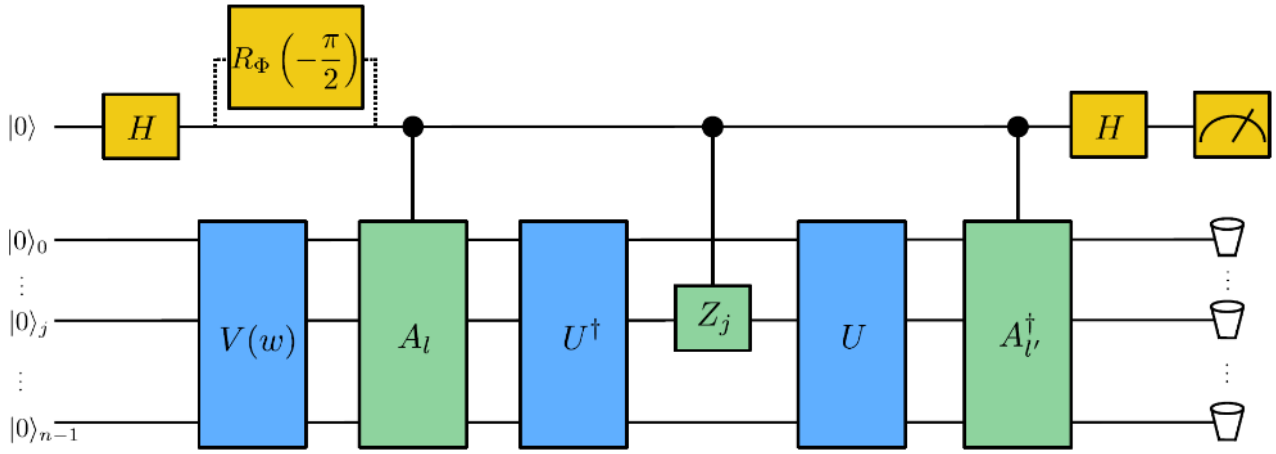


图 8: Hadamard test 的电路表示

B. OpenQASm

要构造受控的 A_l，我们可以先对 $\mathcal{A}$ 做 Pauli 分解，则 A_l 由 Pauli 算子构成，受控的 A_l 则可以由一系列常见的受控门组合构成，见图(9)。

由于 pennylane 不支持转成 OpenQASM 形式，有许多操作需要自己实现，如量子电路的 adjoint，所以，该方案暂时无法提供全 OpenQASM 形式的文件。但我们已经将所有非基础门都转变成基础门的形式，并在一组参数下进行了验证，见 4_circuit 中的 circuit_qasm.ipynb 和 circuit_until.py 文件。图 (10) 中第一个结果是原始电路构成的，使用 qml.ControlledQubitUnitary() 实现受控 A_l 门，第二个结果是由一些基础门构成的，由图(9)与其他函数构成，第二个结果更好。

```python
# 定义受控量子门 A_l
def circuit_with_controlled_pauli(ops_list):
    # 获取识别结果
    identified_terms = identify_pauli_operators(ops_list)
    # 控制位为辅助比特位
    control_wire = ancilla_idx
    # 添加控制位
    for term in identified_terms:
        for op_type, wires in term:
            if op_type == 'X':
                for wire in wires:
                    qml.CNOT(wires=[control_wire, wire])
                    qml.PauliX(wires=wire)
                    qml.CNOT(wires=[control_wire, wire])
            elif op_type == 'Y':
                for wire in wires:
                    qml.CNOT(wires=[control_wire, wire])
                    qml.RY(np.pi / 2, wires=wire)
                    qml.PauliX(wires=wire)
                    qml.RY(-np.pi / 2, wires=wire)
                    qml.CNOT(wires=[control_wire, wire])
            elif op_type == 'Z':
                for wire in wires:
                    qml.CNOT(wires=[control_wire, wire])
                    qml.PauliZ(wires=wire)
                    qml.CNOT(wires=[control_wire, wire])
```

图 9: 由基础门构成的受控 A_l

```
# 最优参数
w = qml_np.array([3.14161205e+00,  5.80377608e+00,  3.93857269e+00,
                  4.71236102e+00,  4.42799279e+00,  5.32962513e+00,
                  2.88556879e-05, -6.91562619e-02,  4.79375314e-01])

loss_value = VQLS_until.cost_loc(w)

print("loss_value = ", loss_value)
✓ 1m 40.5s
loss_value =  5.376021627867544e-09
```

```
loss_value_2 = circuit_until.cost_loc(w)
print("loss_value_2 = ", loss_value_2)
✓ 59.4s
loss_value_2 =  5.551115123125783e-17
```

