

Gottfried Wilhelm Leibniz Universität Hannover

Fakultät für Mathematik und Physik

Institut für Theoretische Physik / Institut für Angewandte Mathematik

---

**Quantum Linear Systems Algorithms  
applied to Partial Differential Equations  
with Poisson's Problem as an Example**

---

Master's Thesis

**Author:** Martin Steinbach

**Supervisors:** Prof. Dr. Tobias J. Osborne, Prof. Dr. Thomas Wick

January 26, 2024



---

# Abstract

---

Quantum linear systems algorithms solve the Quantum Linear Systems Problem (QLSP), where the goal is to prepare a quantum state that is proportional to an approximate solution of a linear system. A well-known algorithm by Childs, Kothari and Somma uses an approach based on a linear combination of unitary operators that approximates the inverse of the coefficient matrix using a Fourier series. In this thesis, we show how this algorithm can be implemented to solve the QLSP for the two-dimensional Poisson equation and compute several derived quantities. We then investigate the performance of the algorithm on the the unit square for finite difference grid sizes up to  $34 \times 34$  using a state vector simulation with up to 36 qubits. The study shows that the algorithm performs as expected even with few qubits. It is therefore a good candidate for solving the QLSP corresponding to other elliptic partial differential equations.



---

# Contents

---

<b>1. Introduction</b>	<b>9</b>
<b>2. Preliminaries</b>	<b>13</b>
2.1. Quantum States . . . . .	13
2.2. Quantum Bits and Quantum Gates . . . . .	15
2.3. Quantum Circuits . . . . .	18
2.4. Computational Complexity . . . . .	21
2.5. Quantum Fourier Transform . . . . .	22
2.6. Quantum Phase Estimation . . . . .	24
2.7. Amplitude Amplification . . . . .	27
2.8. State Preparation . . . . .	30
2.9. Linear Combination of Unitaries . . . . .	32
2.10. Hamiltonian Simulation . . . . .	34
<b>3. Geometric Multigrid Methods</b>	<b>37</b>
3.1. Motivation and Problem Formulation . . . . .	37
3.1.1. Illustrative Example: 1D Poisson . . . . .	38
3.1.2. Problem Formulation . . . . .	39
3.2. Algorithmic Formulation . . . . .	40
3.2.1. Two-Grid Method . . . . .	40
3.2.2. Multigrid Method . . . . .	41
3.2.3. Grid Transfer Operations . . . . .	43
3.3. Cost Complexity . . . . .	44
<b>4. Quantum Linear Systems Algorithms</b>	<b>47</b>
4.1. HHL Algorithm . . . . .	48
4.2. CKS Algorithm . . . . .	53
4.2.1. Derivation of the LCU approach . . . . .	53
4.2.2. LCU Implementation . . . . .	55
4.2.3. Complexity Analysis . . . . .	59
<b>5. Numerics</b>	<b>65</b>
5.1. Problem Formulation . . . . .	65
5.2. Implementation . . . . .	67
5.2.1. Runtime Improvements . . . . .	69

*Contents*

---

5.2.2. Memory Usage Improvements . . . . .	71
5.2.3. Space Complexity . . . . .	74
5.3. Solution Convergence . . . . .	76
5.4. Functional Values . . . . .	80
<b>Conclusion</b>	<b>83</b>
<b>A. Appendix</b>	<b>85</b>
<b>Bibliography</b>	<b>93</b>

---

## List of Figures

---

2.1.	Example of a quantum circuit diagram. . . . .	19
2.2.	Example of a quantum circuit diagram with measurement. . . . .	20
2.3.	Example usage of a quantum oracle. . . . .	20
2.4.	Circuit diagram of the 3-qubit quantum Fourier transform. . . . .	24
2.5.	Circuit diagram of the phase estimation algorithm. . . . .	25
2.6.	Illustration of the subspace rotation used for amplitude amplification. . . . .	29
3.1.	Diagram depicting the V and W multigrid cycles. . . . .	43
3.2.	Illustration of basis functions on different levels. . . . .	44
4.1.	High-level circuit diagram of the HHL algorithm. . . . .	50
4.2.	High-level circuit diagram of the CKS algorithm. . . . .	56
4.3.	Implementation of the LCU operator $U$ . . . . .	59
5.1.	Implementation of the LCU operator $U$ in the simulation. . . . .	74
5.2.	Space complexity of the CKS algorithm applied to Poisson's QLSP. . . . .	76
5.3.	Surface plot and heat map of the solution to Poisson's QLSP. . . . .	77
5.4.	Error of the CKS algorithm on different grid sizes. . . . .	78
5.5.	Noiseless qubits needed to solve Poisson's QLSP on a $6 \times 6$ grid. . . . .	79
5.6.	Noiseless qubits needed to solve Poisson's QLSP on a $10 \times 10$ grid. . . . .	79
5.7.	Error of the functional values as a function of the grid size. . . . .	82
A.1.	Circuit diagram of the $n$ -qubit quantum Fourier transform. . . . .	85
A.2.	Scaling of the condition number with the grid size for Poisson's problem. . . . .	91

---

## List of Tables

---

3.1.	Operations for solving large, sparse and s.p.d. linear systems. . . . .	38
5.1.	Exemplary run times for solving Poisson's QLSP. . . . .	72
5.2.	Memory usage for solving Poisson's QLSP. . . . .	73
5.3.	Target precision values used as input for the CKS algorithm. . . . .	77



# CHAPTER 1.

---

## Introduction

---

Differential equations are at the heart of many physical and mathematical processes. Since exact solutions are not computable in most cases, we resort to approximate solutions which suffice for many applications. There exist various techniques for numerically solving differential equations. Among the most commonly used ones are volume discretization methods such as the Finite Difference Method (FDM), the finite element method and the finite volume method [Wic22]. Moreover, there are spectral methods which aim to globally approximate the solution using suitable basis functions. The boundary element method only discretizes the boundary of the domain of interest and can be applied to integral equations.

Many of the previously mentioned methods yield a system of equations  $Ax = b$  which is linear and sparse under suitable conditions, for instance when applying one of the volume discretization methods to linear elliptic Partial Differential Equations (PDEs) [Wic22]. The numerical methods for solving linear systems can be categorized into direct methods and iterative methods.

Direct methods comprise algorithms such as Gaussian elimination and other matrix factorizations, which yield exact results up to rounding errors. Iterative methods include the stationary iterative methods such as the Jacobi method, Gauss-Seidel method and successive over-relaxation, which use fixed-point iterations based on matrix splittings [Saa03]. Another important class of iterative methods are Krylov subspace methods such as the conjugate gradient method, which generate approximate solutions in an increasing sequence of vector spaces, the Krylov spaces [Sog22]. In certain situations, these methods can even compute an exact solution after a finite number of steps. Multigrid methods use a series of refined meshes and the smoothing property of iteratives solvers to compute approximate solutions with linear time complexity in the number of unknowns [Hac85]. These methods are commonly applied as an efficient solver for elliptic PDEs such as the Poisson equation [Hac78]. Moreover, they are well-known as preconditioners for Krylov subspace methods [GLJ18]. In general, it requires careful consideration to choose a suitable method since there is no algorithm that works best for all types of problems [NRT92]. Since we will later consider the Poisson equation as a prototype for a linear elliptic PDE, we use the multigrid method for comparison. Thus, if we want a quantum algorithm to be asymptotically faster, it must have sublinear time complexity in the input size.

In 2009, Harrow, Hassidim and Lloyd published the first Quantum Linear Systems Al-

gorithm (QLSA), which is known today as the HHL algorithm [HHL09]. The algorithm uses Hamiltonian simulation [Fey82; Llo96] in combination with Quantum Phase Estimation (QPE) [Kit95] to perform a quantum eigenbasis decomposition of the right-hand side  $b$  of the system. This can then be used to find the corresponding eigenvalues and apply the inverse of the given matrix by multiplying with the inverse eigenvalues. However, in the quantum case, the notion of a solution has to be weakened. Specifically, quantum linear systems algorithms aim to prepare a quantum state  $|x\rangle$  that is proportional to an approximate solution of the linear system. This means that the solution is encoded by the amplitudes of the state  $|x\rangle$ , which cannot simply be read out as in the classical case. Since a full reconstruction of the solution vector elements from the given state requires exponentially many repetitions of the algorithm, obtaining a speedup for practical applications is still an open problem.

The time complexity of the HHL algorithm is  $\tilde{O}(\log(N)d^2\kappa^2/\varepsilon)$ , where  $N$  is the size of the square linear system,  $d$  is the maximum number of non-zero elements in any given row,  $\kappa$  is the condition number of the coefficient matrix  $A$  and  $\varepsilon$  is the solution tolerance. The  $\tilde{O}$  notation omits any factors that are polynomials in the logarithms of the parameters of interest. Considering only the time to prepare the state  $|x\rangle$ , the HHL would thus be exponentially faster in the input size than the best classical algorithms.

Since the publication of the HHL algorithm, there have been many different attempts to solve the Quantum Linear Systems Problems (QLSP), where one of the most common ideas is to harness the efficiency of recent Hamiltonian simulation algorithms [Ber+14; BCK15; LC17; CL19; Low19; LC19] as in the case of the HHL algorithm. Ambainis gave an improvement to the HHL algorithm that uses Variable-Time Amplitude Amplification (VTAA) to improve the dependence on the condition number to  $O(\kappa \log^3(\kappa))$  [Amb12]. The main idea is to observe that the amplitude amplification procedure [Bra+02] which is used in the HHL algorithm has to wait for all parts of the controlled Hamiltonian simulation to complete, where some parts take much longer than others. The VTAA algorithm circumvents this problem by amplifying the success probability in multiple stages, to make use of the parts of the computation that finish earlier. In 2013, Clader et al. gave an improved version of the HHL algorithm which uses a preconditioner to achieve an end-to-end run time of  $\tilde{O}(d^7\kappa \log(N)/\varepsilon^2)$  to calculate an electromagnetic scattering cross section using the finite element method [CJS13]. However, Montanaro et al. criticize that the complexity analysis is incomplete and show that, under certain conditions, the overall speedup can be at most polynomial [MP16]. Another result that also uses an efficient Hamiltonian simulation algorithm was given by Childs, Kothari and Somma in 2017, which we call the CKS algorithm [CKS17]. It improves the dependence on the tolerance  $\varepsilon$  while maintaining the dependence on other parameters up to polylogarithmic factors. The authors use an approach based on a linear combination of unitary matrices to approximately implement the inverse of a sparse matrix. Combined with the VTAA approach as shown by Ambainis [Amb12], they obtain a query complexity of  $O(d\kappa \text{poly}(\log(d\kappa/\varepsilon)))$ ,

---

which is optimal in the condition number up to polylogarithmic factors.

Another class of algorithms is based on Adiabatic Quantum Computing (AQC) [Far+00] and similar methods such as the randomization method [Som+08]. The first and most complicated step is to encode the solution of the problem as the ground state of a suitable Hamiltonian  $H_1$ . Next, the algorithm prepares the ground state of a simple Hamiltonian  $H_0$  which can be prepared more easily. It is then possible to obtain the solution ground state by applying small perturbations in the direction of  $H_1$  to the initial state, realizing an adiabatic evolution. According to the adiabatic theorem [BF28], the final state is close to the target ground state if the evolution time is large and the perturbations are small enough, provided that the spectral gap between the Hamiltonians is nonzero for all times. It has been shown that AQC is equivalent to standard quantum computation. Hence, it is also a universal model of computing [Aha+08]. The randomization method uses random evolution times, which has the advantage that better convergence guarantees can sometimes be obtained [Som+08; CXS14]. It is used by Subasi et al. to obtain a QLSA with a time complexity of  $O(\kappa \log(\kappa)/\varepsilon)$  [SSO19]. A recent approach was given by Costa et al. in 2022 [Cos+22]. The authors prove a new adiabatic theorem for discrete-time evolutions to obtain a QLSA with optimal time complexity  $O(\kappa \log(1/\varepsilon))$ .

