Quantum algorithm for linear systems of equations

Harrow, Hassidim, Lloyd

Aviral Janveja
Universität Paderborn

1 Introduction

Here we examine the quantum algorithm for linear systems of equations, also commonly known as the HHL Algorithm, from the author names Harrow, Hassidim and Lloyd. The algorithm is useful in solving large systems of linear equations. Solving systems of linear equations is extremely important in many fields of science and engineering, for instance machine learning.
The aim is to break down the algorithm in a step by step manner, such that any person having a basic background in linear algebra, algorithms and complexity theory can grasp the major ideas of this algorithm.
I would like to thank Prof. Sevag Gharibian and Prof. Johannes Blömer for their valuable time and inputs during this semester.

1.1 Linear Systems of Equations

We should naturally start with a brief primer. A system of linear equations simply refers to two or more equations with common variables. The equations obviously are linear, meaning that the highest power for the variables appearing in these equations is 1.

For example, this is a pair of linear equations in 2 variables :

$$a_1x + b_1y = c_1$$

$$a_2x + b_2y = c_2$$

An equivalent representation of the above pair of equations in the matrix form would look like this:

$$\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

This can be then further written in a shorter format like this:

$$AX = B$$

where $A = \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}$, $X = \begin{bmatrix} x \\ y \end{bmatrix}$ and $B = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$.

Now, the unique solution to the equation $AX = B$ is given by $X = A^{-1}B$, where A is an invertible matrix, when $|A| \neq 0$ (A having a non-zero determinant). So, here we have X as the required solution and B is known from the coefficients of the linear equations.

Therefore, computing the solution to $AX = B$, amounts to computing the expression $A^{-1}B$ or more specifically computing A^{-1}.

1.2 Why do we need a Quantum Algorithm ?

Just how a classical algorithm is a series of instructions that can be performed on a classical computer for solving a given problem. Similarly, a quantum algorithm consists of a series of instructions, which can be performed on a quantum computer.

Quantum computers are devices that can utilize the laws of quantum mechanics such as superposition, entanglement and measurement to perform calculations, in a way that classical computers cannot. For certain problems, quantum algorithms offer significant and sometimes exponential speedups over their classical counterpart. Famous examples being Shor's factoring algorithm and Grover's search algorithm.

In *section 1.1*, $A = \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix}$ is a matrix of order 2. However in many real life situations within science and engineering, it is required that we solve much larger systems of linear equations. Naturally, the order of matrix A is going to be much larger, for instance some big number N.

For a classical computer, even to approximate the solution of N linear equations in N variables, generally requires time that scales at least as N.

This paper presents a quantum algorithm that can estimate features of the solution in time which scales as $logN$; that is, logarithmically in N. Therefore, this algorithm can be as much as exponentially faster than classical algorithms for the same task.[3]

2 The Algorithm

Presented here is an intuitive step by step breakdown of the algorithm following through equation-1 to equation-10, that prioritises getting the essence of the procedure. This section requires the understanding of certain fundamental, prerequisite concepts in quantum computing. The following texts [2], [5], [4] have been used for the same.

2.1 Preliminaries

Starting from where we left of in section-1: Let's say we are given a problem that can be represented by a set of N linear equations in N variables, with N being some very large positive integer. As seen in section-1, These equations can then be represented in matrix form as:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & ... & a_{1N} \\ a_{21} & a_{22} & a_{23} & ... & a_{2N} \\ a_{31} & a_{32} & a_{33} & ... & a_{3N} \\ a_{41} & a_{42} & a_{43} & ... & a_{4N} \\ . & & & & \\ . & & & & \\ . & & & & \\ a_{N1} & a_{N2} & a_{N3} & ... & a_{NN} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ . \\ . \\ . \\ x_N \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ . \\ . \\ . \\ b_N \end{bmatrix}$$

Equivalently, we can write in short:

$$A\vec{x} = \vec{b} \tag{1}$$

Where A and $\vec{b}$ are known from the coefficients of linear equations, whereas $\vec{x}$ is the vector of unknowns that we wish to compute.