图 10: 由基础门构成的 Hadamard test 电路与原始电路的数值对比

C. Result

同样，我们依次展示 VQLS 中不同 ipynb 文件的运算结果，并说明。

1. Introduction VQLS

首先，查看 VQLS_introduction.ipynb，这里展示了，将本文线性方程组 Ax = b 转换成能用 VQLS 求解的形式 $\mathcal{A}|x\rangle = |b\rangle$，再从实际的量子态 $|x\rangle$ 出发，恢复成线性方程组的解 x。

我们采用块编码，将 A 变成厄尔米特矩阵，再通过补 0 变成 $2^3 \times 2^3$ 的矩阵，同时要将 x 和 b 做对应的扩

展后进行归一化，由公式(39)将 $|x\rangle$ 恢复成 $\mathrm{x}^{''}$，不需要计算 $\|\mathrm{x}^{''}\|$，

$$\mathcal{A} = \begin{bmatrix} 0 & \mathrm{A} \\ \mathrm{A}^{\dagger} & 0 \end{bmatrix}, \tag{36}$$

$$\mathrm{x}^{''} = [0, 0, 0, 1200, 500, 300, 0, 0, 0]^{\mathrm{T}}, \tag{37}$$

$$\mathrm{b}^{''} = [1000, 0, -600, 0, 0, 0, 0, 0, 0]^{\mathrm{T}}, \tag{38}$$

$$\mathrm{x}^{''} = \frac{|x\rangle}{\sqrt{\langle x| \mathcal{A}^{\dagger}\mathcal{A} |x\rangle}} \cdot \|\mathrm{b}^{''}\|_2 \tag{39}$$

```
x_norm = np.linalg.norm(x_vec_2)
x_normalized = (x_vec_2 / x_norm).reshape(-1)
print("原向量: ", x_vec_2.reshape(-1))
print("归一化后的向量: ", x_normalized)
```

✓ 0.1s

```
原向量:  [   0.    0.    0. 1200.  500.  300.    0.    0.]
归一化后的向量:  [0.         0.         0.         0.89943803 0.37476584 0.22485951
 0.         0.        ]
```

```
# 从量子态 |x> 恢复成解 x，不需要 ||x||
b_norm * x_normalized/(np.sqrt(x_normalized.reshape(1,-1)
@ H_matrix_2.T.conj() @ H_matrix_2 @ x_normalized.reshape(-1,1)))
```

✓ 0.0s

```
array([[   0.,    0.,    0., 1200.,  500.,  300.,    0.,    0.]])
```

图 11: $|x\rangle$ 恢复成 $\mathrm{x}^{''}$

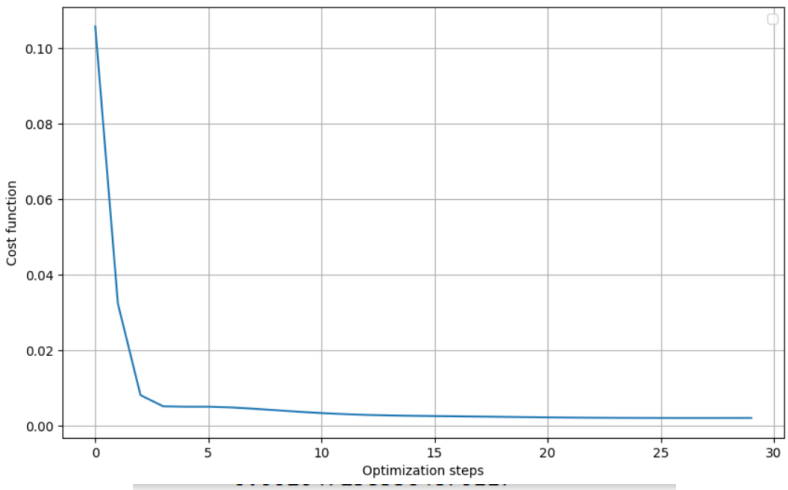
2. VQLS Multiprocessing

其次，我们查看 1_Multiprocessing_demo 中的 multiprocessing_demo.ipynb。由于该算法每计算一次损失值 $\mathcal{C}_L$，要计算不同的 $\mu_{l,l',j}$ 5000+ 次，在代码中形成一个多层循环嵌套，在更新参数阶段会极大的拖慢速度。在本地电脑上，每更新一次参数，需要运算 20min，所以在本文件中，我们采用多进程来运算，将时间压缩到 2min 迭代一次。但 pennylane 的自动微分无法与 multiprocessing 结合，所以需要手写优化器更新参数，拖慢了一部分速度。