Recently, various classically-inspired iterative QLSAs have been proposed, such as quantum gradient descent [Reb+19; KP20] and quantum row and column iteration methods [SX20; Zuo+21; Liu+22]. Finally, the Quantum Singular Value Transform (QSVT) can be used to apply polynomial functions to block-encoded matrices in the sense of the function calculus [Gil+19]. It can be seen as a broader framework including many quantum algorithms such as optimal Hamiltonian simulation, QPE, amplitude amplification and an algorithm to solve the QLSP. The QLSA that can be derived from the QSVT is similar to the CKS algorithm [CKS17].

To solve linear differential equations on a quantum computer in the sense of the QLSP, the most common approach is to use a classical discretization method to obtain a linear system that can then be solved using a suitable QLSA for the given problem. For instance, Childs et al. [CLO21] propose two algorithms based on the adaptive FDM and quantum spectral methods [CLO20] to solve Poisson's problem using the CKS algorithm [CKS17]. Another recent approach called Schrödingerization [JLY22] transforms the system of PDEs to a system of Schrödinger equations which can then be solved directly using Hamiltonian simulation. This analog simulation approach has the advantage that the differential equations do not need to be discretized, which eliminates one source of error. There have been studies of various well-known differential equations such as the Poisson equation [AK23], the heat equation [LMS22; JL23] and the wave equation [JL23]. However, most studies are solely theoretical.

In this thesis, we study one of the algorithms given by Childs, Kothari and Somma [CKS17], which approximates the inverse of the given matrix using a Fourier series. We examine how this algorithm works in detail, how it can be implemented and how many

qubits are needed for the implementation. Using a full state vector simulation with up to 36 qubits, we apply the algorithm to solve the QLSP for the two-dimensional Poisson equation which is discretized using the FDM. This simulation allows us to examine the performance of the algorithm in various scenarios. The main goal is to study the convergence behavior of the error with respect to the number of qubits used. We also calculate goal functionals of the solution as an exemplary application and study how their error behaves compared to the discretization error for a fixed number of qubits. The multigrid method will serve as a benchmark since it is one of the best classical algorithms for the given problem.

In Chapter 2, we begin by introducing the mathematical and quantum mechanical fundamentals and notations for quantum computing. We then introduce the geometric multigrid method in Chapter 3 to understand how one of the best classical algorithms for elliptic PDEs works. In Chapter 4, we present the HHL algorithm [HHL09] and then take a closer look at the CKS algorithm [CKS17]. Finally, we describe the implementation and give the numerical results in Chapter 5.

# CHAPTER 2.

---

## Preliminaries

---

In this chapter, we introduce the basics of quantum computing and explain the quantum subroutines needed for the quantum linear systems algorithms that we study in Chapter 4. We mostly follow the book by Nielsen and Chuang [NC09].

### 2.1. Quantum States

In this section, we introduce the basic notation used throughout this chapter. The Kronecker product or *tensor product*  $\otimes$  plays a fundamental role.

**Definition 2.1 (Kronecker Product).** *Let  $A \in \mathbb{C}^{m \times n}$  and  $B \in \mathbb{C}^{r \times s}$  with  $m, n, r, s \in \mathbb{N}_{>0}$ . Then the Kronecker product of  $A$  and  $B$  is given by*

$$A \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{pmatrix} \in \mathbb{C}^{mr \times sn}. \quad (2.1)$$

Next, let us consider quantum states.

**Definition 2.2 (State).** *Let  $\mathcal{H} = \mathbb{C}^2$  be the two-dimensional complex Hilbert space, equipped with the inner product*

$$\langle x, y \rangle = \bar{x}_1 y_1 + \bar{x}_2 y_2, \quad (2.2)$$

*which is antilinear in the first and linear in the second component. Moreover, let  $\mathcal{H}_n$  be defined as the tensor product of  $n$  copies of  $\mathcal{H}$ :*

$$\mathcal{H}_n = \mathcal{H}^{\otimes n} = \bigotimes_{i=1}^n \mathcal{H} = \text{span} \left\{ \bigotimes_{i=1}^n b_i \mid b \in \{e_0, e_1\}^n \right\}. \quad (2.3)$$

*A basis of this space is given by all Kronecker products of arbitrary  $n$ -tuples of the canonical basis vectors  $e_0 = (1, 0)$  and  $e_1 = (0, 1)$  of  $\mathcal{H}$ . Since  $b_i \in \mathcal{H} \cong \mathbb{C}^{2 \times 1}$ , we have  $\mathcal{H}_n \cong \mathbb{C}^{2^n \times 1} \cong \mathbb{C}^{2^n}$ .*

## 2. Preliminaries

---

Let  $x_1, \dots, x_n, y_1, \dots, y_n \in H$ , then the inner product of  $x = x_1 \otimes \dots \otimes x_n \in \mathcal{H}_n$  and  $y = y_1 \otimes \dots \otimes y_n \in \mathcal{H}_n$  is given by

$$\langle x, y \rangle_{\mathcal{H}_n} = \langle x_1, y_1 \rangle_H \dots \langle x_n, y_n \rangle_H. \quad (2.4)$$

Extending this by linearity defines a scalar product on  $\mathcal{H}_n$  such that  $(\mathcal{H}_n, \langle \cdot, \cdot \rangle_{\mathcal{H}_n})$  is a Hilbert space. Then a (quantum) state is an element  $x \in \mathcal{H}_n$  with  $\|x\|_{\mathcal{H}_n} = 1$ . The set of all states in  $\mathcal{H}_n$  is denoted by  $E_n$ .

In the following, we mostly use the *Dirac notation*.

**Notation 2.3 (Dirac).** Let  $n \in \mathbb{N}$  be the system size. An element  $x \in \mathcal{H}_n$  is denoted by a ket  $|x\rangle \in \mathcal{H}_n$ . The computational basis states are given by  $|0\rangle = e_0 = (1, 0) \in H$  and  $|1\rangle = e_1 = (0, 1) \in H$ . For every ket  $|x\rangle \in \mathcal{H}_n$  there exists an associated bra  $\langle x| \in \mathcal{H}_n^*$ , where  $\mathcal{H}_n^*$  denotes the dual space of  $\mathcal{H}_n$ : Let  $y \in \mathcal{H}_n$ , then  $\langle x| : \mathcal{H}_n \rightarrow \mathbb{C}$  with

$$\langle x|(|y\rangle) := \langle x|y\rangle := \langle x, y \rangle_{\mathcal{H}_n}. \quad (2.5)$$

Using this correspondence, the bra can be identified with the column vector  $\langle x| = |x\rangle^\dagger = \bar{x}^T \in \mathcal{H}_n$ , which is the conjugate transpose of  $x$ . The tensor product of kets with bras is denoted by

$$|x\rangle\langle y| = |x\rangle \otimes \langle y|. \quad (2.6)$$

This corresponds to the rank-one operator mapping  $|z\rangle \in \mathcal{H}_n$  to

$$(|x\rangle\langle y|) \cdot |z\rangle = |x\rangle \langle y|z\rangle. \quad (2.7)$$

The following notation is useful when working both with vectors and states.

**Notation 2.4.** For a given vector  $|x\rangle \in \mathcal{H}_n$ , denote the corresponding state by

$$\mathcal{N}(|x\rangle) = \frac{|x\rangle}{\| |x\rangle \|_{\mathcal{H}_n}} \in E_n. \quad (2.8)$$

Product states play a fundamental role in quantum computing.

**Notation 2.5 (Product State).** A product state is a state  $|\Psi\rangle \in E_n$  that can be written in the form

$$|\Psi\rangle = |\psi_1\rangle \otimes \dots \otimes |\psi_n\rangle = |\psi_1\rangle \dots |\psi_n\rangle. \quad (2.9)$$

for some  $|\psi_1\rangle, \dots, |\psi_n\rangle \in E_1$ . The  $n$ -fold product of a state  $|x\rangle \in E_1$  is abbreviated by

$$|x\rangle \dots |x\rangle := |x\rangle^{\otimes n} := |x^n\rangle. \quad (2.10)$$

In many cases, we work with binary representations of integers. Let the binary representation of  $m$  be given by  $m = m_1 2^{n-1} + \dots + m_n 2^0$ . The basis state encoding this number as an element of  $\mathcal{H}_n$  is denoted by

$$|m\rangle := |m_1\rangle \dots |m_n\rangle \in E_n. \quad (2.11)$$

For instance,

$$|2\rangle = |1\rangle |0\rangle = |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad (2.12)$$

corresponds to the basis vector  $e_2$ . In general, we have the correspondence  $e_m = |m\rangle$  for all  $m \in \mathbb{N}$ . It should be clear from the context if  $|10\rangle = |1\rangle |0\rangle$  or  $|10\rangle = |1\rangle |0\rangle |1\rangle |0\rangle$  is meant.

## 2.2. Quantum Bits and Quantum Gates

A *qubit* is a two-level quantum system that can be in the states  $|0\rangle$ ,  $|1\rangle$  or any superposition

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle \in E_1. \quad (2.13)$$

The values  $\alpha, \beta \in \mathbb{C}$  are called the (probability) *amplitudes* of  $|0\rangle$  and  $|1\rangle$ , respectively. We say that two amplitudes  $\alpha$  and  $\beta$  differ by a *relative phase*  $\varphi \in \mathbb{R}$  if  $\alpha = e^{i\varphi} \beta$ .

Measuring a qubit is an inherently random process and yields one of the classical outcomes 0 or 1 with probability  $p_0 = |\alpha|^2$  and  $p_1 = |\beta|^2$ , respectively. Formally, measurements can be described by projection valued measures.

**Definition 2.6 (Projection Valued Measure).** *Let  $n \in \mathbb{N}$  be the number of qubits. A Projection Valued Measure (PVM) on  $\mathcal{H}_n$  is a set  $\mathcal{M} = \{P_j\}_{j \in J}$  of projection operators  $P_j : \mathcal{H}_n \rightarrow \mathcal{H}_n$  which fulfills the completeness condition*

$$\sum_{j \in J} P_j = I_{2^n}, \quad (2.14)$$

where  $I_{2^n}$  is the identity on  $\mathcal{H}_n$  and where  $J$  is the set of possible outcomes.

It is often said that the state *collapses* due to the measurement, which is an irreversible process. Mathematically, this corresponds to a projection to a subspace followed by renormalization.

