Using HHL Algorithm to Solve Linear System of Equations and its Application in Machine Learning

Shuohao Ping

Rutgers University, New Brunswick

1. Introduction

Linear systems of equations play an important role in the fields of computer science, finance, and engineering. For example, linear systems are used to solve optimization problems in both finance and engineering. In computer science, machine learning is heavily based on linear equations. Usually, a machine learning model is a function with undetermined constant coefficients. We need to train the machine learning model with a dataset in order to determine those coefficients. This is done by doing the least square fitting on the given dataset, which requires solving a linear system of equations. Hence, a speedup in solving a linear system of equations will help to reduce the time needed for training machine learning models. Solving a linear system of equations can be done in polynomial time classically. For instance, for a system $Ax = b$ where A is a $n \times n$ matrix, x and b are $n \times 1$ vectors, we need $O(n^3)$ time to solve it using Gaussian elimination [1]. Some iterative numerical methods can approximate the result with better time complexity, like $O\left(ns\kappa \log \frac{1}{\varepsilon}\right)$ using the conjugate gradient method [2], where s is the maximum number of nonzero entries for every row or column of the matrix, κ is the condition number of the matrix ($\kappa = \|A\|\|A^{-1}\|$), and ε is the accuracy of the approximation that $\frac{\|x_{exact} - x_{approximate}\|}{\|x_{exact} - x_{initial}\|} \leq \varepsilon$, where x_{exact} is the exact solution to the system $Ax = b$, $x_{approximate}$ is the approximate solution after iterations, and $x_{initial}$ is the initial vector used by this iterative method. In 2009, Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd found a quantum algorithm that can for solving linear systems of equations [3]. The Harrow-Hassidim-Lloyd (HHL) algorithm calculates the direction of the solution vector of system $Ax = b$ where A is a $N \times N, N = 2^n, n \in \mathbb{N}$ Hermitian matrix. The measurement result of the HHL algorithm, $x_{measure}$, is a constant multiple of the exact solution, which means $x_{measurement} = cx_{exact}, c \in R$. The time complexity of the HHL algorithm is $O\left(\frac{s^2\kappa^2}{\varepsilon}\log N\right)$, which provides an exponential speedup compared to all classical algorithms. In this project, we will explain how the original HHL algorithm works, and we will explore strategies for solving systems that are not Hermitian or with a dimension of 2^n. We will also discuss the case of A being a rectangular system ($A\ is\ m \times n, m \neq n$).

2. HHL Algorithm

The question that the HHL algorithm can solve is given a $N \times N$ Hermitian matrix A and a $N \times 1$ vector b where $N = 2^n$, solving the system $Ax = b$. Note that the result given by the HHL algorithm is the direction of the solution vector due to the fact that we need to normalize b before we can encode it in the circuit, and computing the exact solution would require $O(n)$ time. We will discuss advanced treatment for A when A is not Hermitian or the dimension of A is not 2^n, or A is not a square matrix in the next section. In this section, we will focus on the simplest case that A is $N \times N$ Hermitian matrix A and a $N \times 1$ vector b where $N = 2^n$ and $||b|| = 1$.

The HHL algorithm solves $Ax = b$ by exploiting the property of the Hermitian matrix. Since A is a Hermitian matrix, then all eigenvectors of A form a basis for $range(A)$, and the spectral decomposition for matrix A is $A = \sum_{i=1}^{N} \lambda_i |u_i\rangle\langle u_i|$, where λ_i is one of the eigenvalues of A, and $|u_i\rangle$ is the eigenvector corresponding to eigenvalue λ_i. Then, $A^{-1} = \sum_{i=1}^{N} \lambda_i^{-1} |u_i\rangle\langle u_i|$ and we can solve for x by computing $x = A^{-1}b$. The HHL algorithm follows similar steps. Firstly, we need to encode vector b into the circuit by encoding entries of b on a set of qubits as amplitudes such that $b = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{N-1} \end{pmatrix} = \beta_0|0\rangle + \beta_1|1\rangle + \cdots + \beta_N|N-1\rangle$.

Then, we will convert A into a unitary matrix $U = e^{iAt}$ where $t \neq 0$, and we will use the unitary matrix U for the quantum phase estimation (QPE) to get the eigenvalues of U, which are encoded on another set of qubits as basis rather than amplitudes. Note that, in general, b is not an eigenvector of U, so instead of having one eigenvalue, we will have a superposition of eigenvalues after QPE. Then, we will rotate the ancilla qubit in order to perform eigenvalue inversion. Finally, we will use inverse quantum phase estimation to calculate $A^{-1}x$, and perform measurements to obtain results. The resulting vector is encoded as amplitudes like b, so we need to perform multiple measurements to obtain the distribution for the result vector. Fig. 1 is the high-level demonstration of the circuit for the HHL algorithm.

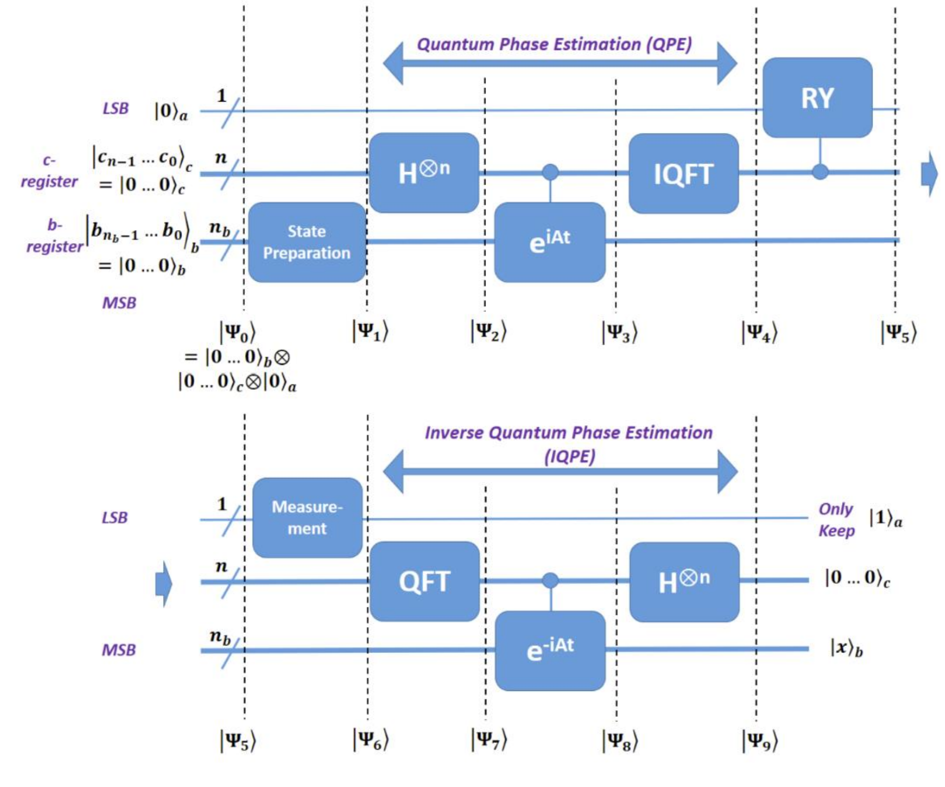


Fig.1 Circuit for HHL Algorithm (Morrell Jr et al., 2023) [4]

In Fig. 1, the qubits are separated into three groups. The first group, $|0\rangle_a$, is the ancilla qubit for eigenvalue inversion. The second group, called c-register, is for storing eigenvalues, and the number of qubits in the c-register depends on the accuracy you want for the eigenvalues. The last group, called b-register, is for storing the initial vector b and result vector x. The number of qubits in this group depends on the dimension of vector b. If b is a $N \times 1$ vector where $N = 2^n$, then we need n qubits in the b-register.

