Implementation of the HHL algorithm

Sameen Aziz
Roll number: 2023-10-0231

April 30, 2023

Contents

1 Abstract

In this report, the quantum algorithm developed to solve a system of linear equations known as the HHL algorithm is discussed. Firstly, the mathematical formulation is presented. Then the implementation methodology is discussed while mapping the evolution of the state after each step. Finally, based on the methodology, a quantum circuit is built for a 2×2 matrix, as well as for a 4×4 matrix, using two different approaches each time. The circuits are tested by computing the expectation values of the Pauli operators and the results are discussed.

2 Introduction

Quantum computers employ the principles of quantum mechanics to perform computations much more efficiently than their classical counterparts. For some problems, such as finding the prime factors of a large integer, quantum computers have been able to provide exponential speed-ups over classical computers [1]. One such domain where quantum computers have proven their superiority over classical computers is that of linear algebra. Solving a system of linear equations [1], quantum verification of matrix product [2], and commutativity testing of a set of matrices [3] are some of the problems that have been quite efficiently tackled with the help of quantum computers. In this report, the algorithm developed to solve a system of linear equations, known as the HHL algorithm, will be discussed.

Linear equations are of prime importance and find their applications in several fields, such as science, engineering, economics, finance, etc. Several classical techniques have been developed to solve systems of linear equations. However, as the size of the data set grows, so does the computation power and time needed to solve the system of linear equations on a classical computer [1].

In 2009, Harrow, Hassidim, and Llyod proposed the HHL algorithm to solve a system of linear equations on a quantum computer. On a classical computer, solving a system of N linear equations in N variables takes time of order N. The HHL algorithm can cut down this time to the order $\log(N)$ in some cases. This algorithm can prove to be useful when the user is not interested in the actual solution vector but is rather interested in the result of applying some operator to it. The said algorithm cannot, however, provide an exact solution to the system of linear equations, and that may be considered one of its limitations [1].

Furthermore, the HHL algorithm imposes a few restrictions on the system of linear equations, which might limit its applications, such as the matrix A should be Hermitian, should be a sparse matrix, meaning that most of its elements should be zero, and should have a low condition number κ, where κ is a measure of how sensitive a matrix is to perturbations [1].

Despite the seemingly hopeless state of affairs, the HHL algorithm has found its applicability in machine learning, such as solving the problem of least-squares fitting efficiently [4]. The HHL algorithm was also applied to the finite element

method, which finds numerical approximations to the solutions of boundary value problems for partial differential equations [5].

3 Mathematical Formulation

Let A be an $N \times N$ matrix and let $\vec{b}$ be an N-dimensional vector. Then, the task is to find some vector $\vec{x}$ such that $A\vec{x} = \vec{b}$. Let us begin by writing the equation in the language of quantum mechanics.

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

Applying A^{-1} on both sides gives the following

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

A can be written in terms of its eigenvalues and eigenvectors as

$$A = \sum_{i=0}^{N-1} \lambda_i |u_i\rangle\langle u_i|,$$

where $|u_i\rangle$ is an eigenvector of A corresponding to the eigenvalue λ_i and $\lambda_i \in \mathbb{R}$ for all values of i. Similarly,

$$A^{-1} = \sum_{i=0}^{N-1} \lambda_i^{-1} |u_i\rangle\langle u_i|.$$

Let us say we are given the quantum state $|b\rangle = \sum_i b_i|i\rangle$. Then $|b\rangle$ can be written in the eigenbasis of A as

$$|b\rangle = \sum_{j=0}^{N-1} b_j |u_j\rangle,$$

where $b_j \in \mathbb{C}$. Then,

$$|x\rangle = \left(\sum_{i=0}^{N-1} \lambda_i^{-1} |u_i\rangle\langle u_i| \right) \sum_{j=0}^{N-1} b_j |u_j\rangle,$$

$$|x\rangle = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \lambda_i^{-1} b_j |u_i\rangle\langle u_i|u_j\rangle,$$

but $\langle u_i|u_j\rangle = \delta_{ij}$, and $\delta_{ij} = 1$ only when $i = j$. Thus, we are left with

$$|x\rangle = \sum_{i=0}^{N-1} \lambda_i^{-1} b_i |u_i\rangle.$$

which is our desired solution vector [6].

