UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Master Degree in Physics of Data

Final Dissertation

Quantum algorithms for the solution of partial differential equations with applications in the aerospace sector

Thesis supervisor
Dr. Ilaria Siloi

Thesis co-supervisors
Prof. Simone Montangero
Mattia Verducci

Candidate
Alessandro Grilli

Academic Year 2023/2024

Abstract

Partial Differential Equations (PDEs) are transversal to all scientific fields such as aero- and fluido-dynamics, plasma physics and finance. A technique for efficiently finding numerical approximations to the PDEs' solutions is based on the introduction of a finite mesh to discretize the space of the parameters, the so-called Finite Element Method (FEM). With this approach the solution of the PDEs ultimately reduces to the solution of a large system of linear equations. This thesis explores the potential of quantum algorithms, with a specific focus on the Harrow-Hassidim-Lloyd (HHL) algorithm, in accelerating the solution of PDEs relevant in the context of electromagnetic simulations for Earth Observation (EO). We present a comprehensive implementation of the HHL algorithm, that allows full control over input parameters and associated subroutines. By classically emulating HHL, we critically examine its limitations, and highlight the regimes where a speedup over classical methods is expected. This work represents the output of a six month internship with the research division of Thales Alenia Space Italia (TASI), aimed at exploring potential applications of quantum computing in the EO scenario.

Contents

Introduction

Partial Differential Equations (PDEs) are fundamental across many scientific fields, including aerodynamics, fluid dynamics, plasma physics, and finance. While solving differential equations analytically is ideal, such solutions are rare. Examples of analytically solvable cases include the electrostatic potential between infinite parallel plates or wave propagation in rectangular, circular, and elliptic waveguides [1]. Most real-world engineering problems do not have analytical solutions, leading to the development of various approximate methods.

The Finite Element Method (FEM) is a widely used numerical approach for solving PDEs [1, 2]. It involves discretizing a continuous domain into smaller geometric elements and approximating the behavior of the solution within each element using simple mathematical functions. This method transforms the PDE into a system of algebraic equations by enforcing the governing equations and boundary conditions at discrete points. Solving this linear system yields an approximate solution to the original PDE. FEM has been extensively used for analyzing electromagnetic fields in antennas, radar scattering, Radio frequency (RF), and microwave engineering. However, traditional methods for EM simulation like FEM, while robust, face significant challenges in scaling to large, complex systems due to their computational intensity [3]. Thus, any speedup for FEM would represent a significant advancement in these fields.

This limitation led us to consider quantum computing as a potential solution, indeed, quantum computing approaches to PDEs have been attracting

considerable attention in recent years and represent an active research field. This work focuses on the Harrow-Hassidim-Lloyd (HHL) quantum algorithm [4], which can solve linear system problems and demonstrates exponential speed-up over classical algorithms under certain conditions. This speed-up is achieved if we are interested in some expectation values of the solution and the input matrix is sparse. FEM poses an attractive prospect for enhancement using the HHL algorithm for several reasons [5]. In particular, FEM naturally generates sparse systems of linear equations whose corresponding matrix usually has a well-defined structure. However, the current stage of quantum hardware development is still in its infancy, limiting the immediate feasibility of solving large-scale EM problems via quantum computation [6]. Given these considerations, this work presents a comprehensive implementation of the HHL algorithm, that allows full control over input parameters and associated subroutines, to emulate it classically using Qiskit [7]. This implementation includes the development of a complete Python library from scratch. The goals of this work include gaining a deeper understanding of the algorithm's functionality, its intrinsic limitations, and its potential applications in the Earth Observation (EO) domain. Specifically, we are interested in the the solution of linear equation systems coming from FEM techniques for EM simulation problems. For the simulations we focused on Toeplitz tridiagonal matrices [8], since similar matrices result from the FEM applied to the 1-D Poisson equation [1]. This choice also enables a full control over the condition number of the matrix and its eigenvalues, such that we can easily test the implemented algorithm in different regimes. We analyze how to fine tune the input parameters, such as the number of qubits and the evolution time parameter t of the Hamiltonian simulation subroutine, discussing how to set them in a generic scenario.
This thesis represents the final output of a six-month internship within the research division of Thales Alenia Space Italia (TASI), aimed at exploring potential applications of quantum computing in the EO scenario. This thesis is structured as follows:

- **Chapter 1** introduces the numerical methods used in Computational Electromagnetism, focusing on relevant use cases for TASI, such as Antenna Design and Optimization. An overview of FEM is provided, discussing its implementation in the 1-D boundary-value problem.
- **Chapter 2** reviews fundamental concepts of quantum computation, such as quantum entanglement and the implementation of one- and two-qubit gates.
- **Chapter 3** describes the main concepts of quantum algorithms, such as quantum parallelism, and discusses the implementation of the primary subroutines of the HHL algorithm.
- **Chapter 4** details the step-by-step implementation of the HHL algorithm, from state preparation to measurement. It also covers how to compute the solution norm and the absolute average of the full solution.
- **Chapter 5** introduces the Qiskit framework and the software implementation of the algorithm, discussing the challenges of simulating quantum algorithms on classical hardware. It presents the results from the numerical simulations and provides an overview of potential future developments to extend this work.

Chapter 1

Computational Electromagnetism for EO problems

Computational Electromagnetism (CEM) plays a crucial role at Thales Alenia Space Italy (TASI). From the mission definition to the validation and verification campaign, numerous design steps of space assets involve electromagnetic (EM) simulation. These applications include antenna design and optimization, antenna calibration, and EM compatibility testing, among others. These problems vary in terms of wavelengths, physical sizes, and the accuracies required, necessitating a wide array of potential solvers. Each solver possesses its own set of advantages and limitations for particular problems, but all converge on the common task of solving a large system of equations. With the rapid growth in the scale and complexity of EM structures, CEM can result in extremely large matrices, which could challenge the capabilities of classical computation. Recently, quantum algorithms have been developed that efficiently solve linear systems, claiming exponential speedups over corresponding classical algorithms. Therefore, the potential for a quantum advantage opens up many new opportunities for the simulation and design of EM structures. This chapter introduces the domain of CEM and presents a use case involving the EM simulation of a slotted antenna for radar Earth Observation (EO) as a relevant industrial problem where quantum speedup *could be beneficial. Although this specific problem is efficiently addressed by classical algorithms, it represents a simplified version of a broader use case (phased array antenna). This broader use case involves the design and optimization of antennas for radar EO using the Finite Element Method (FEM) at a fixed frequency, which is currently impractical with classical methods due to its scale. Assessing the real benefits that quantum algorithms can provide to CEM problems is challenging because large-size problems are currently beyond the reach of Noisy Intermediate-Scale Quantum (NISQ) [6] hardware, while small-size problems are outperformed by classical algorithms and are not industrially relevant. The use case proposed in this thesis represents a good trade-off between industrial relevance and availability of quantum resources. The structure of this chapter is as follows: Sec. 1.1 gives an overview on Maxwell's Equations, Sec. 1.2 introduces the CEM domain and reviews the main solvers, highlighting their strengths and weaknesses, Sec. 1.3 defines the industrial use case and its parameters for solution using the Finite Element Method (FEM).Finally, Sec. 1.4 briefly describes how FEM operates in the case of the 1-dimensional boundary-value problem.*

1.1 Maxwell's Equations

Classical electromagnetism is based on Maxwell's equations which describe how electric and magnetic fields propagate and interact with matter. These equations are essential for understanding and simulating electromagnetic (EM) phenomena in various applications [1, 9]. The general differential form of Maxwell's equations, derived using Gauss's and Stokes's theorems, is given by:

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \quad \text{(Faraday's law)} \tag{1.1.1}$$

$$\nabla \times \mathbf{H} = \frac{\partial \mathbf{D}}{\partial t} + \mathbf{J} \quad \text{(Maxwell-Ampere law)} \tag{1.1.2}$$

$$\nabla \cdot \mathbf{D} = \rho \quad \text{(Gauss's law)} \tag{1.1.3}$$

$$\nabla \cdot \mathbf{B} = 0 \quad \text{(Gauss's law—magnetic)} \tag{1.1.4}$$

$$\nabla \cdot \mathbf{J} = -\frac{\partial \rho}{\partial t} \quad \text{(equation of continuity)} \tag{1.1.5}$$

where **E** is the electric field, **B** is the magnetic field, **H** is the magnetic field intensity, **D** is the electric displacement field, ρ is the electric charge density, and **J** is the current density.

In the static case, where the fields are time-independent, Equations (1.1.1), (1.1.2), and (1.1.5) can be written as:

$$\nabla \times \mathbf{E} = 0 \tag{1.1.6}$$

$$\nabla \times \mathbf{H} = \mathbf{J} \tag{1.1.7}$$

$$\nabla \cdot \mathbf{J} = 0 \tag{1.1.8}$$

1.2 Overview of Classical Solvers and Methods

Computational Electromagnetism (CEM) involves using numerical methods to compute approximate solutions to Maxwell's equations for various applications, including antenna design and optimization, electromagnetic compatibility, radar cross-section, and electromagnetic wave propagation. CEM typically addresses the problem of computing the electric (E) and magnetic (H) fields across the problem domain; from E and H, various other quantities can be derived, such as power flow direction (Poynting vector), scattering, waveguide's normal modes, and currents. CEM includes a wide range of methods, each with its advantages and disadvantages depending on the specific problem. A key parameter in selecting the most appropriate method is the Electrical Size (ES) of the object to be simulated. For conductors, ES is defined as the length of the object measured in wavelengths (relative to the specific frequency f or narrow band of frequencies at which the object operates):

$$ES = \frac{l}{\lambda}, \tag{1.2.1}$$

where l is the physical length of the object and $\lambda = \frac{v_p}{f}$. Here, v_p is the phase velocity of electrical signals along the object. In free space, $v_p = c$.

Based on the ES of the object, CEM methods can be classified as follows:

Low-frequency Methods

For $ES < 20$, the object is considered "electrically short," meaning voltage and current are approximately constant along the conductor. These methods can be further divided into:

- **Differential Methods**: these methods discretize ("mesh") the problem space into regular shapes ("cells") and solve Maxwell's equations simultaneously across all cells. Examples include the Finite Element Method (FEM), suitable for both time and frequency domain simulations, and the Finite Difference Time Domain (FDTD), which is suitable for time domain simulations. Both methods produce sparse matrices, reducing memory requirements, though frequency domain simulations (FEM) require repeated calculations for each frequency within the desired bandwidth, increasing computational cost.

- **Integral Methods:** starting from the Green's function solution G of Maxwell's equations, these methods express the electromagnetic field as an integral of G-weighted equivalent currents on the object's boundaries. The primary integral method is the Method of Moments (MoM), which focuses on the surface current distribution J. By expanding J

into a series of basis functions with unknown coefficients and projecting this current expansion onto a set of weighting functions, a matrix equation is formed and solved.

High-frequency Methods (Asymptotic Methods)

For $ES \geq 20$, traditional full-wave methods result in impractically large matrices. Here, electromagnetic fields are modeled as rays interacting with media boundaries, undergoing reflection, refraction, and transmission. Geometrical Optics (GO) forms the core of this approach. Other methods include the Geometrical Theory of Diffraction (GTD) and Physical Optics (PO).

Hybrid Methods

For complex large-scale problems, hybrid methods combine high-frequency and low-frequency methods by dividing the problem domain into several sub-domains. The main hybrid method is Domain Decomposition Methods (DDM), which divides the computational domain into smaller subdomains, each solved independently. The final solution is obtained by combining these individual solutions through carefully chosen boundary conditions.

All numerical methods reported above require dividing the structure of interest into many cells or elements, using approximations to convert the problem into a solvable form, such as a linear system of equation of the form $A\mathbf{x} = \mathbf{b}$. A common issue is the high demand for computer memory and computation time for solving the resulting linear system problem. For a detailed technical description of the main CEM methods we refer the reader to Ref. [3]. In Table 1.1 we report possible applications for each method.

Method	**Applications**
Finite Difference Time Domain (FDTD)	• Inhomogeneous materials (different materials or dielectric substrates) • Complex geometries (microstrip patch antennas with multi-layer and other small complex planar designs) • Different boundary conditions for various regions • Microscopic details
Finite Element Method (FEM)	• Inhomogeneous materials (different materials or dielectric substrates) • Complex geometries (microstrip patch antennas with multi-layer and other small complex planar designs) • Different boundary conditions for various regions • Microscopic details
Method of Moments (MoM)	• Unbounded problems • Microwave and antenna engineering • Complex geometries • Free space (satellite antennas, like horn and apertures) • Ground planes (accurate simulations of antennas mounted on vehicles) • Stratified and periodic materials • Modeling wire antennas and metallic structures with high conductivity
Geometrical Optics (GO)	• Scattering and diffraction • Radar cross-section (RCS) • Effects of a finite antenna ground plane • Interactions between several antennas and structures
Domain Decomposition Method (DDM)	• Complex structures (e.g., a 3D feed horn near a large metallic object) • Large antenna arrays • Metamaterials • Radar cross-section

Table 1.1: Methods of Computational Electromagnetism and their applications.

1.3 Antenna Design for Radar EO

Antennas are vital components in numerous real-world applications, including spacecraft. They serve as critical elements for a variety of applications, both as primary and secondary payloads, such as telemetry, tracking, and control (TT&C), radar observation, telecommunications, and space situational awareness (SSA). Therefore, antenna design is an essential step in spacecraft development, ranging from simple designs like horns, patches, and dipoles to complex antenna arrays used in Synthetic Aperture Radar (SAR). The design of an antenna is crucial to ensure it fulfills specifications such as center frequency, bandwidth, efficiency, and directivity/gain. Equally critical is its placement; the platform on which it is mounted can significantly affect its installed performance. In complex real-world environments, propagation issues can lead to coverage gaps or co-site interference between radio systems. Essential to this design phase are electromagnetic (EM) simulations, often conducted using commercial software, such as CST. Simulations for antenna arrays and radio frequency (RF) waveguides typically involve up to several hundred thousand finite elements and may require several hours to complete on hardware equipped with GPU acceleration. Thales Alenia Space Italia is universally recognized as one of the leading companies in designing and implementing satellite radar Earth Observation (EO) systems. These systems comprise one or more antennas operating at specific frequencies, generally between 1 and 40 GHz, tailored to particular applications, along with a dedicated digital processing backend. For the purposes of this thesis, the focus will be on arrays of slotted waveguides for EO. An illustrative example is the Sentinel-1's C-SAR antenna, depicted in Fig. 1.3.1, a large $12.03\,\mathrm{m} \times 0.84\,\mathrm{m}$ active phased array antenna. This antenna consists of 14 tiles, each comprising 20 dual-polarized resonant waveguide sub-arrays. These sub-arrays include 40 H/V polarized slotted antennas. Given the small ES, the required frequency domain simulation, and the presence of several different boundary conditions, the Finite Element Method (FEM) is employed for accurate modeling and analysis of this use case. The simulation of the full system is unfeasible for classical algorithms; therefore, the slot antenna has been identified as the building block that enables the entire design and optimization of larger antennas made of active arrays of slotted waveguides for radar Earth Observation (EO).

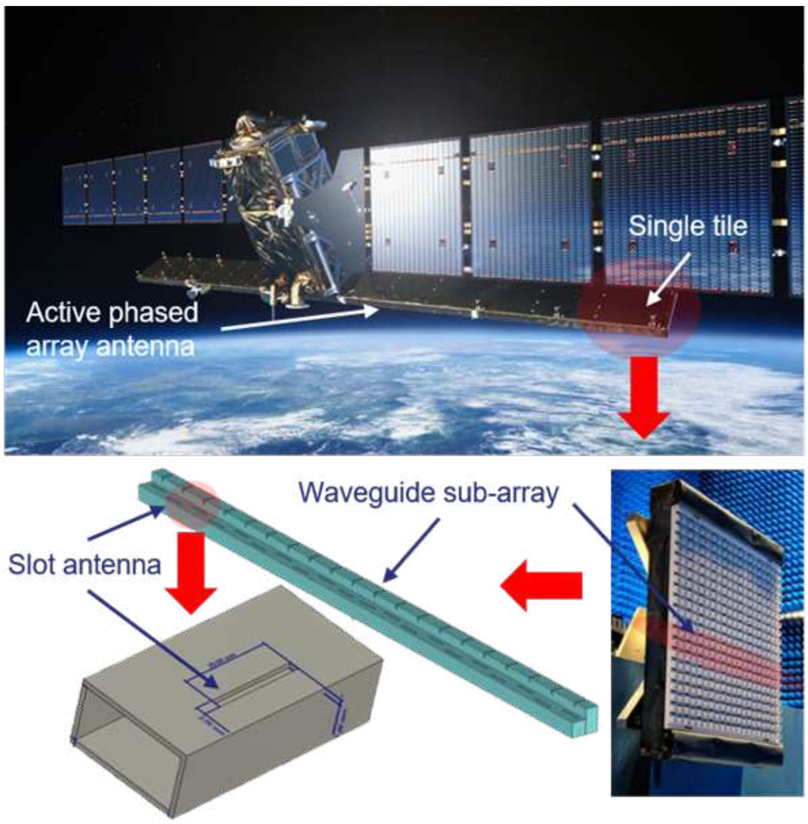


Figure 1.3.1: Image representing the Sentinel-1's C-SAR antenna along with its components.

1.3.1 Slotted Antennas

A slotted antenna is typically constructed from a waveguide or a flat surface with one or more slots. These slots, when excited by a signal propagating through the waveguide, emit electromagnetic waves that contribute to the antenna's radiation pattern. According to Babinet's principle, the radiation pattern of a slot antenna can be approximated using the same theoretical frameworks applicable to rod element antennas, such as dipoles. Slot antennas are predominantly utilized in frequency ranges from 300 MHz to 24 GHz, which encompasses most radar applications. The simplicity of slot antennas allows them to be arranged effectively in large linear arrays. The specific radiation pattern characteristics are determined by the number, orientation, and placement of the slots along the waveguide. Key advantages of employing slotted antenna arrays include:

- Highly directional beams,
- Cost-effectiveness,
- Manufacturing simplicity,
- Low-profile design suitable for integration into complex surfaces and structures,
- Structural robustness.

In this use case, the waveguide is a metallic rectangular structure where the electromagnetic wave propagates and induces current distributions along its perfectly conducting (PEC) walls. These currents can become sources of alternating potential at the edges of a narrow slot cut into the wall. Adjusting the slot's position relative to the waveguide's edge can modify the power radiated by the slot. Typically, the waveguide feeds the slots in the Transverse Electric Mode 01 (TE_{01}), supporting efficient energy transfer and pattern consistency. This scenario was selected due to its significance in antenna theory and its feasibility for analytical solution, making it suitable for benchmarking. Detailed mathematical formulations of the field calculations are documented in [10]. In Fig. 1.3.2 we show the structure and specifications of the slotted antenna. The following parameters will be used for the simulation:

- $a = 22.86mm$

- $b = 10.16mm$
- $f = 10GHz$
- $L = 50mm$
- $S = 20mm$ (width of the slot)

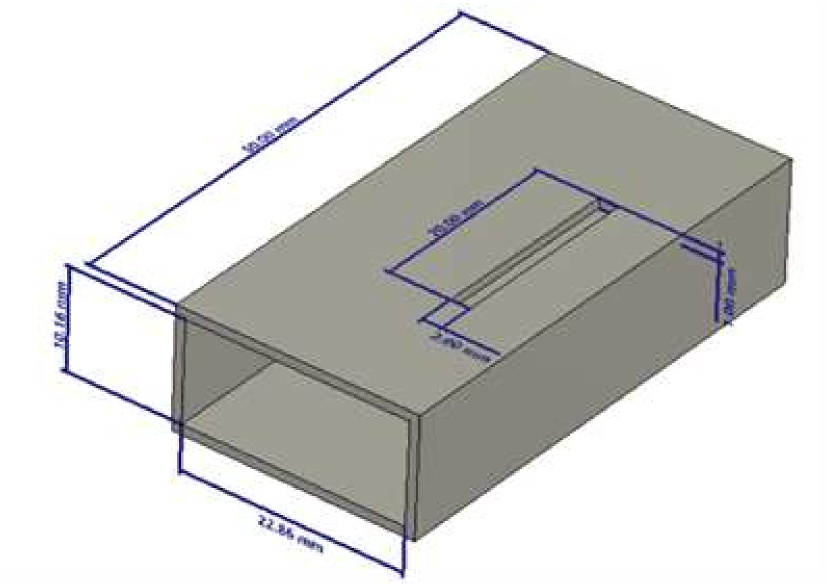


Figure 1.3.2: Slotted antenna structure and specifications.

1.3.2 Criteria for the use case selection

Identifying a single representative use case for the entire computational complexity of EM simulations in space applications is challenging. Additionally, most computations currently use specialized non-open- source software, introducing complications such as:

- **Accessibility and licenses**: Frequently, it is not possible to obtain intermediate outputs crucial for scientific analysis but deemed irrelevant for everyday industrial use, such as the global system matrix (output of the numerical method) or the parameters selected for the computation.

- **High level of optimization on classical Hardware**: This can hidden the quantum benefits when applied on very small scale problems like those that can be currently addressed by quantum computers and/or emulators.

The strategy followed by TASI is to start from a problem currently tackled by commercial software with dedicated computer resources and simplify it reaching a building block of the simulation whose size and solution is still relevant but it could be also feasible to be tackled with next future quantum hardware. Thus, the selected scenario is a simplified version of a larger use case related to the design and optimization of antennas for radar EO using FEM at a fixed frequency.

1.3.3 Quantum Computing approach

Since PDE solvers ultimately solve systems of equations, evaluating the matrices generated by classical solvers and identifying where quantum methods could offer advantages is crucial. A common problem of FEM is the huge demand on computer memory (mesh size) and computation time (system's solution and mesh refinement). The quantum approach will be used to tackle the solution of the linear system arising from FEM discretization.