## 2. Preliminaries

---

**Definition 2.7 (Post-Measurement State).** Let  $n \in \mathbb{N}$ ,  $|\psi\rangle \in E_n$  and let  $\mathcal{M} = \{P_j\}_{j \in J}$  be a PVM on  $\mathcal{H}_n$ . Applying the measurement corresponding to  $\mathcal{M}$  yields the post-measurement state

$$|\psi_j\rangle = \mathcal{N}(P_j |\psi\rangle) \quad (2.15)$$

for a random outcome  $j \in J$  with probability

$$p_j = \langle \psi | P_j | \psi \rangle. \quad (2.16)$$

As an example, consider  $n = 2$  qubits. Then a general state  $|\psi\rangle \in E_n$  has the form

$$|\psi\rangle = \alpha_{00} |00\rangle + \alpha_{01} |01\rangle + \alpha_{10} |10\rangle + \alpha_{11} |11\rangle. \quad (2.17)$$

For a measurement of the first qubit, consider the PVM  $\mathcal{M} = \{|0\rangle\langle 0| \otimes I_2, |1\rangle\langle 1| \otimes I_2\}$  corresponding to a measurement in the computational basis. Then the probabilities to measure 0 and 1 are given by

$$p_0 = \langle \psi | (\alpha_{00} |00\rangle + \alpha_{01} |01\rangle) = |\alpha_{00}|^2 + |\alpha_{01}|^2, \quad (2.18)$$

$$p_1 = \langle \psi | (\alpha_{10} |10\rangle + \alpha_{11} |11\rangle) = |\alpha_{10}|^2 + |\alpha_{11}|^2. \quad (2.19)$$

If the measurement outcome is 0, the post-measurement state is

$$|\psi_0\rangle = \mathcal{N}(|0\rangle\langle 0| \otimes I_2 |\psi\rangle) = \mathcal{N}(\alpha_{00} |00\rangle + \alpha_{01} |01\rangle) = \frac{\alpha_{00} |00\rangle + \alpha_{01} |01\rangle}{\sqrt{p_0}}. \quad (2.20)$$

To operate on qubit registers, we use quantum gates which form the elementary building blocks of the quantum algorithms considered in this thesis.

**Definition 2.8 (Quantum Gate).** Let  $n \in \mathbb{N}$  be the number of qubits that the gate acts on. A quantum gate acting on  $\mathcal{H}_n$  is then given by any matrix  $U \in U(2^n)$ , where  $U(2^n)$  denotes the group of unitary  $2^n \times 2^n$  matrices. We usually say that  $U \in U(2^n)$  is a unitary instead of a unitary matrix.

The unitarity is precisely coming from the condition that the norm  $\|\psi\| = 1$  of the state has to be preserved. Since any unitary matrix  $U$  has an inverse  $U^{-1} = U^\dagger$ , this definition implies that any quantum gate is reversible. It is thus possible to *uncompute* any operation by applying its inverse. This is commonly done to achieve a lower memory (qubit) usage since the qubits can then be reused for other purposes.

Let us consider the case where  $|\psi\rangle \in \mathcal{H}_n$  but the operator  $U \in U(2^m)$  acts only on a subsystem of size  $m < n$ . Denote by  $k \in \{0, \dots, n - m\}$  the first qubit that  $U$  acts on. By extending  $U$  with identities, we obtain the unitary

$$\tilde{U} = I_{2^k} \otimes U \otimes I_{2^{n-(k+m)}} \in U(2^n) \quad (2.21)$$

which acts on  $\mathcal{H}_n$ . Important examples for single-qubit gates are the Pauli gates  $X$ ,  $Y$ ,  $Z$  and the Hadamard gate  $H$ :

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \quad (2.22)$$

The Hadamard gate maps the computational basis  $\{|0\rangle, |1\rangle\}$  to the states

$$|+\rangle := H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \quad (2.23)$$

$$|-\rangle := H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle), \quad (2.24)$$

which also form a basis of  $\mathcal{H}_1$ . For  $x \in \{0, 1\}$ , this can be summarized by

$$H|x\rangle = \frac{1}{\sqrt{2}}(|0\rangle + (-1)^x|1\rangle). \quad (2.25)$$

Applying one Hadamard gate to each qubit in the multi-qubit state yields the *uniform superposition*

$$H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle, \quad (2.26)$$

where each summand has the same amplitude  $2^{-1/2}$ . This follows since expanding all Kronecker products gives us every possible integer in binary representation. For instance,

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2^2}}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) \quad (2.27)$$

$$= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle + |2\rangle + |3\rangle). \quad (2.28)$$

The *CNOT* or *CX* gate is an example of a multi-qubit gate. It is the *controlled* version of the  $X$  gate, meaning that the action on the *target qubit* depends on the value of the *control qubit*:

$$\text{CNOT}|0\rangle|x\rangle = |0\rangle|x\rangle, \quad \text{CNOT}|1\rangle|x\rangle = |1\rangle|\bar{x}\rangle, \quad (2.29)$$

where  $x \in \{0, 1\}$  and  $\bar{x}$  denotes the bitwise NOT operation applied to  $x$ . In this case, the first qubit is the control qubit and the second qubit is the target qubit. The CNOT gate applies an  $X$  gate to the second qubit if and only if the first qubit is in the state  $|1\rangle$ . Knowing how the CNOT gate acts on the basis states, its action can be extended to any other two-qubit state by linearity. The corresponding matrix looks like this:

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} I_2 & 0 \\ 0 & X \end{pmatrix} = |0\rangle\langle 0| \otimes I_2 + |1\rangle\langle 1| \otimes X, \quad (2.30)$$

where  $I_2$  is the  $2 \times 2$  identity matrix and  $|i\rangle\langle i|$  is the Kronecker product  $|i\rangle \otimes \langle i|$  of the column and row vectors  $|i\rangle$  and  $\langle i| = |i\rangle^\dagger$ . The last representation is useful for calculations in the Dirac notation. It can be shown that the set of all single qubit gates combined with the CNOT gate suffices to implement any  $n$ -qubit unitary. Moreover, there are finite sets of gates that can be used to approximate any unitary  $U \in U(2^n)$  in the sense that there exist finitely many  $V_i$  from the set such that

$$\max_{|\psi\rangle \in E_n} \left\| \left( U - \prod_i \tilde{V}_i \right) |\psi\rangle \right\|$$

is arbitrarily small, where  $\tilde{V}_i$  denotes the extension of  $V_i$  to an  $n$ -qubit gate. The standard set of universal gates consists of the Hadamard, CNOT,  $R_1$  and  $R_3$  gates, where the phase shift gate  $R_k$  is given by

$$R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix}. \quad (2.31)$$

Often, it is also helpful to be able to control a gate on multiple qubits. Given a single-qubit gate  $U$ , the CCU gate is the double controlled version of the  $U$  gate, and works in a similar fashion as the CNOT gate:

$$\text{CCU } |x\rangle |y\rangle |z\rangle = \begin{cases} |x\rangle |y\rangle |z\rangle & (x, y) \in \{ (0, 0), (0, 1), (1, 0) \}, \\ |1\rangle |1\rangle U |z\rangle & x = y = 1. \end{cases} \quad (2.32)$$

The corresponding matrix also looks very similar:

$$\text{CCU} = \begin{pmatrix} I_6 & 0 \\ 0 & U \end{pmatrix}. \quad (2.33)$$

It should be noted that controlled gates are inherently different from classical if-constructions in that they can also modify the state of the control qubit. Consider for instance the product state  $|\psi\rangle = |+\rangle |0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) |0\rangle$  where the first qubit is in the state  $|+\rangle$ . Applying a CNOT gate to this state yields a Bell state

$$\text{CNOT } |\psi\rangle = \frac{1}{\sqrt{2}}(\text{CNOT } |0\rangle |0\rangle + \text{CNOT } |1\rangle |0\rangle) = \frac{1}{\sqrt{2}}(|0\rangle |0\rangle + |1\rangle |1\rangle). \quad (2.34)$$

Since this is not a product state, it is not possible to separately specify the state of the first qubit. Hence, the first qubit is not in the  $|+\rangle$  state anymore.

## 2.3. Quantum Circuits

To describe quantum algorithms we wish to have a gate-level description that allows us to quickly see which operations are applied to which qubits. Since most implementations

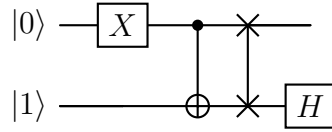


Figure 2.1.: Example of a quantum circuit diagram.

build on two-level systems (qubits) as a base memory resource, such a description is close to the actual hardware. The *quantum circuit model* is a computational model that is inspired by the classical circuit model which uses wires and a set of gates to perform elementary logical operations. In the following, we assume that we can implement arbitrary gates. In practice, we might only be able to implement a set of universal gates directly, depending on the architecture of the quantum computer at hand. It would then be necessary to use a compiler that decomposes these gates using the elementary gates that we know how to implement.

The circuits are commonly depicted as exemplified in Figure 2.1. We first initialize the circuit in the state  $|\psi\rangle = |0\rangle \otimes |1\rangle$ . The next step to understand the action of the gates on  $|\psi\rangle$  is to calculate the output state  $|\psi'\rangle$ . First, we apply an  $X$  gate to the first qubit:

$$|\psi_1\rangle = (X \otimes I_2) \cdot |\psi\rangle = (X \otimes I_2) \cdot |0\rangle |1\rangle = |1\rangle |1\rangle , \quad (2.35)$$

where we extended the  $X$  gate to a two-qubit gate  $X \otimes I_2$  using the 1-qubit identity  $I_{2^1} = I_2$ . We then apply the CNOT gate, which is depicted using a dot on the control qubit and an XOR symbol on the target qubit, connected with a line:

$$|\psi_2\rangle = \text{CNOT} \cdot |\psi_1\rangle = |1\rangle |0\rangle . \quad (2.36)$$

The next gate is the SWAP gate, which is depicted using two connected crosses. The gate simply swaps the two qubits:

$$|\psi_3\rangle = \text{SWAP} \cdot |\psi_2\rangle = |0\rangle |1\rangle . \quad (2.37)$$

Finally, we apply the Hadamard gate to the second qubit, again extending it with an identity:

$$|\psi'\rangle = (I_2 \otimes H) \cdot |\psi_3\rangle = |0\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) . \quad (2.38)$$

Summarized in one line, the circuit performs the following transformation:

$$|\psi'\rangle = (I_2 \otimes H) \cdot \text{SWAP} \cdot \text{CNOT} \cdot (X \otimes I_2) \cdot |\psi\rangle . \quad (2.39)$$

Note that we could also first calculate all matrix-matrix products, which would yield another unitary matrix. However, we would need many more operations than for matrix-vector products.

## 2. Preliminaries

---

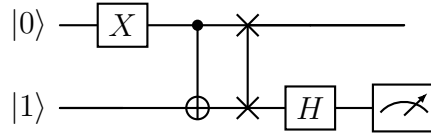


Figure 2.2.: Example of a quantum circuit diagram with measurement.

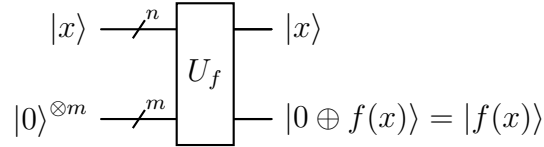


Figure 2.3.: Example usage of a quantum oracle. By leaving the input  $|x\rangle$  intact, we do not destroy any information which allows us to revert the operation by applying  $U_f$  again.

Another important part of quantum circuits are measurements. Measurements are non-unitary operations, so they are mostly used at the end of circuits. For instance, we could extend our above circuit to include a measurement of the second qubit, as seen in Figure 2.2. Since the second qubit is in the state  $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ , we would measure both 0 and 1 with equal probability

$$p_0 = p_1 = \left(\pm \frac{1}{\sqrt{2}}\right)^2 = \frac{1}{2}. \quad (2.40)$$

There are some major differences to classical circuits. Most importantly, we cannot simply read or write the state of a given qubit. This means that the input state should be initialized to a known value, such as  $|0\rangle^{\otimes n}$ . Another important difference is that every gate must be reversible. Since quantum computers can compute any function that is computable by classical computers, there must be a way to compute irreversible functions. This can be realized using a *quantum oracle*.

**Definition 2.9 (Oracle).** Let  $n, m \in \mathbb{N}$  and  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  be a Boolean function. The corresponding oracle that implements  $f$  reversibly is the unitary  $U_f \in U(2^{n+m})$  defined by

$$U_f |x\rangle |y\rangle = |x\rangle |y \oplus f(x)\rangle, \quad (2.41)$$

where  $|x\rangle \in \mathcal{H}_n$ ,  $|y\rangle \in \mathcal{H}_m$  and  $\oplus$  denotes the bitwise XOR operation.

