# Master Semester Project: On Quantum Algorithms for Solving Linear Systems of Equations

Adrien Vandenbroucque
Supervisor: Dr. Nicolas Macris

Fall 2019

## 1 Introduction

Systems of linear equations arise extensively in many scientific fields, and have many applications in real-life problems. An example is in the field of machine learning, where given multidimensional data by an $N \times N$ matrix $A$ and labels by a $N \times 1$ vector $\mathbf{b}$, one must find $\mathbf{x}$ such that $A\mathbf{x}$ equals $\mathbf{b}$, or is as close as possible to $\mathbf{b}$. In fields such as computer vision and others, in practice, the matrix $A$ turns out to be often symmetric. In this case, the best known method is the conjugate gradient method, with runtime complexity $\mathcal{O}(Ns\kappa \log(1/\epsilon))$, where $s$ is the maximum number of non-zero entries in a row (or column) of $A$ and $\kappa$ is its condition number. So in cases where the dimension $N$ of $A$ is very large, one needs to find faster methods in order to solve this problem efficiently.

In 2009, Harrow, Hassidim and Lloyd devised a quantum algorithm to solve linear systems of equations, but not in the same sense as in the classical case. That is, the algorithm produces a quantum state $|x\rangle = \sum_{i=1}^{N} x_i |i\rangle$ where the coefficients $x_i$ correspond to the different components of $\mathbf{x}$. The time complexity of the quantum algorithm is $\mathcal{O}(\log N s^2 \kappa^2/\epsilon)$, providing an exponential speedup in $N$ compared to the classical one.

In this report, a review of this algorithm is presented first, explaining in more detail how one can compute the different parts involved. Secondly, a method is proposed to solve problems such as the satisfiability problems when the input matrix $A$ has a specific structure. This can be extended to more general problems. Thirdly, an implementation of a simple 2x2 system is described together with the obtained results. Finally, a summary is given in the conclusion together with the general trends observed.

## 2 Problem statement

We are given $N$ linear equations with $N$ unknown, which can be expressed as $A\mathbf{x} = \mathbf{b}$, where $A \in \mathbb{C}^{N \times N}$ is the matrix of coefficients, $\mathbf{x} \in \mathbb{C}^N$ is a vector of unknown components, and $\mathbf{b} \in \mathbb{C}^N$ is the vector of solutions. If $A$ is invertible, then we can write the solution as $\mathbf{x} = A^{-1}\mathbf{b}$. This problem is known as the Linear Systems Problem (LSP), and we can more formally define it as follows.

**Definition 2.1** (LSP)**.** *Given a system of linear equations, with $A \in \mathbb{C}^{N \times N}$ and a vector $\mathbf{b} \in \mathbb{C}^N$, find a vector $\mathbf{x} \in \mathbb{C}^N$ such that $A\mathbf{x} = \mathbf{b}$, or output a flag if no solution was found.*

The quantum version of this problem is formulated in Definition 2.2.

**Definition 2.2** (QLSP)**.** *Let $A \in \mathbb{C}^{N \times N}$ be an Hermitian matrix with unit determinant, and $\mathbf{b}, \mathbf{x} \in \mathbb{C}^N$ vectors such that $\mathbf{x} = A^{-1}\mathbf{b}$. Let the quantum state on $\lceil \log(N) \rceil$ qubits $|b\rangle$ and $|x\rangle$ be given by*

$$|b\rangle := \frac{\sum_{i=1}^{N} b_i |i\rangle}{\|\sum_{i=1}^{N} b_i |i\rangle\|_2} \text{ and } |x\rangle := \frac{\sum_{i=1}^{N} x_i |i\rangle}{\|\sum_{i=1}^{N} x_i |i\rangle\|_2} \tag{1}$$

*where $b_i$ and $x_i$ are the $i^{th}$ component of vector $\mathbf{b}$ and $\mathbf{x}$ respectively.*
*Given the matrix $A$ and the state $|b\rangle$, find a state $|\tilde{x}\rangle$ such that $\||\tilde{x}\rangle - |x\rangle\|_2 \leq \epsilon$ with probability $\Omega(1)$ (say, greater than $1/2$), with a flag if no solution was found.*

Although both definitions look very similar, they actually solve distinct problems. In the LSP, the goal is to find the solution $\mathbf{x}$ to the equation $A\mathbf{x} = \mathbf{b}$. In the quantum version, the goal is to prepare a state $|x\rangle$ such that the equation $A|x\rangle = |b\rangle$ is satisfied, but this does not mean that we have access to the classical vector $\mathbf{x}$. Since we are manipulating quantum states, it is also not clear whether getting $|x\rangle$ can be done efficiently. Indeed, for that one should be able to first prepare the state $|b\rangle$ in an efficient way, then perform various computations, and finally read the solution $|x\rangle$. Here, "efficient" means that the algorithm is "polylogarithmic" in $N$, the system size, however we see here that this will be problematic as reading out the whole of $|x\rangle$ would require $\mathcal{O}(N)$ time. So it is crucial to remember that a solution to QLSP must be used in an application where only samples from the vector $\mathbf{x}$ are needed, or if we are interested in computing $\langle x| M |x\rangle$ for some operator $M$.

Looking at the problem definitions given before, it may not be clear that encoding our problem in quantum states, solving using HHL, and reading out $|x\rangle$ will actually give us the components of $\mathbf{x}$. We prove in the following lemma that it is actually equivalent.

**Lemma 2.1.** *Solving QLSP on the encoded version $|b\rangle$ of $\mathbf{b}$ is equivalent to solving LSP on $\mathbf{b}$, in the sense that the solution to QLSP ouputs a state $|x\rangle$ which encodes the components of $\mathbf{x}$, the solution to LSP.*

*Proof.* Let $A = \sum_{j=0}^{N-1} \lambda_j \mathbf{u}_j \mathbf{u}_j^T$ be the spectral decomposition of $A$, where $\lambda_j$s are the eigenvalues and the $\mathbf{u}_j$s are its eigenvectors. Note that since the $\mathbf{u}_j$s are orthonormal, we can write it using the Dirac notation as $A = \sum_{j=0}^{N-1} \lambda_j |u_j\rangle \langle u_j|$.

Now, as in definition 2.2, let $|b\rangle$ and $|x\rangle$ be the encoded versions of $\mathbf{b}$ and $\mathbf{x}$ respectively. We can rewrite them in the basis of the eigenvectors of $A$, so

$$|b\rangle = \frac{\sum_{j=0}^{N-1} \beta_j |u_j\rangle}{\|\mathbf{b}\|_2} \text{ and } |x\rangle = \frac{\sum_{j=0}^{N-1} \tilde{x}_j |u_j\rangle}{\|\mathbf{x}\|_2}. \tag{2}$$

We want to show that $|x\rangle = |A^{-1}\mathbf{b}\rangle$ is proportional to $A^{-1}|b\rangle$, so that our algorithm returns a valid answer. We have

$$A^{-1}|b\rangle = \frac{1}{\|\mathbf{b}\|_2} \sum_{j=0}^{N-1} \frac{\beta_j}{\lambda_j} |u_j\rangle \tag{3}$$

which gives

$$\frac{A^{-1}|b\rangle}{\|A^{-1}|b\rangle\|} = \frac{1}{\frac{1}{\|\mathbf{b}\|_2}\sqrt{\sum_{j=0}^{N-1} \frac{|\beta_j|^2}{|\lambda_j|^2}}} \frac{1}{\|\mathbf{b}\|_2} \sum_{j=0}^{N-1} \frac{\beta_j}{\lambda_j} |u_j\rangle = \frac{1}{\sqrt{\sum_{j=0}^{N-1} \frac{|\beta_j|^2}{|\lambda_j|^2}}} \sum_{j=0}^{N-1} \frac{\beta_j}{\lambda_j} |u_j\rangle \tag{4}$$

when normalized.
Note that the $i^{th}$ component of $A^{-1}|b\rangle$ is

$$\langle u_i| A^{-1} |b\rangle = \frac{1}{\|\mathbf{b}\|_2} \frac{\beta_i}{\lambda_i}. \tag{5}$$

On the other hand, the $i^{th}$ component of $|x\rangle$ is

$$\langle u_i|x\rangle = \frac{1}{\|\mathbf{x}\|_2}\tilde{x_i} = \frac{1}{\|\mathbf{x}\|_2}(\mathbf{u}_i^T A^{-1}\mathbf{b}). \tag{6}$$

Now we would want to use the Dirac notation since $\mathbf{u}_i$ is a unit vector, but the problem is that $\mathbf{b}$ might not be. However, we can still do it using

$$\frac{1}{\|\mathbf{x}\|_2}(\mathbf{u}_i^T A^{-1}\mathbf{b}) = \frac{1}{\|\mathbf{x}\|_2}(\mathbf{u}_i^T A^{-1}\frac{\mathbf{b}}{\|\mathbf{b}\|_2}\|\mathbf{b}\|_2) = \frac{1}{\|\mathbf{x}\|_2}(\langle u_i| A^{-1} |b\rangle \|\mathbf{b}\|_2) = \frac{1}{\|\mathbf{x}\|_2}\frac{\beta_i}{\lambda_i}, \tag{7}$$

so that $\tilde{x_i} = \frac{\beta_i}{\lambda_i}$. A such, we can rewrite

$$|A^{-1}\mathbf{b}\rangle = |x\rangle = \frac{1}{\|\mathbf{x}\|_2}\sum_{j=0}^{N-1}\frac{\beta_j}{\lambda_j}|u_j\rangle \tag{8}$$

which is equal to

$$\frac{|A^{-1}\mathbf{b}\rangle}{\||A^{-1}\mathbf{b}\rangle\|_2} = \frac{1}{\frac{1}{\|\mathbf{x}\|_2}\sqrt{\sum_{j=0}^{N-1}\frac{|\beta_j|^2}{|\lambda_j|^2}}}\frac{1}{\|\mathbf{x}\|_2}\sum_{j=0}^{N-1}\frac{\beta_j}{\lambda_j}|u_j\rangle = \frac{1}{\sqrt{\sum_{j=0}^{N-1}\frac{|\beta_j|^2}{|\lambda_j|^2}}}\sum_{j=0}^{N-1}\frac{\beta_j}{\lambda_j}|u_j\rangle \tag{9}$$

when normalized.
As such, we see that the normalized $|A^{-1}\mathbf{b}\rangle$ and $A^{-1}|b\rangle$ are equal, so we get that

$$|A^{-1}\mathbf{b}\rangle \propto A^{-1}|b\rangle \tag{10}$$

which concludes the proof. □

# 3 Description of the HHL algorithm

## 3.1 Solving Systems of Linear Equations

Before giving a description of the procedure of the HHL algorithm, we briefly review what the algorithm computes in order to solve systems of linear equations.

Recall that $A$ is Hermitian, and as such it has a spectral decomposition. So we can write

$$A = \sum_{j=0}^{N-1}\lambda_j|u_j\rangle\langle u_j|, \tag{11}$$

where the $\lambda_j$'s are the eigenvalues and the $|u_j\rangle$'s are the eigenvectors. The vector $|b\rangle$ can be rewritten in the basis formed by the eigenvectors of $A$, so

$$|b\rangle = \sum_{j=0}^{N-1}\beta_j|u_j\rangle. \tag{12}$$

Observing that the inverse of $A$ can be obtained easily from the spectral decomposition of $A$ as

$$A^{-1} = \sum_{j=0}^{N-1} \frac{1}{\lambda_j} |u_j\rangle \langle u_j| , \tag{13}$$

we finally get

$$|x\rangle = A^{-1} |b\rangle = \sum_{j=0}^{N-1} \frac{\beta_j}{\lambda_j} |u_j\rangle \langle u_j| , \tag{14}$$

which is the state that the HHL algorithm tries to reach.

## 3.2 High-level description

Here, we give a high-level explanation of the HHL algorithm, and explain each of the steps in more detail in separate sections.

The steps are as follows:

1. Prepare the input state $|0\rangle |b\rangle = \sum_{j=0}^{N-1} \beta_j |0\rangle |u_j\rangle$, where $\log N$ is the number of qubits needed to represent $|b\rangle$.