在 multiprocessing_demo.ipynb 中，我们选择随机生成参数，进行优化，

$$\begin{aligned} \theta_{init} = (&4.06956402, 6.26379276, 3.25973705, \\ &4.13504424, 3.7640268, 4.73166161, \\ &0.85606595, 0.02586861, 0.939392), \\ \theta_{end} = (&4.47641446, 5.63449017, 4.48172876, \\ &3.65588352, 2.92052642, 4.36835527, \\ &1.20978976, 0.48213525, 0.31008941). \end{aligned} \tag{40}$$

迭代终止后, 我们损失值最终收敛到 0.002 左右, 但这一级别的损失值并不能让我们此解析出正确的答案。图(12)表示迭代过程，图(14) 是归一化得到的量子态 $|x\rangle$，与变分电路由迭代终止后的参数生成的量子态，它们之间的测量概率对比，可以看到，概率不一致，恢复的答案 x″(图(13))，与实际值也相差巨大。




```
Cost_L = 0.0020360
time =  137.48465418815613
step  =  26
0.0020360293891863446
Cost_L = 0.0020314
time =  135.0967161655426
step  =  27
0.002031378611779766
Cost_L = 0.0020315
time =  138.10048294067383
step  =  28
0.002031505449243909
Cost_L = 0.0020349
time =  137.464035987854
step  =  29
0.0020348577524452738
Cost_L = 0.0020401
time =  136.02001929283142
```

图 12: VQLS: 初始随机参数的优化

```
  0.22796859+0.j -0.75215802+0.j -0.12070895+0.j  0.11860557+0.j]
x_state_value=  [[  5.18231708+0.j   7.35846203+0.j   7.21155662+0.j  47.20255012+0.j
  18.62705005+0.j -61.45796275+0.j  -9.86298859+0.j   9.69112388+0.j]]
/root/miniconda3/envs/pennylane_gpu/lib/python3.9/site-packages/pennylane/_qubit_device.py
```

图 13: 变分电路生成的量子态恢复的解 x″

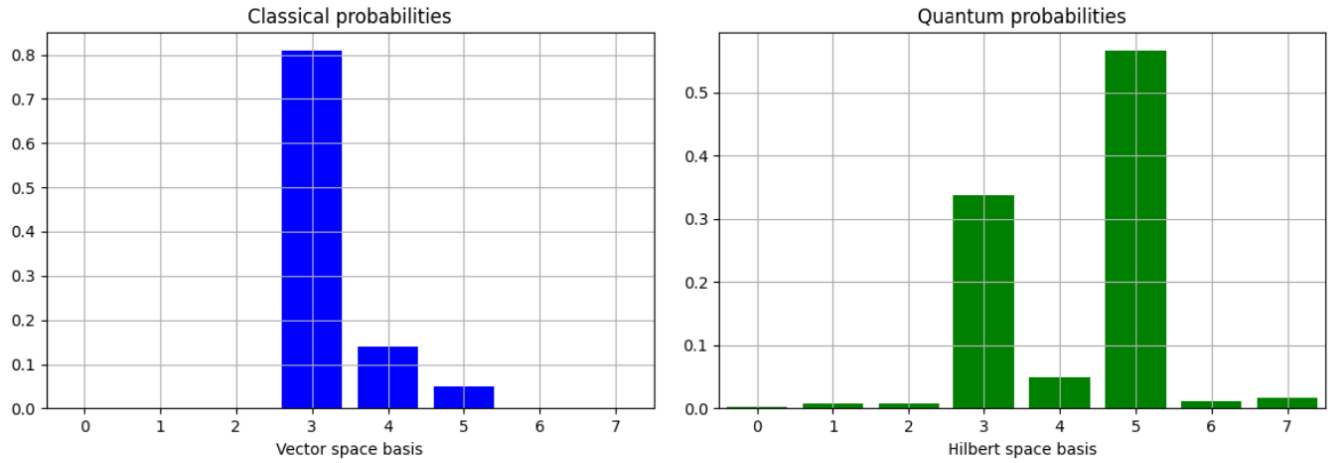


图 14: $|x\rangle$ 与变分电路生成的量子态的测量概率对比

启示

尽管损失值达到了 0.002 的级别，但依然无法让我学到最优解，需要进一步提高终止条件，达到 10^{-4} 或更低的量级。同时，还由于变分电路门的参数量较多，导致随机选取的初始参数很难学到最优解。