Further, If A is invertible, then the solution for equation-1 will be given by:

$$\vec{x} = A^{-1}\vec{b} \tag{2}$$

2.2 Quantum States

In order to utilize a quantum computer for this task, we will first need to encode the above equation-1 in its corresponding quantum mechanical form. Hence, $\vec{x}$ and $\vec{b}$ are represented as respective quantum states $|x\rangle$ and $|b\rangle$.

$|b\rangle$ here represents $\vec{b}$ as a quantum state, written in the *Ket* notation. *Kets* are column vectors with complex entries and they represent quantum states. which can be written as:

$$|b\rangle = b_0|0\rangle + b_1|1\rangle + b_2|2\rangle .... + b_{N-1}|N-1\rangle$$

Or Equivalently as:

$$|b\rangle = \sum_{i=0}^{N-1} b_i|i\rangle$$

Here, $|i\rangle$ from $|0\rangle, |1\rangle, ...|N-1\rangle$ form the orthonormal basis for an N dimensional space. The basis is a vector against which we measure things.

In order to understand this, consider the 2 dimensional euclidean plane with X and Y axis. We define any point in that plane as (x,y). For instance a point (3,4) in this plane means a point that is 3 units away on the X-Axis and 4 units away on the Y-Axis. These units of the axis X & Y against which the point (3,4) or any point (x,y) for that matter is defined, can be considered as the analogy for understanding the orthonormal basis mentioned above.

Orthonormal simply means orthogonal plus normalized. Orthogonal meaning perpendicular to each other (just like X and Y axis on a 2D plane) and normalized meaning length 1 (Just like the unit length on the X and Y axis as per which other points are written).

Further b_i from $b_0, b_1, ..., b_{N-1}$ represent the probability amplitude of basis states $|i\rangle$ in $|b\rangle$. Meaning, the quantum state $|b\rangle$ is in all the basis states at the same time. It is in basis state $|0\rangle$ with probability amplitude b_0, in state $|1\rangle$ with probability amplitude b_1 and so on. This quantum state $|b\rangle$ can also be written as an N-dimensional vector of its probability amplitudes:

$$|b\rangle = \begin{bmatrix} b_0 \\ b_1 \\ . \\ . \\ b_{N-1} \end{bmatrix}$$

Hence, we have the equation-1 transformed in its corresponding quantum mechanical form:

$$A|x\rangle = |b\rangle \tag{3}$$

with the i^{th} component of $\vec{b}$ corresponding to the probability amplitude of the i^{th} basis of $|b\rangle$. Therefore, the goal solution state for the algorithm becomes:

$$|x\rangle = A^{-1}|b\rangle \tag{4}$$

2.3 Assumptions

Before preparing this goal solution state, we now look at certain conditions, some of which are required for simplicity, while some are required for the algorithm to run efficiently as we will discuss below:

- Firstly, we will assume that matrix A is Hermitian ($A = A^{\dagger}$).

  $A^{\dagger}$ is the conjugate transpose of A. It is obtained by first taking a transpose and then taking the complex conjugate of each element of matrix A. When A is Non Hermitian, we can define:

  $$A' = \begin{bmatrix} 0 & A \\ A' & 0 \end{bmatrix} b' = \begin{bmatrix} 0 \\ b \end{bmatrix} x' = \begin{bmatrix} x \\ 0 \end{bmatrix}$$

  Then A' is Hermitian and we can solve for $A'|x'\rangle = |b'\rangle$ instead.

- A is s-sparse. An s-sparse matrix is one which as at most s non zero entries per row. Also, we have access to these non zero entries of A via an efficient oracle (some algorithm, utilized as black box for now).

- A is well-conditioned. This means that A must have a low condition number k, where k is the ratio of the largest and smallest eigenvalues of A.

  Remember that, for matrix A, an eigenvalue is a scalar value satisfying $A|a\rangle = \lambda|a\rangle$. Where λ is the eigenvalue and $|a\rangle$ is the corresponding eigenvector.

  when the condition number k is large, the matrix is said to be ill-conditioned. We can think of it this way : when the smallest eigenvalue is close to 0, making the condition number large, then A becomes close to a non-invertible(ill-conditioned) matrix.