4 Implementation methodology

To implement the HHL algorithm on a quantum computer, we begin by creating two quantum registers, both of which are initialized to $|0\rangle$. One of these is used to store the solution vector $|x\rangle$ and is denoted by n_b. The other one is used to store the eigenvalues of A and is denoted by n. There is also an auxiliary qubit denoted by $|0\rangle_a$, which will be measured at the end [6].

Now since A and A^{-1} are not unitary, A^{-1} cannot be directly applied to $|b\rangle$. A can, however, be made unitary in the following manner

$$U = \mathrm{e}^{iAt}.$$

Thus, we can apply quantum phase estimation on U to estimate the eigenvalues of A [1].

Let's say the initial state of the system looks like

$$|\psi\rangle_0 = |0\rangle_{n_b}|0\rangle_n|0\rangle_a.$$

Then, the state $|b\rangle$ is fed into the register labelled as n_b.

$$|\psi\rangle_1 = |b\rangle_{n_b}|0\rangle_n|0\rangle_a.$$

Then, Quantum Phase Estimation (QPE) is applied, after which the state gets transformed to

$$|\psi\rangle_2 = \sum_{j=0}^{N-1} b_j|u_j\rangle_{n_b}|\tilde{\lambda_j}\rangle_n|0\rangle_a,$$

where $N = 2^{n_b}$.

After this, the ancilla qubit is rotated, and we get

$$|\psi\rangle_3 = \sum_{j=0}^{N-1} b_j|u_j\rangle_{n_b}|\tilde{\lambda_j}\rangle_n \left(\sqrt{1 - \frac{C^2}{\tilde{\lambda_j}^2}}|0\rangle_a + \frac{C}{\tilde{\lambda_j}}|1\rangle_a\right),$$

where C is some normalization constant. The next step is to apply inverse QPE, after which we obtain the following state

$$|\psi\rangle_4 = \sum_{j=0}^{N-1} b_j|u_j\rangle_{n_b}|0\rangle_n \left(\sqrt{1 - \frac{C^2}{\tilde{\lambda_j}^2}}|0\rangle_a + \frac{C}{\tilde{\lambda_j}}|1\rangle_a\right).$$

Then, as the final step, the auxiliary qubit is measured. If the outcome is $|0\rangle_a$, the computation is repeated. If we get $|1\rangle_a$ as our measurement outcome, then the final state of the register is

$$|\psi\rangle_5 = \frac{1}{\sqrt{\sum_{j=0}^{N-1} |\frac{b_j C}{\tilde{\lambda_j}}|^2}} \sum_{j=0}^{N-1} \frac{b_j C}{\tilde{\lambda_j}}|u_j\rangle_{n_b}|0\rangle_n|1\rangle_a,$$

$$|\psi\rangle_5 = \frac{1}{\sqrt{\sum_{j=0}^{N-1} |\frac{b_j}{\tilde{\lambda_j}}|^2}}|x\rangle_{n_b}|0\rangle_n|1\rangle_a.$$

Thus, the solution is stored in the n_b register [6].