1.4 The Finite Element Method

The Finite Element Method (FEM) [1, 2] is a numerical approach for solving partial differential equations (PDEs). It involves discretizing a continuous domain into smaller geometric elements and approximating the behavior within each element using simple mathematical functions. The method transforms the PDE into a system of algebraic equations by enforcing the governing equations and boundary conditions at discrete points. Solving this linear system yields an approximate solution to the original PDE, allowing for accurate simulations of electromagnetic fields in heterogeneous media [9].

1.4.1 The Boundary-Value Problem

For this analysis, we follow the work presented in Ref. [1]. A typical boundary-value problem can be defined through a differential equation within a certain domain Ω:

$$\mathcal{L}\phi = f \tag{1.4.1}$$

together with the boundary conditions on the boundary of the domain Γ. In Eq. (1.4.1), $\mathcal{L}$ is a differential operator, f is the forcing function, and ϕ is the unknown quantity we seek to find by solving the differential equation.

In electromagnetism, the form of Eq. (1.4.1) ranges from simple Poisson equations to more complicated wave equations. Similarly, the boundary conditions vary from simple Neumann and Dirichlet conditions to more complex higher-order ones. When possible, it is always preferable to solve the differential equation analytically. However, such problems are the exception rather than the rule. Some examples of analytically solvable problems include the static potential between infinite parallel plates or wave propagation in rectangular, circular, and elliptic waveguides.

Many real-life engineering problems do not have an analytical solution, prompting the development of various approximate methods, such as the Ritz and Galerkin methods.

1.4.2 One-Dimensional FEM

A generic boundary-value problem can be described by the differential equation

$$-\frac{d}{dx}\left(\alpha\frac{d\phi}{dx}\right) + \beta\phi = f, \quad x \in (0, L) \tag{1.4.2}$$

where ϕ is the unknown function, α and β are known parameters or functions associated with the physical properties of the solution domain, and f is a known source. The standard one-dimensional Poisson equation is a special form of Eq. (1.4.2) with $\beta = 0$. The solution of Eq. (1.4.2) can be obtained by solving the equivalent variational problem defined by

$$\delta F(\phi) = 0, \quad \phi|_{x=0} = p, \tag{1.4.3}$$

where

$$F(\phi) = \frac{1}{2}\int_0^L \left[\alpha\left(\frac{d\phi}{dx}\right)^2 + \beta\phi^2\right] dx - \int_0^L f\phi\, dx + \left[\frac{\gamma}{2}\phi^2 - q\phi\right]_{x=L}. \tag{1.4.4}$$

Eq. (1.4.3) indicates that we seek the stationary point of $F(\phi)$ under the given Dirichlet boundary condition. In this context, ϕ represents the trial function rather than the exact solution. For simplicity, we will use linear functions as basis functions for the interpolation.

Discretization and Interpolation

The first step of the Finite Element Method consists of discretizing the domain $(0, L)$ into small subdomains, corresponding to short line segments in this case. Let $l^e(e = 1, 2, \ldots, M)$ denote the length of the e-th element and M the total number of elements. Let $x_i(i = 1, 2, \ldots, N)$ denote the position of the i-th node, with $x_0 = 0$ and $x_N = L$.

The second step of the Finite Element Method is to select the interpolation functions. For simplicity, we use linear functions. Thus, for the e-th element, $\phi(x)$ can be approximated by

$$\phi^e = a^e + b^e x, \tag{1.4.5}$$

where a^e and b^e are constants to be determined. For one-dimensional systems, there are two nodes associated with each element:

$$\begin{aligned} \phi_1^e &= a^e + b^e x_1, \\ \phi_2^e &= a^e + b^e x_2, \end{aligned} \tag{1.4.6}$$

leading to

$$\phi^e(x) = \sum_{i=1}^{2} N_i^e(x)\phi_i^e, \tag{1.4.7}$$

where N_1^e and N_2^e denote the interpolation or basis functions given by

$$\begin{aligned} N_1^e &= \frac{x_2^e - x}{l^e}, \\ N_2^e &= \frac{x - x_1^e}{l^e}, \end{aligned} \tag{1.4.8}$$

with $l^e = x_2^e - x_1^e$.

Derivation of Elemental Equations via Ritz Method

During this analysis we keep the same notation as in Ref. [1], where $\{\cdot\}$ describes a column vector, while $[\ \cdot\ ]$ represents a matrix. For simplicity, let us consider the case of homogeneous Neumann conditions $\gamma = q = 0$. The functional can be rewritten as

$$F(\phi) = \sum_{e=1}^{M} F^e(\phi^e), \tag{1.4.9}$$

where

$$F^e(\phi^e) = \frac{1}{2}\int_{x_1^e}^{x_2^e} \left[\alpha\left(\frac{d\phi^e}{dx}\right)^2 + \beta(\phi^e)^2\right] dx - \int_{x_1^e}^{x_2^e} f\phi^e\, dx. \tag{1.4.10}$$

Taking the derivative of F^e with respect to ϕ^e and rewriting in matrix form, we obtain

$$\left\{\frac{\partial F^e}{\partial \phi^e}\right\} = [K^e]\{\phi^e\} - \{b^e\}, \tag{1.4.11}$$

where

$$\begin{aligned} K_{ij}^e &= \int_{x_1^e}^{x_2^e} \left(\alpha \frac{dN_i^e}{dx}\frac{dN_j^e}{dx} + \beta N_i^e N_j^e\right) dx, \\ b_i^e &= \int_{x_1^e}^{x_2^e} N_i^e f\, dx. \end{aligned} \tag{1.4.12}$$

If we consider α, β, and f to be constant within each finite element, we can evaluate the elements of the matrices analytically with the final result

$$\begin{aligned} K_{11}^e = K_{22}^e &= \frac{\alpha^e}{l^e} + \beta^e \frac{l^e}{3}, \\ K_{12}^e = K_{21}^e &= -\frac{\alpha^e}{l^e} + \beta^e \frac{l^e}{6}. \end{aligned} \tag{1.4.13}$$

Similarly, for b_i^e: $b_1^e = b_2^e = f^e \frac{l^e}{2}$.

Assembly: Retrieving the System of Equations

The global system of equations can be obtained by summing over all the elements and imposing the stationary requirement:

$$\left\{ \frac{\partial F}{\partial \phi} \right\} = \sum_{e=1}^{M} \left\{ \overline{\frac{\partial F}{\partial \phi^e}} \right\} = \sum_{e=1}^{M} \left(\left[\overline{K^e} \right] \left\{ \overline{\phi^e} \right\} - \left\{ \overline{b^e} \right\} \right) = \{0\}, \tag{1.4.14}$$

where the matrices and vectors behind the summation signs are expanded or augmented to matrix $M \times M$ and column vectors $M \times 1$ by zero filling. This leads to the following final results for $[K]$:

$$\begin{aligned} K_{11} = K_{11}^1 &= \frac{\alpha^1}{l^1} + \beta^1 \frac{l^1}{3}, \\ K_{NN} = K_{22}^M &= \frac{\alpha^M}{l^M} + \beta^M \frac{l^M}{3}, \\ K_{ii} = K_{22}^{i-1} + K_{11}^i &= \frac{\alpha^{i-1}}{l^{i-1}} + \beta^{i-1} \frac{l^{i-1}}{3} + \frac{\alpha^i}{l^i} + \beta^i \frac{l^i}{3}, \\ K_{i+1,i} = K_{i,i+1} = K_{12}^i &= -\frac{\alpha^i}{l^i} + \beta^i \frac{l^i}{6}. \end{aligned} \tag{1.4.15}$$

and for b:

$$\begin{aligned} b_1 = b_1^1 &= f^1 \frac{l^1}{2}, \\ b_N = b_2^M &= f^M \frac{l^M}{2}, \\ b_i = b_2^{i-1} + b_1^i &= f^{i-1} \frac{l^{i-1}}{2} + f^i \frac{l^i}{2}. \end{aligned} \tag{1.4.16}$$

Finally, we can impose the Dirichlet boundary condition $\phi|_{x=0} = p$ by setting

$$K_{11} = 1, \quad b_1 = p, \quad K_{1,j} = 0 \quad \text{for} \quad j = 2, 3, \ldots, M \tag{1.4.17}$$

resulting in a linear system of the type

$$[K]\phi = \{b\}. \tag{1.4.18}$$

This analysis can be extended to higher dimensions [1] and more complex boundary conditions. As seen from Eq. (1.4.15), the matrix resulting from FEM analysis to solve a 1-D boundary value problem is a tridiagonal matrix. An important characteristic of the linear systems of equations resulting from FEM is that the corresponding matrices are sparse and have a well-defined structure. This property is crucial for achieving exponential speed-up with the HHL algorithm, as will be discussed throughout this work.

Example: 1-D Poisson Equation

We now review a simple example. Considering Eq. (1.4.2) with $\beta = 0$ and $\alpha = -1$, we retrieve the 1-D Poisson equation. Let's consider the case where $f = \sin(x + \pi)$ with boundary conditions $\phi(0) = 0$ and $\phi(\pi) = 1$. Using the previously mentioned procedure, we can set the number of finite elements M for the discretization. In this simple 1-D case, the number of nodes n_{nodes} corresponds to $n_{\text{nodes}} = M + 1$. For example, we can choose $M = 7$. The comparison between the analytical solution and the numerical one using the FEM method is shown in Fig. 1.4.1.

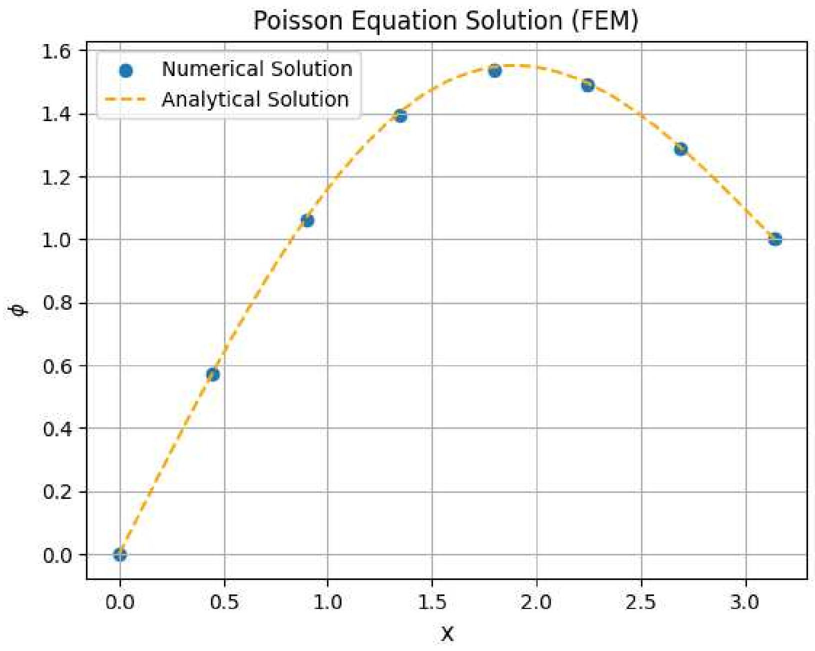


Figure 1.4.1: Comparison between the analytical solution and the numerical solution obtained using FEM to solve a 1-D Poisson equation.

Chapter 2

Quantum Computing

In this chapter we introduce the fundamental concepts of Quantum Computing, starting from the classical computation to understand how it can be extended within the quantum framework. In Sec.2.2 we introduce the Quantum Bit (Qubit) as well as some of its important properties, such as Superposition and Entanglement. Then, in Sec.2.3 we describe the main Quantum operators used in Quantum Computing, and describe the notation we will use throughout this work to draw quantum circuits.

2.1 Classical computation

The *bit* is the fundamental unit of classical information, it is a binary variable which can assume values $\{0, 1\}$. In classical computation each operation translates in computing functions from n-bit to m-bit:

$$f : \{0, 1\}^n \hookrightarrow \{0, 1\}^m \tag{2.1.1}$$

where 0 and 1 are represented as distinguishable states of an appropriate classical system. These operations are called gates. It can be demonstrated that every computation can be performed using only a restricted set of elementary logic gates, called *universal gates*, which we will now specify. Starting from the 1-bit logic gates, we have only two possibilities:

- COPY, which returns the input bit state;
- NOT, which negates the input bit state.

Conversely, examples of 2-bit gates are:

- AND, which outputs 1 if and only if both inputs are 1;
- OR, which outputs 1 if either of the inputs is 1.

The corresponding truth tables are reported in Table 2.1

Input	**COPY**	**NOT**
0	0	1
1	1	0

(a) One-Bit Logical Gates

A	**B**	**AND**	**OR**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

(b) Two-Bit Logical Gates

Table 2.1: Truth Tables of elementary operations in classical computation.

A set of universal classical gates is given by *AND*, *OR*, *NOT*, *COPY*, while it can be demonstrated that a *minimal* set is composed of *COPY* and a chosen gate between *NAND* and *NOR*. We can notice that some of these operations are *irreversible* or *non-invertible*, i.e they do not have a one-to-one mapping between inputs and outputs. In other words, given the output of the gate, it is not possible to uniquely determine the input values that produced that output.
The last observation leads to the fact that these operations cannot be understood as unitary gates, which is instead a fundamental property in quantum mechanics, as we will see in Sec. 2.3.
Now, having defined the elementary states and operations of classical computation, we can understand how they can be extended in the quantum framework, starting by introducing the quantum bit.

2.2 Qubit

Qubits, as their classical counterpart, are physical systems, in particular two-level quantum mechanical systems. However, one can build the general framework of *quantum computation* and *quantum information* by treating qubits as *abstract mathematical objects*, each representing a two-dimensional Hilbert space. The main difference between bits and qubits is that the latter exploit the principles of quantum mechanics, allowing them to exist in superposition states, representing both 0 and 1 simultaneously:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad \text{where} \quad \alpha, \beta \in \mathbb{C}, |\alpha|^2 + |\beta|^2 = 1 \tag{2.2.1}$$

$|0\rangle$ and $|1\rangle$ are known as *computational basis states* of the qubit, α and β are complex coefficients and $|\alpha|^2$, $|\beta|^2$ correspond to the probability of measuring the qubit in the state $|0\rangle$ and $|1\rangle$ respectively.
Equivalently, we can rewrite α and β as phases, obtaining the relation:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle \quad \text{where} \quad \theta \in [0, \pi], \varphi \in [0, 2\pi] \tag{2.2.2}$$

This expression has a geometric representation, θ and φ are real numbers and define a unit vector on the Bloch Sphere (reported in Figure 2.2.1), which can be used to visualize the state of a single qubit. When measuring a qubit, the output it gives will still end up being either 0 or 1, but which one we get depends on a probability which is set by the direction of the unit vector. If it is exactly on the equator of the Bloch sphere, we get either state with a 50% probability.

2.2.1 Multiple qubits and entanglement

Having understood the main concepts about a single qubit, we can address systems with two qubits where the beautiful and, at the same time, weird properties of quantum mechanics can be properly appreciated.
When we deal with a multiple qubits system, the states representing the com-

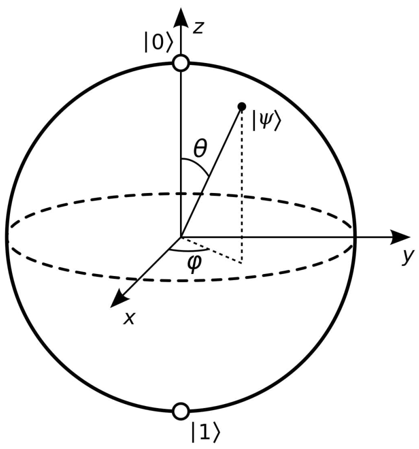


Figure 2.2.1: Qubit state visualization on the Bloch sphere. The basis state $|0\rangle$ corresponds to the unit vector $\hat{z}$, while the basis state $|1\rangle$ corresponds to $-\hat{z}$. Any point on the surface of the sphere represents a qubit state defined by $|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle$.

putational basis of the ensemble can be written as the tensor product $\otimes$ of the single qubit states. For example, if we have a system of two qubits, both in the state $|0\rangle$, the state of the system can be written as $|0\rangle \otimes |0\rangle = |00\rangle$. In general, a system of two qubits has four computational basis states $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$.
Similarly to the single qubit case, a pair of qubits can also exists in a superposition of the the four states listed above, each associated with a complex coefficient, and the state vector of the system can be expressed as:

$$|\psi\rangle = c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle. \tag{2.2.3}$$

When we perform the measurement, we find the system in the state $|i\rangle$ with probability $|c_i|^2$, where $i = 00, 01, 10, 11$ and the normalization condition $\sum_{i\in\{0,1\}^2}|c_i|^2 = 1$ must be satisfied.
An important two qubit state is the *Bell state*

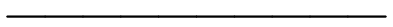
$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}. \tag{2.2.4}$$

This state shows an interesting property: a measurement on one qubit affects the other, meaning that the measurement outcomes are *correlated.* When we measure the first qubit, there are two possible outcomes: $|0\rangle$ and $|1\rangle$, each with a probability of $\frac{1}{2}$. Upon measuring the second qubit, we are guaranteed (with probability 1) to find $|0\rangle$ if we had found $|0\rangle$ in the first measurement, and we are guaranteed to find $|1\rangle$ if we had found $|1\rangle$. This means that the measurement outcome of the first qubit always corresponds to the outcome of the measurement on the second one.
Such states, which cannot be factored into product states, are called *entangled states.* The only way to turn a separable state of the form $|0\rangle|0\rangle$, to into an entangled state of the type in Eq. (2.2.4), is to apply a *collective* unitary transformation to the state, as we will discuss in Sec. 2.3. These correlations have been well-studied in the last century [11, 12]. Specifically, John Bell showed that the measurement correlations in the Bell States are stronger than those that could ever exist between classical systems [13]. In the following sections, we will explore how these states can be created through quantum gates and how they can be useful in practical scenarios such as quantum algorithms.
More generally, if we consider a system of n qubits, the computational basis states have the form $|x_1x_2...x_n\rangle$, where $x_i \in \{0, 1\}$ and the quantum system can be described by 2^n complex numbers. To understand the entity of such number, just think that with $n = 500$ we exceed the number of atoms in the Universe [13] and for a classical computer it would not be feasible to store all these complex numbers, thus one aim of quantum computation is to take advantage of this computational resource that quantum mechanics offers to us.

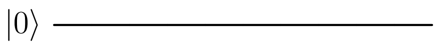
2.3 Quantum Circuits

As we have seen in Sec. 2.1, classical computers can be represented as *wires*, to carry the information, and *logical gates* to perform operations on

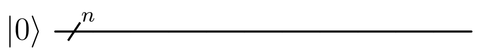
bits , which are the elementary unit of classical computation.
In a similar way, quantum computers are built upon quantum circuits, composed by wires and quantum gates to carry out operations on qubits and control their quantum state's evolution. In this work we use the *quantikz* library on LateX [14] to draw quantum circuits.
Usually we use circuit diagrams to visualize quantum circuits. We build and read these diagrams from left to right, in particular we represent the circuit wire as a line



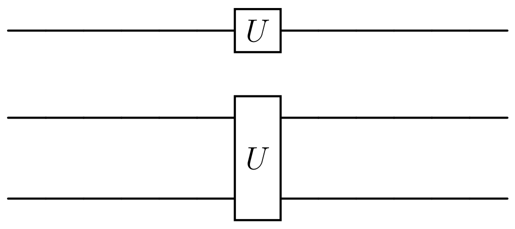
A wire with no gate on it means that the qubit stays in the same state as it was originally prepared.
We denote the initial state of the qubit with a ket on the left of the wire, which is usually chosen to be $|0\rangle$



We denote n number of qubits prepared in the state $|0\rangle$ with a n symbol across the wire.



With the notation established, the next step is to describe the commonly used *quantum operators* or *quantum gates.*
A single-qubit gate is represented as a box containing the corresponding operator's letter, positioned over the qubit line. In contrast, a two-qubit gate is depicted as a box spanning two quantum wires. This notation extends similarly to ternary operators and beyond.
As a general example, here we report the representations for generic one- and two-qubit gates



2.3.1 Single-qubit gates

We start by defining the set of one-qubit operators. Generally, quantum gates acting on a single qubit can be described by two by two matrices. However, because quantum computers adhere to the laws of quantum mechanics, there are constraints on the matrices that can be adopted as quantum gates. Firstly, we have to recall that the normalization condition requires $|\alpha|^2 + |\beta|^2 = 1$ for a quantum state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. This must be true also for the state $|\psi'\rangle = \alpha'|0\rangle + \beta'|1\rangle$ we retrieve after the application of the quantum gate. Consequently, the matrix representing the single-qubit gate must be *unitary*, meaning $U^\dagger U = \mathbb{I}$, where $U^\dagger$ is the adjoint of U. This Unitary constraint is the only requirement for quantum gates [13]. For this reason, quantum gates must be reversible, which is a key distinction between classical and quantum computation. Indeed, not all classical gates are reversible (as mentioned Sec.2.1) and, therefore, cannot be used in Quantum Computing. An example of non-invertible gates includes the *AND* and *OR* operators, whereas the *NOT* gate acts linearly and can thus be extended to Quantum Computation, as we will see later.