This is because, if a matrix has any eigenvalue equal to zero, then its determinant, $|A| = 0$. As the determinant of a matrix is same as the product of its eigenvalues.

Thus making A non-invertible because only matrices with non-zero determinants ($|A| \neq 0$) have an inverse (A^{-1}).

- An efficient procedure to prepare $|b\rangle$. We assume a unitary matrix B which when applied to an initial state $|initial\rangle$ produces the required state $|b\rangle$.

  Where a Unitary matrix is one where conjugate transpose is also its inverse, $B.B^{\dagger} = I$. Unitary matrices are important because they preserve the probability amplitudes of a quantum state. Multiplication by a unitary matrix and measurement are two ways a quantum state can evolve over time.

  Thereby in interest of simplicity and focussing on the essence of this algorithm, for the rest of this report, we will assume a hermitian, s-sparse and well-conditioned matrix A.

2.4 Hamiltonian Simulation

(Certain established facts from [5] are presented unaltered in this section in order to preserve correctness.)
The matrix A can be written in terms of its spectral decomposition as follows:

$$A = \sum_{j=1}^{N} \lambda_j |a_j\rangle\langle a_j| \tag{5}$$

where λ_j are the eigenvalues and $|a_j\rangle$ the corresponding eigenvectors of A.
The spectral decomposition is useful because it allows us to use the eigenvectors of A to form an orthonormal basis for our N-dimensional space. Meaning, any vector in our N-dimensional space can be written in terms of the eigenvector basis of A, including $|b\rangle$. Therefore, writing $|b\rangle$ in terms of the eigenvector basis of A:

$$|b\rangle = \sum_{j=1}^{N} \beta_j |a_j\rangle \tag{6}$$

Where β_j are the modified coefficients of $|b\rangle$ as per the new orthonormal basis (eigenvectors of A). We need not know these coefficients for the following procedure.
While using the basis $|a_j\rangle$, A^{-1} can be simply written as:

$$A^{-1} = \sum_{j=1}^{N} \frac{1}{\lambda_j} |a_j\rangle\langle a_j| \tag{7}$$

Thus implementing A^{-1} essentially means implementing $|a_j\rangle \mapsto \frac{1}{\lambda_j}|a_j\rangle$.
We would now like to apply A^{-1} to $|b\rangle$ to obtain the required solution state $|x\rangle$, such that:

$$|x\rangle = A^{-1}|b\rangle = \sum_{j=1}^{N} \beta_j \lambda_j^{-1} |a_j\rangle \tag{8}$$

However, we cannot do that directly because the only way a quantum state (vector of probability amplitudes) can evolve is via measurement or by multiplications with a unitary matrix. Whereas A and A^{-1} are not unitary.
Instead, we can implement a unitary $U_A = e^{iA}$ which has the same eigenvectors as matrix A through the process of *Hamiltonian Simulation*. The Hamiltonian matrix H describes the physical characteristics of the system, for instance its total energy which is a sum of kinetic energy, potential energy, etc. They also determine the evolution of the quantum state in time. The equation that determines this evolution of a quantum state in time, is called the *Schrödinger Equation*:

$$i\hbar \frac{d|\psi(t)\rangle}{dt} = H|\psi(t)\rangle$$

It is a linear differential equation that relates the time-derivative of the current state to that state itself and to the *Hamiltonian*.
The solution to this equation is given by the following unitary: $U = e^{iHt}$. Where t time steps of evolution induced by the the *Hamiltonian* H, corresponds to applying the unitary matrix $U = e^{ih}$, t times. Where t need not be an integer, as this evolution is continuous in time.
Applying this function $f(x) = e^{-ix}$ to A, is same as applying $f(x)$ to its eigenvalues. So, if $A = \sum_{j=1}^{N} \lambda_j |a_j\rangle\langle a_j|$ from equation-5, then $f(A)$:

