Review of a Quantum Algorithm for Linear Systems of Equations

A Deep Dive on the HHL Algorithm

Michael DeLeo

Abstract

This paper explores the HHL algorithm, a quantum approach to efficiently solving linear systems of equations. Investigating its foundational aspects and advantages over classical methods, the paper covers key phases of the algorithm, such as state preparation, quantum phase estimation, and conditional rotation of the ancilla. The paper also describes and provides links to an implementation of the algorithm. The algorithm's strict requirements, limitations, and potential applications are outlined.

Background

The research I'm proposing is based on a paper I found when I was searching through the Quantum Zoo site [1]. Originally, I had planned to do a paper on a quantum machine learning algorithm, and I found a paper that was describing how there is more to gaining a speedup in machine learning than just throwing the problem at a quantum computer. The paper was called *Quantum Machine Learning Algorithms: Read the Fine Print* which explained that most quantum algorithms being used for machine learning are in most part either derived from or wholly using the HHL algorithm which was developed by the researcher's colleagues [2]. I decided that what was better than learning from a derivative was researching the more substantive and foundational source algorithm. The Harrow-Hassim-Lloyd (HHL) algorithm is named after the authors of the *Quantum algorithm for linear systems of equations* paper and is described as an algorithm that is more efficient than classical algorithms at solving a linear system of equations [3]. For a given linear system

Equation 1

$$A\vec{x} = \vec{b}$$

They consider the case where one doesn't need to know the solution $\vec{x}$ itself, but rather an approximation of the expectation value of some operator associated with $\vec{x}$, for example $\vec{x}^{\dagger}M\vec{x}$ for some matrix M. In the case of sparse matrices where the A matrix is N-by-N, the matrix has a condition number k which determines the time complexity [3]. In the case of classical algorithms, they're able to achieve a time complexity of $O(N\sqrt{k})$. The complexity achieved in the paper is a polynomial time of $O(\log(N, k))$ which is an exponential speedup over the classical approach [3].

Overview of the HHL Algorithm

The general flow of the algorithm is shown in Figure 1. The algorithm utilizes concepts/circuits such as amplitude encoding, quantum phase estimation and eigenvalue inversion to estimate a function of the solution.

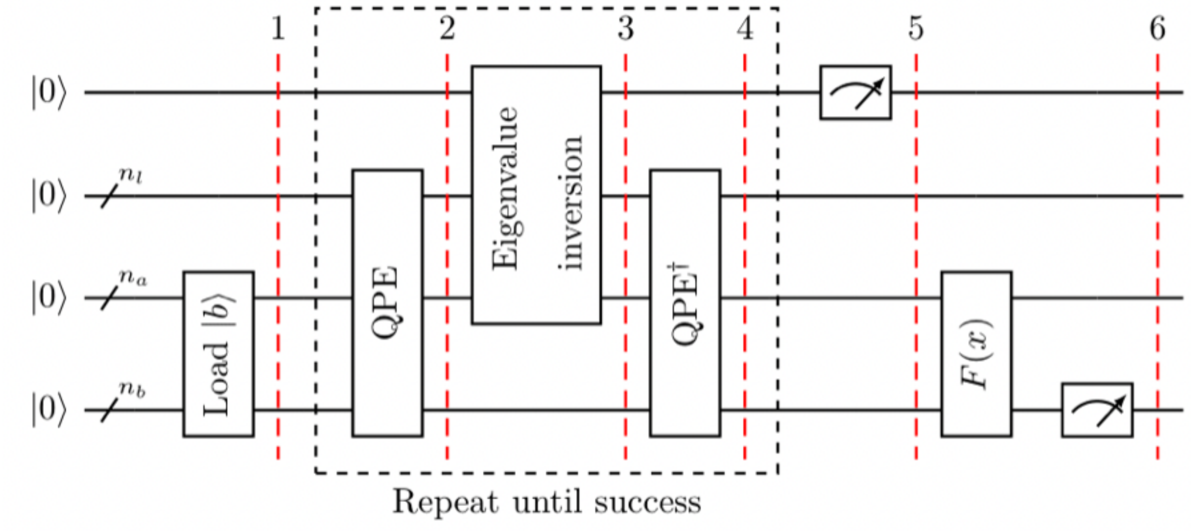


Figure 1 - Qiskit HHL [4]

The algorithm has strict requirements to function correctly. The input matrix A must be Hermitian and be of the space $A \epsilon \mathbb{C}^{N \times N}$. Similarly, the vector $\vec{b} \epsilon \mathbb{C}^{N}$ and the resulting vector will be $\vec{x} \epsilon \mathbb{C}^{N}$.