The first operators we examine are *Pauli operators.* These three matrices, together with the Identity matrix and their products with the factors $\pm i$ and ± 1, form the so called *Pauli group.*

X operator

The X operator is the quantum version of the NOT operator in classical computation, indeed it is also known as NOT or bit flip operator. Its matrix representation is given by

$$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.3.1}$$

If we apply X to a state $\alpha|0\rangle + \beta|1\rangle$ we have

$$X \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix} \tag{2.3.2}$$

which corresponds to $X \equiv |0\rangle\langle 1| + |1\rangle\langle 0|$ in ket notation. We can also represent the X operator in circuit diagrams

$$\text{———}\boxed{X}\text{———}$$

If we start with the qubit in the state $|0\rangle$, after applying the NOT operator we get the state $|1\rangle$

$$|0\rangle \;\text{———}\boxed{X}\text{———}\; |1\rangle$$

Y operator

Y operator, when applied to an input state, performs a rotation along the y axis of the Bloch sphere

$$Y \equiv \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{2.3.3}$$

and when applied to $|0\rangle$ we obtain

$$|0\rangle \;\text{———}\boxed{Y}\text{———}\; \text{i}\,|1\rangle$$

Z operator

The Z operator rotates the state vector along the z axis of the Bloch Sphere, it is also called *phase flip* operator since the rotation angle is 180 degrees

$$Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{2.3.4}$$

The Z gate essentially leaves the state $|0\rangle$ unchanged while flipping the sign of $|1\rangle$, as shown by the following circuit representation:

$$\alpha\,|0\rangle + \beta\,|1\rangle \;\text{———}\boxed{Z}\text{———}\; \alpha\,|0\rangle - \beta\,|1\rangle$$

Rϕ operator

In fact, the Z operator is just a peculiar case of the more general R_ϕ gate where $\phi = \pi$:

$$R_\phi \equiv \begin{bmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{bmatrix}. \tag{2.3.5}$$

By recalling the Euler's Identity $e^{i\pi} = -1$, we retrieve the Z matrix. When we apply R_ϕ to a quantum state, it leaves the state $|0\rangle$ unchanged while performing a rotation of phase ϕ to the state $|1\rangle$

$$\alpha|0\rangle + \beta|1\rangle \;\text{———}\boxed{R_\phi}\text{———}\; \alpha|0\rangle + e^{i\phi}\beta|1\rangle$$

RY operator

Another useful gate we will use throughout this work is the *RY* gate, a rotational gate that performs a rotation around the y-axis by an angle θ. Its matrix representation is:

$$RY \equiv \begin{bmatrix} cos(\frac{\theta}{2}) & -sin(\frac{\theta}{2}) \\ sin(\frac{\theta}{2}) & cos(\frac{\theta}{2}) \end{bmatrix}. \tag{2.3.6}$$

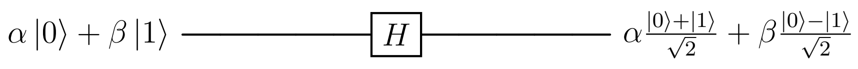
For example, if we chose the rotation angle $\theta = \pi$, applying RY to $|0\rangle$ yields $|1\rangle$, while applying it to $|1\rangle$ results in $-|0\rangle$. In a later chapter, we will see how to use this gate during the implementation of the HHL algorithm.

Hadamard gate

Finally, we introduce the so called *Hadamard gate*

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{2.3.7}$$

This operator is one of the most useful gates in quantum computing, as it allows a qubit to transition from a computational basis state into a superposition of two states. Considering again the Bloch sphere, *Hadamard gate* is composed by a rotation of 90 degrees along the y-axis and a rotation about the x-axis by 180 degrees. We can see how this gate acts on a generic state by the following circuit representation


$\alpha |0\rangle + \beta |1\rangle$
H
$\alpha \frac{|0\rangle+|1\rangle}{\sqrt{2}} + \beta \frac{|0\rangle-|1\rangle}{\sqrt{2}}$

2.3.2 Multiple qubit gates

Let us now consider multiple qubit gates. As discussed in Chapter 2.2.1, in a two-qubit system we have four computational basis states. Therefore, to act on such a Hilbert space, we need operators with a 4x4 matrix representation.

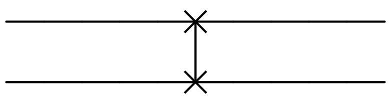
Firstly, we introduce the *SWAP* operator, which simply swap the two qubits, and if applied to the four computational basis states acts as follow

$$\begin{aligned} \text{SWAP}|00\rangle &= |00\rangle \\ \text{SWAP}|01\rangle &= |10\rangle \\ \text{SWAP}|10\rangle &= |01\rangle \\ \text{SWAP}|11\rangle &= |11\rangle \end{aligned} \tag{2.3.8}$$

We can represent this operator with the following matrix

$$\text{SWAP} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3.9}$$

This gate is incapable of generating entanglement because, if the qubits start in a product state, swapping their components will still yield a product state. We can represent its circuit diagram as follow



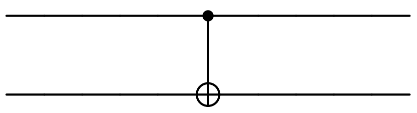
The most important operator in quantum computing is the *controlled-NOT* or *CNOT* gate. This operator takes two qubits as input, where the first qubit is identified as the *control* qubit and the second as the *target* qubit. If the control qubit is in the state $|0\rangle$, then the target qubit is left unchanged. However, if the control qubit is in the state $|1\rangle$, then we apply the *NOT* operator 2.3.1 to the target qubit. We can represent the *CNOT* operator with the following matrix

$$\text{CNOT} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.3.10}$$

The *CNOT* gate is not only invertible but also possesses the property of being its own inverse. This means that if two CNOT gates are applied in series, where the output of the first gate serves as the input to the second gate, the final output will match the initial state. When the quantum *CNOT* gate is applied to qubits in the state $|0\rangle$ or $|1\rangle$, without involving any superpositions, the computation is identical to that of a classical *XOR* gate using binary values 0 and 1. Thus, this gate is a generalization of the classical *XOR* gate, in fact it acts on the computational basis as follows

$$\begin{aligned} \text{CNOT}|00\rangle &= |00\rangle \\ \text{CNOT}|01\rangle &= |01\rangle \\ \text{CNOT}|10\rangle &= |11\rangle \\ \text{CNOT}|11\rangle &= |10\rangle \end{aligned} \tag{2.3.11}$$

To draw the quantum circuit representation of this gate we use the following notation:



This gate is so important in Quantum Computing since we use it to entangle two qubits. For example, we can create entangled states of the form 2.2.4 just with a combination of *Hadamard* and *CNOT* gates

$$\mathrm{CNOT}(\mathrm{H}|0\rangle \otimes |0\rangle) = \mathrm{CNOT}\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

Quantum circuits are designed by placing and linking quantum gates in a specific order. The arrangement and connections of these gates determine how information flows and the resulting computations. Crafting quantum circuits involves the careful selection of gates, considering how qubits are interconnected, and optimizing the circuit to achieve the desired computational outcomes. Furthermore, it is important to exploit the unique properties of quantum mechanics, such as Quantum Superposition and Entanglement to build efficient Quantum algorithms, which can solve problems that are beyond the capabilities of classical computers.

Chapter 3

Quantum Algorithms

In this chapter, we introduce the three principal subroutines utilized in the Harrow-Hassidim-Lloyd (HHL) algorithm. We begin with a brief overview of quantum parallelism. Then, in Sec. 3.2, we discuss the Quantum Fourier Transform (QFT), a critical building block for several quantum algorithms, including the Quantum Phase Estimation (QPE). The QPE is further explored in Sec. 3.3, where we detail its theoretical frameworks and practical implementations. Finally, Sec. 3.4 reviews Hamiltonian simulation algorithms, essential for the efficient time evolution of a quantum state. This section also provides an overview of state-of-the-art algorithms and discusses their integration within the HHL algorithm, highlighting both practical applications and theoretical advancements.

3.1 Quantum Parallelism

Quantum parallelism is a fundamental feature of quantum computers [15], enabling them to perform parallel computations by leveraging the superposition of quantum states. This concept, foundational in quantum computing, involves several critical aspects that distinguish it from classical parallelism. Reference [16] examines the core aspects of quantum parallelism; it contrasts classical parallelism with its quantum equivalent, providing insights into their distinct applications, and introducing methods for evaluating quantum parallelism's role in practical settings. We give an overview of these concepts, resuming the work done in [16].

Interference in Quantum Parallelism

Quantum parallelism is governed by the principles of quantum interference, which can be either constructive or destructive. When a qubit is in a superposition state (where both $\alpha \neq 0$ and $\beta \neq 0$ in Eq. (2.2.1)), multiple quantum threads run simultaneously. The interaction of these threads, influenced by complex numbers, leads to interference patterns. Unlike classical probability, where probabilities are additive, quantum probabilities can cancel each other out through destructive interference. Thus, quantum parallelism creates an interference pattern from the interaction of quantum states, similar to how signals combine in antenna arrays. Optimizing this interference is crucial, as constructive interference amplifies correct solutions while destructive interference suppresses incorrect ones, guiding the system toward the right outcomes.

Probabilistic Composition and Quantum States

Quantum parallelism exploits the composition of elementary probabilistic system. In quantum computing, composing single quantum states results in an exponential increase in the number of possible quantum states. This exponential growth differentiates quantum systems from classical probabilistic systems, where events are mutually exclusive. Quantum computing systems simultaneously explore and process multiple computational paths through superposition. Moreover, quantum gates are described by unitary transformations, which ensure the total probability remains conserved.

Entanglement and Quantum Parallelism

Entanglement is essential for quantum parallelism, enabling the exploration of states and threads that single-qubit operations alone cannot achieve.

Entangled states, created through controlled operations, are not representable by simple tensor products of single-qubit states. This complexity poses challenges for classical computer systems, particularly in simulating quantum computers. While non-entangled states can be efficiently encoded, highly entangled states require exponentially more resources, demonstrating the unique capabilities and demands of quantum systems.

Quantum algorithms structure

In developing quantum algorithms, it is crucial to rethink the application of quantum parallelism. Unlike classical parallelism, which distributes work across threads to solve parts of a problem, quantum algorithms exploit maximum parallelism to explore all potential solutions. Quantum algorithms typically follow three phases:

- *Initialization*: All available quantum parallelism is initiated from a classical input, often using Hadamard gates to create superpositions. This operation is also known as *Hadamard transform* and an example is reported in Fig. 3.1.1 The input is encoded, or an oracle is queried, leveraging quantum parallelism.

- *Exploration and Interference*: Quantum parallelism explores all possible solutions, with correct solutions amplified through constructive interference and incorrect ones suppressed via destructive interference. Entanglement expands the range of solutions explored beyond single-qubit operations.

- *Measurement*: The superposition is collapsed to a classical state through measurement. Efficient quantum applications utilize small quantum parallelism before measurement, reducing the number of possible outcomes.

In summary, quantum parallelism and interference optimization are pivotal in quantum computing. Leveraging these principles these principles, quantum systems can (theoretically) solve complex problems more efficiently than classical systems by exploring a vast solution space and selectively amplifying correct solutions.

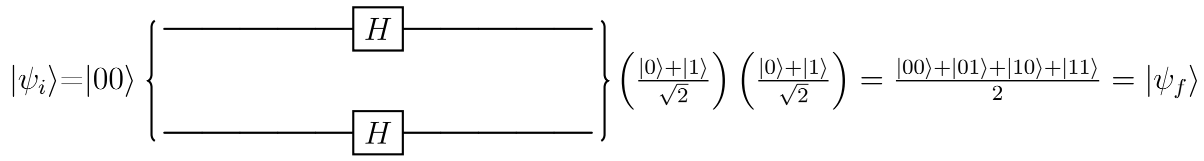

$|\psi_i\rangle=|00\rangle$
H
H
$\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right)=\frac{|00\rangle+|01\rangle+|10\rangle+|11\rangle}{2}=|\psi_f\rangle$

Figure 3.1.1: Circuit representation of Hadamard transform. This operation involves n Hadamard gates acting in parallel on n qubits. To simplify the calculations, we consider a system of two qubits initially prepared in the state $|0\rangle$.

3.2 Quantum Fourier Transform

The *quantum Fourier transform* (QFT) [17] is the quantum analogue of the *inverse discrete Fourier transform* (DFT). QFT and its inverse , iQFT, are building blocks for many quantum algorithms, such as the well-known *Shor algorithm* [18] used for the prime factorization of n-bit integers, as well as the Quantum Phase Estimation (QPE) and the HHL algorithm, that we will implement throughout this work.
The *inverse discrete Fourier transform*, given a sequence of N complex numbers $\{f_k\}_{k=0,\ldots,N-1}$, maps them to a sequence of N complex numbers $\{\tilde{f}_k\}_{k=0,\ldots,N-1}$ defined by:

$$\tilde{f}_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{\frac{2\pi ijk}{N}} f_j. \tag{3.2.1}$$

Analogously, the *QFT* is a linear transformation that, when performed on n qubits, acts on the computational basis states $\{|j\rangle\}_{j=0,\ldots,2^n-1}$ according to:

$$\text{QFT}|j\rangle \equiv \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{\frac{2\pi ijk}{N}} |k\rangle \tag{3.2.2}$$

and equivalently for an arbitrary state:

$$\mathrm{QFT}\sum_{j=0}^{N-1} x_j|j\rangle \equiv \frac{1}{\sqrt{N}}\sum_{k=0}^{2^n-1} \tilde{f}_k|k\rangle \tag{3.2.3}$$

The classical discrete Fourier transform (DFT) [19] involves multiplication by an $N \times N$ matrix, where $N = 2^n$. Essentially, this transform would require $O(N^2)$ elementary operations. However, there is a well-known classical method, the *fast Fourier transform* (FFT) [19], which reduces the scaling to $O(N \log N)$. With quantum parallelism, we can achieve an even more efficient result. Following similar passages as done in [20], we can express j and k in their binary representation:

$$\begin{aligned} j &= j_{n-1}2^{n-1} + ... + j_0 2^0 = \sum_{i=0}^{n-1} j_i 2^i \\ k &= k_{n-1}2^{n-1} + ... + k_0 2^0 = \sum_{i=0}^{n-1} k_i 2^i. \end{aligned} \tag{3.2.4}$$

We define k_i as the i_{th} digit, where k_0 represents the least significant bit (LSB), which will corresponds to the topmost qubit in the circuit diagram (this is the same convention used in Qiskit [7]), and the most significant bit (MSB) is k_{n-1}. This notation allows us to represent the elements of the computational basis as the tensor product of the local bases of single qubits:

$$|k\rangle = \bigotimes_{i=1}^{n} |k_{n-l}\rangle \equiv |k_{n-1}\cdots k_0\rangle. \tag{3.2.5}$$

Using this notation, the expression in (3.2.2) becomes:

$$\mathrm{QFT}|j\rangle = \frac{1}{2^{\frac{n}{2}}}\sum_{k_{n-1}=0}^{1}\cdots\sum_{k_0=0}^{1} \exp\left(\frac{2\pi i j}{2^n}\sum_{m=0}^{n-1} 2^m k_m\right)|k_{n-1}\cdots k_0\rangle. \tag{3.2.6}$$

Now, if we define $l = n - m$, we can see that:

$$\sum_{m=0}^{n-1} \frac{2^m k_m}{2^n} = \sum_{l=1}^{n} \frac{k_{n-l}}{2^l}$$

In addition to this, we can also decompose the main exponential in a product of exponential acting on the local bases states, thus the expression (3.2.6) becomes:

$$\mathrm{QFT}|j\rangle = \frac{1}{2^{\frac{n}{2}}} \sum_{k_{n-1}=0}^{1} \cdots \sum_{k_0=0}^{1} \left[\prod_{l=1}^{n} \exp\left(\frac{2\pi i j}{2^n} k_{n-l}\right)\right] |k_{n-1} \cdots k_0\rangle. \tag{3.2.7}$$

Finally, using the Equation (3.2.5), we get:

$$\begin{aligned}
\mathrm{QFT}|j\rangle &= \frac{1}{2^{\frac{n}{2}}} \sum_{k_{n-1}=0}^{1} \cdots \sum_{k_0=0}^{1} \bigotimes_{l=1}^{n} \exp\left(\frac{2\pi i j}{2^n} k_{n-l}\right) |k_{n-l}\rangle \\
&= \frac{1}{2^{\frac{n}{2}}} \bigotimes_{l=1}^{n} \left[\sum_{k_{n-l}=0}^{1} \exp\left(\frac{2\pi i j}{2^l} k_{n-l}\right) |k_{n-l}\rangle\right] \\
&= \frac{1}{2^{\frac{n}{2}}} \bigotimes_{l=1}^{n} \left[|0\rangle + \exp\left(2\pi i j 2^{-l}\right)|1\rangle\right] \\
&= \frac{\left(|0\rangle + e^{2\pi i 0.j_0}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.j_1 j_0}|1\rangle\right) \cdots \left(|0\rangle + e^{2\pi i 0.j_{n-1}\cdots j_1 j_0}|1\rangle\right)}{2^{\frac{n}{2}}}
\end{aligned} \tag{3.2.8}$$

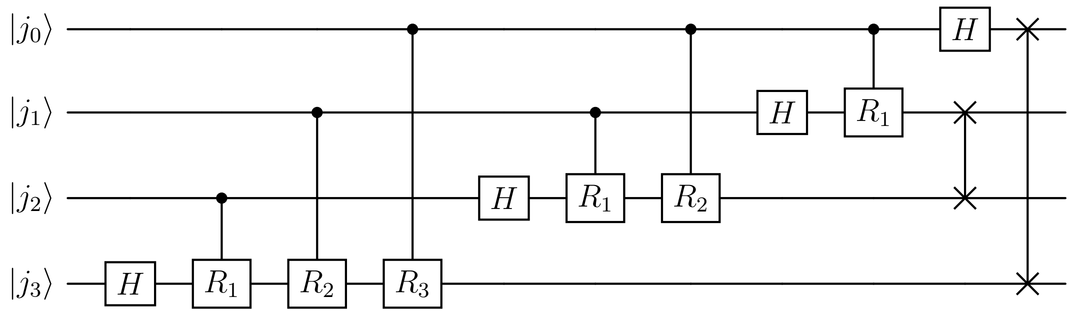
where in the last step we used the binary expansion of j (3.2.4) and we have also adopted the notation

$$0.j_l j_{l+1} \cdots j_m = \frac{1}{2} j_l + \frac{1}{4} j_{l+1} + \cdots + \frac{1}{2^{m-l+1}} j_m$$

so that we can rewrite $j2^{-l}$ as:

$$j2^{-l} = j_{n-1} j_{n-2} \cdots j_{l+1} j_l \cdots j_0.$$

From the last expression in (3.2.8) we can see that QFT transforms each computational basis state into an unentangled state of n qubits. This representation allows us to implement the QFT using an efficient quantum circuit composed of elementary gates.
As an example, we can consider the case for $n = 4$, the resulting QFT circuit diagram is reported below.



Each Hadamard gate acts on the generic state $|j_k\rangle$ as:

$$\mathrm{H}|j_k\rangle = \frac{1}{\sqrt{2}}\big(|0\rangle + e^{2\pi i 0.j_k}|1\rangle\big). \tag{3.2.9}$$

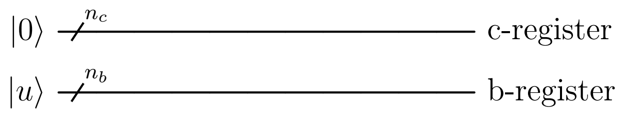
The other contributions to the relative phase of $|0\rangle$ and $|1\rangle$ in the k_{th} qubit is given by conditional R_d gates of the type (2.3.1):

$$R_d \equiv \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/2^d} \end{bmatrix} \tag{3.2.10}$$

where $d = |k - k'|$ is the distance between the two qubits. It is also important to note that a SWAP operation of order $O(n)$ is required, as the QFT changes the original qubits order. In the example above, for $n = 4$ we have four H gates and six controlled-R gates. In general, if we are dealing with a QFT with n qubits, the corresponding circuit requires n H gates and $\frac{1}{2}n(n-1)$ controlled-R, plus the gates involved in the SWAP operation. This results in a scaling of order $O(n^2)$, which is exponentially more efficient if compared with the classical FFT.

3.3 Quantum Phase Estimation

The QFT introduced in Sec.3.2 is a key subroutine of a more complex algorithm known as *Quantum Phase Estimation* (QPE) [13, 21]. If we consider a unitary matrix U with eigenvector $|u\rangle$ and eigenvalue $e^{2\pi i\varphi}$, the goal of the QPE algorithm is to estimate the corresponding phase φ. As we will see, this algorithm is the main subroutine of the HHL algorithm. Therefore, we begin by understanding how QPE works.
The QPE is built upon two quantum registers. The first one, identified as the *clock-register* or *c-register*, contains n_c qubits, where n_c depends on the number of digits of accuracy desired in the solution. The second quantum register, referred to as the *b-register*, contains n_b qubits, which must be sufficient to represent the eigenvector $|u\rangle$. Specifically, if we have a vector of size N, we need $n_b = \log_2(N)$ qubits to encode it in the b-register. We maintain the same convention as in the last section , where the topmost qubit in the circuit diagram corresponds to the most significant bit (MSB). The two quantum registers are represented in the circuit below. Note that the c-register starts with n_c qubits in the state $|0\rangle$, while the b-register encodes the eigenvector $|u\rangle$ of the unitary matrix U.



The QPE algorithm has three main steps:

- Superposition of the clock qubits through an Hadamard transform (as defined in Eq. (3.1.1)) on the c-register;
- n_c controlled U^{2^j}-operations on the b-register, where U is raised to successive powers of two and each operation is controlled on the j_{th} qubit in the c-register;

- Inverse Quantum Fourier Transform (iQFT), where iQFT $\equiv$ QFT†.

Hadamard transform

The initial state of the system is

$$|\psi_{initial}\rangle = |u\rangle_b|0\rangle_c. \tag{3.3.1}$$

The Hadamard transform is applied only to the c-register, transforming its initial state into a superposition of $|0\rangle$ and $|1\rangle$. Thus the state after this first step is

$$|\psi_1\rangle = \frac{1}{2^{\frac{n_c}{2}}}|u\rangle_b \sum_{k=0}^{2^{n_c-1}} |k\rangle \tag{3.3.2}$$

where the sum in the last expression represents the state of the c-register as a superposition of all possible binary states.