$$U_A = e^{iA} = \sum_{j=1}^{N} e^{i\lambda_j} |a_j\rangle\langle a_j| \tag{9}$$

2.5 Phase Estimation & Eigenvalue Inversion

Now that we have a readily applicable unitary implementation of A, the next step is to prepare the eigenvalues associated with A^{-1} which can then be applied to prepare the required solution state $|x\rangle = A^{-1}|b\rangle$ as shown in equation-8.
Intuitively, this is done by first estimating the eigenvalues of A via phase estimation and then inverting the eigenvalues $\lambda_j \mapsto \frac{1}{\lambda_j}$ via rotation. We will go through these steps one by one.

2.5.1 Phase Estimation

It is a well known fact that the eigenvalues of any Unitary matrix have absolute value $|\lambda| = 1$ and these eigenvalues can be represented as $\lambda = e^{i\theta}$.
In order to understand this, imagine the two dimensional X-Y Plane, consisting of a circle with radius equal to 1(unit) and its centre coinciding with the origin.
Our eigenvalues($|\lambda| = 1$) can now be understood as the points on the circumference of this circle connected to the origin via the radius. Whereas, the angle θ formed by these eigenvalues with the $X - axis$ will be the analogous to what we call the *phase* of our eigenvalue as described by the equation $\lambda = e^{i\theta}$.
What phase estimation allows us to do, is to estimate the eigenvalues λ_j by furnishing the corresponding phases (angles θ) with some approximation error which we will ignore here. The Phase Estimation process further utilizes the Quantum Fourier Transform(QFT) subroutine, which is a common subroutine for several quantum algorithms. However, here we utilize it as black box without going into the details of QFT.

2.5.2 Eigenvalue Inversion

Once we have estimated the eigenvalues λ_j, we need to invert our eigenvalues to $\frac{1}{\lambda_j}$ via rotation.
In order to understand the rotation part, picture this: It is a well known fact that matrices can used to perform various geometrical transformations, for instance transforming a geometric object from 3D to 2D, which

finds useful application in the field of computer graphics for example. Similarly flipping or rotating geometric objects via matrix multiplication is possible as well. Taking a simple example, consider the following matrix:

$$R = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$$

This matrix can rotate a given point about origin by the angle θ in anti-clockwise direction. Let us try rotating a point $(2, 3)$ by 45 degrees. Representing this process in matrix form:

$$\begin{bmatrix} cos(45) & -sin(45) \\ sin(45) & cos(45) \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} -0.707 \\ 3.535 \end{bmatrix}$$

Plotting the points $(2, 3)$ and $(-0.707, 3.535)$ on the two dimensional XY-plane, we observe that indeed the point $(2, 3)$ is rotated by 45 degrees. This can be easily verified via a rough pen and paper plot.
This simple example serves as a means to understand the eigenvalue inversion process. As using similar matrix multiplication furnishes the inverted eigenvalues $\frac{1}{\lambda_j}$ via rotation. This makes sense, as we notice that Matrix R above is Unitary, where it can be easily verified that $R.R^{\dagger} = I$.

2.6 Amplitude Amplification

Following from equation-8, simulating A as unitary U_A via hamiltonian simulation, estimating eigenvalues λ_j via phase estimation and then inverting them to $\frac{1}{\lambda_j}$ via rotation.
Therefore, We are now ready to prepare our intended state $|x\rangle$ by applying the above prepared unitary implementation of A^{-1} to $|b\rangle$ which leaves us with the following state:

$$\frac{1}{k} \sum_j \beta_j \frac{1}{\lambda_j} |a_j\rangle$$

Where k is our condition number. This tells us that the probability of obtaining the above created state after our unitary operation is $1/k$. This can be understood to happen, due to the inherent probabilistic nature of quantum operations working on various qubits.
Therefore we apply k rounds of amplitude amplification to have this state

with probability 1 essentially. Intuitively, the amplitude amplification process is similar to the geometric rotation that we performed via the matrix R in our previous section. Where the idea is to have our *quantum state* lie in a certain desired subspace. To understand, imagine rotating a vector on the $2D$ plane to ensure that it lies in the first quadrant between the positive X and positive Y axis.
This prepares our state $|x\rangle$ as intended:

$$\sum_{j=1}^{N} \beta_j \lambda_j^{-1} |a_j\rangle = A^{-1}|b\rangle = |x\rangle \tag{10}$$

Finally, we are ready to apply the relevant measurement operator on $|x\rangle$ to extract its features as required.