Such an oracle can be used in a quantum circuit as seen in Figure 2.3. Assuming that the initial state is  $|0\rangle^{\otimes m}$ , the oracle leaves the input register unchanged and outputs  $f(x)$  in the output register. By *register* we mean a group of consecutive qubits. The diagonal slash shows that the single line in the diagram corresponds to multiple wires. The basic idea of an oracle is to not apply the function in-place, but instead apply it to an additional

set of qubits while keeping the previous information to preserve reversibility. Since the XOR operation  $\oplus$  is self-inverse, we can simply apply it again to invert the action of the oracle. More generally, an oracle is a black box that implements a given operation.

## 2.4. Computational Complexity

In this section, we introduce the notation needed to describe the complexity of algorithms. The goal of complexity theory is to get bounds on the resources needed to solve specific problems. Since there are often many different algorithms for a single computational problem, we want to be able to compare algorithms in terms of their performance. This can be achieved using complexity classes, where we consider the resources used by a specific algorithm in terms of the input size  $n$ . For instance, for a quantum linear systems algorithm, the input size would correspond to the size  $n$  of the matrix  $A \in \mathbb{C}^{n \times n}$ . The big O notation or asymptotic notation is an important concept that we use throughout this thesis [Weg05].

**Definition 2.10 (Asymptotic Notation).** *Let  $n \in \mathbb{N}$  and let  $g : \mathbb{N} \rightarrow \mathbb{R}_+$ . We then denote by  $O(g(n))$  the set*

$$O(g(n)) = \{ f : \mathbb{N} \rightarrow \mathbb{R}_+ \mid \exists k > 0 \exists n_0 \in \mathbb{N} : (\forall n > n_0 : f(n) \leq kg(n)) \}, \quad (2.42)$$

*or in words: The value of  $f$  is asymptotically bounded from above by  $g$  up to a constant factor.*

*Moreover, the set  $\Omega(g(n))$  is given by*

$$\Omega(g(n)) = \{ f : \mathbb{N} \rightarrow \mathbb{R}_+ \mid \exists k > 0 \exists n_0 \in \mathbb{N} : (\forall n > n_0 : f(n) \geq kg(n)) \}, \quad (2.43)$$

*or in words: The value of  $f$  is asymptotically bounded from below by  $g$  up to a constant factor. The set  $\Theta(g(n))$  is then defined by*

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n)). \quad (2.44)$$

Slightly abusing notation, we also write  $f = O(g(n))$  instead of  $f \in O(g(n))$ , where the  $=$  sign is obviously not symmetric in this case. For instance, finding an item in an unsorted list of size  $n$  can be done in linear time  $O(n)$ , since one has to check  $\frac{n}{2} \in O(n)$  items on average. This upper bound can be obtained by counting elementary operations such as array lookups and insertions which can be done in constant time  $O(1)$ .

**Notation 2.11.** *We write  $f \in O(\text{poly}(n))$ , if  $f \in O(n^k)$  for some  $k \in \mathbb{N}$ . Similarly, we write  $O(\text{polylog}(n)) = O(\text{poly}(\log(n)))$  for polynomials of logarithms. Often, we are only interested in the complexity up to polylogarithmic factors. The notation  $f \in \tilde{O}(g(n))$  means that  $f \in O(g(n) \text{poly}(\log(n)))$ .*

We often say that problems which can be solved in time  $O(\text{poly}(n))$  are *tractable* problems, that is, they can be efficiently solved using a digital computer. While this is not accurate due to possibly arbitrarily large powers and constant factors, it is often a useful concept.

In the following, we consider not only classical, but also quantum algorithms. When studying quantum algorithms, we are generally interested in the existence of a *quantum speedup*. That is, there exists a quantum algorithm which runs asymptotically faster than the best known classical algorithm. For comparison, we use the bounds provided by the asymptotic notation. To obtain an upper bound on the run time in the quantum case we can also count elementary operations which would now be the universal 1- and 2-qubit gates. These can directly be implemented on an actual quantum computer and hence make up one *unit* of time. For the quantum algorithms studied in this thesis, we use the following notion of efficiency [CKS17].

**Definition 2.12 (Query and Gate Complexity).** *The query complexity  $Q_{\mathcal{A}}^U$  of a quantum algorithm  $\mathcal{A}$  with respect to an oracle  $U$  is given by the number of applications of  $U$  in the algorithm. By gate complexity we mean the number of 2-qubit gates used in the algorithm. We then say that an algorithm is gate-efficient, if its gate complexity is larger than its query complexity only by polylogarithmic factors in the quantities of interest. Formally, an algorithm with query complexity  $Q_{\mathcal{A}}^U$  and input size  $N$  is gate efficient if its gate complexity is*

$$O(Q_{\mathcal{A}}^U \text{poly}(\log(Q_{\mathcal{A}}^U), \log(N))). \quad (2.45)$$

For instance, considering a quantum linear systems algorithm, the input size  $N = 2^n$  is given by the size of the matrix, where  $n \in \mathbb{N}$  is the number of qubits needed to store the right-hand side  $b$ .

## 2.5. Quantum Fourier Transform

Before we introduce the quantum fourier transform, we consider the discrete Fourier transform (DFT). The DFT maps a vector of uniform samples  $x \in \mathbb{C}^n$  to the Fourier components  $\hat{x} \in \mathbb{C}^n$ . For instance, the sample vector  $x$  could be time-discrete measured data from a periodic signal. Then the DFT  $\hat{x}$  encodes the input signal in the frequency domain. Formally, the DFT is defined as follows.

**Definition 2.13 (Discrete Fourier Transform).** *Let  $N \in \mathbb{N}$ . The discrete fourier transform is the linear map  $\mathcal{F}_N : \mathbb{C}^N \rightarrow \mathbb{C}^N$  given by the transformation matrix*

$$(\mathcal{F}_N)_{jk} = \frac{1}{\sqrt{N}} e^{2\pi i jk/N}. \quad (2.46)$$

Due to the normalization factor  $\frac{1}{\sqrt{N}}$ , it is unitary with the inverse

$$(\mathcal{F}_N^{-1})_{jk} = \frac{1}{\sqrt{N}} e^{-2\pi i j k / N}. \quad (2.47)$$

Since we defined the DFT as a unitary transformation, we already have the matrix representation of the Quantum Fourier Transform (QFT) which is essentially the DFT. We now assume  $N = 2^n$  for simplicity where  $n \in \mathbb{N}$  is the number of qubits that the QFT acts on. The action on a basis state  $|j\rangle \in \{|0\rangle, \dots, |N-1\rangle\}$  is then denoted by

$$\text{QFT}_N |j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k / N} |k\rangle. \quad (2.48)$$

The QFT differs from the classical DFT in that it can be implemented more efficiently on a quantum computer.

**Theorem 2.14 (Quantum Fourier Transform).** *Let  $n \in \mathbb{N}$  be the number of qubits. Then the circuit given in Figure A.1 (appendix) followed by at most  $n/2$  swap gates implements the quantum analogue of the DFT on  $N = 2^n$  samples using  $\Theta(n^2)$  gates.*

This is exponentially less than for the classical DFT, even compared to the  $\Theta(n2^n) = \Theta(N \log(N))$  operations needed for the fast Fourier transform. However, as with every quantum algorithm, this does not directly give us a speedup for practical application since we also need to consider the input and output of the data.

To understand how the implementation of the QFT works, let us consider a simple example with three qubits. We denote the input state by  $|j\rangle = |j_1 \dots j_n\rangle$ , where  $j = j_1 2^{n-1} + j_2 2^{n-2} + \dots + j_n 2^0$ . Moreover, we use the following representation for binary fractions:

$$0.j_1 j_2 \dots j_m = j_1/2 + j_2/4 + \dots + j_m/2^{m-l+1}. \quad (2.49)$$

Using this notation, it can then be shown that the output state  $|k\rangle$  of the QFT can be given using the product representation:

$$\text{QFT}_N |j\rangle = \frac{(|0\rangle + e^{2\pi i 0.j_n} |1\rangle)}{\sqrt{2}} \frac{(|0\rangle + e^{2\pi i 0.j_{n-1}j_n} |1\rangle)}{\sqrt{2}} \dots \frac{(|0\rangle + e^{2\pi i 0.j_1 j_2 \dots j_n} |1\rangle)}{\sqrt{2}}. \quad (2.50)$$

The circuit diagram for the 3-qubit QFT is given in Figure 2.4. The input state is given by  $|\psi_0\rangle = |j_1 j_2 j_3\rangle$ . We begin by applying a Hadamard gate to the first qubit:

$$|\psi_1\rangle = (H \otimes I \otimes I) |\psi_0\rangle \quad (2.51)$$

$$= \frac{1}{\sqrt{2}} (|0\rangle + (-1)^{j_1} |1\rangle) |j_2 j_3\rangle \quad (2.52)$$

$$= \frac{1}{\sqrt{2}} (|0\rangle + e^{2\pi i 0.j_1} |1\rangle) |j_2 j_3\rangle. \quad (2.53)$$

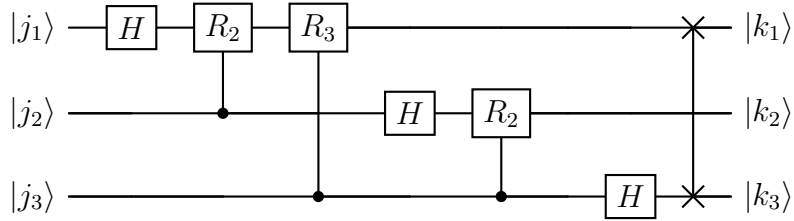


Figure 2.4.: Circuit diagram of the 3-qubit quantum Fourier transform.

Note that due to the swap gates at the end of the circuit, the goal is to build the state of the last qubit in Equation 2.50. We have already created a relative phase of  $2\pi 0.j_1$ , whereas the relative phase of the final state should be  $2\pi 0.j_1j_2j_3$ . This can be achieved using controlled phase gates. In the case of  $j_2 = 0$ , applying the controlled  $R_2$  gate  $\text{CR}_2$  does not change the state. If  $j_2 = 1$ , the  $R_2$  gate adds a relative phase of  $e^{2\pi i/4}$  to  $|1\rangle$ . Since the phase is given by  $2\pi i/4 = 2\pi i 0.j_2$  in both cases and  $0.j_1 + 0.j_2 = 0.j_1j_2$ , this can be summarized as

$$|\psi_2\rangle = (\text{CR}_2 \otimes I) |\psi_1\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 0.j_1j_2} |1\rangle) |j_2j_3\rangle. \quad (2.54)$$

Finally, we apply the  $R_3$  gate controlled on  $j_3$  to obtain the correct state of the last qubit in Equation 2.50. We then apply the same idea for the second and first qubit and finally use swap gates to obtain the same bit order as in the input state. The swapping step is optional and might be omitted, depending on the required bit order. Figure A.1 shows the general  $n$ -qubit QFT circuit excluding the final swap gates. The inverse QFT can be obtained by reversing the order of gates while taking the inverse of each gate. One of the most common applications of the QFT is quantum phase estimation.

## 2.6. Quantum Phase Estimation

In many quantum algorithms, it is useful to know the eigenvalues of a given unitary  $U$ . Assume that we are given a register containing an eigenstate  $|u\rangle$  of  $U$ . Since  $U$  is unitary, the corresponding eigenvalue can be written in the form  $e^{2\pi i\varphi}$ . Suppose that we can implement the controlled powers  $U^{2^j}$  of  $U$  for  $j \in \mathbb{N}$ . Under these assumptions, the Quantum Phase Estimation (QPE) algorithm allows us to estimate the value of  $\varphi \in [0, 1)$  and hence the eigenvalue.