Controlled-U^{2^j}

The controlled-U^{2^j} operation applies U^{2^j} to the b-register, conditioned on the j_{th} qubit of the c-register being in state $|1\rangle$. Thus, the number of unitary operations applied are equal to the number of qubits n_c in the c-register. Since $|u\rangle$ is an eigenvector of U with eigenvalue $e^{2\pi i\varphi}$, we have that $U|u\rangle = e^{2\pi i\varphi}|u\rangle$, thus:

$$U^{2^j}|u\rangle = e^{2\pi i\varphi 2^j}|u\rangle. \tag{3.3.3}$$

Since this operation is applied conditionally to the value of the j_{th} qubit in the c-register, we have that

$$U^{2^j}|\psi_1\rangle = \frac{1}{2^{\frac{n_c}{2}}}e^{2\pi i\varphi 2^j k_j}|u\rangle_b \sum_{k=0}^{2^{n_c-1}} |k\rangle \tag{3.3.4}$$

Where $k_j = 0, 1$. We can extract the phase factor so that it effectively multiplies the computational basis states $|k\rangle$ in the c-register. Therefore, each qubit in the c-register accumulates a phase factor $e^{2\pi i\varphi 2^j}$ depending on its position j in the c-register.

Inverse Quantum Fourier Transform (iQFT)

After these first two steps, the c-register is in the state:

$$\frac{1}{2^{n_c}}\Big(|0\rangle + e^{2\pi i 2^0 \varphi}|1\rangle\Big)\Big(|0\rangle + e^{2\pi i 2^1 \varphi}|1\rangle\Big)\cdots\Big(|0\rangle + e^{2\pi i 2^{n_c-1} \varphi}|1\rangle\Big) \tag{3.3.5}$$

which can be rewritten as

$$\frac{1}{2^{n_c}}\Big(|0\rangle + e^{2\pi i 0.\varphi_0}|1\rangle\Big)\Big(|0\rangle + e^{2\pi i 0.\varphi_1\varphi_0}|1\rangle\Big)\cdots\Big(|0\rangle + e^{2\pi i 0.\varphi_{n_c-1}\cdots\varphi_1\varphi_0}|1\rangle\Big). \tag{3.3.6}$$

The last expression represents the state we obtain after applying a QFT to the state $|\varphi_1\varphi_2\cdots\varphi_{n_c}\rangle$, as one can see from Equation (3.2.8). Therefore, if we apply an inverse QFT (iQFT), we recover the aforementioned product state. Finally, a measure of the c-register in the computational basis gives us the solution φ that we were looking for.
However, the previous considerations apply only to the ideal scenario where φ can be precisely represented with n_c bits. It can be shown [13] that this method yields a close approximation of φ with high probability. To approximate φ to an accuracy of 2^{-n} (i.e., to n bits) with a probability of at least ϵ, we must choose

$$n_c = n + \left\lceil \log_2\left(2 + \frac{1}{2\epsilon}\right)\right\rceil. \tag{3.3.7}$$

Moreover, we are assuming that the eigenvector $|u\rangle$ of the matrix U is known beforehand. If we instead prepare a different state $|\psi\rangle = \sum_u c_u|u\rangle$, the result will be a state close to $\sum_u c_u|\tilde{\varphi}_i\rangle|u\rangle$, where $\tilde{\varphi}_i$ is a good approximation of the state φ_i. The state $\tilde{\varphi}_u$ has a probability $|c_u|^2$, thus allowing us to avoid preparing the state $|u\rangle$ by introducing some randomness into the process. We will better understand this analysis in the next chapter, where we will study and implement the *HHL* algorithm. The resulting circuit of QPE in Fig. 3.3.1.

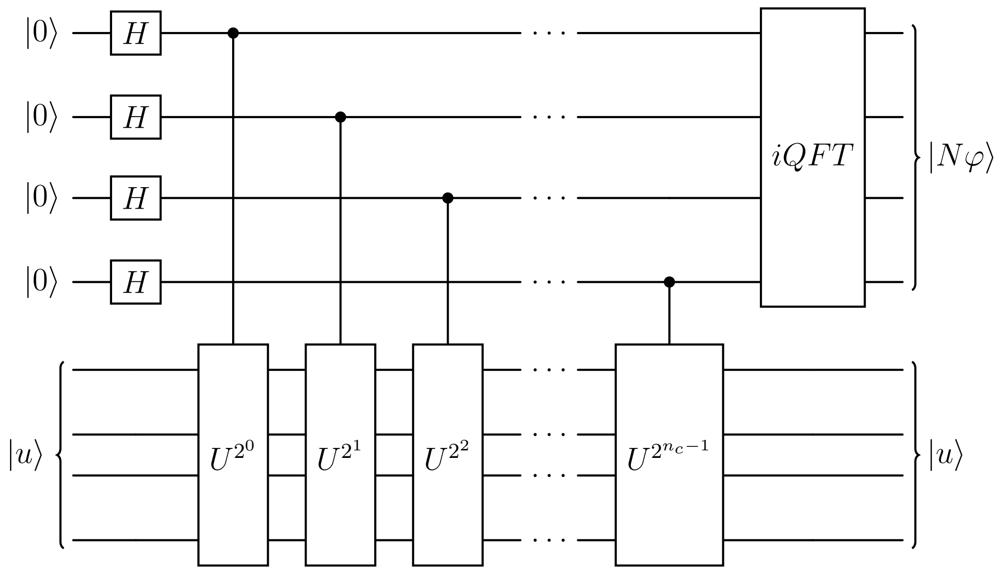


Figure 3.3.1: Quantum circuit diagram of the QPE

3.4 Hamiltonian Simulation

Hamiltonian Simulation (HS) involves the development of efficient methods for performing the temporal evolution of a quantum state according to a specified Hamiltonian, denoted as H. Namely,

$$|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle. \tag{3.4.1}$$

This process is a fundamental component of the HHL algorithm, this will be further explored in the next chapter. Depending on the input state, operator H and resources available, there are many quantum algorithms that can perform this task. We provide a brief overview of this extensive and actively evolving field of research.

Necessity for efficient simulation

In Eq. (3.4.1), we observe the necessity to exponentiate the matrix H, a process that is generally computationally demanding. In quantum systems, Hilbert space's dimensionality grows exponentially with the number of qubits n, namely $N = 2^n$. This results in exponentially large operators. Naive approaches need a runtime of $O(N^3)$ for an $N \times N$ matrix, which is limiting, even for small matrices. For this reason the classical simulation of a quantum system described by a generic Hamiltonian H is a hard task. Therefore, there has been significant focus on developing new efficient methods. Quantum computers, in particular, are capable of simulating efficiently Hamiltonian operators [22], this task is known as *Hamiltonian simulation.* Since H is a Hermitian operator, e^{-iHt} is a unitary operator. This allows it to be implemented using quantum gates. An efficient quantum simulation can be defined as follows:

Definition 3.4.1. *We say that a Hamiltonian H that acts on n qubits can be efficiently simulated if for any $t > 0$, and a simulation error $\epsilon > 0$, there exists a quantum circuits U consisting of $poly(n, t, 1/\epsilon)$ gates such that $\left\|U - e^{-iHt}\right\| \leq \epsilon$, where e^{-iHt} represents the ideal evolution.*

The dependence on t is critical since simulating H for time t requires at least $\Omega(t)$ time, as stated by the no-fast-forwarding theorem [23]. On the other hand, it is possible to achieve running time logarithmic in $1/\epsilon$.
However, we cannot efficiently simulate any arbitrarily Hamiltonians, efficient simulation is possible for specific classes of Hamiltonians, such as those acting nontrivially on a constant number of qubits. Indeed, according to Solovay-Kitaev's theorem, any unitary evolution involving a constant number of qubits can be approximated with an error of at most ϵ using $poly(\log(1/\epsilon))$ one- and two-qubit gates [24].

3.4.1 Product Formulas

Consider an Hamiltonian of the form

$$H = H_1 + H_2 \tag{3.4.2}$$

This involves a sum of terms, which we assume acting on only a constant number of qubits, thus easy to simulate. Generally, if H_1 and H_2 can be efficiently simulated individually, then $H_1 + H_2$ can also be efficiently simulated. If H_1 and H_2 commute we have that $e^{-iH_1 t}e^{-iH_1 t} = e^{-i(H_1+H_1)t}$. Otherwise we can still simulate them using Lie product formula:

$$e^{-i(H_1+H_1)t} = \lim_{m\to\infty} \left(e^{-iH_1 t/m} e^{-iH_2 t/m}\right)^m . \tag{3.4.3}$$

For practical implementations, the product formula can be truncated to a finite number of steps, this introduces a certain error ϵ which we want to keep small as possible. If we want to achieve

$$\left\| e^{-i(H_1+H_2)t} - \left(e^{-iH_1 t/m} e^{-iH_2 t/m}\right)^m \right\| \leq \epsilon, \tag{3.4.4}$$

the number of steps required is $m = O\left(\frac{t^2 \max(\|H_1\|,\|H_2\|)^2}{\epsilon}\right)$, ensuring the error remains within ϵ. With the constraint that $c = \max(\|H_1\|, \|H_2\|)$ must be at most $poly(n)$ for H_1 and H_2 to be efficiently simulable [24].
We can achieve better scaling with respect to t by using higher-order expansions. Considering approximations to the 2nd order, we derive the Trotter-Suzuki formula, expressed as:

$$\left\| e^{-i(H_1+H_2)t} - \left(e^{-iH_1 t/2m} e^{-iH_2 t/m} e^{-iH_1 t/2m}\right)^m \right\| \leq \epsilon, \tag{3.4.5}$$

where a smaller value of m is required for the same accuracy.
This approach is pertinent because any Hermitian matrix can be expressed as a linear combination of Pauli operators, which form a complete basis for the Hilbert space of $2^n \times 2^n$ matrices, where n represents the number of qubits. This enables the decomposition of an arbitrary Hermitian matrix as follows:

$$H = \sum_{j=1}^{N} a_j P_j, \tag{3.4.6}$$

where each P_j is a tensor product of 2×2 matrices from the Pauli group, discussed in Sec. 2.3.1. The coefficients a_j can be calculated using the relation

$$a_j = \frac{1}{2^n} \mathrm{Tr}\left(H P_j\right). \tag{3.4.7}$$

However, an arbitrary matrix can require decomposition into $O(2^{2n})$ Pauli operators. This implies that simulating the unitary operator $e^{-iHt/m}$ with Lie Trotter formula 3.4.4 and using Pauli decomposition would necessitate $O(\frac{2^{6n}t^2}{\epsilon})$ Trotter steps, a computation that scales worse than classical methods due to its exponential dependence on n. Ideally, we seek a polynomial scaling with n, but achieving this is only feasible for specific models [22]. This means the Pauli decomposition is not efficient for arbitrary matrices.

Simulating Sparse Hamiltonians

In the HHL original paper [4], the exponential speedup wrt classical methods is achieved under an important constraint: the matrix A has to be d-sparse and efficiently row-computable, thus we review this definitions.

Definition 3.4.2. *(Sparsity) A $N \times N$ matrix is said to be d-sparse if it has at most d nonzero entries per row.*

Definition 3.4.3. *(Sparse matrix) A $N \times N$ matrix is said to be sparse (in a fixed basis) if it has at most poly$(\log N)$ nonzero entries per row [25].*

Definition 3.4.4. *(Row computability) The entries of a matrix A are efficiently row computable if, given the indices i, j, we can obtain the entries A_{ij} efficiently, i.e. in $O(d)$ time, where d is the sparsity as defined above [26].*

In particular, the simulation methods described previously allow us to efficiently simulate sparse and efficiently row-computable matrices. This is achieved by finding an efficient decomposition of a d-sparse matrix H into 1-sparse matrices [23, 27].

3.4.2 State of the art algorithms

In the literature there are several quantum algorithms proposed for simulating sparse Hamiltonian. However, these algorithms make use of the so-called quantum oracles. The power of quantum computers is often studied in the query model [28]. In this model, we have to compute a function $f(x_1, ..., x_N)$ of an input $x_1, ..., x_N$, with x_i accessible via queries to a black box (or quantum oracle) that, given i, outputs x_i. Thus, the complexity of quantum algorithms using oracles is measured by the number of queries that an algorithm makes [28]. The query and gate complexities of these prominent methods for efficient Hamiltonian simulation are summarized in Table 3.1. In this table, d represents the sparsity of the Hamiltonian, and n indicates the number of qubits involved [29].

3.4.3 Hamiltonian Simulation Implementation

The Hamiltonian Simulation is the most important part of HHL algorithm, as it is performed several times within the QPE module. In this thesis, we implemented HS in two different ways:

- Building the circuit which simulates e^{-iHt} exactly, to test the correctness of the HHL algorithm using small matrices. This means that we first compute classically the exponential matrix e^{-iHt} and then use the Qiskit function *UnitaryGate* [35] to build the corresponding quantum circuit.

- Approximating e^{-iHt}, particularly utilizing the Suzuki-Trotter product formula and the decomposition in Pauli Operators. For this purpose we used the function *TrotterQRTE* [36] available with Qiskit. However, for an arbitrary matrix, we have $\mathcal{O}(2^{2n})$ Pauli Operators, resulting in a simulation that scales exponentially with the number of qubits used.

A further development of this work is to implement an HS based on the state-of-the-art algorithms available in this field, such as those shown in Table 3.1.

Algorithm	**Query complexity**	**Gate complexity** (t, ϵ)	**Gate complexity** (n)
PF 1_{st} order [22]	$O\left(d^4t^2 \,/\, \epsilon\right)$	$O\left(d^4t^2 \,/\, \epsilon\right)$	$O\left(n^5\right)$
PF $2k_{th}$ order [23]	$O\left(5^{2k}d^3t(dt \,/\, \epsilon)^{1/2k}\right)$	$O\left(5^{2k}d^3t(dt \,/\, \epsilon)^{1/2k}\right)$	$O\left(5^{2k}n^{3+1/k}\right)$
Quantum walk [30]	$O\left(dt \,/\, \sqrt{\epsilon}\right)$	$O\left(dt \,/\, \sqrt{\epsilon}\right)$	$O\left(n^4 \log n\right)$
Fractional-query simulation [31]	$O\left(d^2t\frac{\log(dt/\epsilon)}{\log\log(dt/\epsilon)}\right)$	$O\left(d^2t\frac{\log^2(dt/\epsilon)}{\log\log(dt/\epsilon)}\right)$	$O\left(n^4\frac{\log^2 n}{\log\log n}\right)$
Taylor Series (TS) [32]	$O\left(d^2t\frac{\log(dt/\epsilon)}{\log\log(dt/\epsilon)}\right)$	$O\left(d^2t\frac{\log^2(dt/\epsilon)}{\log\log(dt/\epsilon)}\right)$	$O\left(n^3\frac{\log^2 n}{\log\log n}\right)$
Linear Combination of q.walk steps [33]	$O\left(dt\frac{\log(dt/\epsilon)}{\log\log(dt/\epsilon)}\right)$	$O\left(dt\frac{\log^{3.5}(dt/\epsilon)}{\log\log(dt/\epsilon)}\right)$	$O\left(n^4\frac{\log^2 n}{\log\log n}\right)$
Quantum Signal Processing (QSP) [34]	$O\left(dt + \frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$	$O\left(dt + \frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$	$O\left(n^3\right)$

Table 3.1: Comparison of various quantum algorithms for efficient Hamiltonian Simulation [29].

Chapter 4

Harrow-Hassidim-Lloyd algorithm

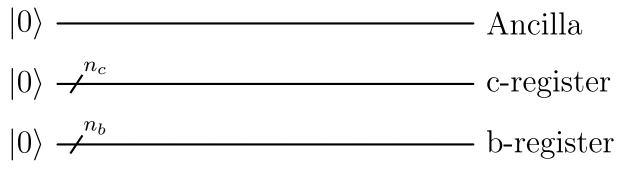
Now that we have established the necessary components, we can examine the central focus of this work: the Harrow-Hassidim-Lloyd (HHL) quantum algorithm. The HHL quantum algorithm [4] can be employed to solve linear system problems and has been shown to achieve exponential speedup over classical methods, such as the classical conjugate gradient method, in specific instances. In Sec.4.1 we will systematically explore each step of the HHL algorithm, clarifying its theoretical foundations, operational mechanics, and the conditions under which it offers computational advantages. This detailed exposition aims to provide readers with a comprehensive understanding of the algorithm's implementation, following the analysis made in [37]. In Sec. 4.2, we discuss how to compute the solution norm and the absolute average of the final solution.

4.1 The HHL Algorithm: Step-by-Step

Let's begin by introducing the problem more rigorously. A linear system of equations can be expressed as

$$A\mathbf{x} = \mathbf{b} \tag{4.1.1}$$

where A is an $N \times N$ matrix, and $\mathbf{x}$ and $\mathbf{b}$ are N-dimensional vectors. The *HHL* algorithm utilizes two quantum registers and an additional ancilla qubit. We will refer to these quantum registers as the c-register and b-register, maintaining the same notation used in the previous sections. As in the QPE framework, we need at least $n_b = \log_2 N$ qubits in the b-register, where we want to store the solution $\mathbf{x}$ of the linear system. Conversely, the number of qubits n_c used inside the c-register depends on the bit-precision we want to achieve, and will be discussed later on. A representation of the two quantum register and the ancilla qubit is reported below



During this analysis, we will keep the same notation of the previous sections, thus the rightmost (ending) qubit represents the least significant bit (LSB) which is classified as the topmost qubit in the circuit representation. The matrix A is assumed to be Hermitian, i.e a complex square matrix that is equal to its own conjugate transpose: $A = A^\dagger$. This is a fundamental constraint that has to be respected during the Hamiltonian Simulation (3.4), otherwise it can be converted to

$$\tilde{A} \equiv \begin{bmatrix} 0 & A \\ A^\dagger & 0 \end{bmatrix} \tag{4.1.2}$$

which is Hermitian by construction. Then, we can solve the equation

$$\tilde{A}\mathbf{y} = \begin{bmatrix} \mathbf{b} \\ 0 \end{bmatrix} \tag{4.1.3}$$

and obtain $\mathbf{y} = \begin{bmatrix} 0 \\ \mathbf{x} \end{bmatrix}$.

The *HHL* quantum algorithm has five main steps:

- State preparation

- Quantum Phase Estimation (QPE)
- Ancilla rotation
- Uncomputation (iQPE)
- Measurement

Each step will now be detailed, aiming to understand the overall evolution of the quantum state. A circuit representation of the full algorithm is depicted in Fig. 4.1.1.