2. Apply the Quantum Phase Estimation (QPE) algorithm with unitary $U = e^{iAt}$ in order to find the eigenvalues of $A$, with $t$ to be specified later. They will be stored in a register of $\log D$ qubits, number which tuned depending on what accuracy and probability of success is desired for the phase estimation. After performing the inverse Quantum Fourier Transform part of the phase estimation, the state is

$$\sum_{j=0}^{N-1} \sum_{k=0}^{D-1} \alpha_{k|j} \beta_j |k\rangle |u_j\rangle . \tag{15}$$

   Note that $|\alpha_{k|j}|$ will be large for values of $k$ such that $k \approx \frac{\lambda_j t}{2\pi}$ (this will be explained in the section about Quantum Phase Estimation). So one can relabel the $|k\rangle$ states as $\tilde{\lambda_k} = \frac{2\pi k}{t}$, giving

$$\sum_{j=0}^{N-1} \sum_{k=0}^{D-1} \alpha_{k|j} \beta_j |\tilde{\lambda_k}\rangle |u_j\rangle . \tag{16}$$

   Supposing that the phase estimation is perfect, the state of the system is

$$\sum_{j=0}^{N-1} \beta_j |\lambda_j\rangle |u_j\rangle . \tag{17}$$

3. Append an ancilla qubit and apply a rotation conditioned on the value of $|\lambda_j\rangle$, resulting in

$$\sum_{j=0}^{N-1} \beta_j |\lambda_j\rangle |u_j\rangle \left( \sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \frac{C}{\lambda_j} |1\rangle \right) . \tag{18}$$

   where $C$ is a constant to be chosen such that the rotation can be done physically. Typically, $C$ is chosen such that $C \leq \min_j |\lambda_j| = O(1/\kappa)$ to ensure that the rotation is valid.

4. Uncompute the values of the register containing the eigenvalues by applying the inverse Quantum Phase Estimation,

$$\sum_{j=0}^{N-1} \beta_j |0\rangle |u_j\rangle \left( \sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \frac{C}{\lambda_j} |1\rangle \right). \tag{19}$$

5. Measure the last qubit in the computational basis. Conditioned on observing $|1\rangle$, the state is

$$\frac{1}{\sqrt{\sum_{j=0}^{N-1} \frac{C^2 |\beta_j|^2}{|\lambda_j|^2}}} \sum_{j=0}^{N-1} \frac{C\beta_j}{\lambda_j} |0\rangle |u_j\rangle, \tag{20}$$

which corresponds to $A^{-1}|b\rangle = |x\rangle$ up to normalization. Note that $\frac{1}{\sqrt{\sum_{j=0}^{N-1} \frac{C^2 |\beta_j|^2}{|\lambda_j|^2}}}$ is the probability of measuring 1 for the ancilla qubit.

# 4 Quantum Phase Estimation

We describe here the Quantum Phase Etimation procedure used in the HHL algorithm. As this is the step that can induce the most errors in the computation, we provide a full error analysis of the algorithm.

## 4.1 Algorithm description

The phase estimation procedure consists in finding the eigenvalues of an operator $U$. More precisely, given a unitary matrix $U$ and one of its eigenvector $|\psi\rangle$ with eigenvalue $e^{2\pi i\lambda}$, the goal of the algorithm is to estimate $\lambda$.

The procedure requires two registers, one containining $d$ qubits all initialized to $|0\rangle$ which will contain the eigenvalue at the end, and another one initialized to $|\psi\rangle$, and containing as many qubits as necessary to represent $|\psi\rangle$. For the first register, the choice of $d$ will depend on the accuracy we want for the estimate of $\lambda$, and also with what probability we wish the phase estimation to be successful.

**Definition 4.1** (Quantum Phase Estimation)**.** *Let $U \in \mathbb{C}^{N\times N}$ be a unitary matrix, $|u\rangle$ one of its eigenvector with eigenvalue $e^{2\pi i\lambda}$, and $d$ the number of bits we wish to use to represent its eigenvalue. The QPE algorithm takes as input the unitary gate representing $U$ together with the state $|0\rangle|\psi\rangle$ and $d$, and outputs the state $|\tilde{\lambda}\rangle|\psi\rangle$. Note here the use of $\tilde{\lambda}$ instead of $\lambda$ since $\tilde{\lambda}$ will be a $d$-bits approximation to $\lambda$.*

An important tool that is needed to perform the QPE is the (inverse) Quantum Fourier Transform (QFT), which we define below.

**Definition 4.2** (Quantum Fourier Transform)**.** *Let $|0\rangle, \dots |N-1\rangle$ be an orthonormal basis. The QFT is defined as a linear operator having the following action on the basis states*

$$|j\rangle \longrightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi ijk/N} |k\rangle \tag{21}$$

*The inverse operator, called the inverse Quantum Fourier Transform, has the following action on the basis states*

$$|j\rangle \longrightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{-2\pi ijk/N} |k\rangle. \tag{22}$$

The circuit representation of the QPE is shown in Figure 1.

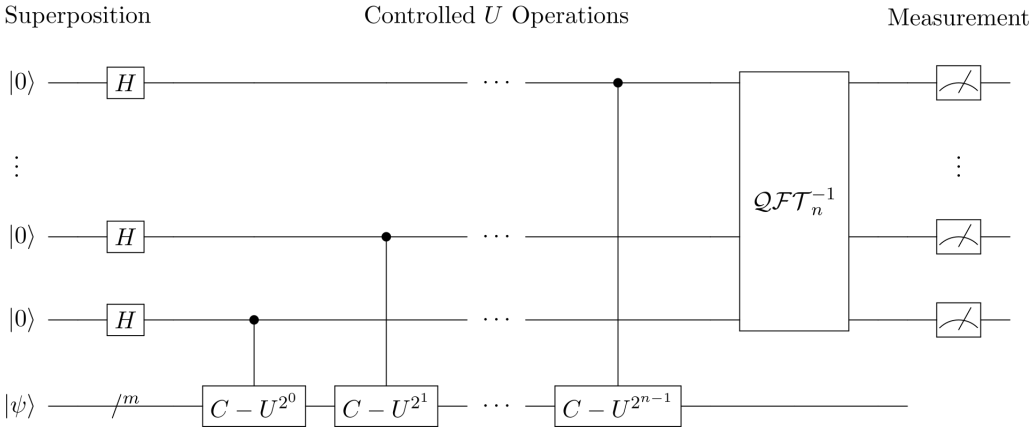


Figure 1: Circuit for the Quantum Phase Estimation

The unitary we apply in HHL for the quantum phase estimation is $U = e^{iAt}$. Since $A$ has spectral decomposition

$$A = \sum_j \lambda_j \left|u_j\right\rangle \left\langle u_j\right|, \tag{23}$$

we can write that

$$e^{iAt} = \sum_j e^{i\lambda_j t} \left|u_j\right\rangle \left\langle u_j\right|. \tag{24}$$

We present below the evolution of a state $\left|0\right\rangle \left|u_j\right\rangle$ throughout the QPE algorithm.

1. Compute a superposition of states in register containing $d$ bits using Hadamard gates:

$$\left|0\right\rangle \left|u_j\right\rangle \longrightarrow \frac{1}{\sqrt{2^d}} \sum_{\tau=0}^{2^d-1} \left|\tau\right\rangle \left|u_j\right\rangle \tag{25}$$

2. Apply $U$ in successive powers of 2, leading to state

$$\frac{1}{\sqrt{2^d}} \sum_{\tau=0}^{2^d-1} \left|\tau\right\rangle U^\tau \left|u_j\right\rangle = \frac{1}{\sqrt{2^d}} \sum_{\tau=0}^{2^d-1} \exp\left(i\lambda_j t\tau\right) \left|\tau\right\rangle \left|u_j\right\rangle. \tag{26}$$

3. Apply the inverse Fourier transform, that is that we do the mapping $\left|\tau\right\rangle \to \frac{1}{\sqrt{2^d}} \sum_{k=0}^{2^d-1} e^{-2\pi i k\tau/2^d} \left|k\right\rangle$. So our state becomes

$$\frac{1}{2^d} \sum_{\tau=0}^{2^d-1} e^{i\lambda_j t\tau} \sum_{k=0}^{2^d-1} e^{\frac{-2\pi i k\tau}{2^d}} \left|k\right\rangle \left|u_j\right\rangle = \frac{1}{2^d} \sum_{k=0}^{2^d-1} \left( \sum_{\tau=0}^{2^d-1} e^{2\pi i\tau\left(\frac{\lambda_j t}{2\pi} - \frac{k}{2^d}\right)} \right) \left|k\right\rangle \left|u_j\right\rangle \tag{27}$$

$$= \sum_{k=0}^{2^d-1} \alpha_{k|j} \left|k\right\rangle \left|u_j\right\rangle, \tag{28}$$

where $\alpha_{k|j} = \frac{1}{2^d} \sum_{\tau=0}^{2^d-1} e^{2\pi i\tau\left(\frac{\lambda_j t}{2\pi} - \frac{k}{2^d}\right)}$ is the probability amplitude of $\left|k\right\rangle$.

4. Measure the $d$-bits register. We denote here $\tilde{\lambda_j}$ the integer in $\{0, 1, \ldots, 2^d - 1\}$ such that $\tilde{\lambda_j}/2^d$ is the best $d$-bit approximation to $\frac{\lambda_j t}{2\pi}$. Assuming that $\frac{\lambda_j t}{2\pi}$ can be exactly represented with $d$ bits, then we are sure to measure it since

$$\left| \langle \tilde{\lambda_j} | \frac{1}{2^d} \sum_{k=0}^{2^d-1} \left( \sum_{\tau=0}^{2^d-1} e^{\frac{2\pi i \tau}{2^d}\left(\frac{\lambda_j t}{2\pi} - \frac{k}{2^d}\right)} \right) |k\rangle \right|^2 = \frac{1}{2^{2d}} \left| \sum_{\tau=0}^{2^d-1} e^{\frac{2\pi i \tau}{2^d}\left(\frac{\lambda_j t}{2\pi} - \frac{\tilde{\lambda_j}}{2^d}\right)} \right|^2 \tag{29}$$

$$= \frac{1}{2^{2d}} \left| \sum_{\tau=0}^{2^d-1} e^{\frac{2\pi i \tau}{2^d}\left(\frac{\lambda_j t}{2\pi} - \frac{\lambda_j t}{2\pi}\right)} \right|^2 \tag{30}$$

$$= \frac{1}{2^{2d}} \left| \sum_{\tau=0}^{2^d-1} e^0 \right|^2 \tag{31}$$

$$= 1. \tag{32}$$

So in this case, we are done with the phase estimation and we are able to recover the eigenvalue as needed, that is we end in state

$$|\tilde{\lambda_j}\rangle |u_j\rangle . \tag{33}$$

## 4.2 Error analysis

In previous section, we presented the procedure and ended it assuming that the eigenvalue can be exactly represented with $d$ bits. However, this may not be the case, and we will show here that even if we do not get the precise output, we can find what value of $d$ to use in order to achieve an error smaller than $\epsilon$. We use proof ideas from [10, 12] and go in detail for every step.

We write $\frac{\lambda_j t}{2\pi} = a/2^d + \delta$, where $a$ is the smallest integer such that $a/2^d$ is the best $d$-bits approximation to $\frac{\lambda_j t}{2\pi}$, and $\delta$ is the rounding error with $0 \leq \delta \leq 2^{-d}$. The probability amplitude of $|(a + k) \bmod 2^d\rangle$ is

$$\alpha_{k|j} = \frac{1}{2^d} \sum_{\tau=0}^{2^d-1} \left[ \exp\left( 2\pi i \left( \frac{\lambda_j t}{2\pi} - \frac{a+k}{2^d} \right) \right) \right]^\tau \tag{34}$$

$$= \frac{1}{2^d} \frac{1 - \exp\left(2\pi i \left(2^d \frac{\lambda_j t}{2\pi} - (a+k)\right)\right)}{1 - \exp\left(2\pi i \left(\frac{\lambda_j t}{2\pi} - \frac{a+k}{2^d}\right)\right)} \tag{35}$$

$$= \frac{1}{2^d} \frac{1 - \exp\left(2\pi i \left(2^d \delta - k\right)\right)}{1 - \exp\left(2\pi i \left(\delta - \frac{k}{2^d}\right)\right)}. \tag{36}$$

We can now bound the probability of getting an error greater than $e/2^d$, that is getting a result $m$ such that $|m - a| > e$. The probability of getting such an $m$ is given by summing over the probability amplitudes of states $|k\rangle$ such that $|k| \geq e$. Remember that $k$ takes values in $\{0, 1, \ldots, 2^d - 1\}$, but here we will take $k$ in $\{-2^{d-1}, \ldots, 2^{d-1} - 1\}$ which is fine since taken mod $2^d$ it gives back the original set of values that $k$ can take. So we have

$$p(|m - a| > e) = \sum_{k=-2^{d-1}}^{-(e+1)} |\alpha_{k|j}|^2 + \sum_{k=e+1}^{2^{d-1}-1} |\alpha_{k|j}|^2, \tag{37}$$

which we will bound here. First, we state a useful lemma.