3. VQLS Idea

由于选择随机的初始参数，损失函数比较难优化到 10^{-4} 及更低的量级，导致恢复出来的解，与实际解差距较大。所以我们在 4_state_circuit 中的 x_state_circuit.ipynb，训练变分电路的参数，使得生成的量子态与归一化后的 x'' 靠近，由此得到一组最优参数 θ_{opt}，

$$\begin{aligned}\theta_{opt} = (&4.13345498, 6.96129538, 3.15416639,\\ &3.8794304, 4.86300243, 4.68250843,\\ &0.41432465, 2.37792433, 1.63689462),\end{aligned} \tag{41}$$

使用参数 θ_{opt} 计算损失函数 $\mathcal{C}_L$，将满足我们小于 10^{-4} 的终止条件，

```
# 最优参数
w = qml_np.array([3.14161205e+00,  5.80377608e+00,  3.93857269e+00,
                  4.71236102e+00,  4.42799279e+00,  5.32962513e+00,
                  2.88556879e-05, -6.91562619e-02,  4.79375314e-01])

loss_value = VQLS_until.cost_loc(w)

print("loss_value = ", loss_value)
```

✓ 1m 28.7s

```
loss_value =  5.376024680980862e-09
```

图 15: 最优参数时的损失值

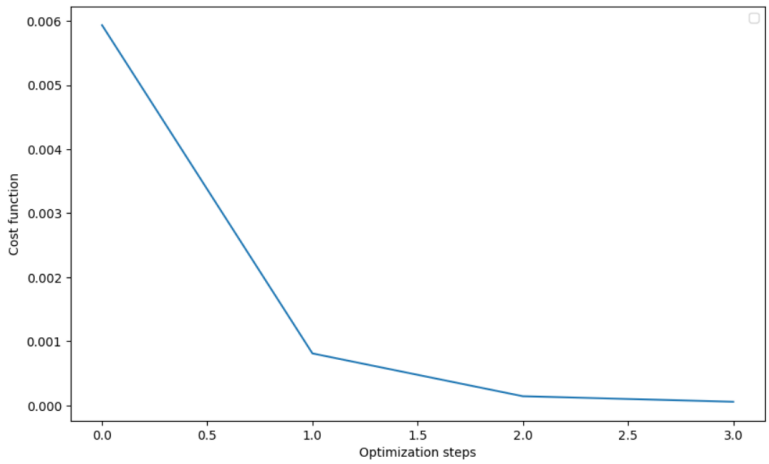
于是，我们在最优参数的基础上，添加随机扰动，由此开始 VQLS 的训练，见 2_Idea_demo 中的 VQLS_opt.ipynb 和 VQLS_train.ipynb，迭代终止后损失值为 0.0000569，此时的初始参数和迭代终止参数如

下:

$$\begin{aligned}\theta_{init} = (&3.14456587, 5.81371435, 3.93894876,\\ &4.71552327, 4.42997406, 5.33468648,\\ &-0.0072462, -0.07907392, 0.47236549),\\ \theta_{end} = (&3.14205787, 5.81537851, 3.94134631,\\ &4.71524587, 4.42473918, 5.33191857,\\ &-0.00133491316, -0.0743916138, 0.474029654).\end{aligned} \tag{42}$$

图(17)展示迭代优化后生成的量子态测量概率与实际 $|x\rangle$ 的基本一致，而图(16)除了展示优化过程，还包括其使用优化后的变分量子态恢复的解为 $(1200.61, 499.05, 300.10)$，保留两位有效数字，与实际解的绝对误差为

$$|1200.61 - 1200| + |499.05 - 500| + |300.10 - 300| = 0.76, \tag{43}$$




```
x_state_value=  [[-5.29424198e-02+0.j -2.09940545e-01+0.j -9.80056423e-02+0.j
  1.20060697e+03+0.j  4.99051166e+02+0.j  3.00098651e+02+0.j
  2.76856218e+00+0.j  1.66147916e-01+0.j]]
/root/miniconda3/envs/pennylane_gpu/lib/python3.9/site-packages/pennylane/_qubit_dev
```