3 Further Remarks

This section borrows established facts from the original paper[3] and basic definitions from [4].

3.1 Run Time

- On face value the HHL algorithm offers an exponential speedup over the best classical algorithm with respect to the size N of the matrix A. However this is not without many caveats which we have either ignored or covered under assumptions in order to focus on the essence of this algorithm.

- Moreover, the quantum algorithm differs from the classical ones in a fundamental way. Where the classical algorithm is able to furnish the complete solution vector $\vec{x}$. The quantum algorithm only prepares an approximation of the state $|x\rangle$ which must then be measured to extract the desired properties of the given linear system.

- We never really get the complete solution vector in the quantum setting as the act of *measurement*, collapses the super position of various states and gives out a single output. The computation going on internally is thereby lost.

- Therefore, to get the complete N dimensional vector, you will actually need to run the algorithm N times, whereas the classical algorithm furnishes the entire $\vec{x}$ in one go. This fact illustrates the inherent power as well as limitations of a quantum computer. Also showcasing the difference of utilities between the quantum and classical algorithms.

- The run time of HHL algorithm is $O(log(N)s^2k^2/\epsilon)$. This conveys how the run time of the algorithm scales with respect to the matrix size N, matrix sparsity number s, condition number k and the error value ϵ which can also be referred to as the desired precision.

3.2 Optimality

- In this section, we aim to understand if it is possible to significantly improve upon the run time of this algorithm, and if yes, by how much. It is also interesting to ask, that how probable is it for classical algorithms to achieve further speedups in future, if they are to perform a similar task. That is, only estimate features of $\vec{x}$.

- In response to these questions, the authors have established that such possibilities are highly unlikely. Using concepts from complexity theory and by essentially showing that the matrix inversion task is equivalent in some sense to performing a general quantum computation.

- Thereby, they have established that matrix inversion is BQP-complete, where BQP is the class of decision problems solvable in polynomial time on a quantum computer with an error probability less than $1/3$.

- This validates their claims regarding the difficulty to significantly improve the algorithm's run time as any substantial improvement with respect to s, k or ϵ leads to highly unlikely complexity theory statements like $BQP = PSPACE$ or $PP \subseteq BQP$.

- Where $PSPACE$ is the class of decision problems solvable by a Turing machine using polynomial amount of space and PP refers to the class of decision problems solvable by a non-deterministic Turing machine in polynomial time with error probability less than $1/2$.

3.3 Future Directions

This section is based on article[1] by Scott Aaronson.

- It is important to note that all the assumptions that we made, so the algorithm could offer the best theoretical speedup, actually represent complex real world challenges, when we try to apply the algorithm to real world problems.
- For instance, the matrix being robustly invertible, with sparsity s, condition number k and an efficient way to prepare and load the state $|b\rangle$ into our quantum computer.
- All of these represent non-trivial challenges which often turn the theoretical exponential speedup into something more modest during real world application.
- Nonetheless, the HHL algorithm and further algorithms that build upon it, together called the *class of HHL algorithms* represent a crucial advance in the field of quantum computation, with exciting possibilities specifically in quantum machine learning.

References

[1] Scott Aaronson. "Quantum Machine Learning Algorithms: Read the Fine Print". In: (2015).

[2] Sevag Gharibian. *Introduction to Quantum Computation*. 2021.

[3] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. "Quantum Algorithm for Linear Systems of Equations". In: *Physical Review Letters* (2009).

[4] Michael Nielsen and Isaac Chuang. *Quantum Computation and Quantum Information*. 2000.

[5] Ronald De Wolf. *Quantum Computing: Lecture Notes*. 2022.