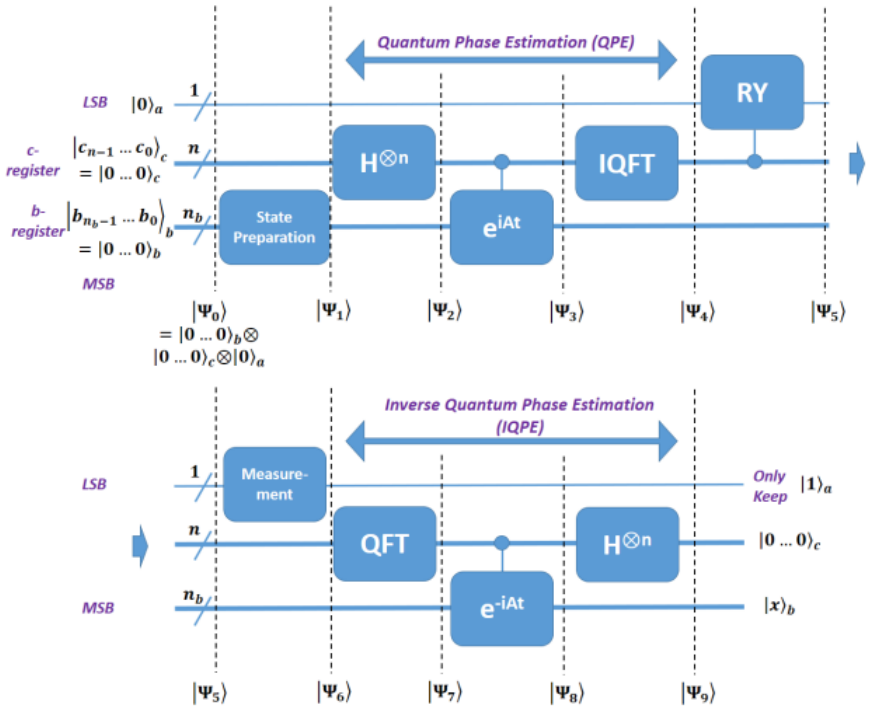


Figure 1: A schematic of the HHL algorithm circuit [6].

5 Implementation for a 2×2 matrix

Let us implement the HHL algorithm for the following system of linear equations.

$$\begin{aligned} \frac{3}{2}x_1 + \frac{1}{2}x_2 &= 0 \\ \frac{1}{2}x_1 + \frac{3}{2}x_2 &= 1 \\ \begin{pmatrix} 3/2 & 1/2 \\ 1/2 & 3/2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ A\vec{x} &= \vec{b} \end{aligned}$$

The solution vector, computed classically, turns out to be

$$\vec{x} = \begin{pmatrix} -1/4 & 3/4 \end{pmatrix}^T,$$

where $|x_0|^2 : |x_1|^2 = 1 : 9$.

First, we have to find $U = e^{iAt}$. To do so, let us begin by diagonalizing A. The eigenvalues of A are $\lambda_0 = 2$ and $\lambda_1 = 1$ and the eigenvectors are $v_0 = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}^T$

and $v_1 = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}^T$ respectively.

$$\begin{aligned} V &= \begin{pmatrix} v_0 & v_1 \end{pmatrix} \\ &= \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \\ A_d &= V^\dagger A V \\ A_d &= \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} \end{aligned}$$

where A_d represents diagonalized A. To get diagonalized U, we exponentiate the diagonal elements of A_d.

$$U_d = \begin{pmatrix} \mathrm{e}^{i1t} & 0 \\ 0 & \mathrm{e}^{i2t} \end{pmatrix}$$

Here t is taken to be $\pi/2$. The value of t should be chosen such that $\lambda_j t/2\pi \in [0, 1)$ so as to obtain accurate results from QPE [7]. Inserting $t = 3\pi/4$ in U_d, we get

$$\begin{aligned} U_d &= \begin{pmatrix} \mathrm{e}^{i\pi/2} & 0 \\ 0 & \mathrm{e}^{i\pi} \end{pmatrix} \\ &= \begin{pmatrix} i & 0 \\ 0 & -1 \end{pmatrix} \end{aligned}$$

To obtain U from U_d, we proceed in the following manner.

$$\begin{aligned} U &= V U_d V^\dagger \\ &= \frac{1}{2} \begin{pmatrix} -1+i & 1+i \\ 1+i & -1+i \end{pmatrix}. \end{aligned}$$

We also need to find U^2 in order to be able to implement QPE successfully.

$$\begin{aligned} U^2 &= V U_d^2 V^\dagger \\ &= \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}. \end{aligned}$$

To implement inverse QPE, we need U^{-1} and U^{-2}.

$$\begin{aligned} U^{-1} &= \frac{1}{2} \begin{pmatrix} -1-i & 1-i \\ 1-i & -1-i \end{pmatrix}, \\ U^{-2} &= \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}. \end{aligned}$$

Now there are a few different methods available to implement these matrices in a circuit, two of which will be discussed in section 5.1 and section 5.2 below.