Now, we will trace the states of the system after each operation in Fig. 1. This part is based on [4].

Initially, all qubits are in $|0\rangle$ state.

$$|\Psi_0\rangle = |0\rangle_a \otimes |0 \dots 0\rangle_c \otimes |0 \dots 0\rangle_b$$

At the state preparation stage, we will encode vector b into b-register as $|b\rangle = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{N-1} \end{pmatrix} =$ $\beta_0|0\rangle + \beta_1|1\rangle + \cdots + \beta_N|N-1\rangle$.

Hence, we have

$$|\Psi_1\rangle = |0\rangle_a \otimes |0 \dots 0\rangle_c \otimes |b\rangle_b$$

Then, we will perform QPE on the b-register and c-register to get the eigenvalues of U, and we need the relationship between eigenvalues and eigenvectors of U and H. Since $U = e^{iAt}$ and A is Hermitian, then $A = P\begin{pmatrix} \lambda_0 & 0 & 0 & 0 \\ 0 & \lambda_1 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \lambda_{2^n-1} \end{pmatrix}P^{-1}$ and $U =$ $P\begin{pmatrix} e^{i\lambda_0 t} & 0 & 0 & 0 \\ 0 & e^{i\lambda_1 t} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & e^{i\lambda_{2^n-1} t} \end{pmatrix}P^{-1}$, where $P = (u_0\ u_1 \dots u_{2^n-1})$ such that u_j $(0 \le j \le 2^n - 1)$ is the eigenvector corresponding to eigenvalue λ_j. Thus, U and A have the same eigenvectors. Let $\tilde{\lambda}_j (0 \le j \le 2^n - 1)$ be the eigenvalue of U, then $\tilde{\lambda}_j = e^{i\lambda_j t}$. Since $U|u_i\rangle = \tilde{\lambda}_j|u_j\rangle = e^{i\lambda_j t}|u_j\rangle = e^{i2\pi\phi_j}|u_j\rangle$, then $\phi_j = \frac{\lambda_j t}{2\pi}$. Also, since all eigenvectors of

A form a basis for $range(A)$, then $|b\rangle = b_0|u_0\rangle + b_1|u_1\rangle + \cdots + b_{2^n-1}|u_{2^n-1}\rangle$ for some constants $b_0, b_1, \dots b_{2^n-1}$. Now, we are ready to discuss $|\Psi_2\rangle, |\Psi_3\rangle, |\Psi_4\rangle$.

We perform a Hadamard transform to $|\Psi_1\rangle$ to get $|\Psi_2\rangle$. Hence,

$$|\Psi_2\rangle = |0\rangle_a \otimes \frac{1}{2^{\frac{n}{2}}}\big((|0\rangle + |1\rangle) \otimes (|0\rangle + |1\rangle) \otimes \dots \otimes (|0\rangle + |1\rangle)\big)_c \otimes |b\rangle_b$$

After applying unitary operations, we have

$$|\Psi_3\rangle = \frac{1}{2^{\frac{n}{2}}}\textstyle\sum_{j=0}^{2^n-1}|0\rangle_a \otimes \Big(\big(|0\rangle + e^{i2\pi\phi_j 2^{n-1}}|1\rangle\big) \otimes \big(|0\rangle + e^{i2\pi\phi_j 2^{n-2}}|1\rangle\big) \otimes \dots \otimes$$
$$\big(|0\rangle + e^{i2\pi\phi_j 2^0}|1\rangle\big)\Big)_c \otimes |u_i\rangle_b = \frac{1}{2^{\frac{n}{2}}}\textstyle\sum_{j=0}^{2^n-1}\sum_{k=0}^{2^n-1} b_j|0\rangle_a \otimes \big(e^{i2\pi\phi_j k}|k\rangle\big)_c \otimes |u_j\rangle_b$$

Then, we apply the inverse quantum Fourier transform.

$$|\Psi_4\rangle = \frac{1}{2^{\frac{n}{2}}}\sum_{j=0}^{2^n-1}\sum_{k=0}^{2^n-1} b_j|0\rangle_a \otimes IQFT\Big(\big(e^{i2\pi\phi_j k}|k\rangle\big)_c\Big) \otimes |u_j\rangle_b$$

$$= \frac{1}{2^n}\sum_{j=0}^{2^n-1} b_j \sum_{y=0}^{2^n-1}\sum_{k=0}^{2^n-1}|0\rangle_a \otimes e^{2\pi ik\left(\phi_j - \frac{y}{2^n}\right)}|y\rangle_c \otimes |u_j\rangle_b$$

When $\phi_j = \frac{y}{2^n}$, we have $\sum_{k=0}^{2^n-1} e^{2\pi ik\left(\phi - \frac{y}{2^n}\right)} = 2^n$, and when $\phi_j \neq \frac{y}{2^n}$, we have $\sum_{k=0}^{2^n-1} e^{2\pi ik\left(\phi - \frac{y}{2^n}\right)} = 0$. Hence, we can rewrite $|\Psi_4\rangle$ as

$$|\Psi_4\rangle = \frac{1}{2^n}\sum_{j=0}^{2^n-1} b_j \sum_{y=0}^{2^n-1}|0\rangle_a \otimes e^{2\pi ik0}|2^n\phi_j\rangle_c \otimes |u_j\rangle_b$$

$$= \sum_{j=0}^{2^n-1} b_j|0\rangle_a \otimes |2^n\phi_j\rangle_c \otimes |u_j\rangle_b$$

$$= \textstyle\sum_{j=0}^{2^n-1} b_j|0\rangle_a \otimes \left|\frac{N\lambda_j t}{2\pi}\right\rangle_c \otimes |u_j\rangle_b \text{ since } \phi_j = \frac{\lambda_j t}{2\pi} \text{ and } N = 2^n$$

For simplicity, let $\tilde{\lambda}_j = \frac{Nt}{2\pi}\lambda_j$, which means $\tilde{\lambda}_j$ is a scaled version of λ_j.