图 16: VQLS: 最优参数随机扰动的优化

4. VQLS Noise

最后，查看 3_Noise_demo 中的文件，我们在变分量子电路的最后添加了去极化噪声。VQLS_Noise_prob.ipynb 中图(18)展示了，三个不同程度的去极化噪声，在最优参数下变分电路生成的量子态的测量概率。可以看到随着噪声级别的正大，正确基向量的测量概率逐步变小。

在 VQLS_Noise_train.ipynb，我们在 Hadamard test 电路后添加 0.01 的去极化噪声，并进行 VQLS 的训

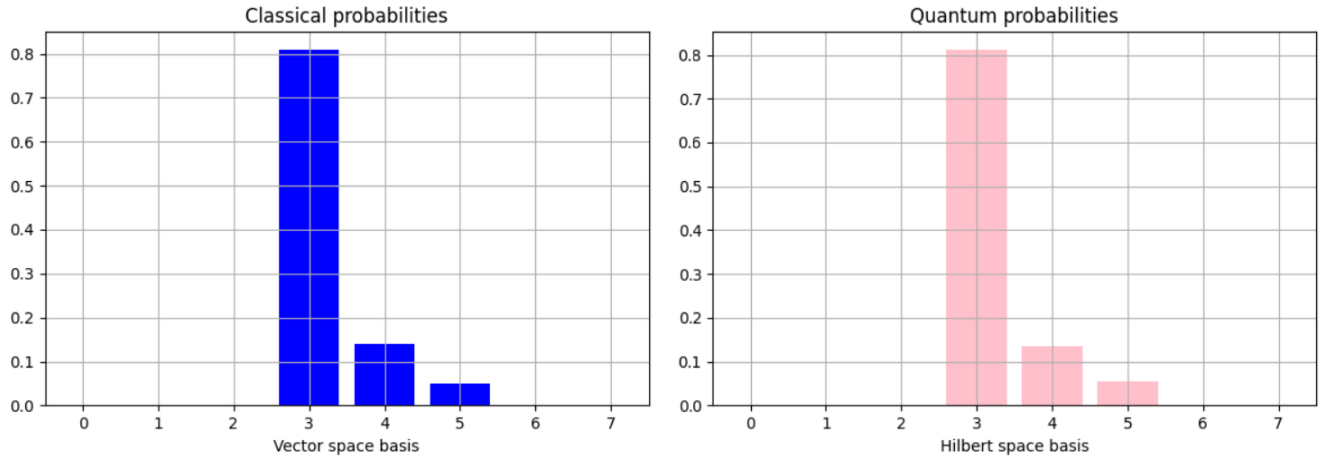


图 17: 最优参数扰动下，$|x\rangle$ 与变分电路生成的量子态的测量概率对比

练，损失值最终达到 0.0059761，也没有满足我们的终止条件，初始参数和终止参数如下：

$$\begin{aligned}\theta_{init} = (&3.14456587, 5.81371435, 3.93894876,\\ &4.71552327, 4.42997406, 5.33468648,\\ &-0.0072462, -0.07907392, 0.47236549),\\ \theta_{end} = (&3.38703632, 5.7479508, 4.01636212,\\ &3.24634907, 4.70685735, 4.18882633,\\ &-0.02817045, -0.19982191, 0.40660194).\end{aligned} \tag{44}$$

图(19)优化过程中损失值还是未能小于终止条件 10^{-4}，其使用优化后的变分量子态恢复的解为 $(859.27, 379.14, 310.45)$，与实际解的总的绝对误差为

$$|859.27 - 1200| + |379.14 - 500| + |310.45 - 300| = 472.04, \tag{45}$$

注意

在噪声模拟器上，无法返回态矢量，只能通过返回测量概率的方式来获得电路的态矢量。从该实验可以看出，在含噪的环境下，使用复杂的 VQA 算法，会极大的影响解的准确性，需要考虑 QEM 中的相关算法来缓解噪声的影响。