After implementing QPE, the next step is to find the angles for the controlled-Y rotation. The angles can be found using $\theta = 2\arcsin(\frac{C}{\tilde{\lambda}_j})$, where $\tilde{\lambda}_j = \frac{N\lambda_j t}{2\pi}$ and $C \leq \tilde{\lambda}_j$.Thus, $\theta = \pi/3$ for $\tilde{\lambda}_0 = 2$ and $\theta = \pi$ for $\tilde{\lambda}_1 = 1$ [6].

Then, inverse QPE is applied, and finally, the ancilla qubit and the n_b register are measured at the end.

5.1 Implementation using the arbitrary unitary matrix

To implement the unitary matrix U, we need to compare it with the general form of a 4-parameter arbitrary unitary gate which is given below

$$U = \begin{pmatrix} e^{i\gamma}\cos(\theta/2) & -e^{i(\gamma+\lambda)}\cos(\theta/2) \\ e^{i(\gamma+\phi)}\cos(\theta/2) & e^{i(\gamma+\phi+\lambda)}\cos(\theta/2) \end{pmatrix}.$$

Comparing this general matrix with our matrices, we see that our U can be implemented by choosing $\theta = \pi/2, \phi = -\pi/2, \lambda = \pi/2, \gamma = 3\pi/4$. U^2 can be implemented by choosing $\theta = \pi, \phi = \pi, \lambda = 0, \gamma = 0$. To implement U^{-1}, we need to set $\theta = \pi/2, \phi = \pi/2, \lambda = -\pi/2, \gamma = -3\pi/4$ and since $U^2 = U^{-2}$, we can use the parameters used for U^2 to implement U^{-2} [6].

5.2 Implementation using the UnitaryGate class

In Qiskit, the UnitaryGate class allows us to convert a unitary matrix into a unitary gate. If decomposing a matrix into a series of rotations manually seems daunting, one need not fear because the UnitaryGate class in Qiskit comes to the rescue. The unitary gate can then also be converted to a control gate using the control() command in Qiskit [8].

6 Implementation for a 4×4 matrix

Let us now try to implement the HHL algorithm for a 4×4 system.

$$A = \begin{pmatrix} 5 & 3 & 1 & -2 \\ 3 & 5 & 2 & -1 \\ 1 & 2 & 5 & -3 \\ -2 & -1 & -3 & 5 \end{pmatrix}, b = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

In this case, implementing $U = e^{iAt}$ is not a straightforward task because, this time, we do not have the luxury of comparing the elements of our matrix U with those of an arbitrary unitary matrix as we did above. There are, however, a few other techniques that can be followed, two of which will be discussed below. The rest of the implementation scheme is the same as for a 2×2 matrix already discussed above and will, therefore, not be explained here.

6.1 Implementation using Trotterization

A Hermitian matrix can be written as a combination of Pauli operators.

$$H = \sum_i \alpha_i P_i,$$

where $\alpha_i = \frac{1}{2^n} Tr(P_i H)$ [9]. Using this scheme, we see that A can be written as

$$A = 5(I \otimes I) + 3(Z \otimes X) + 1(X \otimes Z) + 2(Y \otimes Y).$$

Now that we have written A as a sum of tensor products of the Pauli matrices, we can input it into Qiskit, which approximates e^{iAt} following a technique called Trotterization [10, 8]. Figure 2 shows the circuit that approximates e^{iAt} in Qiskit.

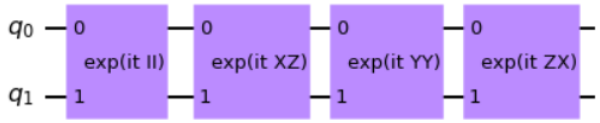


Figure 2: e^{iAt} implementation circuit in Qiskit.

Here, t is taken to be $\pi/8$.