Hence,

$$|\Psi_4\rangle = \sum_{j=0}^{2^n-1} b_j |0\rangle_a \otimes \left|\tilde{\lambda}_j\right\rangle_c \otimes \left|u_j\right\rangle_b$$

Next, we need to apply an RY rotation in order to perform eigenvalue inversion. The goal is to change the state $|\Psi_5\rangle$ such that

$$|\Psi_5\rangle = \sum_{j=0}^{2^n-1} b_j \left(\sqrt{1-\left(\frac{C}{\tilde{\lambda}_j}\right)^2}\, |0\rangle_a + \frac{C}{\tilde{\lambda}_j} |1\rangle_a \right) \otimes \left|\tilde{\lambda}_j\right\rangle_c \otimes \left|u_j\right\rangle_b \; \textit{where C is some constant}$$

In general, we want to make C as large as we can so that we are more likely to measure the state $|1\rangle_a$ in the next step. If the measurement result of the ancilla qubit is $|0\rangle_a$, we will discard the current result and repeat the previous steps until we measure the state $|1\rangle_a$ for the ancilla qubit.

Hence,

$$|\Psi_6\rangle = \frac{1}{\sqrt{\sum_{k=0}^{2^n-1}\left|\frac{b_j C}{\tilde{\lambda}_j}\right|^2}} \sum_{j=0}^{2^n-1} \frac{b_j C}{\tilde{\lambda}_j} |1\rangle_a \otimes \left|\tilde{\lambda}_j\right\rangle_c \otimes \left|u_j\right\rangle_b$$

Finally, we will perform inverse QPE to compute $x = A^{-1}b$. The inverse QPE circuit has three components: QFT, controlled-U operations, and Hadamard transform.

Firstly, after QFT, the state changes to

$$|\Psi_7\rangle = \frac{1}{\sqrt{\sum_{k=0}^{2^n-1}\left|\frac{b_j C}{\tilde{\lambda}_j}\right|^2}} \sum_{j=0}^{2^n-1} \frac{b_j C}{\tilde{\lambda}_j} |1\rangle_a \otimes QFT\left(\left|\tilde{\lambda}_j\right\rangle_c\right) \otimes \left|u_j\right\rangle_b$$

$$= \frac{1}{\sqrt{\sum_{k=0}^{2^n-1}\left|\frac{b_j C}{\tilde{\lambda}_j}\right|^2}} \sum_{j=0}^{2^n-1} \frac{b_j C}{\tilde{\lambda}_j} |1\rangle_a \otimes \left(\frac{1}{2^{\frac{n}{2}}} \sum_{y=0}^{2^n-1} e^{\frac{2\pi i y \tilde{\lambda}_j}{N}} |y\rangle_c \right) \otimes \left|u_j\right\rangle_b$$

Then, we would apply controlled-U operations. Note that $U = e^{-iAt}$ for inverse QPE.

$$|\Psi_8\rangle = \frac{1}{\sqrt{\sum_{k=0}^{2^n-1}\left|\frac{b_jC}{\tilde{\lambda}_j}\right|^2}} \sum_{j=0}^{2^n-1} \frac{b_jC}{\tilde{\lambda}_j} |1\rangle_a \otimes \left(\frac{1}{2^{\frac{n}{2}}} \sum_{y=0}^{2^n-1} e^{\frac{2\pi i y \tilde{\lambda}_j}{N} - 2\pi i \phi_j y} |y\rangle_c\right) \otimes |u_j\rangle_b$$

Since $\phi_j = \frac{\lambda_j t}{2\pi}$ and $\tilde{\lambda}_j = \frac{Nt}{2\pi}\lambda_j$, then $\frac{\tilde{\lambda}_j}{N} = \phi_j$ and we have

$$|\Psi_8\rangle = \frac{1}{\sqrt{\sum_{k=0}^{2^n-1}\left|\frac{b_jC}{\tilde{\lambda}_j}\right|^2}} \sum_{j=0}^{2^n-1} \frac{b_jC}{\tilde{\lambda}_j} |1\rangle_a \otimes \left(\frac{1}{2^{\frac{n}{2}}} \sum_{y=0}^{2^n-1} e^{2\pi i y\left(\frac{\tilde{\lambda}_j}{N} - \phi_j\right)} |y\rangle_c\right) \otimes |u_j\rangle_b$$

$$= \frac{1}{2^{\frac{n}{2}}\sqrt{\sum_{k=0}^{2^n-1}\left|\frac{b_jC}{\tilde{\lambda}_j}\right|^2}} \sum_{j=0}^{2^n-1} \frac{b_jC}{\tilde{\lambda}_j} |1\rangle_a \otimes \left(\sum_{y=0}^{2^n-1} |y\rangle_c\right) \otimes |u_j\rangle_b$$

$$= \frac{C}{2^{\frac{n}{2}}\sqrt{\sum_{k=0}^{2^n-1}\left|\frac{b_jC}{\lambda_j}\right|^2}} \sum_{j=0}^{2^n-1} \frac{b_j}{\lambda_j} |1\rangle_a \otimes \left(\sum_{y=0}^{2^n-1} |y\rangle_c\right) \otimes |u_j\rangle_b$$

Since $x = A^{-1}b$, $b = \sum_{j=0}^{2^n-1} b_j u_j$, and $A^{-1} = \sum_{j=0}^{2^n-1} \frac{1}{\lambda_j} |u_j\rangle\langle u_j|$, then $x = \sum_{j=0}^{2^n-1} \frac{b_j}{\lambda_j} u_j$.

Hence,

$$|\Psi_8\rangle = \frac{C}{2^{\frac{n}{2}}\sqrt{\sum_{k=0}^{2^n-1}\left|\frac{b_jC}{\lambda_j}\right|^2}} |1\rangle_a \otimes \left(\sum_{y=0}^{2^n-1} |y\rangle_c\right) \otimes \left(\sum_{j=0}^{2^n-1} \frac{b_j}{\lambda_j} |u_j\rangle_b\right)$$

$$|\Psi_8\rangle = \frac{C}{2^{\frac{n}{2}}\sqrt{\sum_{k=0}^{2^n-1}\left|\frac{b_jC}{\lambda_j}\right|^2}} |1\rangle_a \otimes \left(\sum_{y=0}^{2^n-1} |y\rangle_c\right) \otimes |x\rangle_b$$

Since $\sum_{y=0}^{2^n-1}|y\rangle_c = \big((|0\rangle + |1\rangle) \otimes (|0\rangle + |1\rangle) \otimes \ldots \otimes (|0\rangle + |1\rangle)\big)_c$, we have

$$|\Psi_8\rangle = \frac{C}{\sqrt{\sum_{k=0}^{2^n-1}\left|\frac{b_j C}{\lambda_j}\right|^2}}|1\rangle_a \otimes \frac{1}{2^{\frac{n}{2}}}\big((|0\rangle + |1\rangle) \otimes (|0\rangle + |1\rangle) \otimes \ldots \otimes (|0\rangle + |1\rangle)\big)_c \otimes |x\rangle_b$$

Finally, we apply Hadamard transform since the inverse of the Hadamard transform is itself.

$$|\Psi_9\rangle = \frac{C}{\sqrt{\sum_{k=0}^{2^n-1}\left|\frac{b_j C}{\lambda_j}\right|^2}}|1\rangle_a \otimes |0\ldots0\rangle_c \otimes |x\rangle_b$$

$= \frac{1}{\sqrt{\sum_{k=0}^{2^n-1}\left|\frac{b_j}{\lambda_j}\right|^2}}|1\rangle_a \otimes |0\ldots0\rangle_c \otimes |x\rangle_b$ by dividing C in the numerator and denominator

Since $|x\rangle = \sum_{j=0}^{2^n-1}\frac{b_j}{\lambda_j}|u_j\rangle$ is a unit vector, then $\sum_{j=0}^{2^n-1}\left(\frac{b_j}{\lambda_j}\right)^2 = 1$.