**Theorem 2.15 (Quantum Phase Estimation).** *Let  $\varepsilon > 0$ , let  $n \in \mathbb{N}$  be the size of the second register and let  $U \in U(2^n)$  be a unitary operator. Assume that the second register is initialized in the eigenstate  $|u\rangle \in \mathcal{H}_n$  with  $U|u\rangle = e^{2\pi i\varphi}|u\rangle$ . Choose the size of the first*

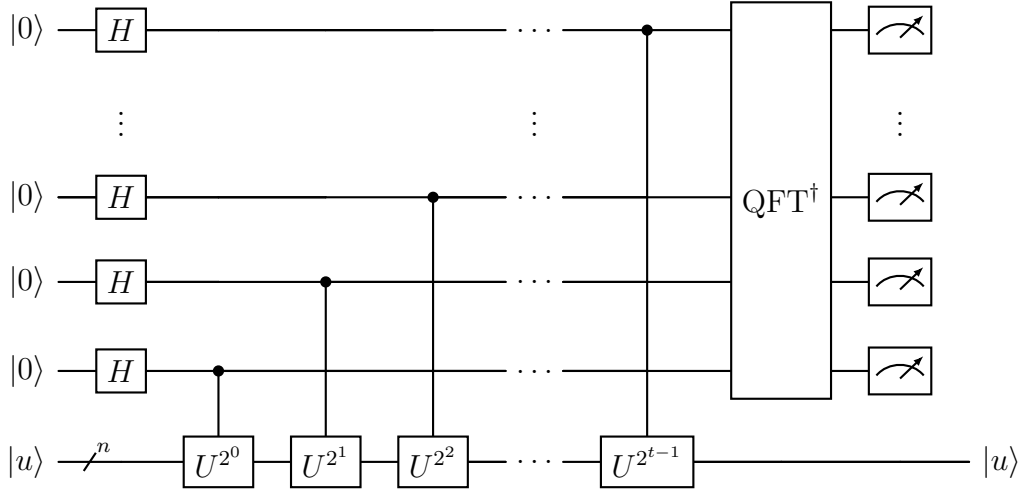


Figure 2.5.: Circuit diagram of the phase estimation algorithm. The last register is initialized in the eigenstate  $|u\rangle$  of  $U$ . By creating a uniform superposition in the first register and applying the controlled  $U^{2^j}$  for  $j \in \{0, \dots, t-1\}$  we obtain a state whose phases encode the approximate eigenvalue  $e^{2\pi i\tilde{\varphi}}$ . These phase are transferred to a binary representation  $|\tilde{\varphi}\rangle$  using the inverse QFT.

register as

$$t = n + \left\lceil \log \left( 2 + \frac{1}{2\varepsilon} \right) \right\rceil, \quad (2.55)$$

which is initialized to  $|0\rangle^{\otimes t}$ . Then applying the quantum phase estimation algorithm given in Figure 2.5 yields an estimate of  $\varphi$  accurate to  $n$  bits with probability of success  $p \geq 1 - \varepsilon$ .

In this case, the probability of success is given by the probability that measuring the first register yields an integer  $b \in \mathbb{N}$  such that  $\tilde{\varphi} = b2^{-t}$  is an approximation of  $\varphi$  accurate to  $n$  bits.

The general idea of the algorithm is to create a state where the relative phases encode an approximation of  $\varphi$  which can then be transferred to a binary representation  $|\varphi\rangle$  using the inverse QFT. To understand how the algorithm works, assume for now that  $\varphi = 0.\varphi_1 \dots \varphi_t$  can be exactly represented using  $t$  bits. The first step is to create a uniform superposition in the first register:

$$|\psi_1\rangle = (H^{\otimes t} |0\rangle^{\otimes t}) |u\rangle = |+\rangle^{\otimes t} |u\rangle. \quad (2.56)$$

Denote by  $C_k$  the controlled unitary  $U^{2^k}$  with control qubit  $t-1-k$ , such that  $C_0$  is the leftmost controlled unitary in Figure 2.5. Then  $C_k |x\rangle |u\rangle = |x\rangle U^{2^k} |u\rangle$  if bit  $t-1-k$  of the binary representation of  $x$  is one, and  $C_k$  performs no action otherwise. Hence,

## 2. Preliminaries

---

applying  $C_k$  for all  $k \in \{0, \dots, t-1\}$ , we obtain

$$|\psi_2\rangle = \left( \prod_{k=0}^{t-1} C_k \right) \left( \bigotimes_{k=0}^{t-1} \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \right) \otimes |u\rangle \quad (2.57)$$

$$= \frac{1}{\sqrt{2^t}} \left( |0\rangle^{\otimes t} |u\rangle + \left( \prod_{k=0}^{t-1} C_k \right) |1\rangle^{\otimes t} |u\rangle \right) \quad (2.58)$$

$$= \frac{1}{\sqrt{2^t}} \left( |0\rangle^{\otimes t} |u\rangle + |1\rangle^{\otimes t} \prod_{k=0}^{t-1} e^{2\pi i 2^k \varphi} |u\rangle \right) \quad (2.59)$$

$$= \frac{1}{\sqrt{2^t}} \bigotimes_{k=t-1}^0 (|0\rangle + e^{2\pi i 2^k \varphi} |1\rangle) |u\rangle \quad (2.60)$$

$$= \frac{1}{\sqrt{2^t}} (|0\rangle + e^{2\pi i 0 \cdot \varphi_t} |1\rangle) (|0\rangle + e^{2\pi i 0 \cdot \varphi_{t-1} \varphi_t} |1\rangle) \dots (|0\rangle + e^{2\pi i 0 \cdot \varphi_1 \varphi_2 \dots \varphi_t} |1\rangle) |u\rangle. \quad (2.61)$$

To see more clearly how the  $C_k$  act on the state, we first split the sum to obtain two summands of the form  $|x\rangle |u\rangle$ . Since all bits of  $|0\rangle^{\otimes t}$  are zero,  $C_k$  performs no action on the first summand. Equation 2.59 follows directly from the definition of  $C_k$  and the eigenvalue relation. By collecting the terms, we see that in each of the tensor factor,  $|1\rangle$  now has a relative phase that depends on  $k$ . The last equation follows from the fact that the complex exponential is periodic together with the assumption that  $\varphi$  can be represented exactly, which implies:

$$e^{2\pi i 2^{t-1} \varphi} = e^{2\pi i 2^{t-1} (\varphi_1 2^{-1} + \dots + \varphi_t 2^{-t})} \quad (2.62)$$

$$= e^{2\pi i \varphi_t 2^{-1}} \prod_{j=2}^t e^{2\pi i \varphi_j 2^{t-j}} \quad (2.63)$$

$$= e^{2\pi i 0 \cdot \varphi_t}, \quad (2.64)$$

where the product is equal to 1 since  $\varphi_j 2^{t-j} \in \mathbb{N}$  for all  $j = 2, \dots, t$ . The same idea can be used to obtain the remaining factors in Equation 2.61, which is exactly the product representation of the QFT given in Equation 2.50. The last step is then to apply the inverse QFT:

$$|\psi_3\rangle = (\text{QFT}^\dagger \otimes I) |\psi_2\rangle = |\varphi\rangle |u\rangle. \quad (2.65)$$

Let us briefly consider the case where  $\varphi$  cannot be exactly represented using  $t$  bits. Expanding all the terms in Equation 2.59, we obtain

$$|\psi_2\rangle = \frac{1}{\sqrt{2^t}} \sum_{k=0}^{2^t-1} e^{2\pi i \varphi k} |k\rangle |u\rangle. \quad (2.66)$$

Applying the inverse QFT then yields

$$|\psi_3\rangle = \frac{1}{2^t} \sum_{k,l=0}^{2^t-1} e^{-2\pi i k l / 2^t} e^{2\pi i \varphi k} |l\rangle |u\rangle \quad (2.67)$$

$$= \sum_{l=0}^{2^t-1} \frac{1}{2^t} \sum_{k=0}^{2^t-1} e^{2\pi i k (\varphi 2^t - l) / 2^t} |l\rangle |u\rangle \quad (2.68)$$

$$= \sum_{l=0}^{2^t-1} c_l |l\rangle |u\rangle, \quad (2.69)$$

where  $c_l = 2^{-t} \sum_{k=0}^{2^t-1} e^{2\pi i k (\varphi 2^t - l) / 2^t}$ . Let  $b \in \{0, \dots, 2^t - 1\}$  be the closest integer to  $\varphi 2^t$ , then  $|\varphi - b/2^t| \leq 2^{-t}$ . It can be shown that the probability  $p_b = |c_b|^2$  of measuring  $|b\rangle$  is high such that measuring the final state will yield an approximation to  $\varphi 2^t$  and thus  $\varphi$  with high probability. Using  $t$  qubits for the estimation register as required by Theorem 2.15, we obtain an estimate of  $\varphi$  that is accurate to  $n$  bits with probability of success of at least  $1 - \varepsilon$ .

Next, consider the case where the input state  $|\psi\rangle \in E_n$  of the QPE routine is not an eigenstate. We can then decompose  $|\psi\rangle$  using the eigenstates  $|u\rangle$  of  $U$ :

$$|\psi\rangle = \sum_u c_u |u\rangle. \quad (2.70)$$

In this case, applying the QPE algorithm yields a state close to  $\sum_u c_u |\tilde{\varphi}\rangle |u\rangle$ , where  $\tilde{\varphi}$  is a good approximation to the phase  $\varphi$ .

## 2.7. Amplitude Amplification

To motivate amplitude amplification, let  $n \in \mathbb{N}$  and consider the unstructured search problem which consists of finding a marked element in a set  $X = \{x_0, \dots, x_{N-1}\}$ . The problem is to find the unique element  $x_j \in X$ . This can be reformulated using a Boolean function  $\chi : \{0, \dots, N-1\} \rightarrow \{0, 1\}$ , by asking instead to find an index  $k$  such that  $\chi(k) = 1$ . In our case,  $\chi(k) = \delta_{jk}$ . Basis states  $|k\rangle \in E_n$  are called *good* if  $\chi(k) = 1$ , and *bad* otherwise. In the classical case, this cannot be solved faster than in time  $\Theta(N)$ , since in the worst case, each element has to be checked to find the solution.

Suppose that we have the ability to check if an element  $x_j \in X$  is a solution to our problem, that is,  $\chi(j) = 1$ . Consider the state obtained by applying  $\mathcal{A} = H^{\otimes n}$  to the initial state  $|0\rangle$ :

$$|\psi\rangle = \mathcal{A}|0\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} |k\rangle = \frac{1}{\sqrt{N}} |j\rangle + \frac{1}{\sqrt{N}} \sum_{\substack{k=0 \\ k \neq j}}^{N-1} |k\rangle. \quad (2.71)$$

## 2. Preliminaries

---

The naive approach would be to measure this state and check if  $\chi(j) = 1$  for each outcome. However, since the probability of measuring  $j$  is only  $p = \frac{1}{N}$ , we would also need  $\Theta(N)$  applications of  $\mathcal{A}$  to find  $j$  which encodes the solution to the problem. In such a situation, amplitude amplification increases the amplitude  $\frac{1}{\sqrt{N}}$  of the summand  $|j\rangle$  with  $\chi(j) = 1$  to obtain a state

$$|\psi'\rangle = \sqrt{p'} |j\rangle + \sqrt{1-p'} \sum_{\substack{k=0 \\ k \neq j}}^{N-1} |k\rangle \quad (2.72)$$

with a higher probability  $p' > \frac{1}{N}$  of measuring  $j$ . Using amplitude amplification, a solution can be found using only  $\Theta(\sqrt{N})$  evaluations of  $\chi$ , which corresponds to a quadratic speedup. The algorithm also works in the case where multiple values  $j_i$  fulfill  $\chi(j_i) = 1$ . The general result is given by the following theorem.