**Lemma 4.1.** *Let $x \in \mathbb{R}$ such that $-\pi \leq x \leq \pi$. Then*

$$|1 - e^{ix}| \geq \frac{2|x|}{\pi}. \tag{38}$$

*Proof.* We have

$$|1 - e^{ix}| = |1 - (\cos(x) + i\sin(x))| = |(1 - \cos(x)) + i\sin(x)| \tag{39}$$

$$= \sqrt{(1 - \cos(x))^2 + \sin(x)^2} = \sqrt{1 - 2\cos(x) + \cos(x)^2 + \sin(x)^2} \tag{40}$$

$$= \sqrt{2(1 - \cos(x))} = 2\sqrt{\frac{1 - \cos(x)}{2}} = 2\left|\sin\left(\frac{x}{2}\right)\right|. \tag{41}$$

We will now show that $\left|\sin\left(\frac{x}{2}\right)\right| \geq \frac{|x|}{\pi}$ for $-\pi \leq x \leq \pi$. Between 0 and $\pi$, $\sin\left(\frac{x}{2}\right)$ is concave, $\frac{x}{\pi}$ is linear, and they take the same value at the extremums, so $\sin\left(\frac{x}{2}\right) \geq \frac{x}{\pi}$. Remembering that $\sin(x)$ and $x$ are both odd funtions, by symmetry, we have $-\sin\left(\frac{x}{2}\right) \geq -\frac{x}{\pi}$ for $x$ between $-\pi$ and 0. As such, we have

$$|1 - e^{ix}| = 2\left|\sin\left(\frac{x}{2}\right)\right| \geq \frac{2|x|}{\pi} \tag{42}$$

when $-\pi \leq x \leq \pi$. □

Let us bound $|\alpha_{l|j}|$. By the triangle inequality, $|1 - e^{ix}| \leq 2$, so

$$|\alpha_{k|j}| = \left|\frac{1}{2^d}\frac{1 - e^{2\pi i\left(2^d\delta - k\right)}}{1 - e^{2\pi i\left(\delta - \frac{k}{2^d}\right)}}\right| = \frac{\left|1 - e^{2\pi i\left(2^d\delta - k\right)}\right|}{2^d\left|1 - e^{2\pi i\left(\delta - \frac{k}{2^d}\right)}\right|} \leq \frac{2}{2^d\left|1 - e^{2\pi i\left(\delta - \frac{k}{2^d}\right)}\right|}. \tag{43}$$

By lemma 4.1, we get

$$|\alpha_{k|j}| \leq \frac{2}{2^d\frac{2}{\pi}\left|2\pi i\left(\delta - \frac{k}{2^d}\right)\right|} = \frac{1}{2^{d+1}\left(\delta - \frac{k}{2^d}\right)}. \tag{44}$$

Combining with equation 37, we have

$$p(|m - a| > e) \leq \sum_{k=-2^{d-1}}^{-(e+1)}\left(\frac{1}{2^{d+1}\left(\delta - \frac{k}{2^d}\right)}\right)^2 + \sum_{k=e+1}^{2^{d-1}-1}\left(\frac{1}{2^{d+1}\left(\delta - \frac{k}{2^d}\right)}\right)^2 \tag{45}$$

$$= \sum_{k=-2^{d-1}}^{-(e+1)}\frac{1}{4\left(2^d\delta - k\right)^2} + \sum_{k=e+1}^{2^{d-1}-1}\frac{1}{4\left(2^d\delta - k\right)^2}. \tag{46}$$

Recall $0 \leq 2^d\delta \leq 1$, so $\frac{1}{(2^d\delta - k)^2} \leq \frac{1}{k^2}$ when $k$ is negative (the RHS is biggest when the denominator is smallest, which happens when $2^d\delta$ takes its smallest value, which is 0), and $\frac{1}{(2^d\delta - k)^2} \leq \frac{1}{(k-1)^2}$ when $k$ is positive (the RHS is biggest when the denominator is smallest, which happens when $2^d\delta$ takes its biggest value, which is 1). Thus,

$$p(|m-a|>e) \leq \sum_{k=-2^{d-1}}^{-(e+1)} \frac{1}{4k^2} + \sum_{k=e+1}^{2^{d-1}-1} \frac{1}{4\left(k-1\right)^2} \tag{47}$$

$$= \sum_{k=e+1}^{2^{d-1}} \frac{1}{4k^2} + \sum_{k=e+1}^{2^{d-1}-1} \frac{1}{4\left(k-1\right)^2} \tag{48}$$

$$= \sum_{k=e+1}^{2^{d-1}} \frac{1}{4k^2} + \sum_{k=e}^{2^{d-1}-2} \frac{1}{4k^2} \tag{49}$$

$$\leq \sum_{k=e}^{2^{d-1}} \frac{1}{4k^2} + \sum_{k=e}^{2^{d-1}} \frac{1}{4k^2} \tag{50}$$

$$= \frac{1}{2} \sum_{k=e}^{2^{d-1}} \frac{1}{k^2} \tag{51}$$

$$\leq \frac{1}{2} \int_{e-1}^{2^{d-1}} \frac{1}{k^2} dk \tag{52}$$

$$= \frac{1}{2(e-1)} - \frac{1}{2^{d-1}} \tag{53}$$

$$\leq \frac{1}{2(e-1)}. \tag{54}$$

This means that if we want to measure the eigenvalue to accuracy $2^{-n}$, we choose $e = 2^{d-n}-1$. We will at least need $n$ qubits for that, but how many extra ones do we need to make the probability of error arbitrarily small? Writing $d = n + p$, we get that the probability of error is $\frac{1}{2(2^p-2)}$, which means that taking $p = \log\left(\frac{1}{2\epsilon} + 2\right)$ gives use the desired bound. Hence, taking $d = n + \log\left(\frac{1}{2\epsilon} + 2\right)$ qubits will give us the correct measurement to accuracy $2^{-n}$ with probability $1 - \epsilon$.

# 5 Hamiltonian Simulation

A core part of the HHL algorithm depends heavily on how efficiently we can compute $e^{iAt}$, the unitary used in the QPE procedure. In this section we discuss how we can perform this operation, without diving much into the details.

The question of whether we can efficiently simulate Hamiltonians is fundamental in quantum computing. It was shown in [9] that this can be done for $k$-local Hamiltonians, i.e. a sum of polynomially many (w.r.t the number of qubits) Hamiltonians that each act on a constant number of qubits, so $k = \mathcal{O}(1)$.

We review here various methods for simulating Hamiltonians, and we will focus particularly on sparse Hamiltonians simulation, as is the case in the HHL algorithm.

There exists multiple schemes for simulating sparse Hamiltonian efficiently, and we use here one presented in [1].

There are 3 main steps in order to simulate $e^{iAt}$:

1. Find a decomposition of $A$ into a sum of Hamiltonians $A = \sum_i H_i$ such that each $H_i$ is 1-sparse.
2. Compute efficiently $e^{iH_i t}$
3. Combine all the $e^{iH_i t}$ using the Lie-Product formula and get a good approximation of $e^{iAt}$

We review the first and third steps in the following sections. For efficiently simulating 1-sparse Hamiltonians, details can be found in chapter 4 of [4].

## 5.1 Trotter-Suzuki methods

These methods are very useful for $k$-local Hamiltonians. Indeed, when we are given a Hamiltonian acting on $n$ qubits

$$\hat{H} = \sum_{i=1}^{L} H_i, \tag{55}$$

where each $H_i$ acts on at most $k = \mathcal{O}(1)$ qubits, and $L$ is polynomial in $n$, it may be hard to compute $e^{iHt}$. However, it is much simpler to approximate $e^{iH_it}$ as it acts on a much smaller number of qubits. But now the question is, how can we reconstruct $e^{iHt}$ from $e^{iH_it}$? In general, the $H_i$'s do not commute, and as such $e^{iHt} \neq \Pi_{i=1}^{L} e^{iH_it}$.

However, a fundamental result in quantum simulation gives us the following asymptotic approximation:

**Theorem 5.1** (Trotter formula)**.** *Let $A$ and $B$ be Hermitian operators. Then for any real $t$*

$$\lim_{m\to\infty} \left(e^{iAt/m}e^{iBt/m}\right)^m = e^{i(A+B)t}. \tag{56}$$

It is important to emphasize here that this holds even if $A$ and $B$ do not commute. If we want to restrict the simulation to an error $\epsilon$, then we can "truncate" the Trotter formula after a certain number of iterations $m$. That is,

$$\left\| e^{i(A+B)t} - \left(e^{iAt/m}e^{iBt/m}\right)^m \right\|_2 \leq \epsilon \tag{57}$$

The following lemma from [6] shows us a choice of $m$ in order to get the precision $\epsilon$ desired.

**Lemma 5.2** (Trotter formula)**.** *Let $A$ and $B$ be Hermitian operators, $m \in \mathbb{N}$ and $t \in \mathbb{R}$. Then*

$$\left\| e^{i(A+B)t} - \left(e^{iAt/m}e^{iBt/m}\right)^m \right\|_2 = \mathcal{O}\left(\frac{t^2 \max\left(\|A\|_2, \|B\|_2\right)^2}{m}\right). \tag{58}$$

As such, taking the number of steps $m$ to be $m = \mathcal{O}\left(\frac{t^2 \max(\|A\|_2, \|B\|_2)^2}{\epsilon}\right)$ gives us an error of less than $\epsilon$.

The theorem and lemma above can be generalized to an arbitrary sum of Hamiltonians $\hat{H} = \sum_{i=1}^{L} H_i$, so

$$\lim_{m\to\infty} \left(e^{iH_1t/m}e^{iH_2t/m}\dots e^{iH_Lt/m}\right)^m = e^{i(H_1+H_2+\dots+H_L)t} = e^{i\hat{H}t}. \tag{59}$$

## 5.2 Decomposition into sum of Hamiltonians

The Trotter formula enables us to compute $e^{iHt}$ for a Hamiltonian $H$ when it is given as a sum of efficiently computable local Hamiltonians. But how to perform the decomposition? We use the technique presented in [1], which finds a decomposition of a Hamiltonian based on ideas from graph theory. Before explaining it, we first introduce some definitions relevant in our discussion.