1. State Preparation

The standard linear system we are trying to solve is

Equation 2

$$Ax = b \rightarrow x = A^{-1}b$$

It has the following representation in complex vector notation

Equation 3

$$A|x\rangle = |b\rangle$$

Where the $|b\rangle$ vector is represented in amplitude encoding. You'll need a fairly involved circuit that will increase the circuit depth. The A operator is represented with a Hamiltonian encoding, and it is required to be Hermitian. If it is not a Hermitian matrix then it needs to be transformed into one. (Eventually into a Unitary?)

In this first phase, we are loading the $|b\rangle$ vector into the circuit such that

$$|0\rangle_{n_b} \mapsto |b\rangle_{n_b}$$

The number of qubits is determined by the length of the vector, and so we assume $|b\rangle \epsilon \mathbb{C}^N$. This implies the length of the vector is N, and we can determine the integer number of qubits n_b with

Equation 4

$$n_b = \log_2 N$$

To load the $|b\rangle$ vector into our circuit, we need to use an amplitude encoding algorithm [5]

Equation 5

$$S_b|0\rangle = |b\rangle = \frac{1}{||b||}\sum_{i=1}^{2^{(n_b)}} b_i|i\rangle$$

Some Qiskit implementations [6] use the isometry [7] circuit for amplitude encoding. However, these tend to limit the dimensions of the input data by exact dimensions of 2^n.

2. Quantum Phase Estimation

We intend to calculate the eigenvalues of A, and to do this need to propagate it through a phase estimation circuit.

Equation 6

$$|\psi_0\rangle \coloneqq \sqrt{\frac{2}{T}}\sum_{\tau=0}^{T-1} \sin\left(\frac{\pi\left(\tau+\frac{1}{2}\right)}{T}\right)|\tau\rangle$$

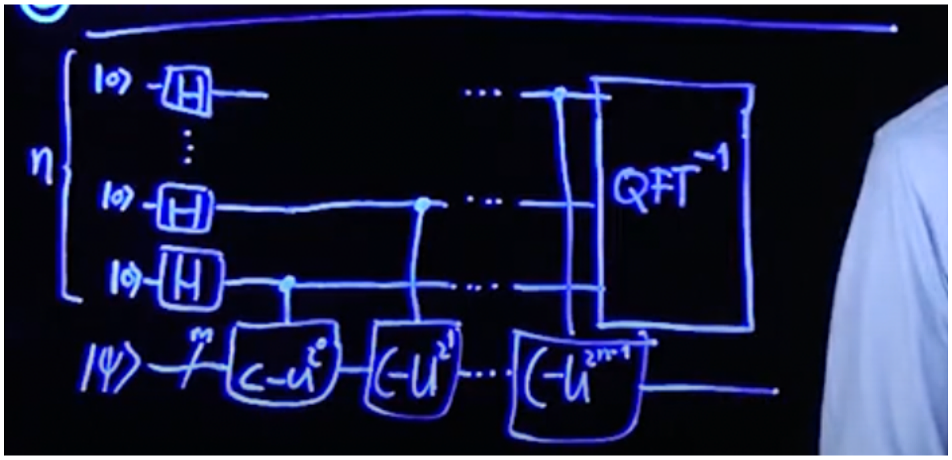


Figure 2 – Phase Estimation Circuit [8]

3. Conditional Rotation of the Ancilla

We create another ancilla register on the quantum circuit.

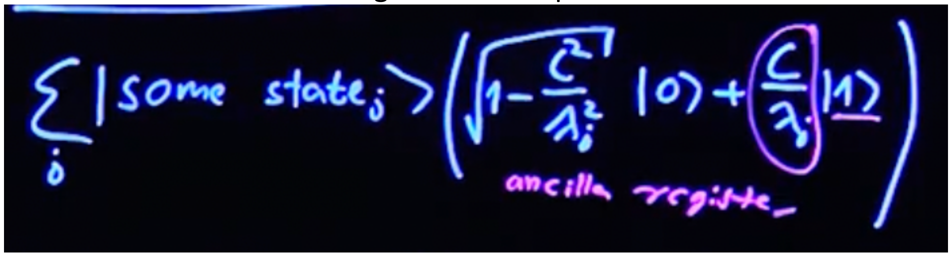

$\sum_j |\text{some state}_j\rangle \left(\sqrt{1-\frac{C^2}{\lambda_j^2}}\,|0\rangle + \frac{C}{\lambda_j}|1\rangle\right)$
ancilla register

Figure 3 – Conditional Rotation as described in [8]

By creating this conditional rotation of the ancilla we will encode the value of a constant times the inverse of the eigenvalue (in the probability amplitude of the $|1\rangle$ state of the ancilla.