**Theorem 2.16 (Quadratic Speedup).** *Let  $\mathcal{A}$  be any quantum algorithm that uses no measurements, let  $n \in \mathbb{N}$  and let  $\chi : \{0, \dots, 2^n - 1\} \rightarrow \{0, 1\}$  be a Boolean function encoding the problem, such that  $|x\rangle$  is a good state if  $\chi(x) = 1$  and  $\chi(x) = 0$  otherwise. Moreover, let  $p$  be the initial success probability of  $\mathcal{A}$ . Suppose  $p > 0$  is known, and set  $m = \left\lfloor \frac{\pi}{4\theta_p} \right\rfloor$ , where  $\theta_p$  is defined so that  $\sin^2(\theta_p) = p$  and  $0 < \theta_p < \pi/2$ . Then, there exists a unitary  $Q$  such that computing  $Q^m \mathcal{A} |0\rangle$  and measuring the resulting state yields a good outcome with probability at least  $\max(1-p, p)$ . Overall, this procedure needs  $(2m+1)/\max(1-p, p) = \Theta(1/\sqrt{p})$  uses of  $\mathcal{A}$  to obtain a good outcome.*

This is Theorem 2 in the article by Brassard et al. [Bra+02]. For simplicity, we only consider the case where the probability of success is known a priori. However, Brassard et al. also give a similar algorithm to obtain a quadratic speedup for the case where  $p$  is unknown.

Let us now take a look at how amplitude amplification works. The algorithm leverages the algorithm  $\mathcal{A}$  and its inverse to build an operator  $Q$  that increases the amplitudes of the good states. First, observe that any Boolean function  $\chi$  automatically partitions  $\mathcal{H}_n$  into a direct sum of subspaces

$$\mathcal{H}_n = \mathcal{H}_g \oplus \mathcal{H}_b, \quad (2.73)$$

where  $\mathcal{H}_g$  is spanned by all good basis states and  $\mathcal{H}_b$  is its orthogonal complement in  $\mathcal{H}_n$ . Denote by  $P_g$  and  $P_b$  the orthogonal projections onto the good and bad subspaces, respectively. The first step of the algorithm is to use the algorithm  $\mathcal{A}$  once to obtain

$$|\psi\rangle := \mathcal{A} |0\rangle = P_g |\psi\rangle + P_b |\psi\rangle = |\psi_g\rangle + |\psi_b\rangle, \quad (2.74)$$

where  $|\psi_g\rangle$  and  $|\psi_b\rangle$  do not need to be known to apply the algorithm. Denote by  $p_g = \langle \psi_g | \psi_g \rangle$  the probability of measuring a good state and by  $p_b = \langle \psi_b | \psi_b \rangle$  the probability of

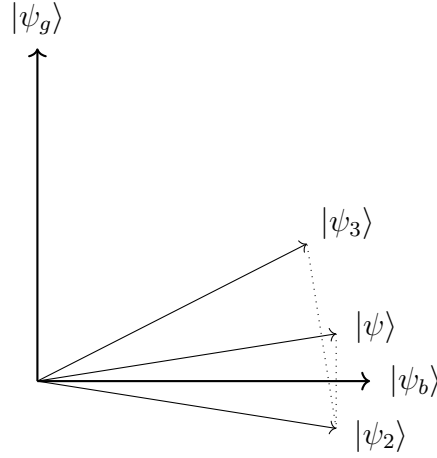


Figure 2.6.: Illustration of the subspace rotation used for amplitude amplification. We begin with the state  $|\psi_1\rangle = |\psi\rangle$  and then reflect about  $|\psi_b\rangle$ , followed by a reflection about  $|\psi\rangle$ . This results in a higher probability to measure a good state.

measuring a bad state. By assumption, the initial success probability  $p_g$  is small compared to  $p_b$ . To understand how  $p_g$  can be increased, consider the two-dimensional subspace

$$\mathcal{H}_\psi = \text{span}\{|\psi_b\rangle, |\psi_g\rangle\} = \text{span}\{|\psi_b\rangle, |\psi\rangle\}. \quad (2.75)$$

Figure 2.6 shows the action of the operator  $Q$  from Theorem 2.16 on this subspace. At first, the angle between  $|\psi\rangle$  and  $|\psi_b\rangle$  is small, corresponding to a high probability  $p_b$ . The idea is to rotate  $|\psi\rangle$  closer towards  $|\psi_g\rangle$ , increasing the probability  $p_g$ . Since such a rotation cannot directly be implemented, it is decomposed into two Householder reflections. The first reflection is across the line given by  $|\psi_b\rangle$ , which does not change  $p_g$ . The next reflection across the line given by  $|\psi\rangle$  then rotates the state closer towards  $|\psi_g\rangle$ . After  $m$  such rotations with  $m$  given by Theorem 2.16, the probability  $p_g$  is as high as possible.

Formally, the action of  $Q$  on the subspace  $\mathcal{H}_\psi$  is given by

$$Q' = U_\psi U_{\psi_b}, \quad (2.76)$$

where

$$U_\psi = I - 2|\psi\rangle\langle\psi| = \mathcal{A}\mathcal{A}^\dagger - 2\mathcal{A}|0\rangle\langle 0|\mathcal{A}^\dagger = \mathcal{A}(I - 2|0\rangle\langle 0|)\mathcal{A}^\dagger \quad (2.77)$$

realizes the reflection across  $|\psi\rangle$  and

$$U_{\psi_b} = I - \frac{2}{p_b}|\psi_b\rangle\langle\psi_b| \quad (2.78)$$

realizes the reflection across  $|\psi_b\rangle$ . The general operator  $Q$  is given by

$$Q = -\mathcal{A}S_0\mathcal{A}^{-1}S_\chi, \quad (2.79)$$

where the conditional sign change operator  $S_\chi$  maps

$$|x\rangle \mapsto \begin{cases} -|x\rangle, & \chi(x) = 1 \\ |x\rangle, & \chi(x) = 0 \end{cases}, \quad (2.80)$$

whereas  $S_0 = I - 2|0\rangle\langle 0|$  changes the sign of  $|x\rangle$  if and only if  $x = 0$ . In Equation 2.77, we used the assumption that  $\mathcal{A}|0\rangle = |\psi\rangle$ .

## 2.8. State Preparation

In general, we assume that a quantum computer can initialize the basis state  $|0 \dots 0\rangle \in E_n$  for some  $n \in \mathbb{N}$ . Quantum state preparation is concerned with the problem of approximately preparing a general superposition

$$|\psi\rangle = \sum_{j=0}^{2^n-1} \alpha_j |j\rangle \in E_n, \quad (2.81)$$

where the amplitudes  $\alpha_i \in \mathbb{C}$  are given. More precisely, for a given tolerance  $\varepsilon > 0$ , the problem is to construct a quantum algorithm  $\mathcal{A}$  such that

$$\| |\psi\rangle - \mathcal{A}|0\rangle \|_2 < \varepsilon. \quad (2.82)$$

For instance, when solving the system

$$A|x\rangle = |b\rangle \quad (2.83)$$

on a quantum computer, encoding the right hand side  $|b\rangle$  can be done using a state preparation algorithm. This is a nontrivial problem due to the inherent restrictions of quantum computing: Instead of simply *writing* the desired state to a sequence of qubits, we need to find a series of unitary operations which gradually modify the input state  $|0\rangle$ . Since we are interested in efficient quantum algorithms that outperform the classical counterparts, the state preparation algorithm also needs to be efficient.

The Grover-Rudolph algorithm is a simple state preparation algorithm that can be applied when  $\alpha_j \geq 0$  [GR02].

**Theorem 2.17 (Grover-Rudolph).** *Let  $p$  be a probability density and  $\{p_j\}_{j \in J}$  a discretization of  $p$  over an index set  $J$  with  $|J| = N = 2^n \in \mathbb{N}$  samples. Assume that the*

density function can be efficiently integrated. That is, there exists an efficient classical algorithm to calculate

$$P(a \leq x \leq b) = \int_a^b p(x) dx \quad (2.84)$$

for all  $a, b \in \mathbb{R}$ . Then there exists an efficient quantum algorithm to prepare the superposition

$$|\psi\rangle = \sum_{j \in J} \sqrt{p_j} |j\rangle, \quad (2.85)$$

where  $\{|j\rangle \mid j \in J\}$  is an orthonormal set of states.

One well-known class of efficiently integrable probability density functions are *log-concave* distributions which fulfill the condition

$$\frac{\partial^2 \log(p(x))}{\partial x^2} < 0 \quad \forall x. \quad (2.86)$$

Most important statistical distribution such as the exponential and normal families are log-concave.

To understand the idea behind the Grover-Rudolph algorithm, let  $m \in \mathbb{N}$  and assume that the distribution is partitioned into  $2^m$  equally sized regions. Denote by  $p_j^{(m)}$  the probability that a sample  $x$  of the random variable corresponding to  $p$  lies in region  $j$ . Moreover, assume that the  $m$ -qubit state

$$|\psi_m\rangle = \sum_{j=0}^{2^m-1} \sqrt{p_j^{(m)}} |j\rangle \quad (2.87)$$

which encodes these probabilities has already been prepared. The idea of the algorithm is that the state  $|\psi_{m+1}\rangle$  encoding the probabilities corresponding to a partition into  $2^{m+1}$  regions can be built from the previous state  $|\psi_m\rangle$ . Denote by  $\alpha_j$  and  $\beta_j$  the probabilities that  $x$  is in the left and right half of region  $j$ , respectively. The algorithm then performs the operation

$$\sqrt{p_j^{(m)}} |j\rangle |0\rangle \rightarrow \sqrt{\alpha_j} |j\rangle |0\rangle + \sqrt{\beta_j} |j\rangle |1\rangle. \quad (2.88)$$

Applied to the state in Equation 2.87, we obtain

$$|\psi_{m+1}\rangle = \sum_{j=0}^{2^{m+1}-1} \sqrt{p_j^{(m+1)}} |j\rangle. \quad (2.89)$$

To obtain the desired state, this process is repeated until  $m = n$ .

As an example, consider a distribution with  $\{p_i\} = \{p_0, p_1, p_2, p_3\}$ . This corresponds to the state

$$|\psi\rangle = \sqrt{p_0} |00\rangle + \sqrt{p_1} |01\rangle + \sqrt{p_2} |10\rangle + \sqrt{p_3} |11\rangle. \quad (2.90)$$

For  $m = 0$ , the initial state is  $|\psi_0\rangle = 1 |00\rangle$ , since there is only one region which corresponds to the probability  $p_0^{(0)} = 1$ . The probabilities for the left and right half are given by  $\alpha_0 = p_0^{(1)} = p_0 + p_1$  and  $\beta_0 = p_1^{(1)} = p_2 + p_3$ . For  $m = 1$ , the state is then given by

$$|\psi_1\rangle = (\sqrt{p_0 + p_1} |0\rangle + \sqrt{p_2 + p_3} |1\rangle) |0\rangle. \quad (2.91)$$

In this case, the probabilities of the left and right halves are the desired probabilities:

$$\alpha_0 = p_0^{(2)} = p_0, \quad (2.92)$$

$$\beta_0 = p_1^{(2)} = p_1, \quad (2.93)$$

$$\alpha_1 = p_2^{(2)} = p_2, \quad (2.94)$$

$$\beta_1 = p_3^{(2)} = p_3. \quad (2.95)$$

Hence, performing the operation

$$\sqrt{p_0 + p_1} |0\rangle |0\rangle \rightarrow \sqrt{\alpha_0} |0\rangle |0\rangle + \sqrt{\beta_0} |0\rangle |1\rangle, \quad (2.96)$$

$$\sqrt{p_2 + p_3} |1\rangle |0\rangle \rightarrow \sqrt{\alpha_1} |1\rangle |0\rangle + \sqrt{\beta_1} |1\rangle |1\rangle, \quad (2.97)$$

we obtain  $|\psi_2\rangle = |\psi\rangle$  as desired.