State Preparation

As previously mentioned, we have a total of $n_b + n_c + n_a$ qubits, with $n_a = 1$ representing the ancilla qubit. Similar to most quantum algorithms, all qubits are initially set to $|0\rangle$. Thus the initial state will be

$$|\psi_{initial}\rangle = |0\rangle_b |0\rangle_c |0\rangle_a.$$

During this step, the objective is to rotate the b-register to encode the values of the vector $\mathbf{b}$ into the amplitudes of the computational basis states. Specifically,

$$\beta_0|0\rangle + \beta_1|1\rangle + \cdots + \beta_{N-1}|N-1\rangle = |b\rangle$$

In practice, the vector encoded in the b-register will be $\frac{\mathbf{b}}{\|\mathbf{b}\|}$, as the normalization constraint must be maintained. To achieve this, we require an efficient procedure for preparing the state $|b\rangle$, otherwise, the exponential speedup could be compromised. There is a significant body of work on state preparation, crucial for many quantum algorithms. Generally, methods to prepare an arbitrary state often have exponential complexity in terms of the number of qubits, n_b, which would undermine any quantum advantage offered by the HHL algorithm. Sub-exponential resource algorithms have been devised for certain quantum states or through approximation techniques [8]. For

example, quantum states characterized by efficiently integrable probability distributions can be prepared with polynomial complexity in n_b [38]. Another approach could be approximating the state preparation using matrix product state [8, 39]. However, for the purpose of studying and understanding the algorithm in a general context, we will assume the existence of an oracle capable of efficiently performing this operation. In particular, we call the *StatePreparation* class available in Qiskit [7], based on Ref. [40].
After the state preparation, we obtain the state

$$|\psi_1\rangle = |b\rangle_b|0\rangle_c|0\rangle_a. \tag{4.1.4}$$

Quantum Phase Estimation

The Quantum Phase Estimation (QPE) is the primary subroutine of the HHL algorithm. It is also the most complex part, as it comprises three components, as discussed in Sec. 3.3. The analysis follows the same principles as the previous section; however, in this case, we aim to encode the *eigenvalues* of the matrix A into the c-register. Thus, we require an additional step to build the unitary operator $U = e^{iAt}$ for use during the controlled rotation step. This is possible through Hamiltonian Simulation algorithms we discussed in Sec. 3.4. By following this procedure, the phase associated with the eigenvalues of the unitary operator U will be proportional to the eigenvalues of A. This will become clearer in the subsequent analysis.
Following the same procedure in Sec. 3.3, we start by applying an Hadamard transform in the c-register, to create a superposition of clock qubits, namely:

$$|\psi_2\rangle = \frac{1}{2^{\frac{n_c}{2}}}|b\rangle_b \sum_{k=0}^{2^{n_c-1}} |k\rangle|0\rangle_a. \tag{4.1.5}$$

At this point, controlled U^{2^j} are applied to $|b\rangle_b$ using the j_{th} clock qubit as control. We start by the ideal scenario, where $|b\rangle$ is an eigenvector of U with eigenvalues $e^{2\pi i\varphi}$

$$U|b\rangle = e^{2\pi i\varphi}|b\rangle. \tag{4.1.6}$$

Therefore, after these operations, we obtain the state

$$
\begin{aligned}
|\psi_3\rangle &= |b\rangle_b \frac{1}{2^{n_c}} \Big(|0\rangle + e^{2\pi i 2^0 \varphi}|1\rangle\Big)\Big(|0\rangle + e^{2\pi i 2^1 \varphi}|1\rangle\Big) \cdots \Big(|0\rangle + e^{2\pi i 2^{n_c-1} \varphi}|1\rangle\Big)|0\rangle_a \\
&= |b\rangle_b \frac{1}{2^{n_c}} \sum_{k=0}^{2^{n_c-1}} e^{2\pi i \varphi k}|k\rangle|0\rangle_a
\end{aligned}
\tag{4.1.7}
$$

Now, applying the iQFT, we menage to encode the phases φ in the c-register

$$|\psi_4\rangle = |b\rangle_b|2^{n_c}\varphi\rangle_c|0\rangle_a. \tag{4.1.8}$$

As explained in Sec. (3.3), each clock qubit j corresponds to a U^{2^j} controlled operation, adding bit precision to the representation of the phase φ.
In this context, the unitary operator U and the matrix A are related by the equation $U = e^{iAt}$, where t is the evolution time parameter of the Hamiltonian simulation, which will be discussed later. Assuming that $|b\rangle$ is an eigenvector of U, i.e.,

$$
\begin{aligned}
|b\rangle &= |u_i\rangle \\
U|b\rangle &= e^{i\lambda_i t}|u_i\rangle
\end{aligned}
\tag{4.1.9}
$$

we have $\varphi = \lambda_i t/2\pi$. Thus, equation (4.1.8) becomes:

$$|\psi_4\rangle = |b\rangle_b|2^{n_c}\lambda t/2\pi\rangle_c|0\rangle_a. \tag{4.1.10}$$

However, in a general case, $|b\rangle$ is not an eigenvector of U. As outlined in Sec. (3.3), the preparation of the eigenvector $|u\rangle$ can be avoided by integrating randomness into the procedure [13]. $|b\rangle$ can be rewritten in the eigenvector basis as

$$|b\rangle = \sum_{i=0}^{2^{n_b-1}} b_i|u_i\rangle.$$

Thus, we have

$$|\psi_4\rangle = \sum_{i=0}^{2^{n_b-1}} b_i |u_i\rangle |2^{n_c}\lambda_i t/2\pi\rangle_c |0\rangle_a. \tag{4.1.11}$$

We have now successfully encoded the eigenvalues of the matrix A within the c-register. Note that we did not store the exact eigenvalues λ_i, but rather a scaled version $\tilde{\lambda_i}$ given by

$$\tilde{\lambda_i} = 2^{n_c}\lambda_i t/2\pi. \tag{4.1.12}$$

The time evolution constant t serves as a normalization parameter that must be set appropriately for the QPE to function correctly. Specifically, the phases φ must satisfy the constraint

$$\varphi \leq 1. \tag{4.1.13}$$

In our case, this constraint translates to

$$\lambda_i t \leq 2\pi. \tag{4.1.14}$$

A detailed analysis of this parameter's fine-tuning will be discussed in a later section.

Ancilla rotation

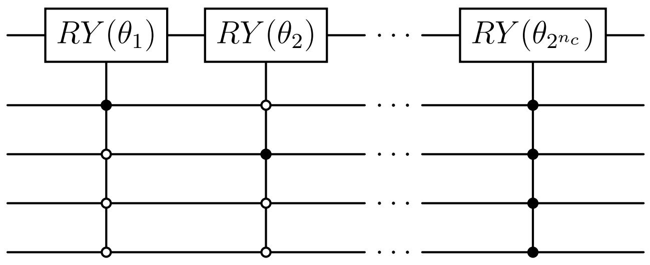
Now we rotate the ancilla qubit $|0\rangle_a$ based on the encoded eigenvalues $\tilde{\lambda_i}$ in the c-register, such that:

$$|\psi_5\rangle = \sum_{i=0}^{2^{n_b-1}} b_i |u_i\rangle |\tilde{\lambda_i}\rangle_c |0\rangle_a \Big(\sqrt{1 - \frac{C^2}{\tilde{\lambda_i}^2}} |0\rangle_a + \frac{C}{\tilde{\lambda_i}} |1\rangle_a\Big). \tag{4.1.15}$$

This step is also known as eigenvalue inversion, as after the rotation, we successfully store the inverted eigenvalues in the amplitude of the ancilla qubit. We are specifically interested in the case where the ancilla collapses to the state $|1\rangle$, corresponding to the amplitude $\frac{C}{\tilde{\lambda_i}}$ we are aiming for. The rotation of the ancilla qubit from $|0\rangle_a$ to $\sqrt{1 - \frac{C^2}{\tilde{\lambda_i}^2}}|0\rangle_a + \frac{C}{\tilde{\lambda_i}}|1\rangle_a$ can be achieved through an RY rotation of the type (2.3.1), where the rotation angle is

$$\theta = 2\arcsin\left(\frac{C}{\tilde{\lambda}_i}\right). \tag{4.1.16}$$

The variable C in Eq. (4.1.15) is an input parameter of the algorithm, which should be set as large as possible to maximize the probability of measuring $|1\rangle_a$. However, we must adhere to the constraint $C \leq \tilde{\lambda}_i$ imposed by the rotation angle in Eq. (4.1.16). Since we do not know the exact $\tilde{\lambda}_i$ beforehand to perform the rotation, we apply 2^{n_c} multi-controlled RY operations, one for each possible value of $\tilde{\lambda}_i$. We use the entire c-register as control qubits and the binary string corresponding to the current $\tilde{\lambda}_i$ as the control state. This way, the RY rotation is applied only if the corresponding state has a non-zero probability. Since the c-register contains a basis encoding of our eigenvalues, the rotation will succeed only in the cases corresponding to the true $\tilde{\lambda}_i$. The corresponding circuit representation is shown below, where $\theta_i = 2\arcsin\left(\frac{C}{i}\right)$:



Note that this approach requires $O(2^{n_c})$ multi-control gates, resulting in a linear scaling with the problem size N. The literature offers efficient implementations of this step for specific cases using approximations, such as the one used in [8] based on *Chebyshev interpolation*. However, as previously mentioned, the aim of this work is to investigate the implementation and limitations of the algorithm in a more general context.

Uncomputation and Measurement

At this point, our state $|\psi\rangle$ closely resembles the classical solution we seek. However, during the process, we applied many multi-qubit gates, resulting in entanglement between the two quantum registers. We need to uncompute the state to obtain the correct results by measuring in the computational basis where the b-register and c-register are unentangled. This is achieved using an inverse Quantum Phase Estimation (iQPE). After this step, the qubits in the c-register revert to the state $|0\rangle$, and the final state is

$$|\psi_6\rangle = \sum_{i=0}^{2^{n_b-1}} b_i|u_i\rangle|0\rangle_c\Big(\sqrt{1-\frac{C^2}{\tilde{\lambda}_i^2}}|0\rangle_a + \frac{C}{\tilde{\lambda}_i}|1\rangle_a\Big) \tag{4.1.17}$$

Now, we can measure the ancilla qubit and discard the cases where it collapses to the state $|0\rangle_a$. Finally, at the end of this process, we obtain the desired solution $|x\rangle$, successfully stored in the b-register:

$$|\psi\rangle = \frac{1}{\sqrt{\sum_{i=0}^{2^{n_b-1}} |\frac{b_i}{\lambda_i}|^2}}|x\rangle_b|0\rangle_c|1\rangle_a. \tag{4.1.18}$$

In Fig. 4.1.1 we report the circuit representation of the full algorithm.

4.2 Computing Observables

As explained in [4], the algorithm produces a quantum-mechanical representation $|x\rangle$ of the desired vector $\mathbf{x}$. To fully decode all components of $\mathbf{x}$ would typically require repeating the procedure at least N times. However, the primary interest often lies in computing some expectation value $\langle x|M|x\rangle$, where M represents a linear operator. By translating M into a quantum-mechanical operator and executing the corresponding quantum measurement, we can derive an estimate of the expectation value. This allows the extraction of various characteristics of the vector $\mathbf{x}$, including its normalization, distribution of weights across different state spaces and more.

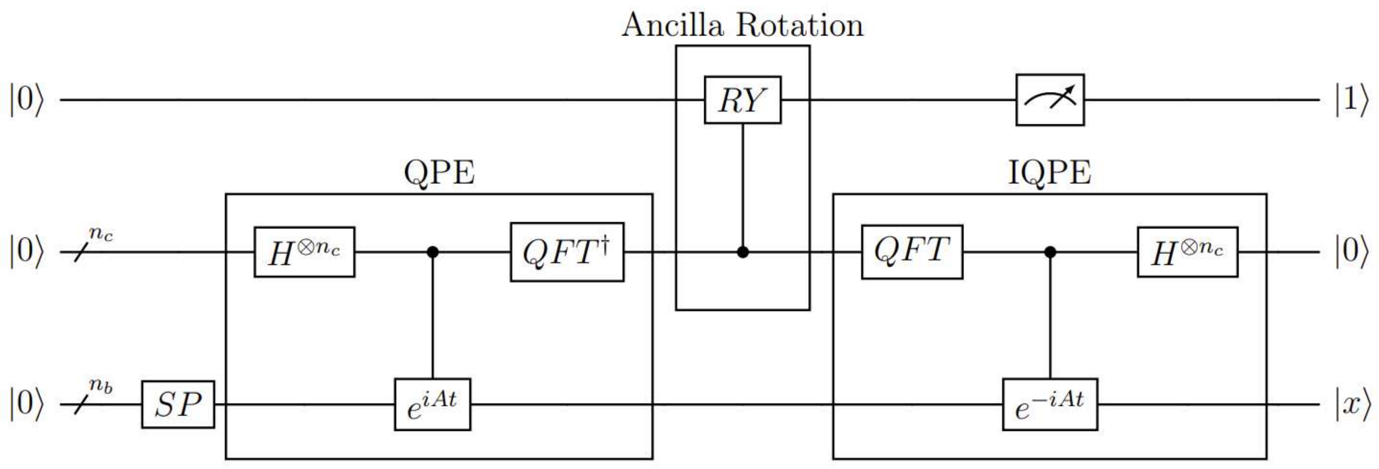


Figure 4.1.1: This circuit depicts the whole HHL algorithm. We can distinguish all the main steps discussed in the previous section.

Solution norm $\|\mathbf{x}\|$

The norm of the solution vector can be employed to determine the actual values of other observables, as our access is limited to the normalized solution vector $\mathbf{x}/\|\mathbf{x}\|$. Another important aspect to emphasize is that, typically, since the normalized vector $\mathbf{b}/\|\mathbf{b}\|$ is loaded during the state preparation step, we have access to the properties of the state $\mathbf{x'} = \mathbf{x}/\|\mathbf{b}\|$

As proven in [4], the normalization $\|\mathbf{x}\|$ can be estimated using the probability of seeing $|1\rangle_a$ when measuring the ancilla qubit. More rigorously, let $M_1 = I_{n_b} \otimes |1\rangle_a\langle 1|_a$. Where $|1\rangle_a\langle 1|_a$ is the projector of the ancilla qubit into the state $|1\rangle$. We recall the final state of the system after the application of HHL circuit:

$$|\psi\rangle = \sum_{i=0}^{2^{n_b-1}} b_i|u_i\rangle|0\rangle_c\Bigg(\sqrt{1-\frac{C^2}{\tilde{\lambda}_i^2}}|0\rangle_a + \frac{C}{\tilde{\lambda}_i}|1\rangle_a\Bigg) \tag{4.2.1}$$

Then the probability of seeing the ancilla qubit in $|1\rangle_a$ after the measurement is

$$\begin{aligned} P_1 &\equiv \langle\psi|M_1^\dagger M_1|\psi\rangle \\ &= \frac{C^2}{\|\mathbf{b}\|^2}\sum_{i=0}^{2^{n_b}-1}\left(\frac{b_i}{\tilde{\lambda}_i}\right)^2 = \frac{(C2^{n_c}t/2\pi)^2\,\|\mathbf{x}\|^2}{\|\mathbf{b}\|^2} \end{aligned} \tag{4.2.2}$$

and we finally find

$$\|\mathbf{x}\| = \frac{2^{n_c}t\sqrt{P_1}\,\|\mathbf{b}\|}{2\pi C}. \tag{4.2.3}$$

Absolute average

Similarly, following the work presented in [8], if we define $M_{0,1} = |0\rangle_b\langle 0|_b \otimes |1\rangle_a\langle 1|_a$ and $|\psi'\rangle = (H^{\otimes n_b} \otimes I)|\psi\rangle$, where ψ is the state in (4.2.1), then

$$P_{0,1} \equiv \langle\psi'|M_{0,1}^\dagger M_{0,1}|\psi'\rangle = \left|\frac{2\pi C}{2^{n_c}t\sqrt{N}\,\|\mathbf{b}\|}\sum_{i=0}^{N-1} x_i\right|^2 \tag{4.2.4}$$

which leads to

$$\overline{\mathbf{x}} \equiv \frac{\sum_{i=0}^{N-1} x_i}{N} = \frac{\sqrt{P_{0,1}}2^{n_c}t\,\|\mathbf{b}\|}{2\pi C\sqrt{N}}. \tag{4.2.5}$$

Thus, the absolute average of the solution can be computed by performing an additional Hadamard transform on the b-register. Then, we can measure each qubit in the b-register in the computational basis, along with measuring the ancilla qubit. The desired solution will be stored in the amplitude corresponding to the state $|0\rangle_b|1\rangle_a$, up to a constant shown in (4.2.5).

Chapter 5

HHL implementation

The main aim of this project is the implementation of the HHL algorithm within the Qiskit framework. In the previous chapter, we described all the subroutines of the algorithm. Now, in Sec. 5.1, we introduce Qiskit and briefly describe the main function implemented along with all its input parameters. Then, we discuss the challenges encountered during the simulations, highlighting the limitations of exactly simulating HHL algorithm on a classical hardware. In Sec. 5.2, we test the algorithm in various scenarios, fine-tuning the algorithm's input parameters for arbitrary-sized problems. Finally, we discuss different aspects and limitations of the HHL algorithm, considering the possible future developments of this work, with applications in the aerospace sector.

5.1 Implementation

5.1.1 Qiskit framework

IBM Quantum Experience is a cloud-based platform that allows researchers and the general scientific community to access and run experiments on real quantum computers. The Quantum Information Science Kit, or Qiskit [7], is an open-source software developed by IBM. Qiskit provides a comprehensive and flexible framework for building quantum algorithms, simulating them on classical computers, and executing them on actual quantum hardware. It supports a wide range of quantum programming tasks, from low-level quantum circuit construction to high-level algorithm implementation, making it an invaluable tool for quantum computing research and application. The primary objective of this study is to implement the Harrow-Hassidh-Lloyd (HHL) quantum algorithm within the Qiskit framework. The majority of existing literature on the HHL algorithm primarily focuses on theoretical analyses, which include discussions on predicted scaling and potential applications of the algorithm. However, the available implementation of the HHL algorithm are often restricted to small matrices (e.g 2×2) [37]. Although some full implementations of the HHL algorithm are available online [41], our work entails a comprehensive implementation that allows for full control over input parameters and all associated subroutines of the algorithm, regardless the size of the targeted matrix. This approach follows and expands upon the work presented in [37], permitting simulations for arbitrary problem sizes. This approach enables the simulation of the algorithm across various contexts and facilitates a detailed examination of several aspects of the algorithm that are not thoroughly covered in theoretical publications, as referenced in [4].

5.1.2 Input parameters

We show now the main function of the interface, my_HHL. This function builds the HHL quantum circuit to solve the linear system $A\vec{x} = \vec{b}$ and takes as inputs the following parameters:

- A : the matrix A of the linear system;
- b: the vector **b** of the linear system;
- n_b : the number of qubits in the b-register used to encode **b**;
- n_c : the number of qubits in the c-register used to represent the eigenvalues of A;

- *n_shots* : the number of times the measurement is repeated for each qubit;
- t : the time evolution parameter of the Hamiltonian simulation;
- C: the normalization constant in Eq. (4.1.15), used to maximize the probability of obtain $|1\rangle_a$ in the measurement of the ancilla qubit;
- *Suzuki_Trotter* : a boolean parameter, if set to True, e^{iAt} is simulated using Suzuki-Trotter decomposition, otherwise the Hamiltonian simulation is performed exactly, as discussed in Sec. 3.4.3;
- *Absolute_Average*: a boolean parameter, if set to True, an additional Hadamard operation is performed on the b-register, which is measured along with the ancilla qubit. Otherwise, only the ancilla qubit is measured.

As an output, the function returns the corresponding quantum circuit and the results of the measurements on the final state. Then, it is possible to compute the norm or the absolute average of the solution, as discussed in Sec. 4.2.

5.1.3 Examples with a 2 × 2 matrix

Let us examine a simple 2×2 matrix example, following the framework introduced in the previous section. This example will be useful for familiarizing ourselves with the various steps of the algorithm. In this scenario, the linear system of equations is defined by the matrix A and the vector $\mathbf{b}$, which determine the problem's dimensions. Consider:

$$A = \begin{bmatrix} 2/5 & 0 \\ 0 & 4/5 \end{bmatrix} \qquad \mathbf{b} = [1, 1]$$

In this instance, the matrix A is already Hermitian; otherwise, it would be necessary to apply the transformation outlined in Eq. (4.1.2), which would double the size of the problem. Additionally, the vector $\mathbf{b}$ is already normalized, eliminating the need for further adjustments. Given that $N = 2$, we require $n_b = 1$ qubits in the b-register to store the vector $\mathbf{b}$ during the state preparation step. As this is a small-scale toy example and the matrix is diagonal, it is straightforward to determine the eigenvalues of A:

$$\lambda_0 = 2/5,$$
$$\lambda_1 = 4/5.$$

Consequently, we can precisely adjust our input parameters for optimal encoding, this means choosing n_c, t such that $\tilde{\lambda}_i = 2^{n_c}\lambda t/2\pi$ (i.e the scaled eigenvalues stored in the c-register) are integer numbers. This results in a perfect QPE. For this implementation, we choose $n_c = 2$ qubits for the c-register and set $t = \frac{5}{4}\pi$, $C = 1$. The resulting quantum circuit implementing the HHL algorithm is shown in Fig. (5.1.1). The circuit is now simulated

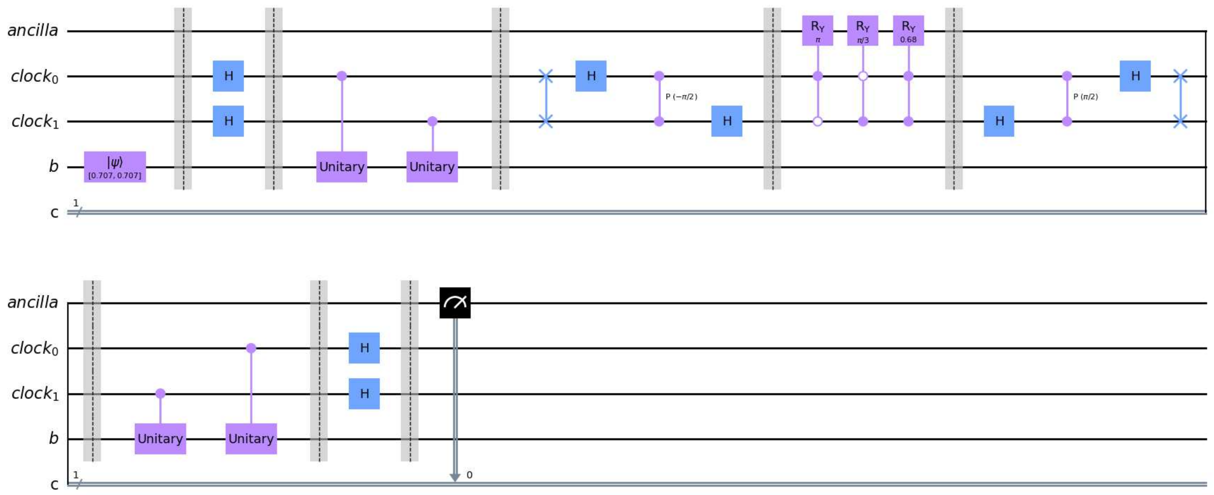


Figure 5.1.1: HHL quantum circuit which solves a 2×2 linear system problem using $n_b = 1$ and $n_c = 2$ qubits. In this case we are interested in computing the solution norm $\|\mathbf{x}\|$, therefore we only need to measure the ancilla qubit.

in the IBM-Q system. However, if we are interested in the solution norm, only the ancilla needs to be measured, as discussed in Sec. 4.2. We then use the probability P_1 of observing the ancilla in the state $|1\rangle_a$ combined with substituting all the input parameters into Eq. (4.2.3) to determine the solution norm. The resulting statistics of the simulation can bee seen in Fig. 5.1.2. In this instance, executing the circuit $n_{shots} = 1000$ times, we obtain $\|\mathbf{x}\|_{HHL} = 2.7225$ to compare with the classical solution $\|\mathbf{x}\|_{classical} = 2.7951$.