To finish the inversion we measure the last qubit. Conditioned on seeing 1, we have the state

Equation 7

$$\sqrt{\frac{1}{\sum_{j=1}^{N} \frac{C^2|\beta_j|^2}{|\lambda_j|^2}}} \sum_{j=1}^{N} \beta_j \frac{C}{\lambda_j} |u_j\rangle$$

The corresponding state of our target $|x\rangle$ is

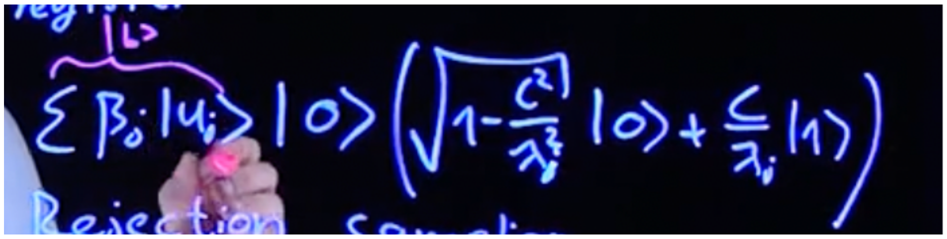
Equation 8

$$|x\rangle = \sum\nolimits_{j=1}^{n} \beta_j \lambda_j^{-1} |u_j\rangle$$

4. Uncompute the Eigenvalue Register

Before we can use this eigenvalue, we need to reverse the quantum phase estimation. We have to do this, because if we act on the registers after the PE then we will find that they are entangled [8].

$\langle \beta_0 | \mu_i \rangle$ is essentially $|b\rangle$b_ket itself in the eigen basis. Additionally, we can see that the ancilla's conditional rotation is untouched.



Equation 9

$$\langle \beta_0 | u_0 \rangle \, |0\rangle \left(\sqrt{1 - \frac{C^2}{\lambda_i^2}} |0\rangle + \frac{C}{\lambda_i} |1\rangle \right)$$

5. Rejection Sampling

We begin with rejection sampling, and we measure the ancilla. If we get, $|0\rangle$ then we restart the calculation and throw it out. If we get $|1\rangle$, then whatever we get is probability proportional to the eigenvalue ($\frac{c}{\lambda}$). This is what we will use to estimate an observable.

HHL Implementation

The implementation of the algorithm can be found at: https://github.com/deleomike/HHL

The setup procedure is specified, as well as any requirements. Seen below is the flow diagram for the algorithm.

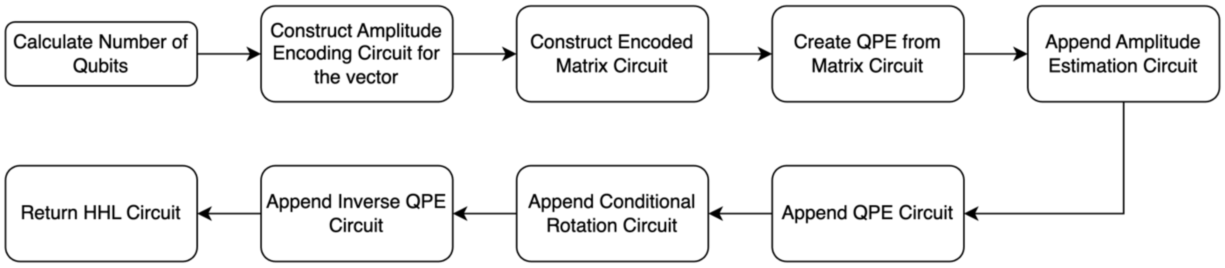

Calculate Number of Qubits
Construct Amplitude Encoding Circuit for the vector
Construct Encoded Matrix Circuit
Create QPE from Matrix Circuit
Append Amplitude Estimation Circuit
Append QPE Circuit
Append Conditional Rotation Circuit
Append Inverse QPE Circuit
Return HHL Circuit

Figure 4 - HHL Flow

For my implementation, I referenced several sets of source material. I initially found a notebook posted on qiskit's docs referencing how to implement the algorithm, although it did not seem extensibility to any other kind of example other than the contemporary example seen in the input data section (Equation 10) [9]. The next major source of material was the official qiskit textbook implementation for the HHL algorithm [4]. The provided material referenced a *linear solvers* library that Qiskit refers to as an official implementation of the HHL algorithm and other quantum solvers [6].