Hence,

$$|\Psi_9\rangle = |1\rangle_a \otimes |0\ldots0\rangle_c \otimes |x\rangle_b$$

We can obtain the result vector $|x\rangle$ by measuring the b-register.

3. Using HHL to solve general linear systems

In part 2, we assume A is a $N \times N$ Hermitian matrix where $N = 2^n$, and b is a unit vector. However, a general linear system may not satisfy any of these conditions. Hence, we need to use some techniques in order to use the HHL algorithm to solve the system. Firstly, we will discuss the case for a random square system (A is a $N \times N$ matrix).

Let A be a $M \times M$ matrix and b be a $M \times 1$ vector. Firstly, we need to normalize b by calculating $b' = \frac{b}{\|b\|}$ so that b' can be encoded as amplitudes in the quantum circuit. Note that this transformation will affect the result x since now we are solving $Ax = b'$ instead of $Ax = b$. This is the reason that the HHL algorithm can only give the direction of the result vector x since x has to be a unit vector because it's encoded as amplitudes, but the exact answer may not be a unit vector.

After normalization, we need to check whether A is Hermitian. If not, we can let $A' = \begin{pmatrix} 0 & A \\ A^\dagger & 0 \end{pmatrix}$ be a $2M \times 2M$ matrix, and $b'' = \begin{pmatrix} b' \\ \vec{0} \end{pmatrix}$ where $\vec{0}$ is the 0 vector with dimension $M \times 1$ and solves for system $A'x' = b''$. We claim that A' is always Hermitian because $A^\dagger = \begin{pmatrix} 0 & A \\ A^\dagger & 0 \end{pmatrix}^\dagger = \begin{pmatrix} 0 & (A^\dagger)^\dagger \\ A^\dagger & 0 \end{pmatrix} = \begin{pmatrix} 0 & A \\ A^\dagger & 0 \end{pmatrix} = A$. After this transformation, the result of the system $Ax = b'$ is the latter half of the answer to the new system $A'x' = b''$, which means if $x' = \begin{pmatrix} x_1 \\ \vdots \\ x_M \\ \vdots \\ x_{2M} \end{pmatrix}$, then the answer to the system $Ax = b'$ is $x = \begin{pmatrix} x_{M+1} \\ \vdots \\ x_{2M} \end{pmatrix}$.

Following the previous step, we will have a new system of $Ax = b$ where A is Hermitian, and b is a unit vector. If A has a dimension $M \times M$ where $M \neq 2^n \, for \, all \, n \in \mathbb{N}$. We need to perform the following transformation. Let $A' = \begin{pmatrix} A & 0 & \cdots & 0 \\ 0 & 1 & 0 & \vdots \\ 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 1 \end{pmatrix}$ such that A is $N \times N$ where $N = 2^n \, for \, some \, n \in \mathbb{N}$ and $b' = \begin{pmatrix} b \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ such that b is $N \times 1$. Then, we can use the HHL algorithm to solve the new system $A'x' = b'$ because A' is Hermitian and with appropriate dimension. The reason that A' is Hermitian is that $(A')^\dagger =$

$$\begin{pmatrix} A & 0 & \cdots & 0 \\ 0 & 1 & 0 & \vdots \\ 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 1 \end{pmatrix}^\dagger = \begin{pmatrix} A^\dagger & 0 & \cdots & 0 \\ 0 & 1 & 0 & \vdots \\ 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 1 \end{pmatrix} = \begin{pmatrix} A & 0 & \cdots & 0 \\ 0 & 1 & 0 & \vdots \\ 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 1 \end{pmatrix} = A'$$

since A is Hermitian. In addition, we can always find the least N such that $N > M$ and $N = 2^n \, for \, some \, n \in \mathbb{N}$. The least possible N is always less than $2M$. The proof is the following:

Suppose $M \in \mathbb{N}$ such that $M \neq 2^n \, for \, all \, n \in \mathbb{N}$. Let $d \in \mathbb{N}$ such that $2^d < M < 2^{d+1}$. Then, $\exists c \in \mathbb{N}$ such that $c > 0 \, and \, M = 2^d + c$. Then, $2M = 2(2^d + c) = 2^{d+1} + 2c$. Let $N = 2^{d+1}$. Then, N satisfies $N > M$ and $N = 2^n \, for \, some \, n \in \mathbb{N}$. Furthermore, $N = 2^{d+1} < 2^{d+1} + 2c = 2M$.

This implies that we can always find A' with a size less than $2M \times 2M$.

After we obtain the result of the system $A'x' = b'$, the solution to the system $Ax = b$ is just the first M entries of x'.

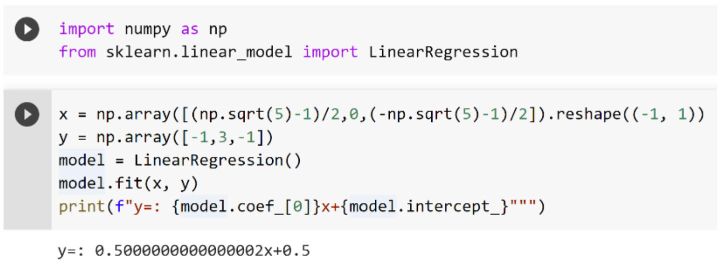
Now, we will analyze how many additional qubits we need to solve the larger system. In the worst case that A is a $M \times M$ non-Hermitian matrix and $M \neq 2^n \, for \, all \, n \in \mathbb{N}$, we need to change A to $A' = \begin{pmatrix} 0 & A \\ A^\dagger & 0 \end{pmatrix}$ that has a size of $2M \times 2M$. Then, we need to expand A' to $A'' = \begin{pmatrix} A & 0 & \cdots & 0 \\ 0 & 1 & 0 & \vdots \\ 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 1 \end{pmatrix}$ such that A'' is a $N \times N$ Hermitian matrix where $N = 2^n \, for \, some \, n \in \mathbb{N}$. We know that the least possible N is always less than $2(2M) = 4M$. Hence, A'' has a size less than $4M \times 4M$. The number of qubits needed to solve this new system is $\lfloor \log_2 4M \rfloor = 2 + \lfloor \log_2 M \rfloor = \lceil \log_2 M \rceil + 1 < \log_2 M + 2$. The number of qubits needed for solving the original system is $\log_2 M$. This means we need at most two additional qubits to solve the larger system, and it's independent of the size of the original system.