**Definition 5.1** (Graph)**.** *A graph is a pair $G = (V, E)$, where $V$ is a set whose elements are called vertices, and $E$ is a set of unordered pairs of vertices, whose elements are called edges.*

**Example 5.1.** *An example of a graph is shown in figure 2. Here, $V = \{1, 2, \dots, 7\}$ and $E = \{(1,2), (2,3), (3,4), (4,5), (5,6), (2,6), (6,7), (2,7)\}$.*

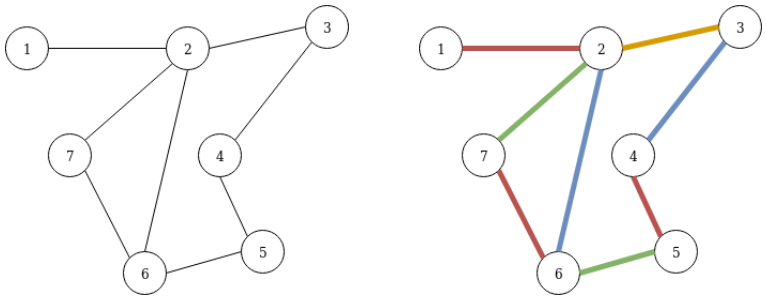


Figure 2: An example of a graph $G$ together with a possible proper $k$-edge coloring of it for $k = 4$

**Definition 5.2** (Adjacency matrix)**.** *Let $G = (V, E)$ be a graph such that $|V| = n$. The adjacency matrix $A$ of $G$ is a $n \times n$ symmetric matrix whose elements are defined as:*

$$A_{ij} = \begin{cases} 1 & \text{if } i, j \in V \text{ and } (i, j) \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{60}$$

*That is, $A_{ij} = 1$ if vertices $i$ and $j$ are connected, and $A_{ij} = 0$ otherwise.*

**Example 5.2.** *The adjacency matrix for the graph in figure 2 is*

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{61}$$

**Definition 5.3** (Graph coloring)**.** *Let $G = (V, E)$ be a graph. An $k$-edge coloring of $G$ is a proper coloring of the edges, meaning an assignment $f : E \to \{1, 2, \ldots, k\}$ of colors to edges so that no vertex is incident to two edges of the same color.*

We also state the following theorem from [8] about graph coloring:

**Theorem 5.3** (Vizing's theorem)**.** *Let $G = (V, E)$ be a graph with maximum degree $d$, i.e. any vertex can be connected to at most $d$ vertices. Then there exists an $k$-edge colouring of $G$ with $k$ at most $d + 1$.*

Let us see how one can decompose a sparse Hamiltonian into a sum of efficiently computable ones.

Since a Hamiltonian can be represented as a square matrix, we can think of it as an adjacency matrix and thus associate a graph to it. In this case, we will consider an adjacency matrix $A$ such that

$$A_{ij} = \begin{cases} 1 & \text{if } H_{ij} \neq 0, \\ 0 & \text{if } H_{ij} = 0. \end{cases} \tag{62}$$

We can do the following:

1. Compute a $k$-edge coloring of the graph.

2. Consider the subgraphs $A_c$ induced by each of the colors. This gives us a decomposition of the adjacency matrix as

$$A = \sum_{c=1}^{k} A_c. \tag{63}$$

*Remark.* This decomposition is indeed good for two reasons. First, by Vizing's theorem, if the Hamiltonian is $s$-sparse, then the number of colors needed, i.e. the number of terms in the sum, will be $\leq s+1$. Second, one can observe that each subgraph induced by a color is just composed of vertices that are connected to at most 1 other vertex. As such the adjacency matrix of each subgraph will be symmetric, 1-sparse, and so we can efficiently simulate them individually.

# 6 Controlled Rotations for Eigenvalues Inversion

Recall that after the Quantum Phase Estimation step of the HHL algorithm, we must perform controlled rotations in order to compute the inverse of the eigenvalues previously computed. More specifically, after the QPE, we are in state

$$\sum_{j=0}^{N-1} \beta_j \left|\tilde{\lambda_j}\right\rangle \left|u_j\right\rangle \tag{64}$$

if it was performed without any errors, where $\tilde{\lambda_j}$ is such that $\tilde{\lambda_j}/2^d$ is the best $d$-bit approximation to $\frac{\lambda_j t}{2\pi}$. After appending an ancilla qubit, the controlled rotations is what allow us to end up in state

$$\sum_{j=0}^{N-1} \beta_j \left|\tilde{\lambda_j}\right\rangle \left|u_j\right\rangle \left(\sqrt{1 - \frac{C^2}{\tilde{\lambda_j}^2}} \left|0\right\rangle + \frac{C}{\tilde{\lambda_j}} \left|1\right\rangle\right). \tag{65}$$

We state the following lemma from [6] which state that using controlled rotations can bring the system to the desired state.

**Lemma 6.1.** *Let $\lambda \in \mathbb{R}$ and $\tilde{\lambda}$ be its $d$-bits finite representation. Then there is a unitary $U_\lambda$ that acts as*

$$U_\lambda : \left|\tilde{\lambda}\right\rangle \left|0\right\rangle \mapsto \left|\tilde{\lambda}\right\rangle \left(\cos(\tilde{\lambda}) \left|0\right\rangle + \sin(\tilde{\lambda}) \left|1\right\rangle\right). \tag{66}$$

It turns out that the operator that can do this is the controlled $Y$ rotation of angle $2\tilde{\lambda}$, where a $Y$ rotation of angle $\theta$ is defined as $R_Y(\theta) = e^{-i\theta\sigma_y/2} = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$. Here, $\sigma_y$ is the Pauli $Y$ matrix.

So this means that in the HHL algorithm, performing a controlled $Y$ rotation of angle $2\arcsin\left(\frac{C}{\tilde{\lambda}}\right)$ gives us mapping that computes

$$\left|\tilde{\lambda}\right\rangle \left|0\right\rangle \mapsto \left|\tilde{\lambda}\right\rangle \left(\cos(\arcsin(C/\tilde{\lambda})) \left|0\right\rangle + \sin(\arcsin(C/\tilde{\lambda})) \left|1\right\rangle\right) \tag{67}$$

which is exactly the mapping

$$\left|\tilde{\lambda}\right\rangle \left|0\right\rangle \mapsto \left|\tilde{\lambda}\right\rangle \left(\sqrt{1 - \frac{C^2}{\tilde{\lambda}^2}} \left|0\right\rangle + \frac{C}{\tilde{\lambda}} \left|1\right\rangle\right) \tag{68}$$

desired.

There is a detail we omitted above which is very important for the implementation of a circuit. The controlled rotation can be easily done when we know the value of the angle as we described before. The problem here is that our angle is a function of $\tilde{\lambda_j}$, which is stored in the circuit after the Quantum Phase Estimation procedure. Thus, we can not simply do $Y$ rotations of angles depending on the eigenvalues as we don't know them a priori. We explain here how the desired state can still be reached by using a procedure known as Polynomial State Preparation, presented in [14, 15].

## 6.1 Polynomial State Preparation

First we define exactly the mapping done by a controlled $Y$ rotation.

**Definition 6.1.** *A controlled $R_Y(\theta)$ rotation with control qubit $|q\rangle$ and target qubit $|0\rangle$ is a gate that performs:*

$$|q\rangle|0\rangle \mapsto |q\rangle e^{-iq\theta\sigma_y/2}|0\rangle = |q\rangle\left(\cos(q\theta/2)|0\rangle + \sin(q\theta/2)|1\rangle\right). \tag{69}$$

*In the same way, a multi-controlled $R_Y(\theta)$ rotation with control qubits $|q_0\rangle, |q_1\rangle, \ldots, |q_k\rangle$ and target qubit $|0\rangle$ is a gate that performs:*

$$|q_0\rangle|q_1\rangle\ldots|q_k\rangle|0\rangle \mapsto |q_0\rangle|q_1\rangle\ldots|q_k\rangle e^{-iq_0q_1\ldots q_k\theta\sigma_y/2}|0\rangle. \tag{70}$$

Observe that the action of performing successive controlled $Y$ rotations can be also written in a compact manner. For example, a controlled $Y$ rotation of angle $\theta_1$ controlled by qubit $q_0$ followed by a mutli-controlled $Y$ rotation of angle $\theta_2$ controlled by qubit $q_0$ and $q_1$ can be written as

$$|q_0\rangle|q_1\rangle|0\rangle \mapsto |q_0\rangle|q_1\rangle e^{-i(q_0q_1\theta_2+q_0\theta_1)\sigma_y/2}|0\rangle. \tag{71}$$

Note how $q_0q_1\theta_2 + q_0\theta_1 = p(q)$ is a polynomial, and $q_1q_0$ is the binary representation of $q$. This shows that by choosing appropriate $\theta_i$'s, we could implement arbitrary polynomials of $q$.

To make things, clearer, we show an example of implementation for a simple case.

**Example 6.1.** *Suppose that we have two qubits $q_0$, $q_1$ as before and an ancilla qubit, and we want to perform a rotation that does the following mapping*

$$|q_0\rangle|q_1\rangle|0\rangle \mapsto |q_0\rangle|q_1\rangle e^{-ip(q)\sigma_y/2}|0\rangle, \tag{72}$$

*where $p(x) = c_0 + c_1x + c_2x^2$. Here, we have $x = 2q_1 + q_0$, which is the binary expansion of $q$. Replacing in $p(x)$ and remembering that $q_i^2 = q_i$ since these are bits, we have*

$$\begin{aligned} p(q) = p(2q_1+q_0) &= c_0 + c_1(2q_1+q_0) + c_2(2q_1+q_0)^2 && (73)\\ &= c_0 + c_1(2q_1+q_0) + c_2(4q_1 + 4q_1q_0 + q_0) && (74)\\ &= c_0 + q_0(c_1+c_2) + q_1(2c_1+4c_2) + q_0q_1 4c_2 && (75)\end{aligned}$$

*which gives us a way to implement it with controlled $Y$ rotations. Indeed, the circuit can be represented as shown in Figure 3.*

The example can of course be generalized to implement polynomials of degree higher than two, and for more than just two qubits. In the case where we have $d$ qubits and a polynomial of degree $k$, we will have

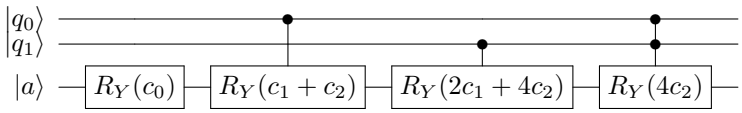


Figure 3: Diagram of circuit with controlled $Y$ rotations for the example polynomial

$$p(q) = p\left(\sum_{i=0}^{d-1} 2^i q_i\right) = \sum_{j=0}^{k} a_j \left(\sum_{i=0}^{d-1} 2^i q_i\right)^i, \tag{76}$$

and this can also be implemented with the help of controlled $Y$ rotations.

In fact, one can perform many functions $f(x) : \mathbb{R} \mapsto \mathbb{R}$ using a properly chosen polynomial $p(q)$ so that the values of $p(q) = f(q)$ for $q \in \{0, \ldots, 2^d - 1\}$, if $q$ is represented using $d$ bits. This would give a way to implement $f(x) = 2\arcsin(C/x)$, the function used in the HHL algorithm, using (multi-)controlled $Y$ rotations.

In the next section, we will see how we can exactly represent functions when there is a $d$-qubit input using the concept of Walsh transform.

## 6.2 Exact Function Representation and Walsh Transform

An arbitrary function $f$ of $d$ qubits can be represented as a polynomial

$$f(q_0, q_1, \ldots, q_{d-1}) = D + D_0 q_0 + D_1 q_1 + \cdots + D_{01} q_0 q_1 + D_{02} q_0 q_2 + \cdots + D_{01\ldots d-1} q_0 q_1 \ldots q_{d-1}, \tag{77}$$

which is the sum of all subsets of the input qubits. Each subset $S_i$ of size $i$ has a corresponding coefficient $D$. Note that later, we will refer to $D(S_i)$ as the coefficient corresponding to subset $S_i$. We want to be able to choose the coefficients so that when evaluating $f(q_0, q_1, \ldots, q_{d-1})$, we get the true value of $f$, that is $f\left(\sum_{i=0}^{d-1} 2^i q_i\right)$.

We claim (see lemma 6.2) that $f$ can also be represented as

$$\begin{aligned} f(q_0, q_1, \ldots, q_{d-1}) = C + C_0(-1)^{q_0} + C_1(-1)^{q_1} + \cdots + C_{01}(-1)^{q_0+q_1} \\ + C_{02}(-1)^{q_0+q_2} + \cdots + C_{01\ldots d-1}(-1)^{q_0+q_1+\cdots+q_{d-1}} \end{aligned} \tag{78}$$

for appropriate choices of $C$'s.