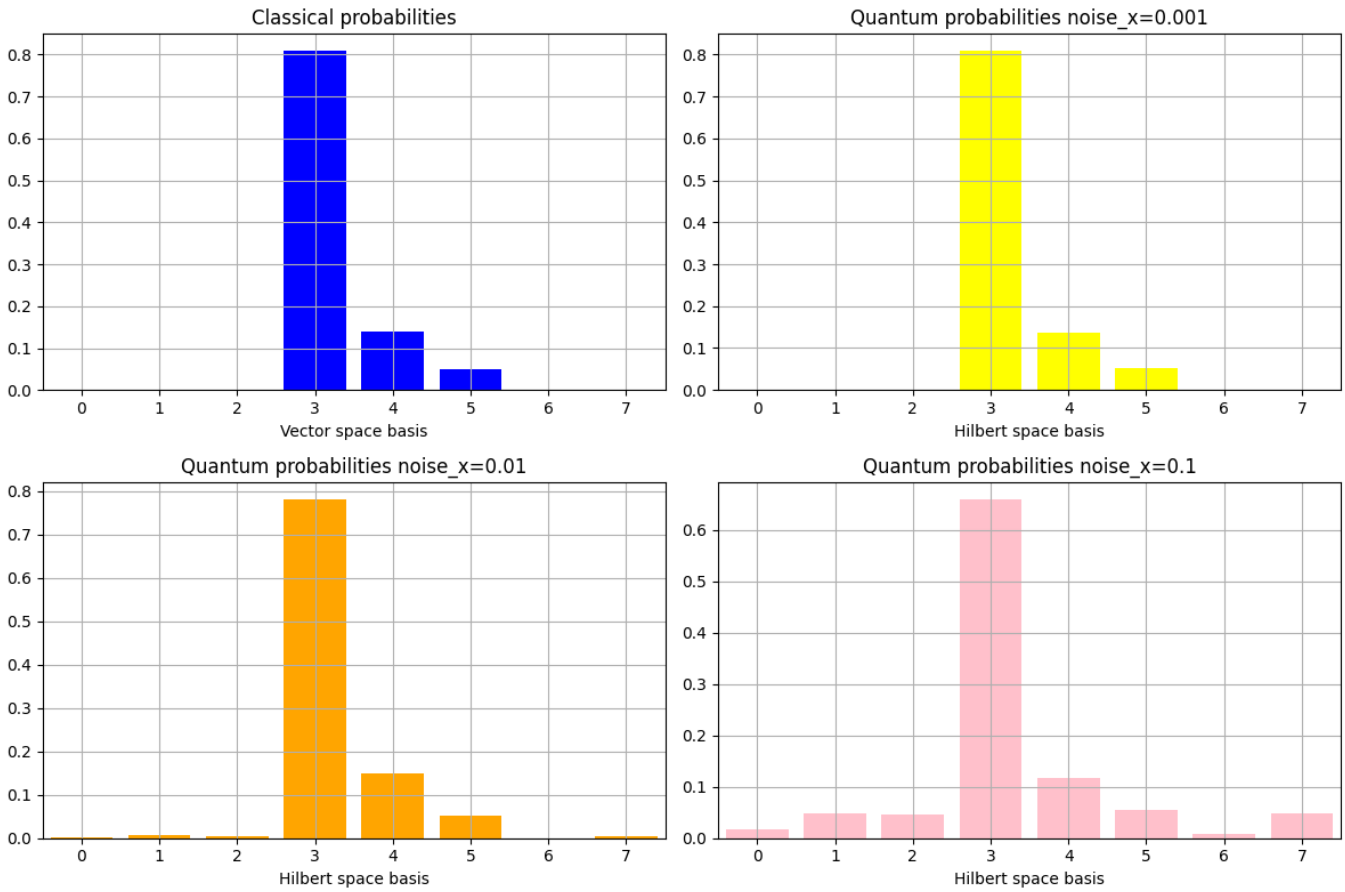
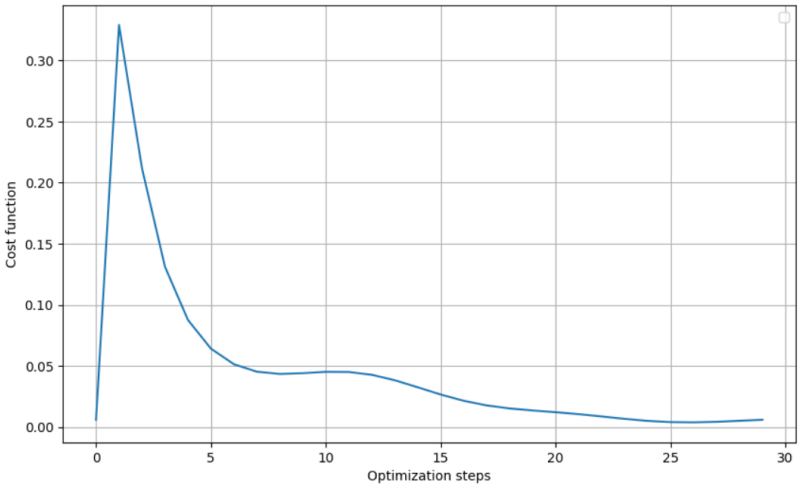