Now, we will discuss the case when A is a rectangular matrix ($M \times N \, and \, M \neq N$). In this case, the system $Ax = b$ can have zero or infinitely many solutions depending on the value of b. So, instead of solving for an exact solution to the system $Ax = b$, we want to find a vector $x \in C^N$ such that $||b - Ax||$ is minimal for all $x \in C^N$. This is the least square solution to the system $Ax = b$. In the case of $Ax = b$ having infinitely many solutions, the least square solution is one of the solutions to the system. If the system has zero solution, then the least square solution provides the best approximation to the system. We can find the least square solution to the system $Ax = b$ by solving $A^\dagger Ax = A^\dagger b$. Since A is a $M \times N$ matrix, then $A^\dagger A$ is a $N \times N$ Hermitian matrix because $(A^\dagger A)^\dagger = A^\dagger (A^\dagger)^\dagger = A^\dagger A$. Hence, we can use the HHL algorithm to solve a rectangular system by solving $A^\dagger Ax = A^\dagger b$. Since

$A^\dagger A$ is guaranteed to be Hermitian, we only need to ensure $A^\dagger A$ has the appropriate size. If $A^\dagger A$ doesn't have the appropriate size, we can change the system to $\begin{pmatrix} A^\dagger A & 0 & \dots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix} x = \begin{pmatrix} A^\dagger b \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ with the appropriate size that can be solved by the HHL algorithm. The size of the new system is less than $2N \times 2N$, so the number of qubits needed for solving this new system is $\lfloor \log_2 2N \rfloor = 1 + \lfloor \log_2 N \rfloor = \lceil \log_2 N \rceil$, which means we only need one additional qubit to solve the larger system. Moreover, the number of qubits only depends on the number of columns of matrix A, which means the number of qubits in the circuit only depends on the number of variables in the system, and it's independent of the number of equations in the system.

4. Using HHL to Solve the Linear Regression Problems

HHL is an important algorithm for solving linear regression problems in machine learning. In fact, it is the foundation of quantum machine learning. In this part, I will use an example to demonstrate how to use the HHL algorithm to solve a linear regression problem. Consider the following problem: we are given three data points $\left(\frac{\sqrt{5}-1}{2}, -1\right), (0,3), \left(\frac{-\sqrt{5}-1}{2}, -1\right)$ and we want to find a line that fits these data points the most. It's clear that there doesn't exist a line that passes through these three points exactly, so we want to find a line $y = c_1 x + c_0$ such that $\left\| y\left(\frac{\sqrt{5}-1}{2}\right) - (-1) \right\| + \| y(0) - 2 \| + \left\| y\left(\frac{-\sqrt{5}-1}{2}\right) - (-1) \right\|$ has the smallest value. Classically, we can use linear regression to solve this problem. Fig.2 shows the result of using linear regression.



```
import numpy as np
from sklearn.linear_model import LinearRegression
```

```
x = np.array([(np.sqrt(5)-1)/2,0,(-np.sqrt(5)-1)/2]).reshape((-1, 1))
y = np.array([-1,3,-1])
model = LinearRegression()
model.fit(x, y)
print(f"y=: {model.coef_[0]}x+{model.intercept_}""")
```

```
y=: 0.5000000000000002x+0.5
```

Fig. 2 Result from Linear Regression

As the graph shows, the best-fit line is $y = 0.5x + 0.5$. Now, we will use the HHL algorithm to solve this problem.

Given three data points, we have

$$\begin{cases} -1 = c_0 + \dfrac{\sqrt{5}-1}{2}c_1 \\ 3 = c_0 \\ -1 = c_0 + \dfrac{-\sqrt{5}-1}{2}c_1 \end{cases} \Rightarrow \begin{pmatrix} 1 & \dfrac{\sqrt{5}-1}{2} \\ 1 & 0 \\ 1 & \dfrac{-\sqrt{5}-1}{2} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} -1 \\ 3 \\ -1 \end{pmatrix}$$

To solve for $\begin{pmatrix} 1 & \frac{\sqrt{5}-1}{2} \\ 1 & 0 \\ 1 & \frac{-\sqrt{5}-1}{2} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} -1 \\ 3 \\ -1 \end{pmatrix}$, we need to solve

$$\begin{pmatrix} 1 & \dfrac{\sqrt{5}-1}{2} \\ 1 & 0 \\ 1 & \dfrac{-\sqrt{5}-1}{2} \end{pmatrix}^{\dagger} \begin{pmatrix} 1 & \dfrac{\sqrt{5}-1}{2} \\ 1 & 0 \\ 1 & \dfrac{-\sqrt{5}-1}{2} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 1 & \dfrac{\sqrt{5}-1}{2} \\ 1 & 0 \\ 1 & \dfrac{-\sqrt{5}-1}{2} \end{pmatrix}^{\dagger} \begin{pmatrix} -1 \\ 3 \\ -1 \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} 1 & 1 & 1 \\ \dfrac{\sqrt{5}-1}{2} & 0 & \dfrac{-\sqrt{5}-1}{2} \end{pmatrix} \begin{pmatrix} 1 & \dfrac{\sqrt{5}-1}{2} \\ 1 & 0 \\ 1 & \dfrac{-\sqrt{5}-1}{2} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ \dfrac{\sqrt{5}-1}{2} & 0 & \dfrac{-\sqrt{5}-1}{2} \end{pmatrix} \begin{pmatrix} -1 \\ 3 \\ -1 \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Now, we will use the HHL algorithm to solve the system $\begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

Firstly, we will normalize vector $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ to $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$, which can be encoded by using an H gate.

Then, we need to determine the unitary operation. Matrix $\begin{pmatrix} 3 & -1 \\ -1 & 3 \end{pmatrix}$ has eigenvalue $\lambda_0 = 2, \lambda_1 = 4$ with corresponding eigenvectors $u_1 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}, u_2 = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$. Hence, the unitary operation $U = \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}e^{i2t} & 0\\ 0 & e^{i4t}\end{pmatrix}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$. For convenience, we set $t = \frac{\pi}{2}$ and then, $U = \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}-1 & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} = \begin{pmatrix}0 & -1\\ -1 & 0\end{pmatrix}$. This unitary operation can be decomposed into Z, X, Z gates since $ZXZ = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix} = \begin{pmatrix}0 & -1\\ -1 & 0\end{pmatrix} = U$. The inverse of unitary operation is itself since $UU = \begin{pmatrix}0 & -1\\ -1 & 0\end{pmatrix}\begin{pmatrix}0 & -1\\ -1 & 0\end{pmatrix} = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$.

Finally, we need to determine the parameters for RY gates. I use two qubits in c-register to encode the eigenvalues since $\lambda_1 : \lambda_2 = 1:2$. Since we want to change the state of ancilla qubit to $\sqrt{1 - \left(\frac{C}{\tilde{\lambda}_j}\right)^2}\,|0\rangle_a + \frac{C}{\tilde{\lambda}_j}|1\rangle_a$. Since $\tilde{\lambda}_j = \frac{N\lambda_j t}{2\pi} = \lambda_j \ \left(N = 2^2 = 4, = \frac{\pi}{4}\right)$, then the largest possible value of C is 2, which is the minimum value of $\tilde{\lambda}_j$. Since $RY(\theta) = \begin{pmatrix}\cos\frac{\theta}{2} & -\sin\frac{\theta}{2}\\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2}\end{pmatrix}$ and ancilla qubit is set to $|0\rangle$ initially, then after $RY(\theta)$, the state of ancilla qubit is $\cos\frac{\theta}{2}|0\rangle + \sin\frac{\theta}{2}|1\rangle$, then $\sin\frac{\theta_j}{2} = \frac{2}{\tilde{\lambda}_j}$. Hence, $\theta_j = 2\arcsin\frac{2}{\tilde{\lambda}_j}$. Then, $\theta_1 = 2\arcsin\frac{2}{2} = \pi, \theta_2 = 2\arcsin\frac{2}{4} = \frac{\pi}{3}$.