6.2 Implementation using the UnitaryGate class

To implement this method, we need only follow the approach described in section 5.2. We begin by diagonalizing A and proceed to find U and its powers, as well as their inverses. In this case, U turns out to be the following matrix

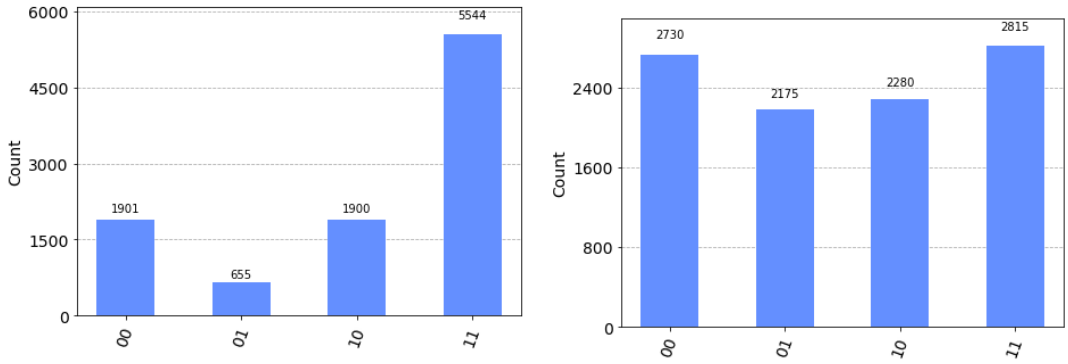
$$U = \begin{pmatrix} e^{i11t} & 0 & 0 & 0 \\ 0 & e^{i5t} & 0 & 0 \\ 0 & 0 & e^{i3t} & 0 \\ 0 & 0 & 0 & e^{i1t} \end{pmatrix}.$$

As already mentioned above, t is taken to be $\pi/8$. The next step is to compute U^2, U^4 and U^8 and their inverses, which can be done quite easily using Python.

7 Results

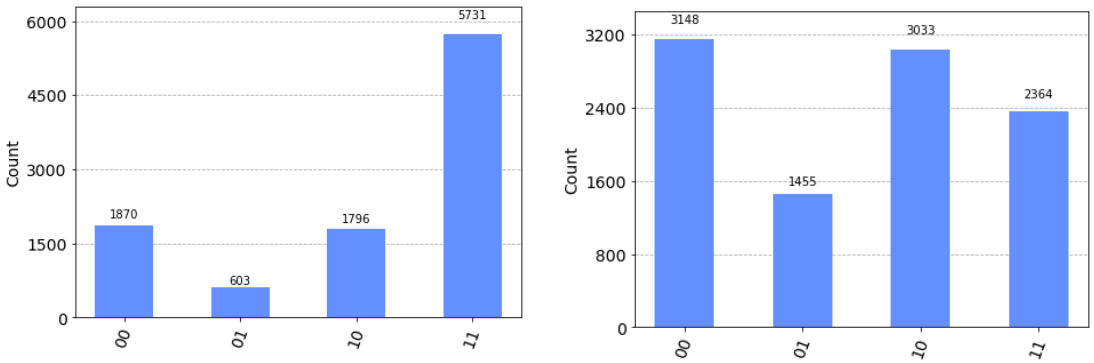
7.1 Results for the 2×2 case

Attached below are the results of the implementation of the HHL algorithm for the 2×2 matrix case using both methods described above.



(a) Simulator results. (b) IBM quantum computer results.

Figure 3: Results for the 2×2 implementation (arbitrary U method).



(a) Simulator results. (b) IBM quantum computer results.

Figure 4: Results for the 2×2 implementation (UnitaryGate class method).

In the circuit, the ancillary qubit and the register n_b were measured at the end. The simulator results show that the ancillary qubit collapses to the state $|1\rangle$ most of the time. The IBM quantum computer results seem, however, quite hopeless, and the blame may be placed on the noise and errors that may occur on real hardware.

We also note that the probability ratio of $|0\rangle_b|1\rangle_a$ and $|1\rangle_b|1\rangle_a$ in the simulator result is close to the expected value in both cases, which turns out to be $1 : 9$. More information regarding how we reach this conclusion can be found in [6].

The veracity of the circuit was further tested by computing the expectation values of the Pauli operators. The expectation values were computed using three different methods: using our circuit, using Qiskit's HHL solver and using classical methods. The results are shown below.

Operator	Our HHL circuit	Qiskit's HHL circuit	Classical method
X	0.705	0.896	-0.375
Y	0.0	0.0	$0i$
Z	-0.498	-0.277	-0.5

Figure 5: Expectation value computation results (arbitrary U method).