In order to make the notation more compact, we will represent the qubits $q_0, q_1, \ldots, q_{d-1}$ as a vector $\mathbf{q} \in \{0,1\}^d$. As such we can now write (78) as

$$f(\mathbf{q}) = \sum_{\mathbf{b} \in \{0,1\}^d} C(\mathbf{b})(-1)^{\langle \mathbf{q}, \mathbf{b} \rangle}. \tag{79}$$

**Example 6.2.** *For the case $d = 2$, we get*

$$f(\mathbf{q}) = f\left(\begin{bmatrix} q_0 \\ q_1 \end{bmatrix}\right) = C\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) + C\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}\right)(-1)^{q_0} + C\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix}\right)(-1)^{q_1} + C\left(\begin{bmatrix} 1 \\ 1 \end{bmatrix}\right)(-1)^{q_0+q_1}. \tag{80}$$

It is good to recall what the overall goal is. We would like to compute a function $f : \mathbb{R} \to \mathbb{R}$ with inputs restricted to $\{0, 1, \ldots, 2^d - 1\}$. Moreover, as shown in (84), we want to find a representation of $f(\mathbf{q})$ such that we can implement it using $Y$ rotations. At the same time, observe that we know the actual value of $f(\mathbf{q})$ since we can compute it as $f\left(\sum_{i=0}^{d-1} 2^i q_i\right)$. So

really, the aim is to find what the values of the coefficients $C(\mathbf{b})$, from which we can find the coefficients $D(S_i)$ which correspond to the polynomial representation of the function. From there, we can use the Polynomial State Preparation ideas presented in previous section in order to choose appropriate controlled $Y$ rotations for the implementation.

### 6.2.1 Walsh Transform

It turns out that there is a simple equation that relates $f(\mathbf{q})$ to the coefficients $C(\mathbf{b})$. Equation (84) revealed how to compute $f$ from the $C$'s, and we can in fact compute the $C$'s from $f$ in a similar fashion. The two entities are related through what is called the Walsh Transform, which is the analogue of the Fourier Transform but for boolean functions. We can find the values of the coefficients as

$$C(\mathbf{b}) = \frac{1}{2^n} \sum_{\mathbf{q}\in\{0,1\}^d} f(\mathbf{q})(-1)^{\langle \mathbf{q},\mathbf{b}\rangle}. \tag{81}$$

as described in [11]. Note that we know the actual value of $f(\mathbf{q})$ which is $f\left(\sum_{i=0}^{d-1} 2^i q_i\right)$, so we can compute all the values $C(\mathbf{b})$.

### 6.2.2 Computing the right coefficients

In fact, we are interested in computing the coefficients $D(S_i)$ in order to implement using controlled $Y$ rotations. But how do the coefficients $C(\mathbf{b})$ relate to coefficients $D(S_i)$?

We have the following.

**Lemma 6.2.** *Let $S_i$ be a subset of size $i$ of the qubits, which we represent with a set of indices, i.e. $S_i = \{k_0, k_1, \ldots, k_{i-1}\}$. Then one can compute the corresponding coefficient $D(S_i)$ in equation (77) using*

$$D(S_i) = (-2)^i \sum_{\mathbf{b}\in\{0,1\}^d} C(\mathbf{b}) \prod_{j=0}^{i-1} b_{k_j}, \tag{82}$$

*where $b_i$ is the $i^{th}$ component of vector $\mathbf{b}$, and $C(\mathbf{b})$ is as defined in equation (81).*

*Proof.* First note that we can rewrite (77) as a sum over all subsets of qubits, each subset having a "weight" equals to the corresponding coefficient, which we want to compute. So we can write $f$ as

$$f(\mathbf{q}) = \sum_{i=0}^{d} \sum_{S_i} D(S_i) \prod_{j=0}^{i-1} q_{k_j}, \tag{83}$$

where again here, $S_i = \{k_0, k_1, \ldots, k_{i-1}\}$ is a set of $i$ indices, representing a subset of the qubits, and $D(S_i)$ is the corresponding coefficient.

On the other hand, we have the representation of $f$ from (84) with

$$f(\mathbf{q}) = \sum_{\mathbf{b}\in\{0,1\}^d} C(\mathbf{b})(-1)^{\langle \mathbf{q},\mathbf{b}\rangle}. \tag{84}$$

We expand this formula to turn it into something of the form (83). So we have

$$f(\mathbf{q}) = \sum_{\mathbf{b}\in\{0,1\}^d} C(\mathbf{b})(-1)^{\langle \mathbf{q},\mathbf{b}\rangle} \tag{85}$$

$$= \sum_{\mathbf{b}\in\{0,1\}^d} C(\mathbf{b})(-1)^{b_0 q_0 + b_1 q_1 + \cdots + b_{d-1} q_{d-1}}. \tag{86}$$

Since we have qubits, we can write $(-1)^{q_i b_i} = 1 - 2q_i b_i$. We get

$$\sum_{\mathbf{b}\in\{0,1\}^d} C(\mathbf{b})(-1)^{b_0 q_0 + b_1 q_1 + \cdots + b_{d-1} q_{d-1}} = \sum_{\mathbf{b}\in\{0,1\}^d} C(\mathbf{b})(1-2b_0 q_0)(1-2b_1 q_1)\ldots(1-2b_{d-1}q_{d-1}). \tag{87}$$

We can then use a generalized form of the binomial expansion, in the sense that

$$(1+y_0)(1+y_1)\ldots(1+y_{d-1}) = \sum_{i=0}^{d}\sum_{S_i}\prod_{j=0}^{i-1} y_{k_j}, \tag{88}$$

with $S_i$ still a subset of $i$ indices. In our case, we get

$$\sum_{\mathbf{b}\in\{0,1\}^d} C(\mathbf{b})(1-2b_0 q_0)\ldots(1-2b_{d-1}q_{d-1}) = \sum_{\mathbf{b}\in\{0,1\}^d} C(\mathbf{b}) \sum_{i=0}^{d}\sum_{S_i}\prod_{j=0}^{i-1}(-2q_{k_j}b_{k_j}) \tag{89}$$

$$= \sum_{i=0}^{d}\sum_{S_i}(-2)^i \sum_{\mathbf{b}\in\{0,1\}^d} C(\mathbf{b}) \prod_{j=0}^{i-1}(q_{k_j}b_{k_j}) \tag{90}$$

$$= \sum_{i=0}^{d}\sum_{S_i}\left((-2)^i \sum_{\mathbf{b}\in\{0,1\}^d} C(\mathbf{b}) \prod_{j=0}^{i-1} b_{k_j}\right)\prod_{j=0}^{i-1} q_{k_j} \tag{91}$$

$$= \sum_{i=0}^{d}\sum_{S_i} D(S_i) \prod_{j=0}^{i-1} q_{k_j}, \tag{92}$$

where we defined $D(S_i) = (-2)^i \sum_{\mathbf{b}\in\{0,1\}^d} C(\mathbf{b}) \prod_{j=0}^{i-1} b_{k_j}$. □

In summary, the implementation of a function $f$ can be done using controlled $Y$ rotations using the following steps:

1. Compute the coefficients $C(\mathbf{b})$ using equation (81).
2. Compute the coefficients $D(S_i)$ corresponding to some subset of size $i$ of the qubits using the previously computed coefficients $C$ through the formula of lemma 6.2, for all $i$.
3. Each subset of qubits of size $i$ has now a corresponding coefficient $D(S_i)$ for all $i$. So in the circuit, use a $R_Y(D(S_i))$ gate controlled by qubits with indices in $S_i$ and do this for all subset $S_i$ and for all $i$.

## 6.3 Gate Analysis

We present here an analysis for the number of gates needed to realize the controlled $Y$ rotations. First, we present a way to build arbitrary multi-controlled $Y$ rotation gates, and then assuming a set of basic gates at our disposal, we describe the number of such gates needed for computing exactly an arbitrary function of $d$ bits.

### 6.3.1 Implementation of multi-controlled $Y$ rotations

In this section we present circuits to build controlled $Y$ rotation. There is in fact a compact recursive construction of such circuits, which we decide to present here.

The circuit for a single control qubit is shown in figure 4. The basic idea comes from the fact that $XR_Y(\theta)X = R_Y(-\theta)$. As such, when the control qubit is set to $|0\rangle$, the we apply two

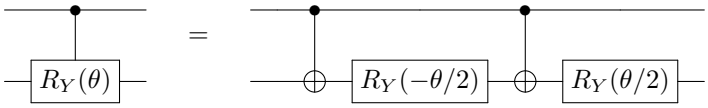


Figure 4: Diagram of circuit for a controlled $Y$ rotation

rotations of opposite angles, resulting in no rotation at all, and when the control qubit is set to $|0\rangle$, we will apply two rotations of angle $\theta/2$, resulting in an overall rotation of $\theta$.

The circuit can be simply extended to have multiple qubits as control. We depict the case $d = 2$ controls in Figure 5 and the general case in Figure 6.

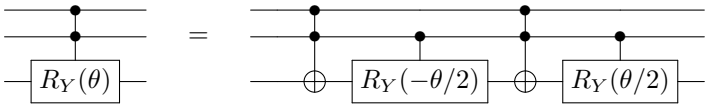


Figure 5: Diagram of circuit for a 2-controlled $Y$ rotation

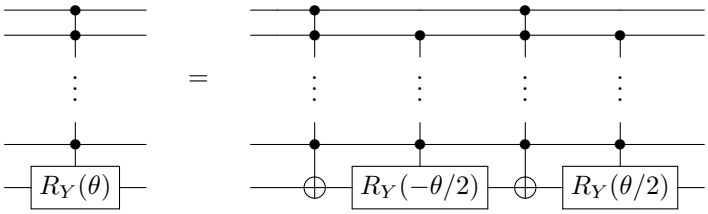


Figure 6: Circuit for a $d$-controlled $Y$ rotation

Observe that a rotation with $d$ controls involves two rotations with $d-1$ controls together with two $d$-controlled Toffoli gates. Implementing the $d$-controlled gate recursively from the basic circuit would give us exponentially many gates in $d$, but we can do much better than that provided we use "work" qubits. Indeed, as presented in Figure 4.10 in [10] and shown here in Figure 7, the number of gates is greatly reduced.

We make the assumption that adding those extra qubits is not a problem, and continue with the complexity analysis.

### 6.3.2 Analysis

Suppose that the set of gates that can be used is $\{\text{CNOT}, \text{Toffoli}, R_Y(\cdot)\}$. Then we have the following:

**Lemma 6.3.** *One needs $\mathcal{O}(d)$ Toffoli gates, $\mathcal{O}(1)$ CNOTs, $\mathcal{O}(1)$ $R_Y(\cdot)$ gates, and $d-1$ additional "work" qubits to implement a $d$-controlled $Y$ rotation.*

*Proof.* From Figure 4, the use of 2 CNOTs and 2 $R_Y(\cdot)$ gates is needed to build a controlled $Y$ rotation. For an arbitrary $d$-controlled rotation, following the construction from [10], we need $d-1$ additional qubits, $2(d-1)$ Toffoli gates, and the 2 CNOTs and 2 $R_Y(\cdot)$ for the simple controlled $Y$. □

Now, for arbitrary functions, recall from section 6.2 that in the worst-case, we need rotations for each combination of the input qubits. Before deriving results about the complexity, we state a useful lemma.

**Lemma 6.4.** *Let $d$ be a positive integer. Then $\sum_{i=0}^{d} \binom{d}{i} = 2^d$ and $\sum_{i=0}^{d} \binom{d}{i} i = d2^{d-1}$.*

*Proof.* Write the binomial expansion formula

$$(x+1)^d = \sum_{i=0}^{d} \binom{d}{i} x^i. \tag{93}$$

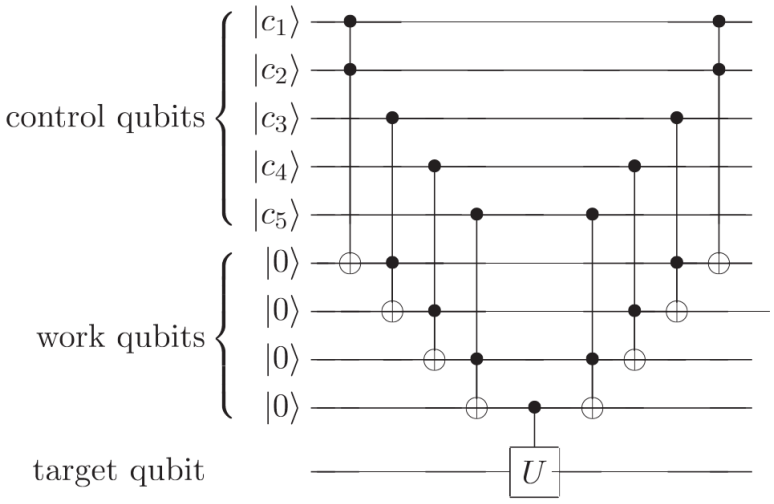


Figure 7: Circuit for an arbitrary 5-controlled $U$ operation

Replacing $x$ by 1 gives us the first result. For the second one, differentiating (93) with respect to $x$ on both sides gives

$$d(x+1)^{d-1} = \sum_{i=0}^{d} \binom{d}{i} i x^{i-1}. \tag{94}$$

Once again, setting $x = 1$ proves the equality.