Now, we have the parameters for all gates in the circuit.

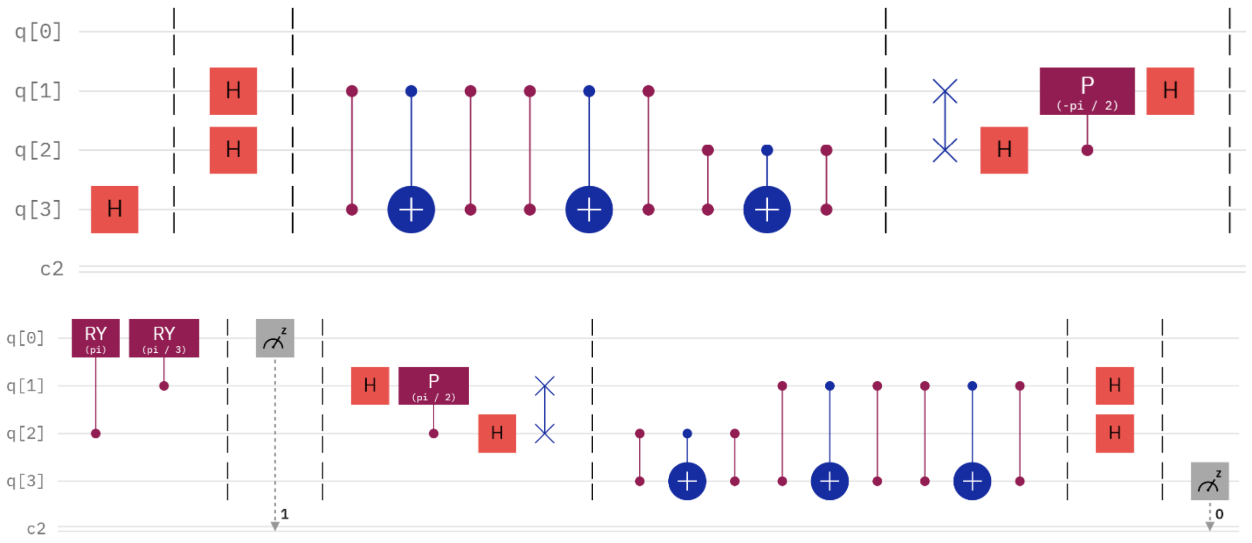


Fig. 3 Full HHL circuit for solving the system

However, this circuit can be simplified because $U^2 = I$.

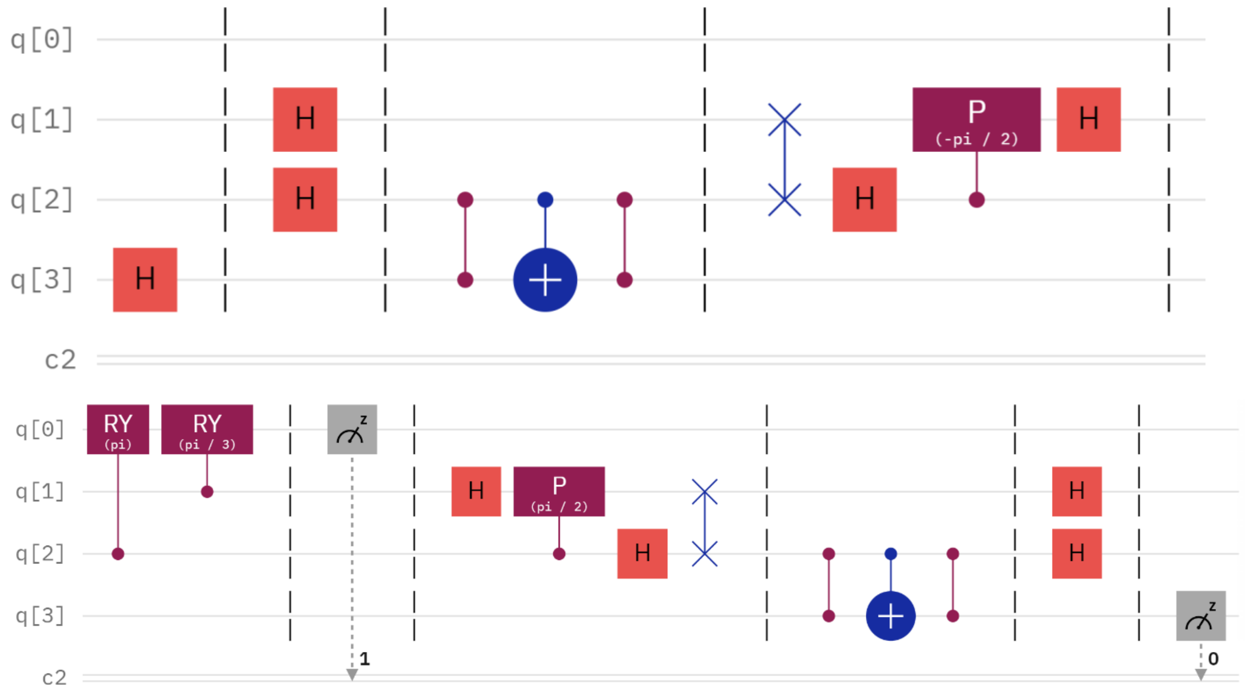


Fig. 4 Simplified HHL circuit

I ran the simplified HHL circuit on the simulator and quantum computer (ibm_nairobi) with 10000 shots. The following two graphs show the results from the simulation and ibm_nairobi.

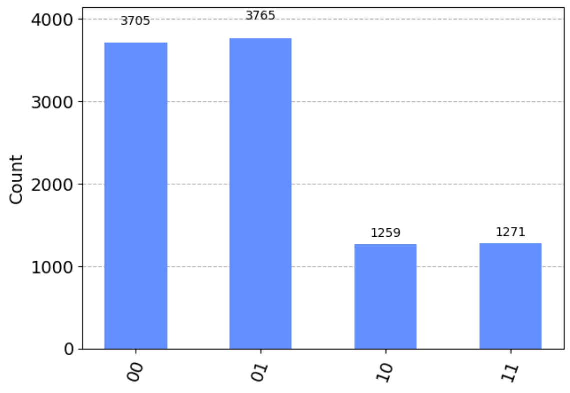


Fig. 5 Simulation result

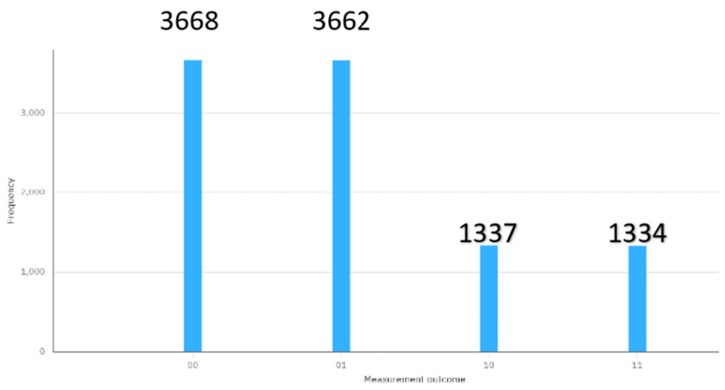


Fig. 6 ibm_nairobi result

The above graphs follow textbook convention, which means the first qubit is the ancilla qubit, and the second qubit encodes x. We need to discard the case when the first qubit is