For sparse states with  $d \ll N$  nonzero vector components, the Grover-Rudolph algorithm has a complexity of  $O(dn^2)$ , which is not optimal. However, Ramacciotti et al. gave an improvement of the algorithm which only prepares the nonzero elements and then permutes them to their correct position, which results in a complexity of  $O(dn)$  [RLR23].

## 2.9. Linear Combination of Unitaries

While unitary quantum gates are sufficient for universal quantum computation, it is often useful to employ non-unitary operations in a quantum algorithm. Assume that we have such a non-unitary operation  $M$  which can be decomposed in terms of a Linear Combination of Unitaries (LCU)

$$M = \sum_{i=1}^n \alpha_i U_i, \quad (2.98)$$

where  $\alpha_i > 0$  without loss of generality, since for any unitary  $U$  and  $c \in \mathbb{C}$  with  $|c| = 1$ ,  $cU$  is still unitary. We can then implement  $M$  as follows:

**Theorem 2.18 (Linear Combination of Unitaries).** *Let  $I$  be a finite index set,  $m, n \in \mathbb{N}$  be the sizes of the first and second register, and let  $M = \sum_{i \in I} \alpha_i U_i$  be a linear*

combination of unitaries  $U_i \in U(2^n)$  with  $\alpha_i > 0$  for all  $i \in I$ . Moreover, let  $V \in U(2^m)$  be any unitary that satisfies

$$V |0^m\rangle = \frac{1}{\sqrt{\alpha}} \sum_{i \in I} \sqrt{\alpha_i} |i\rangle, \quad (2.99)$$

where  $\alpha = \sum_{i \in I} \alpha_i$  and let  $\Pi = |0^m\rangle\langle 0^m| \otimes I_{2^n}$ . Then

$$W = (V^\dagger \otimes I_{2^n})U(V \otimes I_{2^n}) \in U(2^{m+n}) \quad (2.100)$$

satisfies

$$W |0^m\rangle |\psi\rangle = \frac{1}{\alpha} |0^m\rangle M |\psi\rangle + |\Psi^\perp\rangle \quad (2.101)$$

for all states  $|\psi\rangle \in \mathcal{H}_m$ , where  $U = \sum_{i \in I} |i\rangle\langle i| \otimes U_i$  and

$$\Pi |\Psi^\perp\rangle = 0. \quad (2.102)$$

Finally, let  $\mathcal{P}_B$  be an algorithm for preparing a state  $|b\rangle \in \mathcal{H}_n$ . Then there is a quantum algorithm that exactly prepares the state  $\mathcal{N}(M |b\rangle)$  with constant success probability. This algorithm makes  $O(\alpha)$  uses of  $\mathcal{P}_B$ ,  $U$  and  $V$  in expectation and outputs a bit indicating whether it was successful.

*Proof.* We first show Equation 2.101. Note that

$$\langle 0^m | V^\dagger = (V |0^m\rangle)^\dagger = \frac{1}{\sqrt{\alpha}} \sum_{i \in I} \sqrt{\alpha_i} \langle i|. \quad (2.103)$$

We thus obtain

$$W |0^m\rangle |\psi\rangle = (V^\dagger \otimes I_{2^n})U(V \otimes I_{2^n}) |0^m\rangle |\psi\rangle \quad (2.104)$$

$$= (V^\dagger \otimes I)U \frac{1}{\sqrt{\alpha}} \sum_{i \in I} \sqrt{\alpha_i} |i\rangle |\psi\rangle \quad (2.105)$$

$$= (V^\dagger \otimes I) \frac{1}{\sqrt{\alpha}} \sum_{i \in I} \sqrt{\alpha_i} |i\rangle U_i |\psi\rangle \quad (2.106)$$

$$= \Pi V^\dagger \frac{1}{\sqrt{\alpha}} \sum_{i \in I} \sqrt{\alpha_i} |i\rangle U_i |\psi\rangle + (I_{2^m} - \Pi) V^\dagger \frac{1}{\sqrt{\alpha}} \sum_{i \in I} \sqrt{\alpha_i} |i\rangle U_i |\psi\rangle \quad (2.107)$$

$$= \left( |0^m\rangle \frac{1}{\sqrt{\alpha}} \sum_{i \in I} \sqrt{\alpha_i} \langle i| \right) \left( \frac{1}{\sqrt{\alpha}} \sum_{j \in I} \sqrt{\alpha_j} |j\rangle U_j \right) |\psi\rangle + |\Psi^\perp\rangle \quad (2.108)$$

$$= \frac{1}{\alpha} |0^m\rangle M |\psi\rangle + |\Psi^\perp\rangle, \quad (2.109)$$

where

$$|\Psi^\perp\rangle = (I - \Pi)V^\dagger \frac{1}{\sqrt{\alpha}} \sum_{i \in I} \sqrt{\alpha_i} |i\rangle U_i |\psi\rangle \quad (2.110)$$

satisfies  $\Pi |\Psi^\perp\rangle = 0$  since  $\Pi(I - \Pi) = 0$ .

To prove the second statement, consider  $|\psi\rangle = |b\rangle$ . We obtain

$$W |0^m\rangle |b\rangle = \frac{1}{\alpha} |0^m\rangle M |b\rangle + |\Psi^\perp\rangle = \left( \frac{1}{\alpha} \|M |b\rangle\| \right) |0^m\rangle \mathcal{N}(M |b\rangle) + |\Psi^\perp\rangle. \quad (2.111)$$

Hence, applying  $W$  to  $|0^m\rangle |b\rangle$  followed by a measurement on the first  $m$  qubits will yield the desired state  $\mathcal{N}(M |b\rangle)$  with probability  $(\|M |b\rangle\|/\alpha)^2$ . Since  $\mathcal{P}_B |0^n\rangle = |b\rangle$ , we can also use the operator  $(I \otimes \mathcal{P}_B)(I - 2|0^{m+n}\rangle\langle 0^{m+n}|)(I \otimes \mathcal{P}_B^\dagger)$  to reflect across the state  $|0^m\rangle |b\rangle$ , as seen in the section on amplitude amplification. Given an algorithm that creates a desired state from an initial state with probability  $q$ , and the ability to reflect across the initial state, we can use amplitude amplification to create the desired state with probability (say)  $\frac{1}{2}$  that makes  $O(1/\sqrt{q})$  uses of the algorithm and the reflection map in expectation, as seen in Theorem 2.16. Note that amplitude amplification also works when the probability  $q$  is not known in advance, and hence we do not need an estimate of  $\|M |b\rangle\|$ . If we want an algorithm with a worst-case guarantee on cost, then we need to know an upper bound on the probability  $q$ . Using amplitude amplification, we obtain an algorithm that creates the quantum state  $\mathcal{N}(M |b\rangle)$  and makes  $O(\alpha/\|M |b\rangle\|) = O(\alpha)$  uses of  $W$  and  $\mathcal{P}_B$  in expectation.  $\square$

This is a combination of Lemma 2.1, [Kot14] and Lemma 7 from the paper by Childs, Kothari and Somma [CKS17]. The property in Equation 2.102 allows us to check if the LCU operation was successful: We first measure the first register. An outcome of only zeros then implies that the post-measurement state is proportional to  $M |\psi\rangle$ , since  $|\Psi^\perp\rangle$  has no overlap with the first state in the superposition in Equation 2.101. To sample from  $M |\psi\rangle$ , we then only have to discard all results where the measurement of the first  $m$  qubits is not  $0\dots 0$ . Since  $U$  has the form of a general multi-controlled unitary and  $V$  can be implemented by a state preparation algorithm, the implementation of the LCU reduces to well-known results.

## 2.10. Hamiltonian Simulation

Finally, let us briefly consider Hamiltonian simulation. Let  $n \in \mathbb{N}$  be the number of qubits,  $N = 2^n$  and  $H \in \mathbb{C}^{N \times N}$ . Hamiltonian simulation is concerned with the time evolution of a quantum system  $|\psi(0)\rangle \in \mathcal{H}_n$  according to  $H$ :

$$|\psi(t)\rangle = e^{-iHt} |\psi(0)\rangle, \quad (2.112)$$

where  $t > 0$  is the evolution time. In practice, the goal is to find a unitary  $U_t \in U(N)$  such that

$$\|(U_t - e^{-iHt})|\psi(0)\rangle\| < \varepsilon \quad (2.113)$$

for a given  $\varepsilon > 0$ .

As an example, consider the special case where the Hamiltonian can be written as a sum over  $L \in O(\text{poly}(n))$  local interactions:

$$H = \sum_{k=1}^L H_k, \quad (2.114)$$

where each  $H_k$  acts on at most  $c$  qubits. This is the case for many physical systems. One of the simplest methods for Hamiltonian simulation is motivated by the Trotter formula. Let  $A$  and  $B$  be Hermitian operators. Then, for any  $t \in \mathbb{R}$  and  $r \in \mathbb{N}$  large enough, it holds

$$e^{i(A+B)t} = \lim_{n \rightarrow \infty} (e^{iAt/n} e^{iBt/n})^n \approx (e^{iAt/r} e^{iBt/r})^r. \quad (2.115)$$

Applied to the Hamiltonian  $H$ , it can be shown that

$$e^{-iHt} = \left( \prod_{k=1}^L e^{-iH_k t/r} \right)^r + O((nt)^2/r). \quad (2.116)$$

To obtain a target precision of  $\varepsilon$ , we need to use  $O(n^3 t^2 / \varepsilon)$  gates [Lea+23].

Recently, there have been various developments in the field of Hamiltonian simulation. For the quantum linear systems algorithms described in Chapter 4, we need Hamiltonian simulation algorithms for sparse matrices.

**Definition 2.19 ( $d$ -sparse).** *Let  $A \in \mathbb{C}^{N \times N}$ . Then  $A$  is  $d$ -sparse if it has at most  $d$  nonzero entries in any row or column. Instead of  $d$ -sparse, we also say sparse when  $d \ll N$ , but the specific value of  $d$  is not of interest.*

For instance, the HHL algorithm [HHL09] uses an efficient Hamiltonian simulation algorithm for sparse Hamiltonians [Ber+06]. In this case, each  $H_k$  is a 1-sparse Hamiltonian. The CKS algorithm [CKS17] uses an improved Hamiltonian simulation algorithm for sparse Hamiltonians which is based on quantum random walks instead of a product decomposition [BCK15]. We will not consider these algorithms in more detail and only use them as a black box.



# CHAPTER 3.

---

## Geometric Multigrid Methods

---

Multigrid (MG) methods have proved to be efficient solvers for the sparse linear systems which arise in the discretization of elliptic Partial Differential Equations (PDEs). In this chapter, we consider the geometric multigrid method which we motivate and explain in a simplified way to underline the main concepts. We describe the algorithm in the context of the finite element method, but any other discretization method that yields a sparse linear system can be used. A more detailed description can be found in the books by Hackbusch [Hac85] and Braess [Bra07].