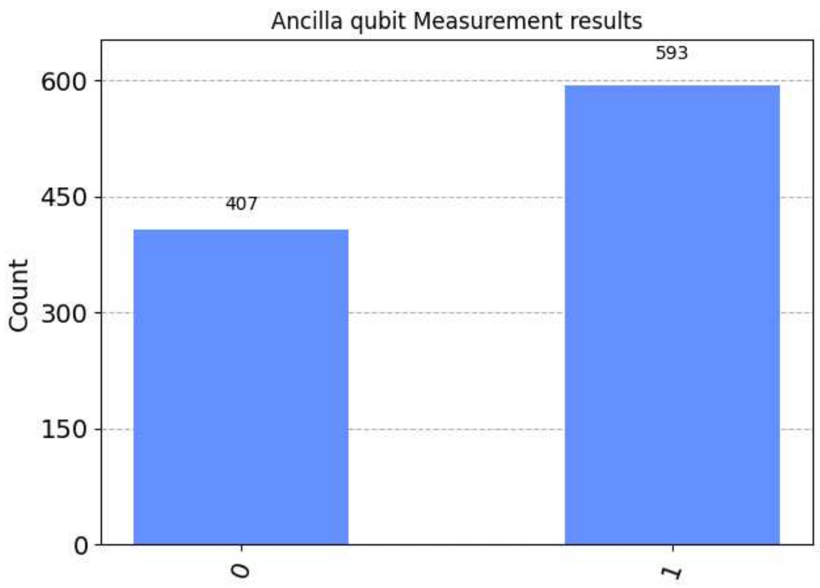


Figure 5.1.2: Results of the measurements on the Ancilla qubit, since we are interested in computing the solution norm, we only have to consider the probability associated to the state $|1\rangle$.

Conversely, to compute the average solution, an additional Hadamard transform to the b-register is incorporated, and the final circuit configuration is depicted in Fig. 5.1.3. Subsequently, as illustrated in the circuit, both the ancilla and the b-register are measured, specifically targeting the probability $P_{0,1}$ corresponding to the state $|0\rangle_b|1\rangle_a$, which can be extracted from the counts in Fig. 5.1.4 . Given that the b-register contains only $n_b = 1$ qubit, our focus is solely on the state $|01\rangle$. Finally, we apply Eq. (4.2.5) to obtain the desired result. In this case, given the values of the counts in Fig. 5.1.4, we obtain $\overline{\mathbf{x}}_{HHL} = 1.896$ to compare with the classical solution $\overline{\mathbf{x}}_{classical} = 1.875$. Clearly, our algorithm implemented using Qiskit can

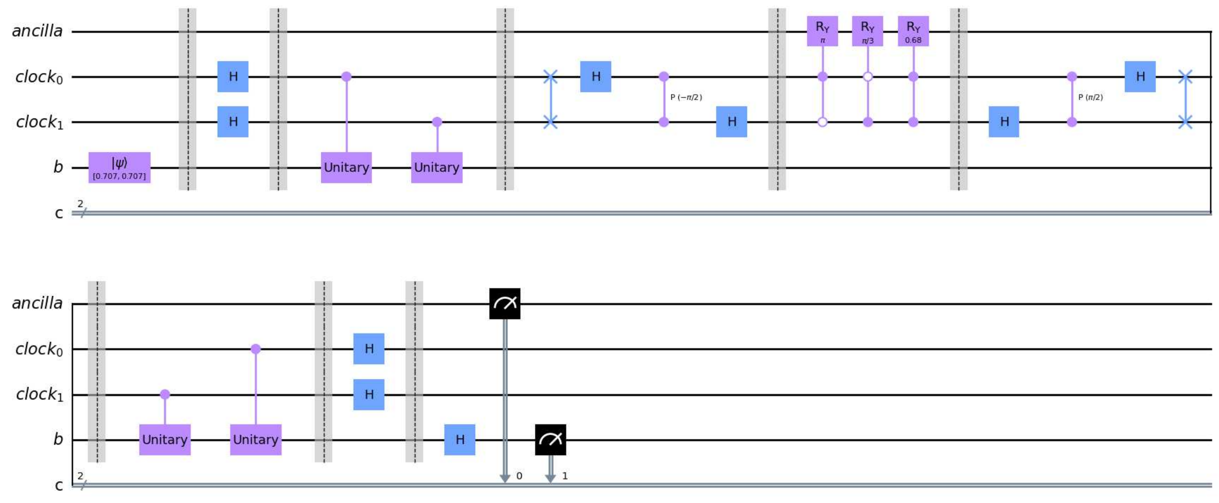


Figure 5.1.3: HHL quantum circuit which solves a 2×2 linear system problem using $n_b = 1$ and $n_c = 2$ qubits. Here we are interested in computing the absolute average $\overline{\mathbf{x}}$ of the solution. Thus, an additional Hadamard operation on the b-register is required. Finally, we need to measure the ancilla qubit and the qubits inside the b-register.

handle matrices of arbitrary size, providing direct control over all the input parameters discussed in Sec. 5.1.2. We present these simple examples to help the reader familiarize themselves with the main steps of the algorithm and to visualize the corresponding quantum circuits, which become too large to depict as N increases. Examples with larger N will be further explored in the next sections.

5.1.4 Challenges in Simulating the HHL Algorithm

One of the main challenges we encountered during these analyses was the limitation imposed by the computational time required for simulations. This issue arises because we utilized the Qiskit framework as an *emulator* for quantum circuits. Unlike actual quantum computers, which can execute quantum operations in parallel and inherently exploit quantum parallelism, emulators simulate these operations classically. Firstly, to perform a simulation, we need to construct the quantum circuit. This involves defining and

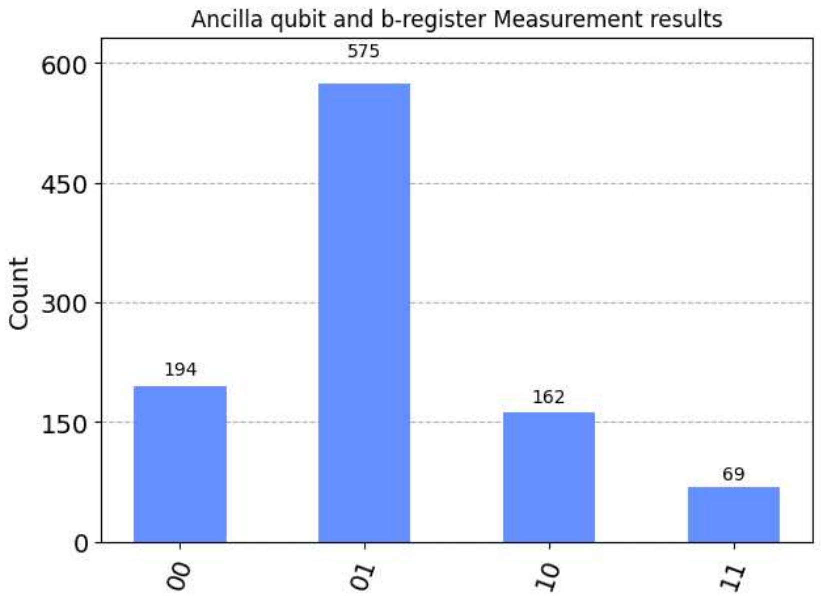


Figure 5.1.4: Results of the measurements on the Ancilla and the b-register, since we are interested in computing the Absolute Average, we only have to consider the probability associated to the state $|01\rangle$, as discussed in Sec.4.2.

implementing all the necessary quantum gates and operations. In a classical emulator, these gate operations are simulated through matrix multiplications. Each gate operation corresponds to a matrix that must be multiplied with the state vector representing the quantum system. As the problem size increases, the number of gates grows exponentially, leading to a significant increase in the computational resources required. This is because we implemented the ancilla rotation and the Hamiltonian simulation exactly. Thus, even for relatively small quantum systems, the *depth* of the quantum circuit, which is defined as the length of the longest path from the input to the output (or a measurement gate), moving forward in time along each qubit wire [20], becomes a critical factor in determining the simulation time. In Fig. 5.1.5 we report the Circuit Depth of the circuit as N increases, in logarithmic scale. The primary bottleneck in this process is the exponential growth in the size of the matrices involved. For an n-qubit system, the state vector has 2^n entries, and each gate operation is represented by a $2^n \times 2^n$ matrix. As a result, both memory usage and computational time increase exponentially with the number of qubits. This exponential scaling restricts the feasible size of the simulations. In our experiments, we found that practical simulations were limited to systems with $n_b \leq 6$ qubits. Thus, if we choose $n_c = n_b + 1$, we have a total of $n = 1 + n_b + n_c = 14$ corresponding to state vectors of size $2^{14} = 16384$. Furthermore, the cumulative effect of simulating a sequence of gates exacerbates the problem. Each additional gate requires another matrix multiplication, further increasing computational load. In conclusion, while the Qiskit framework provides a powerful tool for simulating quantum algorithms, the classical emulation of quantum operations inherently suffers from scalability issues. These issues manifest as increased simulation times, significantly limiting the size and complexity of problems that can be efficiently simulated. Overcoming these limitations requires either the development of more efficient classical algorithms for emulation or the use of actual quantum hardware as it becomes more accessible and capable. Another approach could be using a Tensor Network emulator for quantum circuits, which can allow the access to instances of significant size. A prominent work in this field is *Quantum Matcha Tea* [20] [42], a Tensor Network emulator developed by the quantum research group at the University of Padua.

5.2 Results: fine-tuning of input parameters for arbitrary size

As we have seen in Sec. 1.4, if we use the FEM to solve 1-D Poison equation, we have to solve a linear system $A\mathbf{x} = \mathbf{b}$ where A is a tridiagonal matrix. The FEM poses an attractive prospect for enhancement using the HHL algorithm for various reasons [5]. Firstly, the vast systems of linear equations inherent in the FEM are generated algorithmically, as opposed to being directly supplied as input. This eliminates efficiency issues associated with the necessity to access data via a quantum RAM. Secondly, the FEM naturally gives rise to sparse systems of linear equations, a condition often essential for

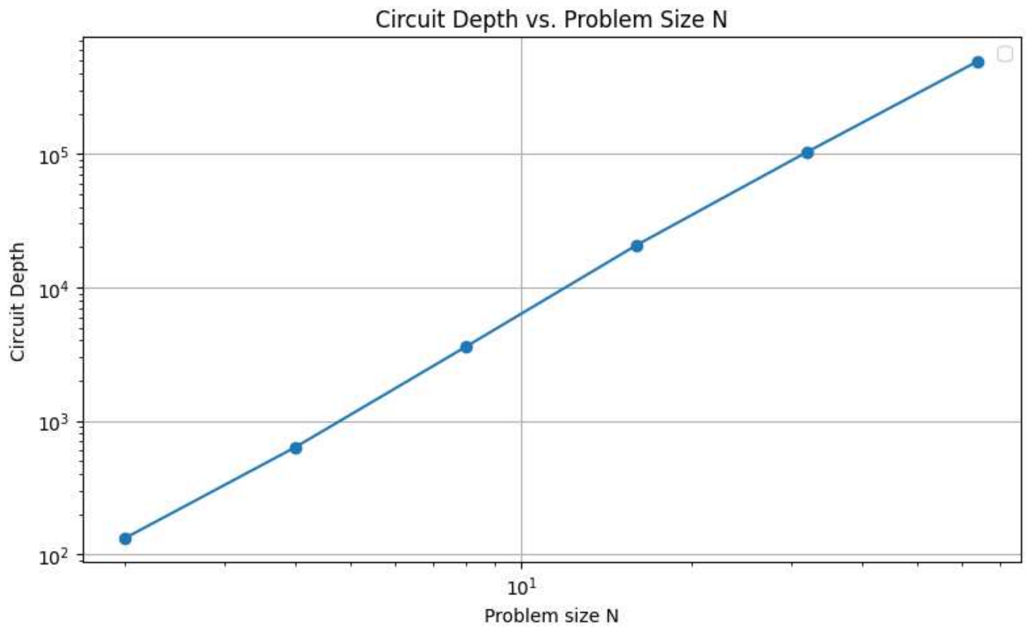


Figure 5.1.5: Circuit Depth of the HHL circuit implemented as N increases. Both N and the Circuit Depth are represented on a logarithmic scale, demonstrating exponential scaling. This scaling comes from the ancilla rotation step, which has been implemented "exactly", requiring 2^{n_c} multi-control gates. To achieve the logarithmic scaling with N shown in the original paper [4], also an efficient Hamiltonian simulation in required.

achieving quantum speed-up through the Hamiltonian Simulation algorithm, which is a key module in HHL. Furthermore, these matrices often have a well-defined structure, making them efficiently row-computable, which is a constrained in many Hamiltonian simulation algorithms, as we have seen in Sec. 3.4. In this section, we conduct an analysis on the fine-tuning of input parameters of HHL algorithm, focusing on the number n_c of qubit inside the c-register and the evolution time parameter t of the Hamiltonian Simulation subroutine.

5.2.1 Selection of Input Matrices for Simulations

As outlined in Sec 1.4, the application of the Finite Element Method (FEM) often leads to the formation of sparse linear equation systems. A common scenario occurs when FEM is used to solve the one-dimensional Poisson equation, resulting in tridiagonal matrices. In this study, we particularly examine Toeplitz tridiagonal matrices defined as:

$$\mathrm{A} \equiv \begin{pmatrix} a & c & 0 & 0 \\ c & a & c & 0 \\ 0 & c & a & c \\ 0 & 0 & c & a \end{pmatrix}, \tag{5.2.1}$$

where $a, c \in \mathbb{R}$. The eigenvalues of such matrices can be exactly described as [8]:

$$\lambda_j = a - 2c\cos\left(\frac{j\pi}{N+1}\right), \quad j = 1, 2, ..., N \tag{5.2.2}$$

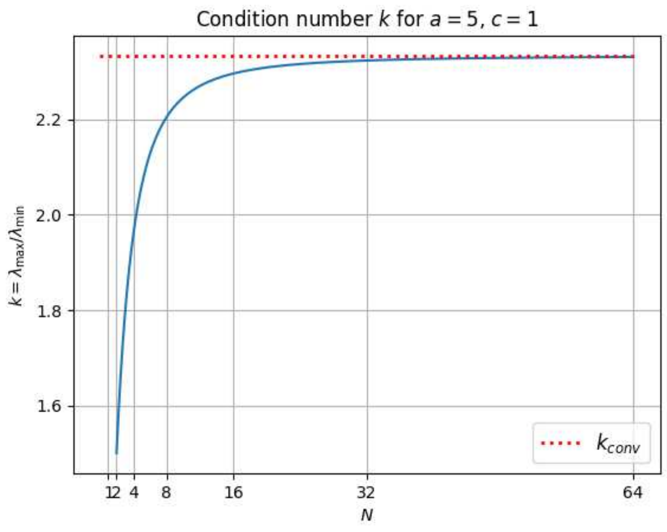
allowing for the pre-estimation of the matrix condition number and the distribution range of the eigenvalues. This predictive capability is particularly useful for adjusting the input parameters as we discuss in Sec. 5.2.3. In particular, the condition number k of the matrix has a defined upper bound k_{conv} when a and c are fixed, which is defined by

$$k_{\text{conv}} = \frac{a+2c}{a-2c}. \tag{5.2.3}$$

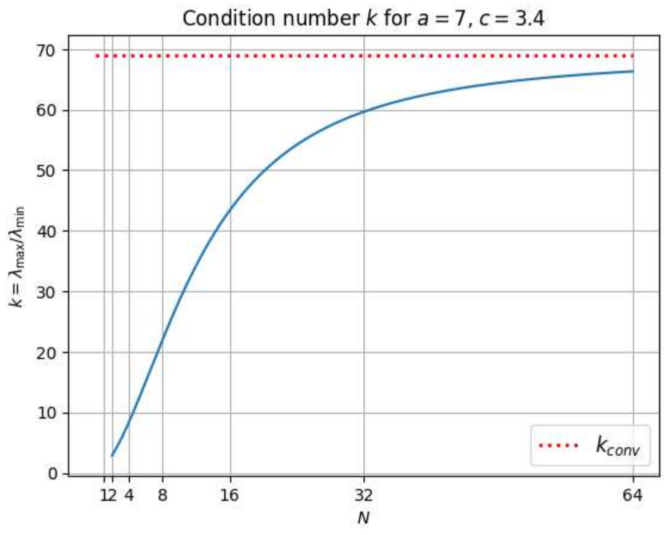
This implies that a and c can be set such that k falls within a specific range. This flexibility allows us to study the behavior of the algorithm under different conditions, especially when k is small compared to the size of the matrix, namely $k = O(\log_2 N)$, which is the regime where HHL achieves an exponential speedup over its classical counterpart [4], and also when k is higher. We consider two distinct cases:

- For $a = 5$ and $c = 1$, the condition number reaches its plateau k_{conv} for $N = 32$ and its relatively small with respect to N, as shown in 5.2.1a.
- For $a = 7$ and $c = 3.4$, the condition number increases with N and does not converges for $N \leq 64$, as illustrated in Fig. 5.2.1b.

For both the cases considered, we used $\mathbf{b} = [1, 1]$.



(a) Condition number for $a = 5$, $c = 1$



(b) Condition number for $a = 7$, $c = 3.4$

Figure 5.2.1: Relationship between the matrix size N and the condition number k of a Toeplitz tridiagonal matrix, for fixed a and c as N increases. The red dotted line represents the upper bound of k defined in Eq. 5.2.3.

5.2.2 The number of clock-qubits $\mathbf{n}_c$

Selecting the appropriate number of qubits to represent the eigenvalues in the HHL algorithm is crucial, as it significantly influences the precision and accuracy of the obtained solution. Indeed, we have to consider different factors to properly set n_c:

- *The problem size N*: a matrix of dimension N results in N (generally) different eigenvalues, which necessitate at least $n_c = n_b = \log_2(N)$ qubits to be represented.
- *The condition number $k = \frac{\lambda_{\max}}{\lambda_{\min}}$* (ratio of the largest to the smallest eigenvalue) influences the required precision. Higher condition numbers generally require more qubits to accurately represent the eigenvalues and achieve a precise solution. Thus, for large k, we need at least $n_c = \lceil \log_2(k) \rceil$.
- *Precision Requirement n*: The number of qubit n_c used to represent the eigenvalues directly affects the precision of the quantum phase estimation. Utilizing more qubits enhances bit precision but also elevates resource demands, as shown in Eq. (3.3.7) and demonstrated in [13], where n refers to a 2^{-n} accuracy.

A general way to set the parameter n_c and consider all the previous factors is given by

$$n_c = \max\left(n_b + 1, \left\lceil \log_2(k) + 1 \right\rceil, n + \left\lceil \log_2\left(2 + \frac{1}{2\epsilon}\right)\right\rceil\right). \tag{5.2.4}$$

However, as outlined in [4], it is crucial that the number n_c is logarithmic with respect to the problem size N to achieve an exponential speedup over classical methods.

Simulations results

To determine the optimal setting for the parameter n_c, we focus on small matrices due to the computational limitations outlined in Sec. 5.1.4. The number of qubits in the clock-register significantly influences both the depth of the quantum circuit and the dimension of the full state vector, rendering the analysis of larger matrices prohibitively expensive. Another important aspect concerns the probabilistic nature of our analysis. We recall that the final state of our algorithm is expressed as:

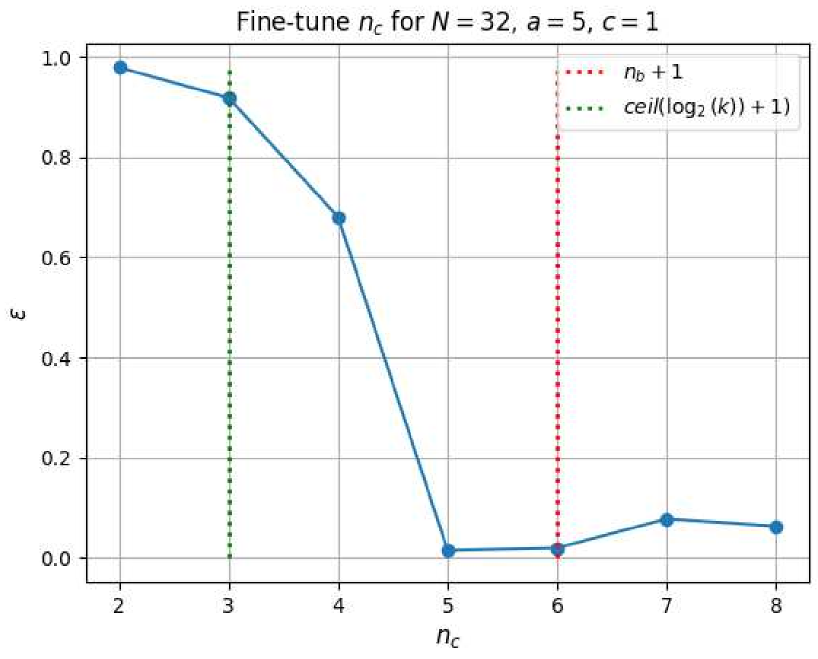
$$|\psi\rangle = \sum_{i=0}^{2^{n_b-1}} b_i|u_i\rangle|0\rangle_c \left(\sqrt{1-\frac{C^2}{\tilde{\lambda}_i^2}}|0\rangle_a + \frac{C}{\tilde{\lambda}_i}|1\rangle_a\right), \tag{5.2.5}$$

where $\tilde{\lambda}_i = 2^{n_c}\lambda_i t/2\pi$ represents a scaled version of the true eigenvalues of the input matrix A. Higher values of n_c lead to a lower probability of measuring the ancilla qubit in the state $|1\rangle_a$. Consequently, although increasing the number of qubits in the clock-register enhances the bit-precision of the eigenvalue representation, setting n_c too high may actually reduce the precision of the solution if the number of algorithm executions, or "shots", is not correspondingly increased to ensure robust statistical outcomes. We tested the algorithms for different n_c, considering the two different cases presented in Sec. 5.2.1. We utilized the percent error in the Absolute Average of the HHL solution, $\overline{x}_{\text{HHL}}$, with respect to the classical solution $\overline{x}_{\text{cl}}$ as a parameter for assessing precision, namely:

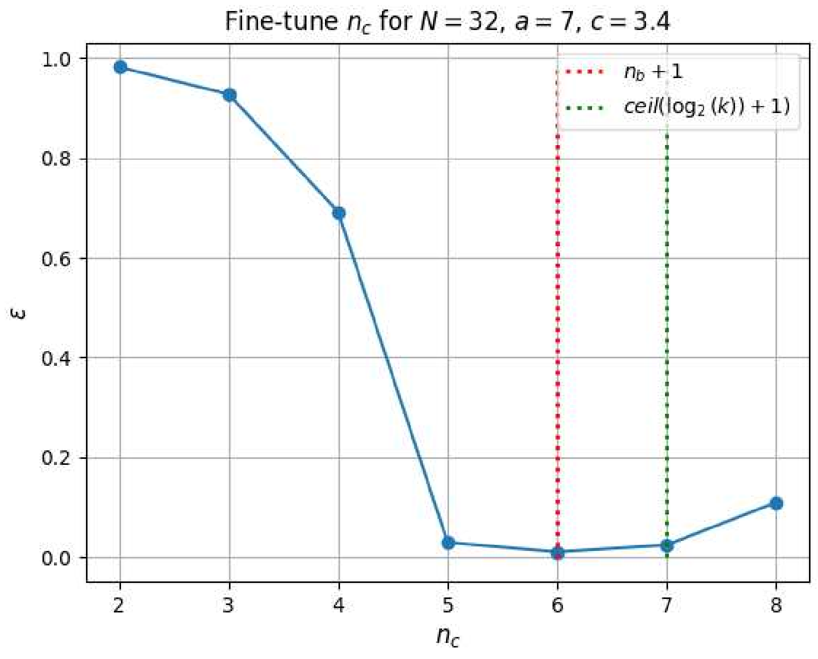
$$\epsilon = \frac{(\overline{x}_{\text{HHL}} - \overline{x}_{\text{cl}})}{\overline{x}_{\text{cl}}}, \tag{5.2.6}$$

where $\overline{x}_{\text{HHL}}$ is computed through Eq. 4.2.5. In Fig. 5.2.2 and Fig. 5.2.3, we can clearly see that if $n_c \leq n_b$, the percent error ϵ is higher because we do not have enough qubits to represent all the eigenvalues. However, as shown in the plots, if we carefully choose a "good encoding" through the parameter t (we will see how in the next section) and if the condition number of the matrix is of the order of the matrix size N, we achieve good accuracy just by setting $n_c = n_b + 1$. As we can see in Fig. 5.2.1, if we consider the case for $a = 7$ and $c = 3.4$, the condition number is higher (but of the same order) of N. However, we get worse results increasing the number of qubits. This is because we kept the same number of shots (number of times the circuit is executed) for all the simulations, and, as explained before, for higher n_c the probabilty of measuring the ancilla in $|1\rangle_a$ decreases, affecting the statistics.