0, as we discussed earlier. For the second qubit, $|0\rangle$ represents $\begin{pmatrix}1\\0\end{pmatrix}$, and $|1\rangle$ represents $\begin{pmatrix}0\\1\end{pmatrix}$. Hence, the result from the simulation and ibm_nairobi are $x_{sim} = \begin{pmatrix}1259\\1271\end{pmatrix}, x_{quantum} = \begin{pmatrix}1337\\1334\end{pmatrix}$. To get the exact solution, we need to substitute these two vectors into one of the equations in the system $\begin{pmatrix}3 & -1\\-1 & 3\end{pmatrix}\begin{pmatrix}c_0\\c_1\end{pmatrix} = \begin{pmatrix}1\\1\end{pmatrix}$ where $\begin{pmatrix}c_0\\c_1\end{pmatrix} = cx, c\ is\ a\ constant.$ Substituting into the first equation, we have $3 * 1259c_{sim} - 1271c_{sim} = 1$ and $3 * 1337c_{quantum} - 1334c_{quantum} = 1$. Solving for c, we have $c_{sim} = \frac{1}{2506}, c_{quantum} = \frac{1}{2677}$. Hence, we have $\begin{pmatrix}c_0\\c_1\end{pmatrix} = \begin{pmatrix}\frac{1259}{2506}\\\frac{1271}{2506}\end{pmatrix} \approx \begin{pmatrix}0.502\\0.507\end{pmatrix}$ for simulation and $\begin{pmatrix}c_0\\c_1\end{pmatrix} = \begin{pmatrix}\frac{1337}{2667}\\\frac{1334}{2667}\end{pmatrix} \approx \begin{pmatrix}0.501\\0.500\end{pmatrix}$ for ibm_nairobi. Then, the best-fitting line is

$$y_{sim} = 0.507x + 0.502$$

$$y_{quantum} = 0.5x + 0.501$$

These two answers are very close to the exact solution $y = 0.5x + 0.5$.

5. Conclusion

In this project, I explained the original HHL algorithm and discussed techniques for using HHL to solve arbitrary linear systems. Though these techniques require us to solve a larger system, I have shown that for a system with N variables, we need at most $\lceil \log_2 N \rceil + 1$ qubits to solve the larger system, which means at most 2 additional qubits are needed since the original system requires $\log_2 N$ qubits and $\lceil \log_2 N \rceil + 1 - \log_2 N < 2$. In addition, I showed how to apply HHL to solve problems in machine learning. This algorithm has an important application in areas that requires solving linear systems of equations.

6. Reference

[1]. *8 computational complexity - NDSU.EDU*. (n.d.). Retrieved April 27, 2023, from https://www.ndsu.edu/pubweb/~novozhil/Teaching/488%20Data/08.pdf

[2]. J. R. Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Technical Report CMU-CS-94-125, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, March 1994.

[3]. A. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," Phys. Rev. Lett. 103, 150502 (2009).

[4]. Morrell Jr, H. J., Zaman, A., & Wong, H. Y. (2023, March 25). *Step-by-step HHL algorithm walkthrough to enhance the understanding of critical quantum computing concepts*. arXiv.org. Retrieved April 28, 2023, from https://arxiv.org/abs/2108.09004

In [4]:

```
import numpy as np
from sklearn.linear_model import LinearRegression
```

In [ ]:

```
x = np.array([(np.sqrt(5)-1)/2,0,(-np.sqrt(5)-1)/2]).reshape((-1, 1))
y = np.array([-1,3,-1])
model = LinearRegression()
model.fit(x, y)
print(f"y=: {model.coef_[0]}x+{model.intercept_}""")
```

```
y=: 0.5000000000000002x+0.5
```

In [2]:

```
from qiskit import QuantumCircuit, assemble, Aer, transpile
from qiskit.visualization import plot_histogram, plot_bloch_vector
```

In [21]:

```
state_prep=QuantumCircuit(1,0)
state_prep.h(0)
print(state_prep)
```

```
   ┌───┐
q: ┤ H ├
   └───┘
```

In [22]:

```
qft = QuantumCircuit(2,0)
qft.h(0)
qft.cs(1,0)
qft.h(1)
qft.swap(0,1)
print(qft)
```

```
     ┌───┐┌───┐           
q_0: ┤ H ├┤ S ├─────────X─
     └───┘└─┬─┘┌───┐    │ 
q_1: ───────■──┤ H ├────X─
               └───┘      
```

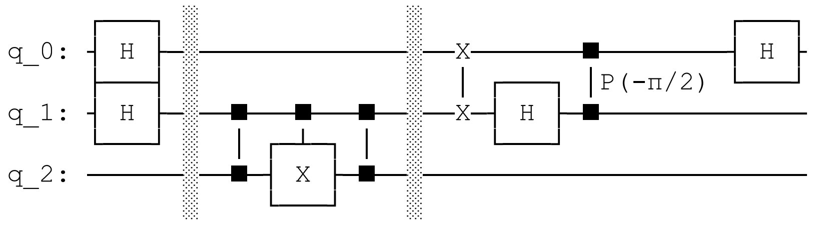
In [23]:

```
iqft = QuantumCircuit(2,0)
iqft.swap(0,1)
iqft.h(1)
iqft.cp(-np.pi/2,1,0)
iqft.h(0)
print(iqft)
```

```
                          ┌───┐
q_0: ─X─────────■─────────┤ H ├
      │ ┌───┐   │P(-π/2)  └───┘
q_1: ─X─┤ H ├───■──────────────
        └───┘                  
```

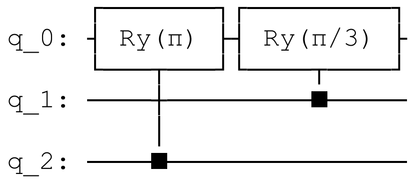
In [33]:

```
qpe=QuantumCircuit(3,0)
qpe.h([0,1])
qpe.barrier()
qpe.cz(1,2)
qpe.cx(1,2)
qpe.cz(1,2)
qpe.barrier()
qpe=qpe.compose(iqft,[0,1])
print(qpe)
```



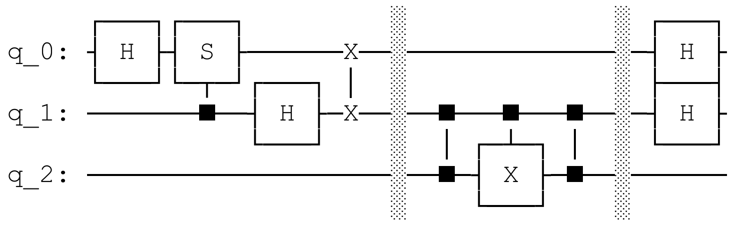
In [25]:

```
rys=QuantumCircuit(3,0)
rys.cry(np.pi,2,0)
rys.cry(np.pi/3,1,0)
print(rys)
```