□

**Lemma 6.5.** *One needs $\mathcal{O}(d2^d)$ Toffoli gates, $\mathcal{O}(2^{d+1})$ CNOTs, $\mathcal{O}(2^{d+1})$ $R_Y(\cdot)$ gates, and $d-1$ additional "work" qubits to implement exactly an arbitrary function of $d$ qubits.*

*Proof.* There are $\binom{d}{i}$ combination of $i$ qubits from a set of $d$ qubits. This gives us the number of rotations controlled by $i$ qubits needed for the implementation of a function of $d$ qubits. We need then to sum on $i$ from 0 to $d$. We get from lemma 6.3

$$\text{Overall complexity } = \sum_{i=0}^{d} \binom{d}{i} \cdot \text{Complexity}(i\text{-controlled } Y \text{ rotation}) \tag{95}$$

$$= \sum_{i=0}^{d} \binom{d}{i} \cdot \left(2(i-1)\text{Toffoli} + 2\text{CNOTs} + 2R_Y(\cdot)\right). \tag{96}$$

We thus find the answer for the different gates in our set separatly. For the Toffoli gates, the exact number needed is

$$\sum_{i=0}^{d} \binom{d}{i} 2(i-1) = \sum_{i=0}^{d} \binom{d}{i} i - 2\sum_{i=0}^{d} \binom{d}{i} = d2^{d-1} - 2^d = 2^{d-1}(d-2). \tag{97}$$

For the CNOTs and $R_Y(\cdot)$, the exact number needed is

$$\sum_{i=0}^{d} \binom{d}{i} 2 = 2 \cdot 2^d = 2^{d+1}. \tag{98}$$

Note that since the rotations are performed one after the other, we can always reuse the same $d-1$ "work" qubits. □

This seems like bad news if we want to compute the function exactly. However one could be content with the use of a polynomial of lower degree that approximate the function. In that case, the complexity slightly changes and we get the following.

**Lemma 6.6.** *One needs $\mathcal{O}(kd^k)$ Toffoli gates, $\mathcal{O}(d^k)$ CNOTs, $\mathcal{O}(d^k)$ $R_Y(\cdot)$ gates, and $d-1$ additional "work" qubits to implement a polynomial of degree $k$ on $d$ qubits.*

Before describing the proof, we state a useful lemma.

**Lemma 6.7.** *Let $d, k$ be positive integers such that $k \leq d$. Then $\sum_{i=0}^{k} \binom{d}{i} = \mathcal{O}(d^k)$ and $\sum_{i=0}^{k} \binom{d}{i} i = \mathcal{O}(kd^k)$.*

*Proof.* We use the fact that $\binom{d}{i} \leq d^i$, As such, we get

$$\sum_{i=0}^{k} \binom{d}{i} \leq \sum_{i=0}^{k} d^i = \frac{d^{k+1}-1}{d-1} = \mathcal{O}(d^k) \tag{99}$$

for the first result.
Differentiating with respect to $d$ we have

$$\sum_{i=0}^{k} d^{i-1} i = \mathcal{O}(kd^{k-1}). \tag{100}$$

So we find the second result by using

$$\sum_{i=0}^{k} d^i i = d \sum_{i=0}^{k} d^{i-1} i = \mathcal{O}(kd^k). \tag{101}$$

□

Using these, we can prove lemma 6.6.

*Proof.* Here, we note that the idea is the same as for computing the complexity for an arbitrary function, except that since the degree of the polynomial is $k$, there are no combinations of more than $k$ qubits. So just as before, this involves a summation, but here it is truncated. That is, we have

$$\text{Overall complexity } = \sum_{i=0}^{k} \binom{d}{i} \cdot \text{Complexity}(i\text{-controlled } Y \text{ rotation}) \tag{102}$$

$$= \sum_{i=0}^{k} \binom{d}{i} \cdot \left(2(i-1)\text{Toffoli} + 2\text{CNOTs} + 2R_Y(\cdot)\right). \tag{103}$$

We thus find the answer for the different gates in our set separatly. For the Toffoli gates, the exact number needed is

$$\sum_{i=0}^{k} \binom{d}{i} 2(i-1) = \sum_{i=0}^{k} \binom{d}{i} i - 2 \sum_{i=0}^{k} \binom{d}{i} = \mathcal{O}(kd^k) \tag{104}$$

For the CNOTs and $R_Y(\cdot)$, the exact number needed is

$$\sum_{i=0}^{k} \binom{d}{i} 2 = \mathcal{O}(d^k). \tag{105}$$

Note that since the rotations are performed one after the other, we can always reuse the same $d-1$ "work" qubits. □

One important note to make here, is that if we want to perform all computations exactly throughout the HHL algorithm, there is a hidden exponential complexity. In the overall algorithm, there is a dependency on $\epsilon$ which corresponds to the precision we want. But this $\epsilon$ should be chosen so as to make the Quantum Phase Estimation more accurate, and for that one needs to increase the number of qubits (we called it $d$ in this text) used to represent the eigenvalues of $A$.

The important point is that the complexity of the controlled rotations depends on $d$, and is in fact exponential in $d$ as proved in lemma 6.5 for an exact computation of $2\arcsin(C/x)$. Hence, one should keep in mind these considerations when observing the overall complexity of the HHL algorithm.

# 7 Application of HHL to Various Problems

We now turn to the subject of how the HHL algorithm can be used to solve linear systems of equations in order to solve various problems. It turns out that for some specific matrices, solving the system using the HHL algorithm can allow us to solve the same system but considering it in a finite field for example.

## 7.1 Solving the XORSAT Problem and QLSP over Finite Fields

We present here how to solve a specific problem of satisfiability using the HHL algorithm. We start by giving some definitions.

**Definition 7.1** (Boolean matrices)**.** *A boolean matrix is a matrix whose elements belong to* $\{0, 1\}$*.*

**Definition 7.2** (XOR Satisfiability Problem)**.** *The XOR satisfiability (XORSAT) problem is the problem of determining whether there exists an assignment of n Boolean variables that satisfy m exclusive XOR clauses, where each clause constrains a subset of the variables.*

Note that the XORSAT problem can be viewed as a system of linear equations *mod* 2 on the field $\mathbb{F}_2$, and as such can be solved using Gaussian elimination with $\mathcal{O}(n^3)$ complexity. Of course, one can wonder whether it is possible to achieve a better runtime, and we show here that in some cases, on can solve the XORSAT using the HHL algorithm.

As explained just before, the XORSAT problem can expressed as $A\mathbf{x} = \mathbf{b}$, where $A \in \mathbb{F}_2^{m\times n}$ and $\mathbf{x}, \mathbf{b} \in \mathbb{F}_2^n$. Can we get a solution valid in $\mathbb{F}_2$ but solving the problem in $\mathbb{C}$, like in the setting of the HHL algorithm? If the answer is yes, then this would mean that we could use the HHL algorithm to solve the problem, and get the corresponding speedup.

We first go through some examples to get an idea of what relationship there is between solving a linear system of equations in $\mathbb{F}_2$ versus in $\mathbb{C}$.

**Example 7.1.** *Let $A$ be a boolean matrix and $\boldsymbol{b}$ a boolean vector, defined as*

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

*respectively. Solving $A\mathbf{x} = \mathbf{b}$ in $\mathbb{F}_2$ gives $\mathbf{x} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$, and solving in $\mathbb{C}$ gives $\mathbf{x} = \begin{bmatrix} 2 \\ -1 \\ 0 \\ -1 \end{bmatrix}$, the same solution. So it seems that here, taking the solution in $\mathbb{C}$ mod 2 gives us the corresponding solution in $\mathbb{F}_2$.*

**Example 7.2.** *Let $A$ be a boolean matrix and $\mathbf{b}$ a boolean vector, defined as*

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

*respectively. There is no solution to $A\mathbf{x} = \mathbf{b}$ in $\mathbb{F}_2$, while solving in $\mathbb{C}$ gives $\mathbf{x} = \begin{bmatrix} \frac{1}{2} \\ -1 \\ \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}$. There are cases where even though there is a solution in $\mathbb{C}$, this will be of no help in $\mathbb{F}_2$.*

The corollary below explains the relationship between solving a linear system of equations in $\mathbb{C}$ compared to $\mathbb{F}_2$ where the matrix $A$ has boolean entries. This is derived from a stronger lemma, which we prove next.

**Corollary 7.0.1.** *Let $A \in \mathbb{C}^{n\times n}$ be a boolean matrix and $\mathbf{b} \in \mathbb{C}^n$ a boolean vector such that $det(A) \neq 0$. If $det(A)$ is odd, then we can get a solution to the same problem, but considering the operations of addition and multiplication over $\mathbb{F}_2$, i.e., we can find $\mathbf{x} \in \mathbb{F}_2^n$ such that $A\mathbf{x} = \mathbf{b}$.*

We state the more general form of this result in the following lemma.

**Lemma 7.1.** *Let $A \in \mathbb{C}^{n\times n}$ be matrix and $\mathbf{b} \in \mathbb{C}^n$ a vector both with entries restricted to some finite field $\mathbb{F}_p \subset \mathbb{C}$, and such that $det(A)$ mod $p \neq 0$. Then we can get a solution to the same problem, but considering the operations of addition and multiplication over $\mathbb{F}_p$, i.e., we can find $\mathbf{x} \in \mathbb{F}_p^n$ such that $A\mathbf{x} = \mathbf{b}$.*

*In particular, when $det(A) = 1$, the solution in $\mathbb{F}_p$ is given by taking the solution $\mathbf{x}$ to the problem in $\mathbb{C}$ directly.*

Before proving this lemma we will prove some useful statements.

**Lemma 7.2.** *Let $A \in \mathbb{C}^{n\times n}$ be a matrix with elements restricted to some finite field $\mathbb{F}_p \subset \mathbb{C}$. Then its determinant is an integer, i.e., $det(A) \in \mathbb{Z}$.*

*Proof.* Recall Laplace's formula for the computation of the determinant of a matrix:

$$det(A) = \sum_{j=1}^{n} (-1)^{i+j} A_{ij} M_{ij} \tag{106}$$

for a fixed $1 \leq i \leq n$. Here, $M_{ij}$ corresponds to the determinant of the submatrix of $A$ obtained by removing the $i^{th}$ line and $j^{th}$ column, which is also a boolean matrix. Note that for a $2 \times 2$ matrix $\in \mathbb{F}_p$, its determinant can only be an integer since the operation consists in mutliplying and subtracting integers (since the elements are restricted to $\mathbb{F}_p$). By induction and using Laplace's formula, we get that for an arbitrary $n \times n$ boolean matrix, its determinant is an integer. □

This lemma gives rise to the following corollary.

**Corollary 7.2.1.** *Let $A \in \mathbb{C}^{n\times n}$ be a boolean matrix. Then its determinant is an integer, i.e., $det(A) \in \mathbb{Z}$.*

**Lemma 7.3.** *Let $A \in \mathbb{C}^{n\times n}$ be matrix and $\mathbf{b} \in \mathbb{C}^n$ a vector both with entries restricted to some finite field $\mathbb{F}_p \subset \mathbb{C}$, and such that $det(A)$ mod $p \neq 0$. Then the solution to $A\mathbf{x} = \mathbf{b}$ is such that $\mathbf{x} \in \mathbb{Q}^n$.*

*Proof.* Since the determinant in non-zero, we can solve the system using Cramer's rule. So we find that the $i^{th}$ component of the solution vector $\mathbf{x}$ is given by $\mathbf{x}_i = \frac{det(A_i)}{det(A)}$, where $A_i$ is the matrix obtained by replacing the $i^{th}$ column of $A$ by the vector $\mathbf{b}$. Note that both $A$ and $A_i$ are boolean matrices. As such, from Lemma 7.2, we get that $det(A_i) \in \mathbb{Z}$ and $det(A) \in \mathbb{Z}$. Hence, $\mathbf{x} \in \mathbb{Q}^n$. □

Once again we get a corresponding corollary for boolean matrices.