I implemented the functionality of the HHL in a functional design pattern which allowed me to deconstruct the flow of the algorithm from an atomic point of view (Figure 4 - HHL Flow). Outlined below are the functions and their specific purpose (Table 1 - HHL Functions).

Table 1 - HHL Functions

Function	Purpose
Random_diag	Generate random diagonal matrix with given properties
Random_hermitian	Generate random hermitian (sparse) matrix with given properties
Resize_matrix	Resizes the matrix to match the appropriate dimensionality
Calc_qubits	Given a vector, calculates the number of qubits needed to encode it into a circuit.
AE_circuit	Amplitude Encoding Circuit.
Create_matrix_circuit	Encodes the matrix
Calculate_nl	Update the number of qubits required to represent the eigenvalues
Get_delta	Calculates the scaling factor to represent exactly λmin on nl binary digits
Get_kappa	Calculates/retrieves the kappa value for the circuit.
Set_eigenbounds	Calculates and sets the eigenbounds for the encoded matrix circuit.
Calculate_norm	Calculates the value of the euclidean norm of the solution.
Construct_circuit	Constructs the HHL circuit
HHL	Creates the HHL circuit (construct_circuit), calculates the norm (calculate_norm), and returns a linear solver result.
Compute_x	Computes the x vector given the stored linear solver result.
Hhl_solver	Creates the HHL circuit (construct_circuit), calculates the norm (calculate_norm), and returns a linear solver result.

I was able to match the output of the HHL algorithm for most sets of input data, however past the two-by-two dimension I experienced some issues with receiving the correct dimensions for X.

Something I was unable to figure out was that both mine and the official implementation could not reliably calculate the system of equations outside of the test example below. In other words, the classical algorithm was more accurate more often than the quantum algorithm. There were some cases where the quantum algorithm was sufficiently close, and this is shown in the notebook. I'm not sure why this is the case and would need to get a better understanding of the underlying implementations and algorithms used with Qiskit. I suspect the isometry used to encode the data may be of issue and that a different algorithm should be used instead.

Input Data

The following data is rather standard for testing out the HHL algorithm and is seen in several examples around the internet [4].

Equation 10

$$A = \begin{pmatrix} 1 & -\frac{1}{3} \\ -\frac{1}{3} & 1 \end{pmatrix}, x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, b = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

This data is passed to the hhl_solver function which returns the corresponding x value. Alternatively, the data can be given to HHL which would create the quantum circuit without evaluating.

Output Data

When the data is passed to the HHL or construct_circuit functions the return value will contain a quantum circuit. The corresponding quantum circuit for the example input data (Equation 10) is seen in Figure 5.

The HHL circuit that has been constructed has four sections Amplitude Encoding, QPE, Conditional Rotation and the inverse QPE. The amplitude encoding encodes the input data into the input qubits by encoding the values into the probabilities [3]. The QPE phase computes a Phase estimation of the input qubit which becomes an entangled state with the other qubits [3]. The conditional rotation is the key component of the process which performs an inversion of A and b and reveals the eigen information for x. The inverse QPE is to undo the entanglement from prior. To retrieve the eigen information for the unknown vector, the last set of qubits in the circuit are evaluated. They can be identified in that they were not inputs for the QPE process. In prior discussion, this was referred to as the ancillary qubit(s). This qubit and the other input qubits scale in proportion to the input dimensions. At the end of this circuit we are really just trying to solve for is $|x\rangle$ in Equation 8.