Further simulations with higher problem sizes N and higher condition number (and thus a larger number of qubits n_c in the c-register from Eq. (5.2.4) should be performed, knowing that performing these simulations classically requires a huge amount of time. Indeed, we expect that for large N and k, it becomes very difficult to find a good encoding of the eigenvalues. Thus, the number of qubits inside the c-register, n_c, plays a crucial role to get precise solutions. For the subsequent analyses, we will always set $n_c = n_b + 1$.

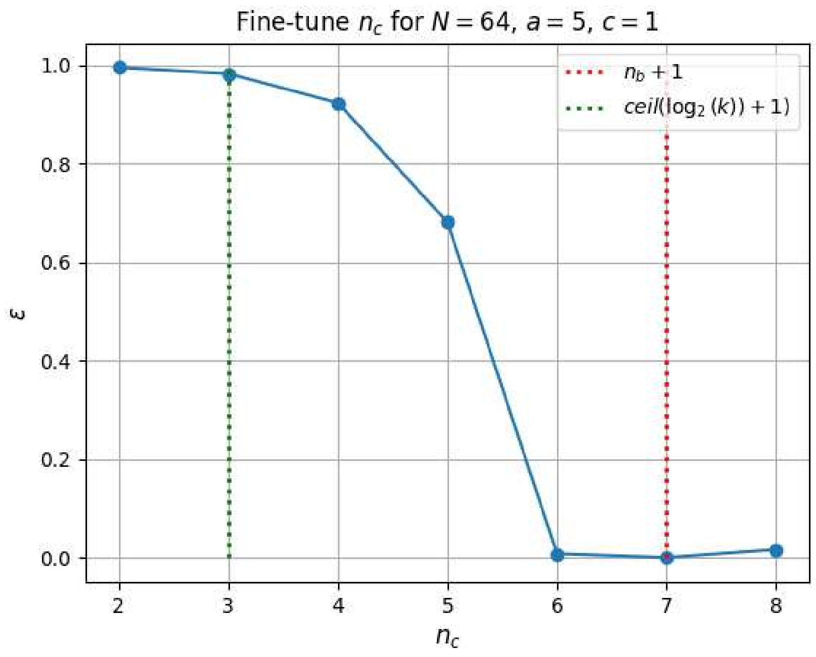


(a) Fine-tuning n_c for a 32×32 matrix defined in Eq. (5.2.1), where $a = 5$, $c = 1$

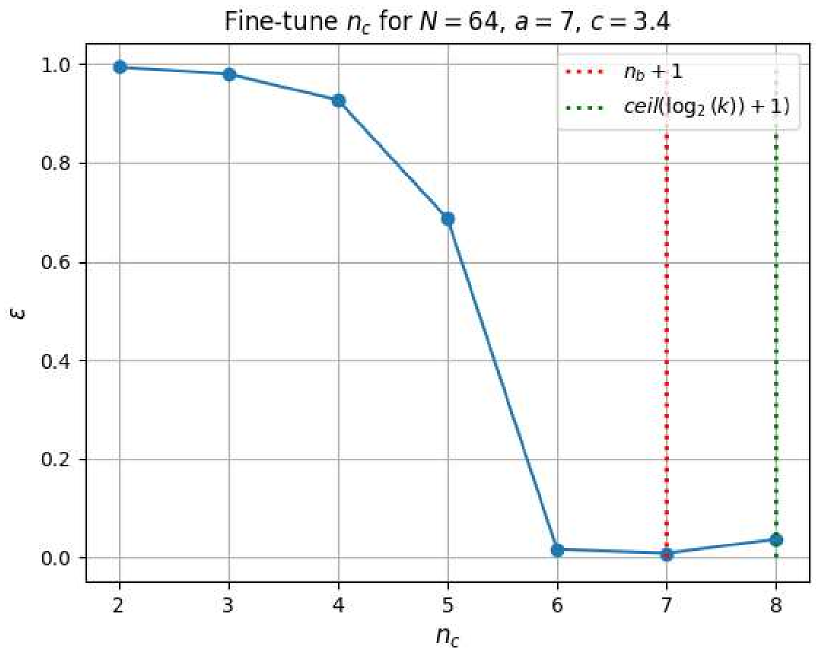


(b) Fine-tuning n_c for a 32×32 matrix defined in Eq. (5.2.1), where $a = 7$, $c = 3.4$

Figure 5.2.2



(a) Fine-tuning n_c for a 64×64 matrix defined in Eq. (5.2.1), where $a = 5$, $c = 1$



(b) Fine-tuning n_c for a 64×64 matrix defined in Eq. (5.2.1), where $a = 7$, $c = 3.4$

Figure 5.2.3

5.2.3 The evolution time parameter t

In the framework of Quantum Phase Estimation (QPE), the phase values φ are constrained within the interval $[0, 1]$. This constraint arises from the periodic nature of phase angles in quantum mechanics and the definition of the eigenvalues of unitary operators.
As shown in the previous chapter, in HHL algorithm this translates to

$$\lambda t \leq 2\pi. \tag{5.2.7}$$

Thus, the parameter t in the Hamiltonian simulation is typically set based on the largest eigenvalue of the matrix A. A common choice is

$$t = \frac{2\pi}{\lambda_{max}}. \tag{5.2.8}$$

In this manner, all eigenvalues of the matrix are confined to the range $[0, 2\pi]$, balancing the demand for accuracy against the practical constraints of the quantum register utilized in the QPE.
We can consider the effects of the parameter t on the precision and complexity of the final solution. A smaller t reduces the phase shift λt, making it more challenging to accurately resolve different eigenvalues. Consequently, a too small t results in lower precision in the Quantum Phase Estimation (QPE) step. On the other hand, a larger t decreases the probability of finding the ancilla in the $|1\rangle_a$. This necessitates additional executions of the algorithm to gather reliable statistics, which could potentially increase the algorithm's complexity. In addition, we get a perfect encoding of our eigenvalues inside the c-register if $\tilde{\lambda_i} = 2^{n_c}\lambda_i t/2\pi$ are integer numbers. Thus, the best encoding we can achieve corresponds to the one representing exactly the minimum eigenvalue λ_{min}, since it contributes the most to the solution of the linear system, as one can see from Eq. (5.2.5). Therefore, one could just set $t = \frac{2\pi}{\lambda_{min}2^{n_c}}$. However, this choice has a main problem: in most cases we do not know λ_{min} in advance. Despite lots of upper bounds about the maximum

eigenvalue are known, there are only few about the smallest eigenvalue [43]. Furthermore, we also have to ensure that $t \leq \frac{2\pi}{\lambda_{\max}}$, as discussed previously.

Simulations results

Here we report the plots of simulations for different values of t. Specifically, we have simulated our algorithm over the range $t \in \left[\frac{2\pi}{10\lambda_{\max}N}, 10\frac{2\pi}{\lambda_{\max}N}\right]$ for different problem sizes N. We also considered the two different scenarios presented in Sec. 5.2.1. We utilized the percent error in the Absolute Average of the HHL solution in Eq. (5.2.6) as a parameter for assessing precision, as done in the previous section. In addition, we have repeated each simulation 50 times, plotting the mean value and the standard deviation of the results. As we can see in the Figures below, we found a good trade-off for setting t in the following equation:

$$t = \frac{2\pi}{\lambda_{\max}N}. \tag{5.2.9}$$

In particular, we can prove and discuss some theoretical considerations we made above. We start by considering small matrices, particularly for $N = 4$ and $N = 8$. In Fig. 5.2.4a and Fig. 5.2.4b, we can see how the accuracy of the algorithm increases dramatically when $t \geq \frac{2\pi}{\lambda_{\max}}$, because the eigenvalues are not all confined in the range $[0, 2\pi]$, which is a fundamental constraint that must be respected for the QPE to be accurate. Additionally, we can see that in the first case, setting $t = \frac{2\pi}{\lambda_{\min}2^{n_c}}$ yields good results. However, in the second case, this is not true, since $\lambda_{\min}$ is too small, resulting in a t that is greater than $\frac{2\pi}{\lambda_{\max}}$, proving the considerations made above. Similar results are obtained simulating the algorithm for $N = 8$. Finally, we can see that both in Fig. 5.2.4 and 5.2.5, $t = \frac{2\pi}{\lambda_{\max}N}$ yields good accuracies despite the two cases considered having different eigenvalues and condition numbers k. Here we can also note that, for very small t, the error increases until $\epsilon = 1$. This is due to the fact that, if t is too small, the eigenvalues are constrained in a very small interval and the QPE cannot distinguish from one eigenvalue to another. A crucial aspect to consider is the fluctuations observed in the results. This study focuses on small matrices, thereby involving a limited number of qubits,

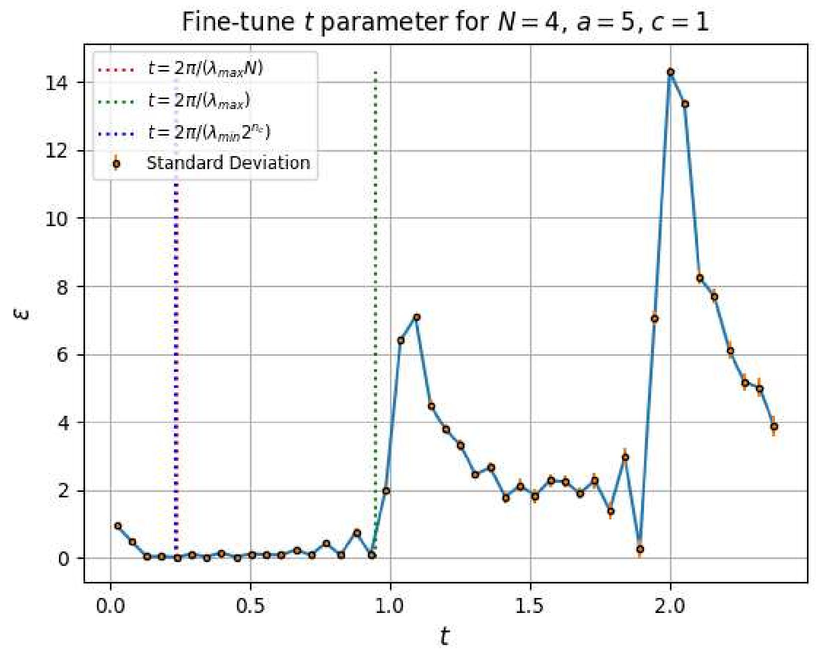
n_c, in the c-register. Consequently, there is a higher probability of measuring the ancilla in the state $|0\rangle_a$, as detailed in Sec. 5.2.2. Additionally, it is important to note that as t increases, the standard deviation also rises, which aligns with the theoretical discussion previously outlined. Now, if we consider the simulations done for $N = 32$, we get very similar results, $t = \frac{2\pi}{\lambda_{\max} N}$ is again a good fit for both the cases in Fig. 5.2.6a and 5.2.6b. However, in this instance the fluctuations increase significantly, this is because n_c rises with the problem size N, leading to a smaller probability of having the ancilla collapsing in $|1\rangle$. For larger matrices this can be solved in different ways:

- Setting $t = 2\pi/2^{n_c}$, so that the term 2^{n_c} cancels out in the Eq: 5.2.5. Always checking that $t \leq 2\pi/\lambda_{\max}$;

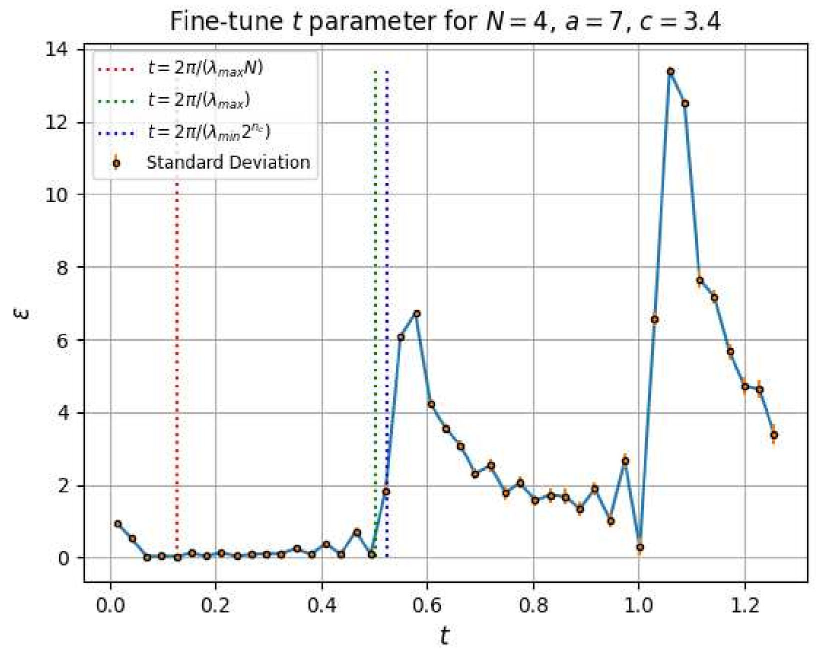
- Increasing the number of shots, i.e the number of times the HHL circuit is executed, so that we can achieve a reliable statistics to compute the solution norm or the absolute average with Eq. (4.2.3) and (4.2.5) respectively.

- Just set $t \leq 2\pi/\lambda_{\max}$ and use some Amplitude Amplification technique [44] to increase the probability of measuring tha ancilla in $|1\rangle$.

In conclusion, this means that we have to extract information about the system in advance, such as the maximum eigenvalue of the matrix $\lambda_{\max}$. For some systems this can be done precisely, such as the Toeplitz tridiagonal matrices we have seen in Sec. 5.2.1. Otherwise we shall use an upper bound for $\lambda_{\max}$.

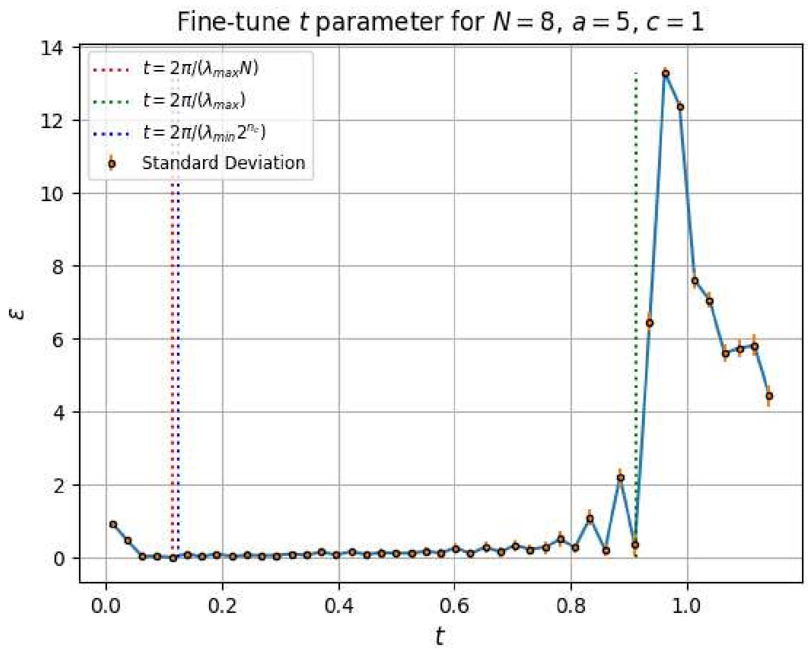


(a) Fine-tuning t for a 4×4 matrix defined in Eq. (5.2.1), where $a = 5$, $c = 1$

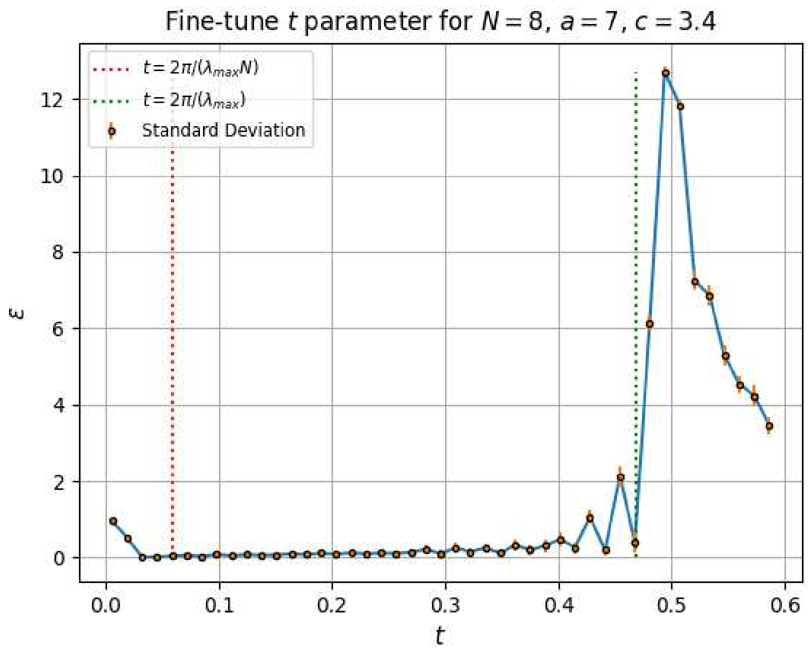


(b) Fine-tuning t for a 4×4 matrix defined in Eq. (5.2.1), where $a = 7$, $c = 3.4$

Figure 5.2.4: Percent error in the Absolute Average of $\mathbf{x}$ as a function of the parameter t . The figure illustrates the effect of fine-tuning t on the precision of the HHL algorithm.