**Corollary 7.3.1.** *Let $A \in \mathbb{C}^{n\times n}$ be a boolean matrix such that $det(A) \neq 0$, and let $\mathbf{b} \in \mathbb{C}^n$ a boolean vector. Then the solution to $A\mathbf{x} = \mathbf{b}$ is such that $\mathbf{x} \in \mathbb{Q}^n$.*

We can now prove Lemma 7.1 using the lemmas stated above.

*Proof.* (Lemma 7.0.1) Consider the $i^{th}$ of the equation in the linear system. That is

$$\sum_{j=1}^{n} A_{ij}\mathbf{x}_i = \mathbf{b}_i. \tag{107}$$

By Lemma 7.3, we can write it as

$$\sum_{j=1}^{n} A_{ij}\frac{det(A_i)}{det(A)} = \mathbf{b}_i \Leftrightarrow \sum_{j=1}^{n} A_{ij}det(A_i) = det(A)\mathbf{b}_i. \tag{108}$$

Note that in this last equation, both the left-hand and right-hand sides are integers. We can thus take each side *mod* $p$, and we get

$$\sum_{j=1}^{n} A_{ij}det(A_i) \ mod\ p = det(A)\mathbf{b}_i \ mod\ p. \tag{109}$$

But since $A_ij$ and $\mathbf{b}_i$ are both elements of $\mathbb{F}_p$, we can see the sum on the left-hand side also *mod* p and moreover, we have

$$\sum_{j=1}^{n} A_{ij}(det(A_i) \ mod\ p) = (det(A) \ mod\ p)\mathbf{b}_i. \tag{110}$$

If $det(A) \ mod\ p \neq 0$, we have that the system

$$\sum_{j=1}^{n} A_{ij}\frac{det(A_i) \ mod\ p}{det(A) \ mod\ p} = \mathbf{b}_i. \tag{111}$$

is a linear system in $\mathbb{F}_p$, and we see that the solution is given by $\tilde{\mathbf{x}_i} = \frac{det(A_i) \ mod\ p}{det(A) \ mod\ p} \in \mathbb{F}_p$. As such, given the solution $\mathbf{x} = \frac{det(A_i)}{det(A)}$ to the problem in $\mathbb{C}$, we can recover the one in $\mathbb{F}_p$, call it $\tilde{\mathbf{x}}$ by computing $\tilde{\mathbf{x}_i} = \frac{(\mathbf{x}_i \cdot det(A)) \ mod\ p}{det(A) \ mod\ p}$ for all entries $i$ of the vectors, and where the division is in $\mathbb{F}_p$.

So in the specific case where $det(A) = 1$, the solution to the problem in $\mathbb{C}$ coincide with the solution in $\mathbb{F}_p$. □

One can observe from the proof that the only value needed to get the solution of the system in $\mathbb{F}_p$ is the determinant of $A$. This might be hard to compute in general, so this method is not appropriate for every matrix. In the special case where $det(A) = 1$ however, solving the system using the HHL algorithm will give us the answer to the problem in any $\mathbb{F}_p$.

# 8 Implementation

We describe an implementation for a specific case using the Qiskit software development framework. All the code can be found at *https://github.com/Adirlou/epfl_master_semester_project*. In our implementation, we use some of what is presented in [17, 18], which describe implementations of the HHL algorithm on specific $2 \times 2$ and $4 \times 4$ matrices. However, they choose very specific matrices and do multiple approximations in order to reduce the circuit size.

Here, we propose an exact implementation for a particular $2 \times 2$ matrix.

We choose

$$A = \begin{bmatrix} 4 & 1 \\ 1 & 4 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

Solving using usual methods gives us as solution $\mathbf{x} = \begin{bmatrix} 4/15 \\ -1/15 \end{bmatrix}$. Since here, we just want to do a simple simulation, we will choose the number of qubits for the register holding the eigenvalues by finding them first using regular methods. Of course, for a completely proper implementation, one should choose this number according to the maximum value that an eigenvalue of $A$ can take. In our case the eigenvalues are found to be $\lambda_1 = 3$ and $\lambda_1 = 5$. As such, 3 bits are sufficient to represent them, and that is why we chose this number in our implementation.

We show here the different steps used to build the corresponding quantum circuit. To remind the reader of how the circuit looks like, we describe a simplified version of it in figure 8. The circuit is composed of the following qubits: $|b\rangle$ corresponds to $\mathbf{b}$, the $|c_i\rangle$'s hold the eigenvalues, and $|a\rangle$ is the ancilla qubit which is used when performing the controlled $Y$ rotations.

It starts by performing the Quantum Phase estimation (the Quantum Fourier Transform is denoted by $\mathcal{FT}^{-1}$). Then the controlled rotations are represented by the box $C - R_Y$, and the actual circuit can be found in figure 9. Finally we perform the inverse of the Quantum Phase Estimation, and measure the ancilla qubit.

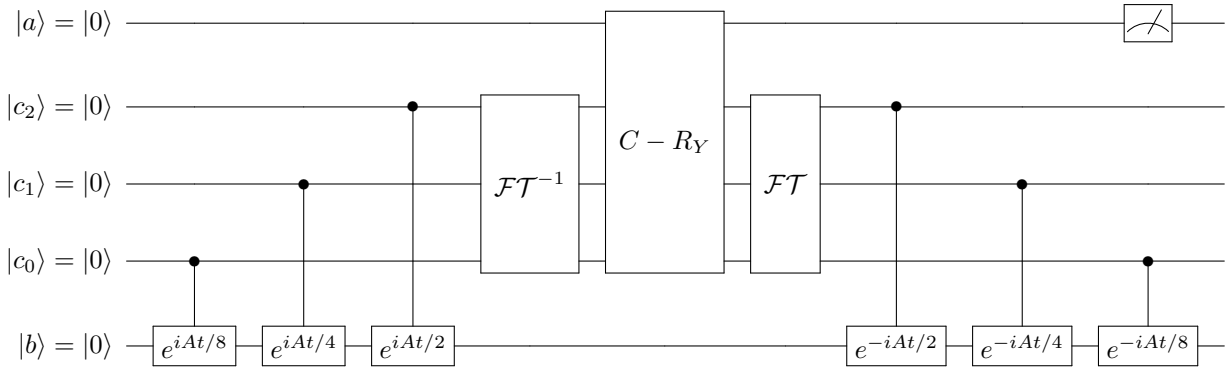


Figure 8: Diagram of circuit for the implementation of the HHL algorithm

## 8.1 Quantum Phase Estimation

The first step is to perform the Quantum Phase Estimation procedure. As shown on the circuit in figure 8, we will need to compute the values $e^{iAt/2^i}$ for $i = 1, 2, 3$. Here, we use tools such as *Wolfram Alpha* to compute the matrix exponential. Also, note that here, we take $t = 2\pi$. We get

$$e^{iAt/2} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}, e^{iAt/4} = \begin{bmatrix} 0 & i \\ i & 0 \end{bmatrix}, e^{iAt/8} = \begin{bmatrix} -1/\sqrt{2} & -i/\sqrt{2} \\ -i/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}, \tag{112}$$

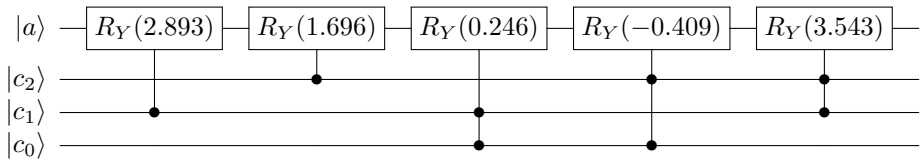


Figure 9: Diagram of circuit with controlled $Y$ rotations for equation (115)

and using the fact that $R_X(\theta) = \begin{bmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$ and $R_Z(\theta) = \begin{bmatrix} \exp(-i\theta/2) & 0 \\ 0 & \exp(i\theta/2) \end{bmatrix}$ we have

$$e^{iAt/2} = R_Z(2\pi), e^{iAt/4} = R_X(-\pi), e^{iAt/8} = R_X(3\pi/2), \tag{113}$$

so the matrix exponentials are just simple $X$ and $Z$ rotations.

## 8.2 Controlled $Y$ Rotations

In this part of the circuit, we would like to perform $Y$ rotations on the ancilla qubit in order to implement a mapping of the form

$$|\tilde{\lambda}\rangle |0\rangle \mapsto |\tilde{\lambda}\rangle \left( \sqrt{1 - \frac{C^2}{\tilde{\lambda}^2}} |0\rangle + \frac{C}{\tilde{\lambda}} |1\rangle \right). \tag{114}$$

Recall that in order to do this, we had to implement the function $f(x) = 2\arcsin(C/x)$. We first need to make a choice for $C$ but recall that we need to restrict it so that $C \leq \min_i |\lambda_i|$, where $\lambda_i$ are the eigenvalues of $A$. So here we take $C = \min_i |\lambda_i| = 3$.

Here we used the following representation for $f$, which needs to return correct values for values $\{3, 4, 5, 6, 7\}$. Indeed, we know the smallest eigenvalue which is 3 and we have 3 qubits, so the highest value we must be able to compute is 7. A possible polynomial representation of $f$ for these inputs is given by

$$f(c_0, c_1, c_2) = 2.893c_1 + 1.696c_2 + 0.246c_0c_1 - 0.409c_0c_2 + 3.543c_1c_2 \tag{115}$$

which can be found using ideas presented in section 6.2. Translating this to controlled $Y$ rotations, it gives rise to the circuit depicted in figure 9.

## 8.3 Results

We present here the results obtained when using the Qiskit package. Recall that the solution of the problem is given by $\mathbf{x} = \begin{bmatrix} 4/15 \\ -1/15 \end{bmatrix}$. However the quantum circuit will output a unit length vector, so $|x\rangle = \begin{bmatrix} 4/\sqrt{17} \\ -1/\sqrt{17} \end{bmatrix}$.

### 8.3.1 Simulator

As expected, from our implementation which should be exact, we are able to compute the amplitudes of the final quantum states, which are depicted in figure 10. Recall that before measuring the ancilla bit, we are in state

$$\sum_{j=0}^{N-1} \beta_j |0\rangle |u_j\rangle \left( \sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \frac{C}{\lambda_j} |1\rangle \right). \tag{116}$$

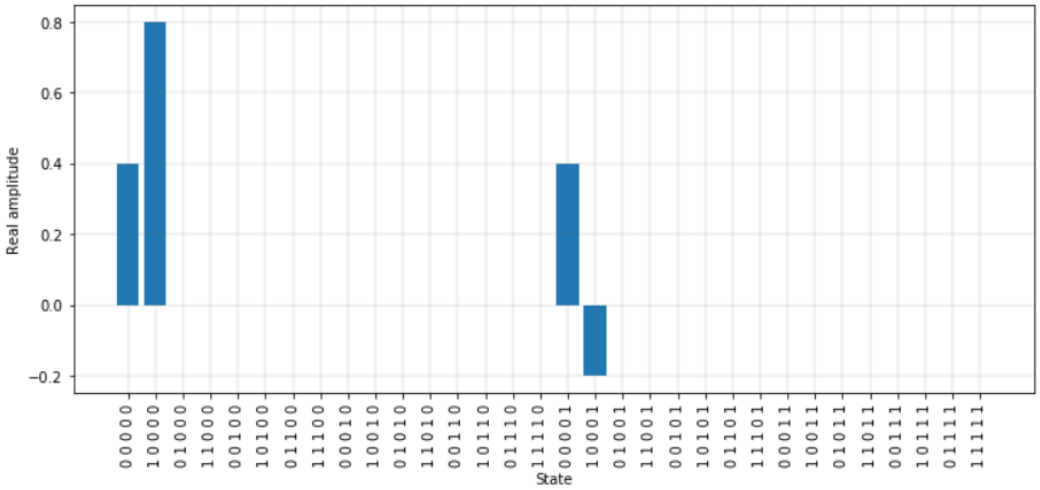


Figure 10: Real part of the amplitudes for all possible final states with simulator

Figure 10 confirms this in the sense that amplitudes greater than zero only for states such that the register containing the eigenvalues is at $|000\rangle$. Then depending on the value of the ancilla, we have different possible amplitudes for the bit representing $|x\rangle$. What we see is that conditioned on the ancilla bit being one, we have a positive amplitude for $|x\rangle$ being zero and a negative one for $|x\rangle$ being one. This is explained by recalling that the output of the quantum circuit, conditioned on the ancilla qubit being one, is $|x\rangle = \begin{bmatrix} 4/\sqrt{17} \\ 1/\sqrt{17} \end{bmatrix}$. In fact, with the conditioning on the ancilla qubit, we retrieve the exact answer in our implementation.