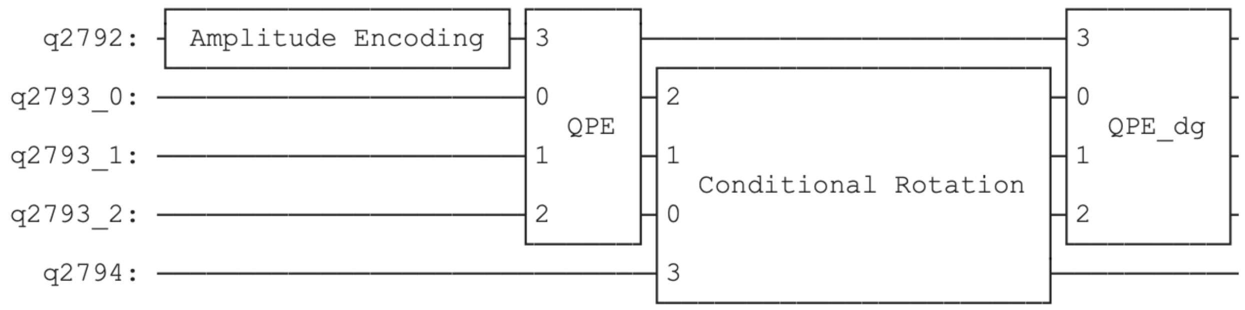


Figure 5 - HHL Circuit

For an inside look to how the actual eigen information is being extracted, refer to the compute_x function.

Limitations

As has been stated throughout the paper, the limitations for the algorithm are not overwhelming but they do severely limit the applicability. For instance, the input matrix A **must** be a hermitian matrix and be of the order of 2^n with respect to its dimensionality (i.e. $2 \times 2, 4 \times 4, 16 \times 16$).

For the applications of artificial intelligence, the dimensionality limits the shape and size of deep learning models. Not to mention as well that its highly unlikely that a given set of weights in a network will be Hermitian. For these reasons it is difficult to understand where the algorithms applicability is within Artificial Intelligence, as it would not be suited for the calculations in the backpropagations of a deep learning network.

For other scientific applications it’s difficult to imagine a use case for this specific kind of algorithm unless the system being solved was so large that it was unmanageable by an ordinary classical algorithm. This goes back to the idea of what really makes a quantum algorithm better than a classical algorithm. In my belief you can meet the technical requirement of being superior, but if it isn’t practical for the intended use case then it doesn’t matter.

In addition to these issues, I found that for random, dimensionally correct, Hermitian matrices I was unable to match the numpy “ground truth” answer for a given matrix. This held for both my implementation and the official Qiskit implementation [6].

Conclusion

In conclusion, the HHL algorithm Is a promising algorithm for solving linear systems of equations in a new way. This paper serves as a deep dive into the inner workings of the algorithm in how it really works, what the limitations are, and what its applicability is. Overall, there is potential for its use in larger scale algorithms, however, it's not going to be as applicable for cases where the data has not already been formatted in one off use cases. Instead, it might have broader applicability in algorithms which can guarantee hermitian inputs and are also confined by the same bounds in terms of dimensionality requirements.

Bibliography

[1] S. Jordan, "Quantum Zoo," [Online]. Available: https://quantumalgorithmzoo.org. [Accessed November 2023].

[2] S. Aaronson, "Quantum Machine Learning Algorithms: Read the Fine Print," *Nature Physics,* no. 11, 2015.

[3] A. Harrow, A. Hassidim and S. Lloyd, "Quantum Algorithm for Linear Systems of Equations," *Physical Review Letters,* vol. 103, no. 15, 2009.

[4] Qiskit, "Solving linear systems of equations using HHL and its Qiskit implementation," IBM, [Online]. Available: https://learn.qiskit.org/course/ch-applications/solving-linear-systems-of-equations-using-hhl-and-its-qiskit-implementation#implementation. [Accessed November 2023].

[5] R. LaRose and B. Coyle, "Robust data encodings for quantum classifiers," *Physical Review A,* September 2020.

[6] C. A. Vazquez, "Quantum Linear Solvers," [Online]. Available: https://github.com/anedumla/quantum_linear_solvers. [Accessed November 2023].

[7] R. Iten, R. Colbeck, I. Kukuljan, J. Home and M. Christandl, "Quantum circuits for isometries," *Physical Review A,* vol. 93, March 2016.

[8] "Quantum Machine Learning - 37 - Overview of the HHL Algorithm," 28 December 2019. [Online]. Available: https://www.youtube.com/watch?v=hQpdPM-6wtU. [Accessed November 2023].

[9] J. Wootton, "Quantum Algorithm for Linear System of Equations," 2018. [Online]. Available: https://github.com/CQCL/qiskit-tutorial/blob/master/community/awards/teach_me_qiskit_2018/quantum_machine_learning/2_HHL/Quantum%20Algorithm%20for%20Linear%20System%20of%20Equations.ipynb. [Accessed November 2023].