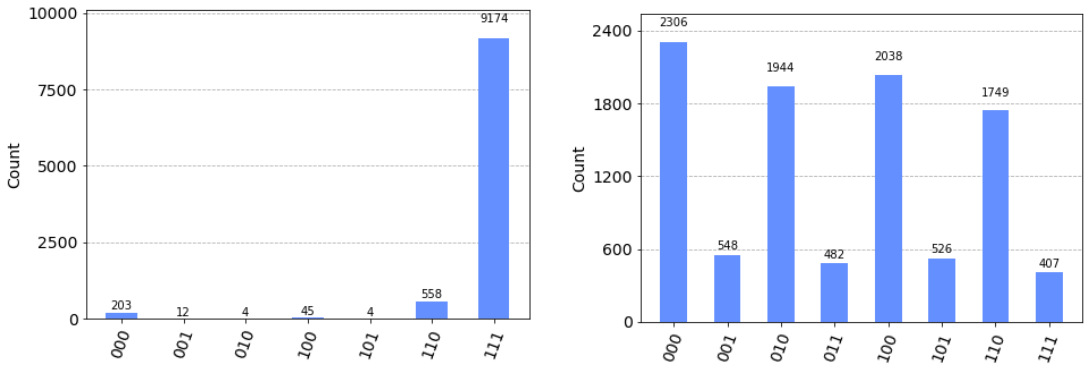
Operator	Our HHL circuit	Qiskit's HHL circuit	Classical method
X	0.742	0.873	-0.375
Y	0.0	0.0	$0i$
Z	-0.498	-0.236	-0.5

Figure 6: Expectation value computation results (UnitaryGate class method).

We can see that our results are either close to or better than Qiskit's HHL solver results indicating that our circuit has been somewhat successful and bringing back some of the dignity our circuit lost at the IBM results.

7.2 Results for the 4×4 case

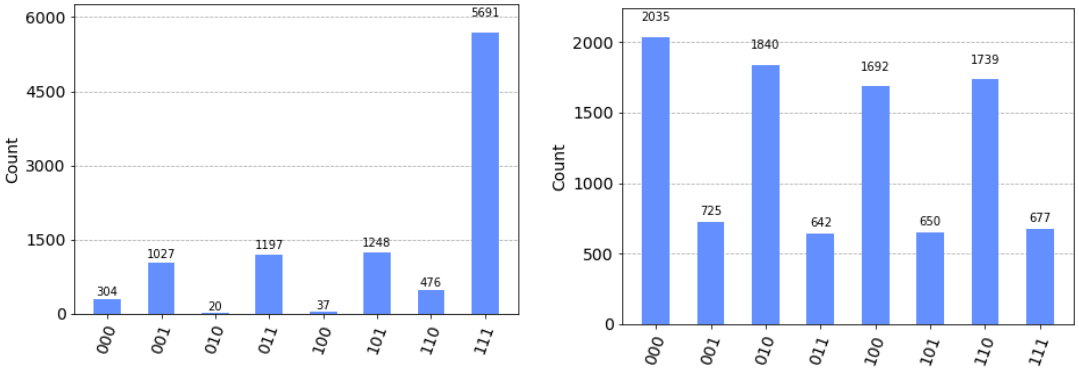
Attached below are the results of the implementation of the HHL algorithm for the 4×4 matrix case using Trotterization as well as the UnitaryGate class method.



(a) Simulator results.

(b) IBM quantum computer results.

Figure 7: Results for the 4×4 implementation (Trotterization method).



(a) Simulator results.

(b) IBM quantum computer results.

Figure 8: Results for the 4×4 implementation (UnitaryGate class method).

The simulator results show that the ancillary qubit collapses to the state $|1\rangle$ most of the time. However, IBM quantum computer results again seem quite disappointing, and the blame may again be placed on noise in an actual quantum computer. However, in this case, we see that the results obtained using Trotterization, even on the simulator, are also quite saddening, and as of right now, we are unsure where to place the blame for that.