图 18: 不同程度去极化噪声环境下，量子态的测量概率




```
 0.28726213 0.07328775]
x_state_sample_value=  [[121.25016677 721.29979412 110.6857524  859.27130172 379.14246206
 310.4467343  371.52619895  94.78561552]]
```

图 19: VQLS: 含噪环境下，最优参数随机扰动的优化

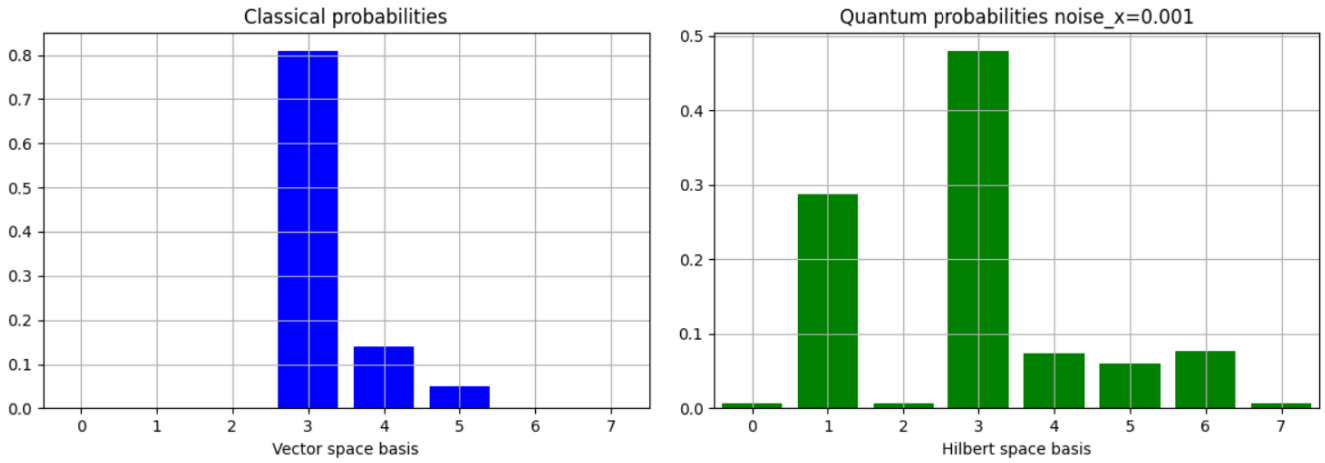


图 20: 去极化噪声水平为 0.1，经过 VQLS 训练后，量子态的测量概率

[BPLC+23] Carlos Bravo-Prieto, Ryan LaRose, Marco Cerezo, Yigit Subasi, Lukasz Cincio, and Patrick J Coles. Variational quantum linear solver. Quantum, 7:1188, 2023.

[FLH+24] Jing-Kai Fang, Yue-Feng Lin, Jun-Han Huang, Yibo Chen, Gao-Ming Fan, Yuhui Sun, Guanru Feng, Cong Guo, Tiejun Meng, Yong Zhang, et al. Divide-and-conquer quantum algorithm for hybrid de novo genome assembly of short and long reads. PRX Life, 2(2):023006, 2024.

[FPGA22] Dmitry A Fedorov, Bo Peng, Niranjan Govind, and Yuri Alexeev. Vqe method: a short survey and recent developments. Materials Theory, 6(1):2, 2022.

[Jun24] Kyungtaek Jun. Qubo formulations for a system of linear equations. Results in Control and Optimization, 14:100380, 2024.