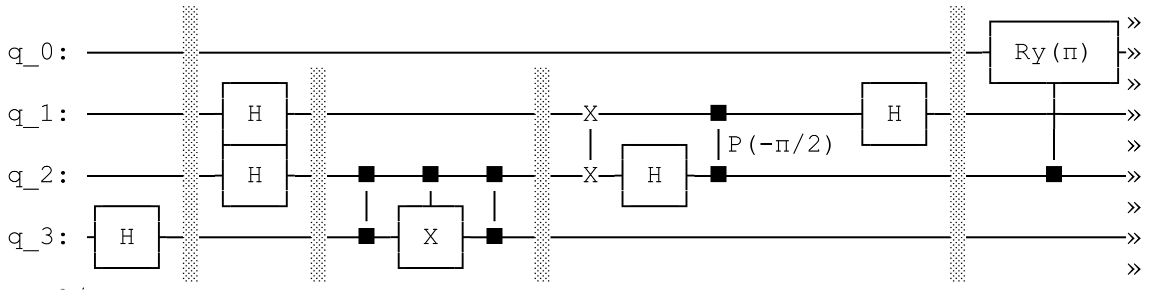
In [34]:

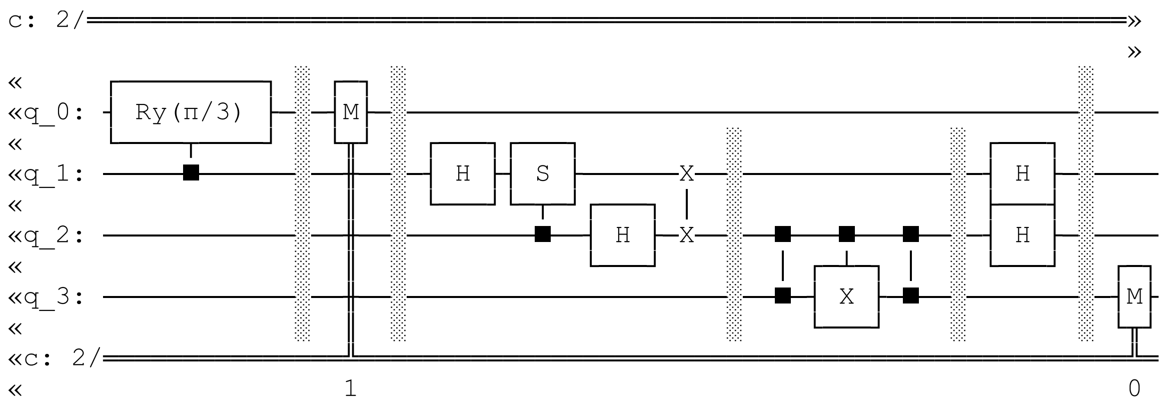
```
iqpe=QuantumCircuit(3,0)
iqpe=iqpe.compose(qft,[0,1])
iqpe.barrier()
iqpe.cz(1,2)
iqpe.cx(1,2)
iqpe.cz(1,2)
iqpe.barrier()
iqpe.h([0,1])
print(iqpe)
```



In [36]:

```
hhl=QuantumCircuit(4,2)
hhl=hhl.compose(state_prep,[3])
hhl.barrier()
hhl=hhl.compose(qpe,[1,2,3])
hhl.barrier()
hhl=hhl.compose(rys,[0,1,2])
hhl.barrier()
hhl.measure(0,1)
hhl.barrier()
hhl=hhl.compose(iqpe,[1,2,3])
hhl.barrier()
hhl.measure(3,0)
print(hhl)
```

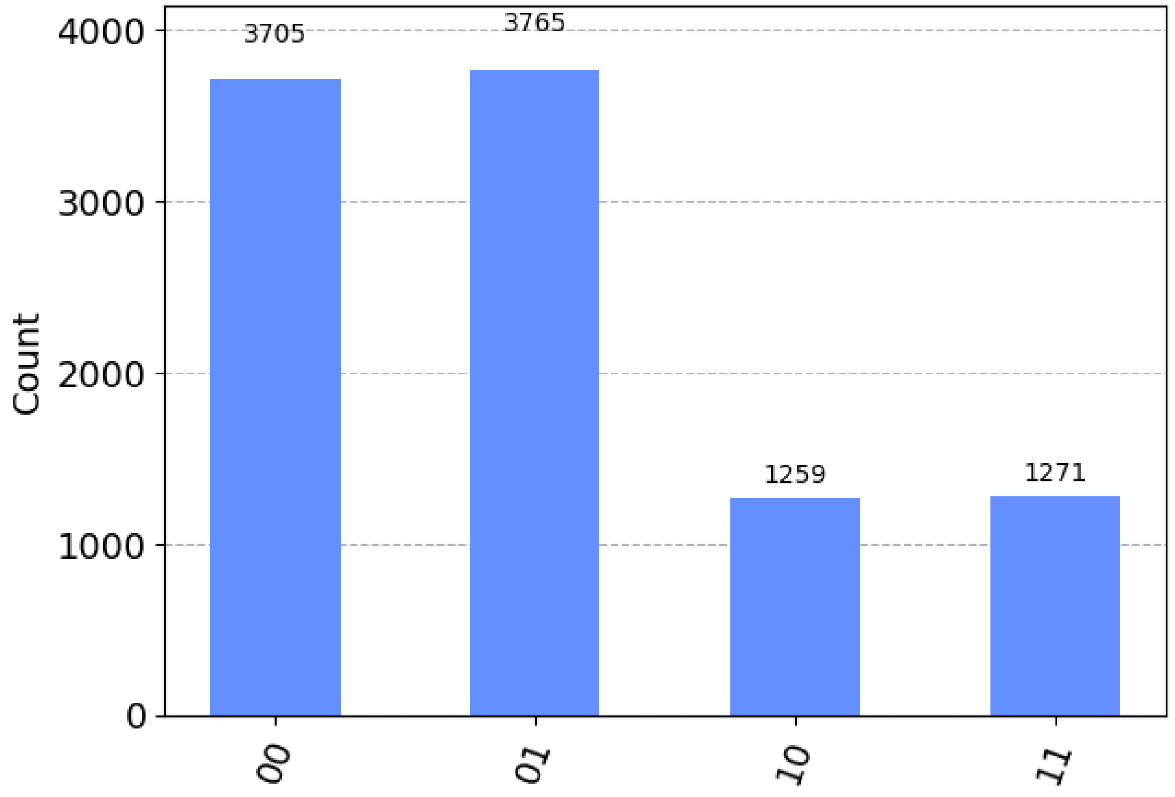




In [43]:

```python
sim = Aer.get_backend('aer_simulator')
qobj = transpile(hhl, sim)
result = sim.run(qobj,shots=10000).result()
counts = result.get_counts()
plot_histogram(counts)
```

Out[43]:


Count