Here again, we note that the probability ratio of $|00\rangle_b|1\rangle_a$ to $|11\rangle_b|1\rangle_a$ in the UnitaryGate method simulator results is close to the expected value which is 0.228. Also, the probability ratio of $|01\rangle_b|1\rangle_a$ to $|11\rangle_b|1\rangle_a$ is also somewhat close to the expected values of 0.118 but that of $|10\rangle_b|1\rangle_a$ to $|11\rangle_b|1\rangle_a$ seems to be not quite right.

The expectation values of the Pauli matrices were computed, and the results are shown here.

Operator	**Our HHL circuit**	**Qiskit's HHL circuit**	**Classical method**
X	0.118	0.0	0.084
Y	-0.137	0.0	-0.230
Z	0.984	0.400	0.115

Figure 9: Expectation value computation results (Trotterization method).

Operator	**Our HHL circuit**	**Qiskit's HHL circuit**	**Classical method**
X	0.769	0.0	0.084
Y	-0.301	0.0	-0.230
Z	0.523	0.398	0.115

Figure 10: Expectation value computation results (UnitaryGate class method).

Looking at the results, we see that we can't make a definitive statement as to whether our circuit has been successful or not. We see that the value for $\langle X \rangle$ obtained is close to the actual value using the Trotterization method but seems to have gone astray in the UnitaryGate class method. The results obtained for $\langle Y \rangle$ are somewhat close to the actual value in both cases. For $\langle Z \rangle$, the result obtained using the UnitaryGate method is close to the one obtained through Qiskit's HHL circuit, but neither is close to the actual value.

8 Conclusion

Looking at the results of all the cases, we can come to the conclusion that our circuit implementation of the HHL algorithm has been successful for the 2×2 matrix. For the 4×4 matrix case, however, the results have not been able to stand up to our expectations, especially in the empolyment of the Trotterization scheme. However, the UnitaryGate class method results seem convincing, indicating that the problem most likely lies somewhere in the Trotterization scheme and not in the general implementation of the circuit. As to where the problem lies is, however, yet to be determined and needs further research.

Bibliography

[1] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. "Quantum Algorithm for Linear Systems of Equations". In: *Physical Review Letters* 103.15 (2009). DOI: 10.1103/physrevlett.103.150502. URL: https://doi.org/10.1103%2Fphysrevlett.103.150502.

[2] Harry Buhrman and Robert Spalek. *Quantum Verification of Matrix Products*. 2005. arXiv: quant-ph/0409035 [quant-ph].

[3] Yuki Kelly Itakura. *Quantum Algorithm for Commutativity Testing of a Matrix Set*. 2005. arXiv: quant-ph/0509206 [quant-ph].

[4] Nathan Wiebe, Daniel Braun, and Seth Lloyd. "Quantum Algorithm for Data Fitting". In: *Phys. Rev. Lett.* 109 (5 2012), p. 050505. DOI: 10.1103/PhysRevLett.109.050505. URL: https://link.aps.org/doi/10.1103/PhysRevLett.109.050505.

[5] Ashley Montanaro and Sam Pallister. "Quantum algorithms and the finite element method". In: *Physical Review A* 93.3 (2016). DOI: 10.1103/physreva.93.032324. URL: https://doi.org/10.1103%2Fphysreva.93.032324.

[6] Hector Jose Morrell Jr au2, Anika Zaman, and Hiu Yung Wong. *Step-by-Step HHL Algorithm Walkthrough to Enhance the Understanding of Critical Quantum Computing Concepts*. 2023. arXiv: 2108.09004 [quant-ph].

[7] Almudena Carrera Vazquez, Ralf Hiptmair, and Stefan Woerner. "Enhancing the Quantum Linear Systems Algorithm Using Richardson Extrapolation". In: *ACM Transactions on Quantum Computing* 3.1 (2022), pp. 1–37. DOI: 10.1145/3490631. URL: https://doi.org/10.1145%2F3490631.

[8] Qiskit contributors. *Qiskit: An Open-source Framework for Quantum Computing*. 2023. DOI: 10.5281/zenodo.2573505.

[9] Sebastián V. Romero and Juan Santos-Suárez. *PauliComposer: Compute Tensor Products of Pauli Matrices Efficiently*. 2023. arXiv: 2301.00560 [quant-ph].

[10] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.