We can also view the result of the circuit in terms of the statistics of the measurements. In that case, we should have good estimations of the probabilities. In our case, again with the conditioning on the ancilla, we should have probability $\left|4/\sqrt{17}\right|^2 \approx 0.941$ of measuring 0 for $|x\rangle$ and probability $\left|-1/\sqrt{17}\right|^2 \approx 0.059$ of measuring 1. Figure 11 shows the results of the measurements of the ancilla qubit and the qubit supposed to contain $|x\rangle$.

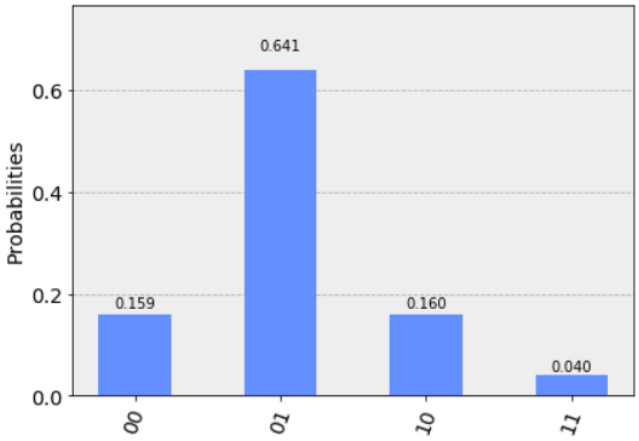


Figure 11: Statistics of measurements when measuring ancilla qubit (top one) and the input qubit (bottom one) with simulator

Once again, rescaling after conditioning on the ancilla qubit being one, we get an estimated probability of $0.641/(0.641 + 0.04) \approx 0.941$ of measuring 0 for $|x\rangle$ and an estimated probability of $0.04/(0.641 + 0.04) \approx 0.059$ of measuring 1, as expected.

### 8.3.2 IBM Quantum Computers

The results we obtain on the IBM quantum computers (more precisely on the one called *ibmq_essex*) are much less precise than on the simulator. This is due to the relatively large depth of the circuit in terms of gates, which incurs a lot of noise that can clearly be seen on the results.

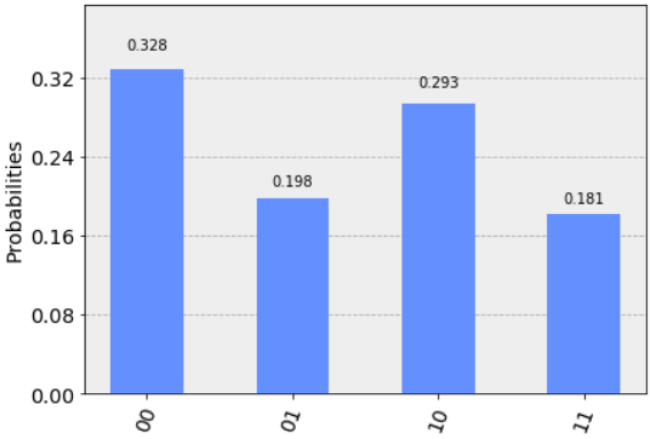


Figure 12: Statistics of measurements when measuring ancilla qubit (top one) and the input qubit (bottom one) with quantum computer

We can compute the probabilities when conditioning on the ancilla qubit being one, and we have estimated probability $0.198/(0.198 + 0.181) \approx 0.522$ of measuring 0 for $|x\rangle$ and probability $0.81/(0.198 + 0.181) \approx 0.478$ of measuring 1, which are not close at all to the desired result.

In fact this can be explained by only computing the Quantum Phase estimation part of the algorithm. Here most of the amplitudes should be on $\lambda_1 = 3$ and $\lambda_2 = 5$. This is the case as can be observed in figure 13, however we note that there is also quite a lot of probability on wrong values. This step of the HHL being critical, it is not a surprise that the noise corrupts the results at the end of circuit.

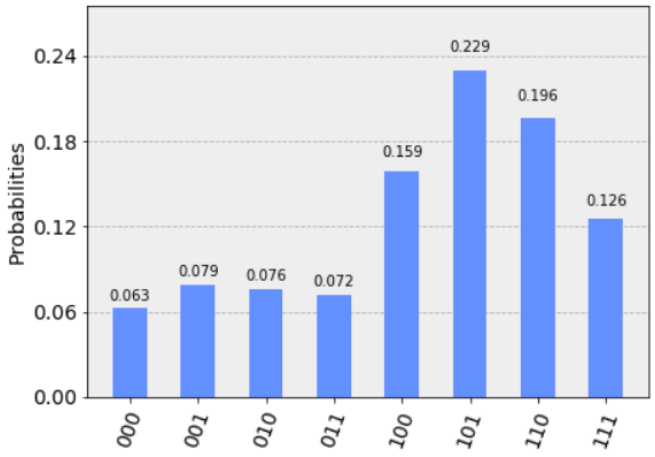


Figure 13: Statistics of measurements when measuring the result of the QPE with quantum computer

We propose then to solve the same problem, but this time using only two bits to hold the eigenvalues in order to make the circuit less deep. This results in an imprecise answer even in

the simulator, but it is interesting to see if the quantum computers can do better in that case. However, here too, the output of the circuit on the quantum computers give noisy results. This can be seen by only computing the QPE and Inverse QPE and checking if most of the probability is on state $|00\rangle$. Figure 14 shows these results, and even though we have quite a lot of probability on the desired state, there is already much noise on other states. Thus when considering the full HHL circuit, that is including the controlled rotations, the obtained results are still too noisy to give useful insights.

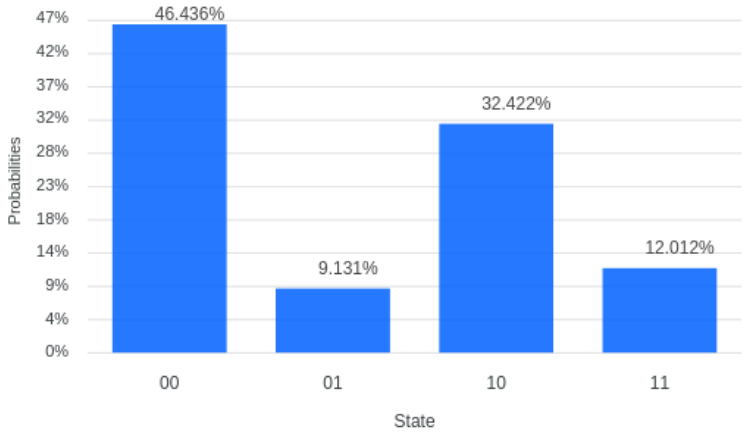


Figure 14: Statistics of measurements when measuring the result of the QPE and Inverse QPE with 2 qubits with quantum computer

It is good to note that when using IBM's software, only the circuit is described, however the way it is actually run depends on the quantum computer used, and this translation step is done automatically. So one could potentially get much more reliable results by studying the topology of the quantum computer used, designing a circuit that takes advantage of it, and manually taking care of running it in an optimal way for that quantum computer.

# 9 Conclusion

In this project we presented and reviewed the HHL algorithm used to solve linear systems of equations. In the first sections, we explained what the three main procedures are: Quantum Phase Estimation, Hamiltonian Simulation and Controlled Rotations. More precisely, we saw how the Quantum Phase Estimation affects the probability of success and precision of the whole algorithm, and some possibilities for performing an efficient Hamiltonian simulation.

Section 6 was dedicated to the controlled rotations, which required a careful analysis. We observed how they can be used to compute arbitrary functions of the qubits, using tools such as the Walsh Transform. Moreover, after describing how to implement such circuits, our various analyses showed that an exact computation may induce an exponential complexity in the number of gates used, which was also dependent on how the Quantum Phase Estimation was performed.

In section 7, we saw how one could use the HHL to solve problems such as solving a system of linear equations in finite fields for some specific cases.

Finally, the implementation presented in section 8 gave insights about the correctness of the HHL algorithm. When simulating, we always obtained the expected results. However, when using actual quantum computers, things got more complicated as it became hard to control the noise when there are many gates. Nevertheless, for simple cases, it confirmed that using quantum effects can help computationally. As time goes, quantum computers keep getting better, but there is still a lot to be discovered and improved in order to have reliable quantum systems.

## Acknowledgments

I would like to first thank Nicolas Macris for helping me throughout the project. It was a lot of work and sessions to get a good understanding of the algorithm, and he helped me push myself to better understand the core concepts. I would also like to acknowledge the Quantum Computing Association at EPFL, who organized many interesting events and deepened my interest in quantum computing. Finally, I acknowledge the support of the Qiskit software development framework for giving the possibility to simulate quantum circuits.

# References

[1] E. Deotto E. Farhi S. Gutmann D. A. Spielman A. M. Childs, R. Cleve. Exponential algorithmic speedup by quantum walk. *arXiv:quant-ph/0209131*, 2002.

[2] R. D. Somma A. M. Childs, R. Kothari. Quantum algorithm for systems of linear equations with exponentially improved dependence on precision. *arXiv:1511.02306 [quant-ph]*, 2017.

[3] S. Lloyd A. W. Harrow, A. Hassidim. Quantum algorithm for solving linear systems of equations. *arXiv:0811.3171 [quant-ph]*, 2009.

[4] G. R. Ahokas. Improved algorithms for approximate quantum fourier transforms and sparse hamiltonian simulations. *University of Calgary*, 2004.

[5] A. Ambainis. Variable time amplitude amplification and a faster quantum algorithm for solving systems of linear equations. *arXiv:1010.4458 [quant-ph]*, 2010.

[6] P. Mountney S. Severini N. Usher L. Wossnig D. Dervovic, M. Herbster. Quantum linear systems algorithms: a primer. *arXiv:1802.08227 [quant-ph]*, 2018.

[7] M. Mosca A. Tapp G. Brassard, P. Hoyer. Quantum amplitude amplification and estimation. *arXiv:quant-ph/0005055*, 2000.

[8] U. S. R. Murty J. A. Bondy. *Graph Theory with Applications*. The Macmillan Press Ltd., 1976.

[9] S. Lloyd. Universal quantum simulators. *Science 273, 1073*, 1996.

[10] I. L. Chuang M. A. Nilsen. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, 2010.

[11] Kaisa Nyberg. T-79.5501 lecture notes: Cryptology, 2007. URL: http://www.tcs.hut.fi/Studies/T-79.5501/2007SPR/lectures/boolean.pdf.

[12] C. Macchiavello M. Mosca R. Cleve, A. Ekert. Quantum algorithms revisited. *arXiv:quant-ph/9708016*, 1997.

[13] S. Dutta S. Roy B. K. Behera P. K. Panigrahi S. Dutta, A. Suau. Demonstration of a quantum circuit design methodology for multiple regression. *arXiv:1811.01726 [quant-ph]*, 2018.

[14] D. J. Egger S. Woerner. Quantum risk analysis. *arXiv:1806.06893 [quant-ph]*, 2018.

[15] A. C. Vásquez. Quantum algorithm for solving tri-diagonal linear systems of equations. *ETH Zürich*, 2018.

[16] M. M. Wilde. *Quantum Information Theory*. Cambridge University Press, Cambridge, 2013.

[17] S. Frankel S. Kais Y. Cao, A. Daskin. Quantum circuit design for solving linear systems of equations. *arXiv:1110.2232 [quant-ph]*, 2011.

[18] S. Frankel S. Kais Y. Cao, A. Daskin. Quantum circuits for solving linear systems of equations. *arXiv:1110.2232 [quant-ph]*, 2013.