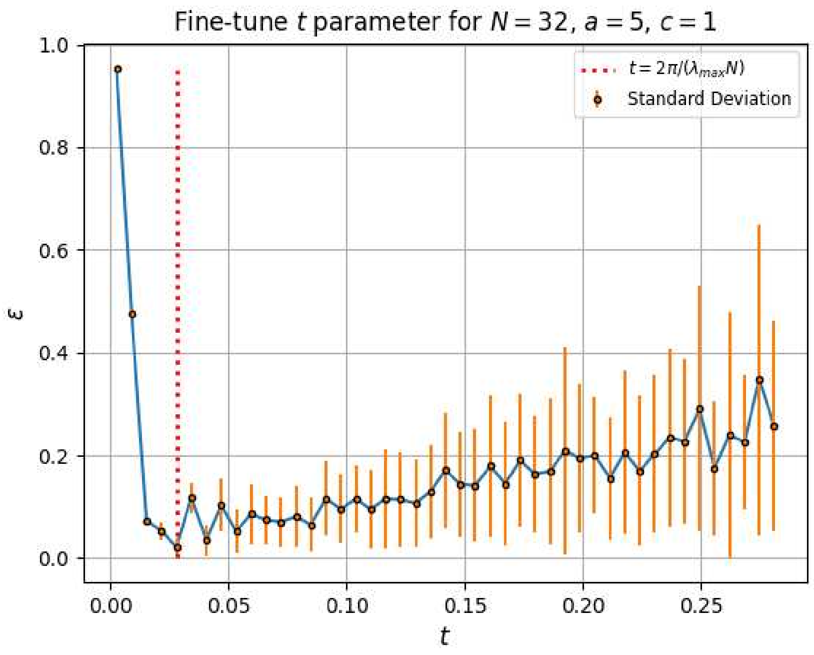


(a) Fine-tuning t for a 8×8 matrix defined in Eq. (5.2.1), where $a = 5$, $c = 1$

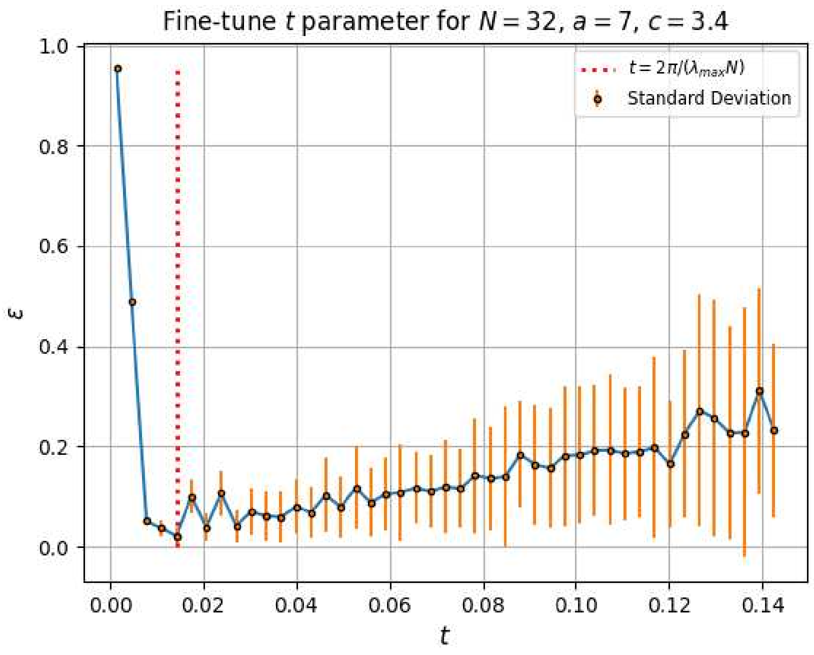


(b) Fine-tuning t for a 8×8 matrix defined in Eq. (5.2.1), where $a = 7$, $c = 3.4$

Figure 5.2.5: Percent error in the Absolute Average of $\mathbf{x}$ as a function of the parameter t . The figure illustrates the effect of fine-tuning t on the precision of the HHL algorithm.



(a) Fine-tuning t for a 32×32 matrix defined in Eq. (5.2.1), where $a = 5$, $c = 1$



(b) Fine-tuning t for a 32×32 matrix defined in Eq. (5.2.1), where $a = 7$, $c = 3.4$

Figure 5.2.6: Percent error in the Absolute Average of $\mathbf{x}$ as a function of the parameter t . The figure illustrates the effect of fine-tuning t on the precision of the HHL algorithm.

To avoid to determine the maximum Eigenvalue in advance, we revolve to the definition of upper bounds. Several upper bounds for $|\lambda_{\max}|$ exist [43], and some are detailed below:

- $\lambda_{\max} \leq \sqrt{\mathrm{Tr}(AA^\dagger)}$.
- $\lambda_{\max} \leq \|A\|_1 = \max_j \sum_i |a_{ij}|$.
- $\lambda_{\max} \leq \|A\|_2 = \sqrt{\sum_{i,j} |a_{ij}|^2}$.
- $\lambda_{\max} \leq M\|A\|_{\max} = M \max_{i,j} |a_{ij}|$.

The chosen upper bounds will influence the overall complexity of the algorithm, as the matrix computations described are challenging on both classical and quantum computers.
For instance, let's choose $\lambda_{\max} = \sqrt{Tr(AA^\dagger)}$ and compare the results obtained using the real $\lambda_{\max}$. We consider again the case for $a = 5$, $c = 1$ and using $n_{shots} = 1000$. The results are reported in Fig. 5.2.7.

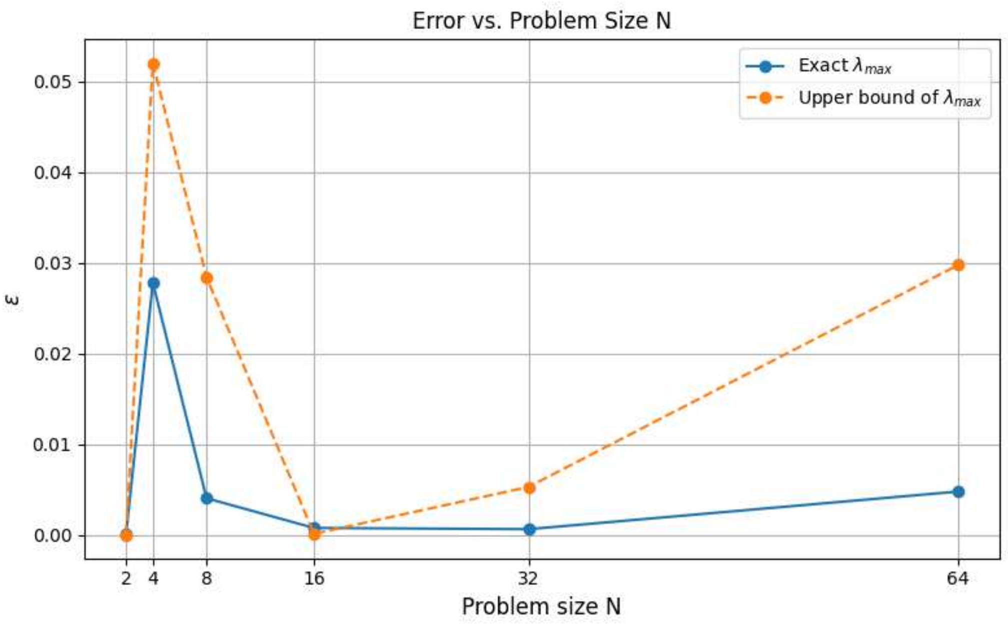


Figure 5.2.7: Percent error in the Absolute Average of the solution as N increases. We set $t = \frac{2\pi}{\lambda_{\max} N}$, using the exact $\lambda_{\max}$ and one of its upper bound, $\overline{\lambda}_{\max} = \sqrt{\mathrm{Tr}(AA^{\dagger})}$. As we can see in the plot, the upper bound is a good approximation of the real $\lambda_{\max}$, leading to a small percent error. This plot concerns the result of a single simulation.

5.2.4 Scaling of the Error with the System size

Now that we know how to set the input parameters of HHL algorithm, we want to test out how it performs in terms of precision when the size of the system increases. Specifically we choose:

- $t = \frac{2\pi}{\lambda_{\max} N}$,
- $n_c = n_b + 1$.

We consider again the case for $a = 5$, $c = 1$, using $n_{shots} = 1000$. The algorithm is simulated 50 times and we report the mean and the standard deviation of the results for different N. As we can see in Fig. 5.2.8, if we set our input parameters as previously mentioned, the precision does not increase with the size of the system. However, we stressed our algorithm with small problem sizes since we are limited by the simulations on classical hardware, as discussed in Sec. 5.1.4. Further cases should be analyzed, in particular for larger matrices and different structures.

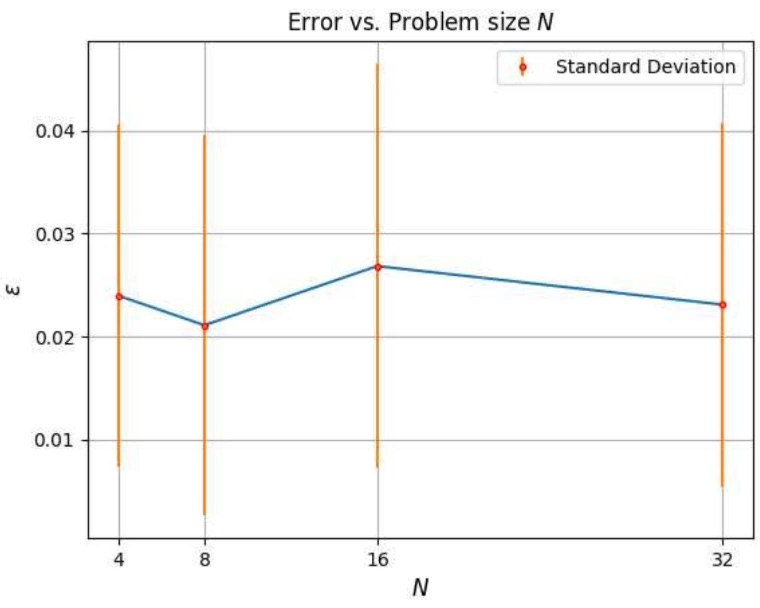


Figure 5.2.8: Percent error in the Absolute Average of the solution as N increases.

Chapter 6

Conclusions and Future Development

In this thesis, we explored the potential of quantum computing, specifically the Harrow-Hassidim-Lloyd (HHL) algorithm, for addressing computational electromagnetism (CEM) problems relevant to Earth observation. We began by highlighting the importance of CEM at Thales Alenia Space Italy (TASI), emphasizing the role of electromagnetic (EM) simulations, in various stages of space mission design and verification. We also identified a relevant use case in the Radio Frequency propagation in a rectangular waveguide for feeding a slotted antenna. This example is a simplified version of a larger use case related to the design and optimization of antennas for radar EO. Traditional methods for EM simulation, while robust, face significant challenges in scaling to large, complex systems due to their computational intensity. This limitation led us to consider quantum computing as a potential solution.

Quantum computing, grounded in the principles of quantum mechanics, offers a fundamentally different approach to computation compared to classical methods. We introduced key quantum concepts such as superposition and entanglement, along with quantum operators and circuit notations, to provide the necessary background. These foundational elements are critical for understanding the complexities and advantages of quantum algorithms.

We then discuss the principal subroutines utilized in the HHL algorithm, including the Quantum Fourier Transform (QFT) and the Quantum Phase Estimation (QPE). These components are essential for the effective implementation of the HHL algorithm, which promises exponential speedup over classical methods under certain conditions. Additionally, we briefly review Hamiltonian simulation algorithms, which represents one of the main steps of the HHL algorithm.

A detailed analysis of the HHL algorithm was conducted, explaining all its steps, from the state preparation to the final measurement of the quantum registers. We examined the algorithm's ability to compute key quantities such as the norm and the absolute average of the final solution. This thorough analysis was crucial for the practical implementation of the HHL algorithm.

This work entails a comprehensive Qiskit implementation of the HHL quantum algorithm from scratch, that allows for full control over input parameters and all associated subroutines of the algorithm. We detailed the main function and input parameters and discussed the challenges encountered during simulations. Various scenarios were tested to fine-tune the algorithm's input parameters for different problem sizes. In particular, we focused on the number of qubits n_c needed to encode the eigenvalues of the input matrix inside the quantum register, and on the evolution time parameter t of the Hamiltonian Simulation step. We highlighted the limitations of simulating the HHL algorithm on classical hardware and discussed potential future developments, such as simulating the algorithm with a Tensor Network emulator.

Our findings underscore the potential of quantum algorithms to address complex CEM problems more efficiently than classical methods. By providing a comprehensive implementation of the HHL algorithm and testing it under various conditions, we contribute to the growing body of research exploring quantum advantages in practical applications. However, the current capabilities of Noisy Intermediate-Scale Quantum (NISQ) hardware limit the size of problems that can be effectively tackled. While small problems are outperformed by classical algorithms and lack industrial relevance, larger problems remain beyond the reach of existing quantum hardware. Therefore, besides advancing quantum hardware, future research should explore hybrid quantum-classical approaches that leverage the strength of both paradigms.

A future development of this work is to implement, within the HHL algorithm, an Hamiltonian Simulation subroutine based on the state-of-the-art algorithms available in this field. Additionally, we plan to simulate the use case presented in Chapter 1, related to RF propagation in a slotted antenna, with the ultimate goal of extending it to the real-world scenario involving the design and optimization of antennas for radar EO applications. Another objective is to execute the algorithm on a real quantum hardware, conducting the same analysis performed in this study.

In conclusion, we have discussed why quantum computing holds significant promise in addressing PDEs, particularly in the context of Earth observation and aerospace applications. While there are challenges to overcome, the potential for improved efficiency justifies continued research and development in this exciting and rapidly evolving field.

Bibliography

[1] Jian-Ming Jin. *The finite element method in electromagnetics.* John Wiley & Sons, 2015.

[2] Owe Axelsson, Va Barker, and Dj Benson. "Finite Element Solution of Boundary Value Problems: Theory and Computation. Classics in Applied Math, Vol. 35". In: *Applied Mechanics Reviews - APPL MECH REV* 55 (May 2002). DOI: `10.1115/1.1470667`.

[3] David B. Davidson. *Computational Electromagnetics for RF and Microwave Engineering.* 2nd ed. Cambridge University Press, 2010.

[4] A. Hassidim A. Harrow and S. Lloyd. "Quantum Algorithm for Linear Systems of Equations". In: *Phys. Rev. Lett. 103* 150502 (2009).

[5] Ashley Montanaro and Sam Pallister. "Quantum algorithms and the finite element method". In: *Physical Review A* 93 (2015), p. 032324. URL: `https://api.semanticscholar.org/CorpusID:44004935`.

[6] John Preskill. "Quantum Computing in the NISQ era and beyond". In: *Quantum* (2018). URL: `https://api.semanticscholar.org/CorpusID:44098998`.

[7] Ali Javadi-Abhari et al. *Quantum computing with Qiskit.* 2024. DOI: `10.48550/arXiv.2405.08810`. arXiv: `2405.08810 [quant-ph]`.

[8] Almudena Carrera Vazquez, Ralf Hiptmair, and Stefan Woerner. "Enhancing the Quantum Linear Systems Algorithm Using Richardson Extrapolation". In: *ACM Transactions on Quantum Computing* 3 (Mar. 2022), pp. 1–37. DOI: `10.1145/3490631`.

[9] John Leonidas Volakis et al. *Finite element method for electromagnetics*. Universities Press, 1998.

[10] *Overview of Sentinel-1 Mission*. URL: https://sentiwiki.copernicus.eu/web/s1-mission.

[11] A. Einstein, B. Podolsky, and N. Rosen. "Can Quantum-Mechanical Description of Physical Reality Be Considered Complete?" In: *Phys. Rev.* 47 (10 May 1935), pp. 777–780. DOI: 10.1103/PhysRev.47.777. URL: https://link.aps.org/doi/10.1103/PhysRev.47.777.

[12] J. S. Bell. "On the Einstein Podolsky Rosen paradox". In: *Physics Physique Fizika* 1 (3 Nov. 1964), pp. 195–200. DOI: 10.1103/PhysicsPhysiqueFizika.1.195. URL: https://link.aps.org/doi/10.1103/PhysicsPhysiqueFizika.1.195.

[13] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. 10th. USA: Cambridge University Press, 2011. ISBN: 1107002176.

[14] Alastair Kay. "Tutorial on the Quantikz Package". In: (Sept. 2018). DOI: 10.17637/rh.7000520.

[15] David Deutsch and Richard Jozsa. "Rapid solution of problems by quantum computation". In: *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 439 (1992), pp. 553–558. URL: https://api.semanticscholar.org/CorpusID:121702767.

[16] Stefano Markidis. "What is Quantum Parallelism, Anyhow?" In: *ISC High Performance 2024 Research Paper Proceedings (39th International Conference)* (2024). URL: https://api.semanticscholar.org/CorpusID:269736904.

[17] Don Coppersmith. "An approximate Fourier transform useful in quantum factoring". In: *arXiv: Quantum Physics* (2002). URL: https://api.semanticscholar.org/CorpusID:17450629.

[18] Peter W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science* (1994), pp. 124–134. URL: https://api.semanticscholar.org/CorpusID:15291489.

[19] James W. Cooley, Peter A. W. Lewis, and Peter D. Welch. "The Fast Fourier Transform and Its Applications". In: *IEEE Transactions on Education* 12 (1969), pp. 27–34. URL: https://api.semanticscholar.org/CorpusID:10563630.

[20] Marco Ballarin. "Quantum computer simulation via tensor networks". In: (). URL: https://thesis.unipd.it/retrieve/117b8335-eed3-4881-b654-66352af4d5e4/BALLARIN_Marco_Thesis_final.pdf.

[21] Alexei Y. Kitaev. "Quantum measurements and the Abelian Stabilizer Problem". In: *Electron. Colloquium Comput. Complex.* TR96 (1995). URL: https://api.semanticscholar.org/CorpusID:17023060.

[22] Seth Lloyd. "Universal Quantum Simulators". In: *Science* 273 (1996), pp. 1073–1078. URL: https://api.semanticscholar.org/CorpusID:43496899.

[23] Dominic W. Berry et al. "Efficient Quantum Algorithms for Simulating Sparse Hamiltonians". In: *Communications in Mathematical Physics* 270 (2005), pp. 359–371. URL: https://api.semanticscholar.org/CorpusID:37923044.

[24] Andrew M Childs. "Lecture notes on quantum algorithms". In: (). URL: https://www.cs.umd.edu/~amchilds/qa/qa.pdf.

[25] Dorit Aharonov and Amnon Ta-Shma. *Adiabatic Quantum State Generation and Statistical Zero Knowledge.* 2003. arXiv: quant-ph/0301023 [quant-ph]. URL: https://arxiv.org/abs/quant-ph/0301023.

[26] Danial Dervovic et al. *Quantum linear systems algorithms: a primer.* 2018. arXiv: 1802.08227 [quant-ph]. URL: https://arxiv.org/abs/1802.08227.

[27] Andrew M. Childs and Robin Kothari. "Simulating Sparse Hamiltonians with Star Decompositions". In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011, pp. 94–103. ISBN: 9783642180736. DOI: 10.1007/978-3-642-18073-6_8. URL: http://dx.doi.org/10.1007/978-3-642-18073-6_8.

[28] Andris Ambainis. *Understanding Quantum Algorithms via Query Complexity*. 2017. arXiv: 1712.06349 [quant-ph]. URL: https://arxiv.org/abs/1712.06349.

[29] Bo-jia Duan et al. "A survey on HHL algorithm: From theory to application in quantum machine learning". In: *Physics Letters A* 384 (2020), p. 126595. URL: https://api.semanticscholar.org/CorpusID:219930336.

[30] In: *Quantum Information and Computation* 12.1 2 (Jan. 2012). ISSN: 1533-7146. DOI: 10.26421/qic12.1-2. URL: http://dx.doi.org/10.26421/QIC12.1-2.

[31] Dominic W. Berry et al. "Exponential improvement in precision for simulating sparse Hamiltonians". In: *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*. STOC '14. ACM, May 2014. DOI: 10.1145/2591796.2591854. URL: http://dx.doi.org/10.1145/2591796.2591854.

[32] Dominic W. Berry et al. "Simulating Hamiltonian Dynamics with a Truncated Taylor Series". In: *Phys. Rev. Lett.* 114 (9 Mar. 2015), p. 090502. DOI: 10.1103/PhysRevLett.114.090502. URL: https://link.aps.org/doi/10.1103/PhysRevLett.114.090502.

[33] Dominic W. Berry, Andrew M. Childs, and Robin Kothari. "Hamiltonian Simulation with Nearly Optimal Dependence on all Parameters". In: *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. IEEE, Oct. 2015. DOI: 10.1109/focs.2015.54. URL: http://dx.doi.org/10.1109/FOCS.2015.54.

[34] Guang Hao Low and Isaac L. Chuang. "Hamiltonian Simulation by Qubitization". In: *Quantum* 3 (2019), p. 163. DOI: `10.22331/q-2019-07-12-163`. arXiv: `1610.06546 [quant-ph]`.

[35] *UnitaryGate function available in Qiskit library.* URL: `https://docs.quantum.ibm.com/api/qiskit/qiskit.circuit.library.UnitaryGate`.

[36] *TrotterQRTE function available in Qiskit library.* URL: `https://docs.quantum.ibm.com/api/qiskit/0.37/qiskit.algorithms.TrotterQRTE`.

[37] Anika Zaman, Hector Morrell, and Hiu Wong. "A Step-by-Step HHL Algorithm Walkthrough to Enhance Understanding of Critical Quantum Computing Concepts". In: *IEEE Access* PP (Jan. 2023), pp. 1–1. DOI: `10.1109/ACCESS.2023.3297658`.

[38] Lov K. Grover and Terry Rudolph. "Creating superpositions that correspond to efficiently integrable probability distributions". In: *arXiv: Quantum Physics* (2002). URL: `https://api.semanticscholar.org/CorpusID:118380132`.

[39] Adam Holmes and Anne Y. Matsuura. "Efficient Quantum Circuits for Accurate State Preparation of Smooth, Differentiable Functions". In: *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)* (2020), pp. 169–179. URL: `https://api.semanticscholar.org/CorpusID:218581252`.

[40] Vivek V. Shende, Stephen S. Bullock, and Igor L. Markov. "Synthesis of quantum-logic circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25 (2004), pp. 1000–1010. URL: `https://api.semanticscholar.org/CorpusID:265038781`.

[41] *HHL function available in Qiskit library.* URL: `https://docs.quantum.ibm.com/api/qiskit/0.35/qiskit.algorithms.linear_solvers.HHL`.

[42] *Quantum Matcha Tea.* URL: `https://pypi.org/project/qmatchatea/`.

[43] Changpeng Shao. “Reconsider HHL algorithm and its related quantum machine learning algorithms”. In: *arXiv: Quantum Physics* (2018). URL: https://api.semanticscholar.org/CorpusID:119367568.

[44] Gilles Brassard et al. “Quantum Amplitude Amplification and Estimation”. In: *arXiv: Quantum Physics* (2000). URL: https://api.semanticscholar.org/CorpusID:54753.