### 3.1. Motivation and Problem Formulation

To solve a given PDE, we first discretize it using a method which yields a system of equations such as the finite element method. This system of equations can then be solved with various methods. For small systems, we can use direct methods such as Gaussian elimination or matrix decompositions. As the systems are typically too large for this approach in practice, we could try to use iterative methods such as the Jacobi method, Gauss-Seidel method, conjugate gradient method or Successive Over-Relaxation (SOR). Table 3.1 shows how the number of operations scales asymptotically for various methods. The multigrid method outperforms any other method with the optimal complexity of  $O(n)$ .

An interesting property of certain iterative methods is that they can quickly reduce error components with a high spatial frequency. Simply put, this means that small spikes in the error are quickly smoothed whereas much more iterations are needed to reduce the low frequency components. For instance, the Jacobi method, the Gauss-Seidel method and the conjugate gradient method fulfill this property [Hac85]. Due to this behavior, these methods are also called *smoothers*. The following observation combined with the previous property is the main idea behind multigrid methods: By transferring low frequency components of the error to a coarser grid, their spatial frequency increases. We can thus first reduce high frequency components on a fine grid and then transfer the error to a coarser grid where the same smoother works more efficiently - if the grid is coarse enough we can even directly solve it using the smoother. Finally, we have to transfer the error back to the fine grid to obtain an approximation to the solution. To see more clearly

Scheme	$d = 2$	$d = 3$
Gauss (direct)	$n^3$	$n^3$
Banded-Gauss (direct)	$n^2$	$n^{7/3}$
Jacobi (iterative)	$n^2$	$n^{5/3}$
Gauss-Seidel (iterative)	$n^2$	$n^{5/3}$
Conjugate Gradients	$n^{3/2}$	$n^{4/3}$
SOR with optimal $\omega$	$n^{3/2}$	$n^{4/3}$
Multigrid	$n$	$n$

Table 3.1.: Operations for solving  $Ax = b$  with  $A \in \mathbb{R}^{n \times n}$  being large, sparse and symmetric positive definite [Wic22].

why it makes sense to use different grid sizes to reduce the error, let us consider a simple example [Wic22].

### 3.1.1. Illustrative Example: 1D Poisson

Suppose we want to solve

$$\begin{cases} -u''(x) = f(x), & x \in \Omega = (0, 1), \\ u(0) = u(1) = 0. \end{cases} \quad (3.1)$$

We first discretize the problem using the finite difference method on a mesh  $T_h = \{x_i\}_{i=0, \dots, n+1}$  with mesh size  $h = 1/(n+1)$  to obtain the linear system

$$A = \begin{pmatrix} -1 & 2 & -1 & 0 & 0 \\ & \ddots & & & \\ 0 & -1 & 2 & -1 & 0 \\ & & & \ddots & \\ 0 & 0 & -1 & 2 & -1 \end{pmatrix}, \quad b = h^2 \begin{pmatrix} f(x_1) \\ \vdots \\ \vdots \\ \vdots \\ f(x_n) \end{pmatrix}. \quad (3.2)$$

In Chapter 5, we consider the derivation in more detail. This matrix has eigenvalues

$$\lambda_j = 4 \sin^2\left(\frac{\theta_j}{2}\right), \quad \theta_j = \frac{j\pi}{n+1}, \quad j = 1, \dots, n, \quad (3.3)$$

with the corresponding eigenvectors

$$v_j = (\sin(\theta_j), \sin(2\theta_j), \dots, \sin(n\theta_j)), \quad j = 1, \dots, n. \quad (3.4)$$

Due to the frequency of the sine increasing with  $j$ , we distinguish between the *low frequency eigenvectors* with small  $j$  and the *high frequency eigenvectors* with  $j \approx n$ . It can be shown that the error reduction factor  $\eta$  using Richardson smoothing is given by

$$\eta = 1 - \sin^2\left(\frac{j\pi h}{2}\right). \quad (3.5)$$

The error reduction factor determines how the error  $e_k$  decreases with each iteration step:

$$e_{k+1} = \eta e_k. \quad (3.6)$$

For small eigenvalues ( $j \approx 1$ ), we obtain

$$\eta = 1 - \sin^2\left(\frac{j\pi h}{2}\right) \approx 1 - \left(\frac{\pi h}{2}\right)^2 = 1 - O(h^2) \xrightarrow{h \rightarrow 0} 1, \quad (3.7)$$

since  $\sin(h) \approx h$  for  $h \ll 1$ . Note that in particular,  $\eta$  is dependent on  $h$ : The smaller  $h$ , the worse. For large eigenvalues  $\lambda_j$  with  $j > \frac{n}{2}$ , we have

$$\eta = 1 - \sin^2\left(\frac{j\pi h}{2}\right) \leq \frac{1}{2}. \quad (3.8)$$

Since the bound is independent of  $n$  and therefore  $h$ , this implies that high frequency errors are damped quickly using a Richardson smoother, while low frequency errors with  $i \leq \frac{n}{2}$  are damped slowly. It thus makes sense to work on multiple grids to efficiently reduce both low and high frequency components.

### 3.1.2. Problem Formulation

Consider the following variational formulation: Assume that we are given the task to find  $u \in V$  such that

$$a(u, \varphi) = f(\varphi) \quad \forall \varphi \in V, \quad (3.9)$$

where  $V$  is a function space,  $a$  is a bilinear form and  $f$  is a linear form. In the following, we discretize the problem on  $L + 1 \in \mathbb{N}$  uniformly refined meshes  $T_0, \dots, T_L$  with mesh sizes  $h_l = \alpha^l h_0$ , where  $h_0, \alpha > 0$  and  $0 \leq l \leq L$ . These correspond to the function spaces

$$V_0 \subset V_1 \subset \dots \subset V_L, \quad (3.10)$$

which we assume to be nested. To obtain an approximate solution, we consider the discrete variational formulation: The task is to find  $u_l \in V_l$  such that

$$a(u_l, \varphi_l) = f(\varphi_l) \quad \forall \varphi_l \in V_l. \quad (3.11)$$

The discrete formulation can be rewritten as a linear system of equations

$$A_l x_l = b_l \quad (3.12)$$

with  $A_l \in \mathbb{R}^{n_l \times n_l}$  and  $x_l, b_l \in \mathbb{R}^{n_l}$ .

## 3.2. Algorithmic Formulation

We have already seen the main idea behind the multigrid algorithm: Since smoothers only reduce high-frequency errors quickly, we increase the frequency of the error components by transferring the error to a coarser grid and then apply the smoother there. To explain the multigrid algorithm, it is easier to first consider two grids only.

### 3.2.1. Two-Grid Method

Let  $h > 0$ , and consider two grids  $T_{2h}$  and  $T_h$ . Denote the problem on a grid with size  $h$  by

$$A_h x_h = b_h \tag{3.13}$$

and denote by  $x_h^k$  the  $k$ -th iterate on this grid. To solve the problem as given above on the fine grid  $T_h$ , we can use Algorithm 1.

---

**Algorithm 1** Two-Grid Method.

---

- 1: **procedure** TWOGRIDMETHOD( $x_h^k$ )
  - 2:     1. Pre-Smoothing: Apply  $\nu_1$  smoothing steps using an iterative method  $S_1$ .
  - 3:      $x_h^{k,1} = S_1^{\nu_1} x_h^k$
  - 4:     2. Calculate the residuum  $r_h$  on the fine grid and restrict it to the coarse grid.
  - 5:      $r_h = b_h - A_h x_h^{k,1}$
  - 6:      $r_{2h} = R_h^{2h} r_h$
  - 7:     3. Solve for the correct error on the coarse grid.
  - 8:      $e_{2h} = A_{2h}^{-1} r_{2h}$
  - 9:     4. Apply the coarse grid correction by prolongating the error to the fine grid.
  - 10:     $e_h = P_{2h}^h e_{2h}$
  - 11:     $x_h^{k,2} = x_h^{k,1} + e_h$
  - 12:    5. Post-Smoothing: Apply  $\nu_2$  smoothing steps using an iterative method  $S_2$ .
  - 13:     $x_h^{k+1} = S_2^{\nu_2} x_h^{k,2}$
  - 14:    **return**  $x_h^{k+1}$
- 

The pre-smoothing step is needed to reduce the high-frequent error components, which can be done using few (usually less than five) iterations of the smoother  $S_1$ . Since the smoothing property only applies to the error components, the following operations use a reformulation of the problem which proceeds with the error: Instead of solving  $Ax = b$ , we write

$$x = x_k + e_k \tag{3.14}$$

where  $x_k$  is the approximate solution and  $e_k = x - x_k$  is the corresponding error after  $k$  iterations. The problem is then equivalent to finding  $e_k$  such that

$$Ax = Ax_k + Ae_k = b. \quad (3.15)$$

This can be rewritten as

$$e_k = A^{-1}(b - Ax_k), \quad (3.16)$$

which is done in step 3 after restricting the residuum

$$r_k = b - Ax_k \quad (3.17)$$

to the coarse grid using the restriction operator  $R_h^{2h}$  which we later explain in more detail. Applying the inverse  $A^{-1}$  can be understood as solving the linear system, which can again be done using an iterative solver such as the two-grid method. We can thus use a hierarchy of as many grids as needed and use a two-grid method in step 3 everywhere but on the coarsest grid. Given that the error  $e_{2h}$  on the coarse grid is known, we obtain the error on the fine grid using the prolongation operator  $P_{2h}^h$ . The solution on the coarse grid can then be obtained by adding the error onto the previous iterate:

$$x_{k+1} = x_h + e_h. \quad (3.18)$$

Since we solved for the correct error  $e_h$ , this yields a more accurate approximation. By finally applying some more smoothing steps, the algorithm reduce reduces high-frequency error components that might have been introduced in the prolongation step, for instance when using piecewise linear interpolation. Let us now consider the multigrid method.

### 3.2.2. Multigrid Method

The multigrid method uses a hierarchy of  $L + 1$  grids with level indices  $0 \leq l \leq L$ , where the problem on level  $l$  is denoted by

$$A_l x_l = b_l, \quad (3.19)$$

such that  $A_l \in \mathbb{R}^{n_l \times n_l}$  and  $x_l^k \in \mathbb{R}^{n_l}$  is the  $k$ -th iterate on level  $l$ . It is given in Algorithm 2. The algorithm is very similar to the two-grid method except for the recursion part in line 14. Instead of using a smoother directly in step 3, the algorithm uses the two-grid method recursively up to the coarsest grid where the smoother can finally be used to directly solve the system. Moreover, there is a new parameter  $\mu \in \mathbb{N}_{>0}$  which determines the type of multigrid *cycle*. For  $\mu = 1$  we obtain the so-called *V-cycle*. The V-cycle operates by restricting all the way down to the coarsest grid, solving the smallest problem directly, and then prolongating back up again. Another common choice is  $\mu = 2$ , leading to the *W-cycle*. A diagram showing how the cycles operate can be seen in Figure 3.1. As we will

---

**Algorithm 2** Multigrid Method.

---

```

1: procedure MULTGRIDMETHOD( $l, x_l^k, b_l$ )
2:   1. Pre-Smoothing: Apply  $\nu_1$  smoothing steps using an iterative method  $S_1$ .
3:    $x_l^{k,1} = S_1^{\nu_1} x_l^k$ 
4:   2. Calculate the residuum  $r_h$  on the current grid and restrict it to a coarser grid.
5:    $r_l = b_l - A_l x_l^{k,1}$ 
6:    $r_{l-1} = R_l^{l-1} r_l$ 
7:   3. Solve for the correct error on the coarser grid.
8:   if  $l == 1$  then
9:     Directly solve the system on the coarsest grid
10:     $e_0 = A_0^{-1} r_0$ 
11:   else
12:     $e_{l-1} = 0$ 
13:    for  $i = 0; i < \mu; i++$  do
14:       $e_{l-1} = \text{multigridMethod}(l-1, e_{l-1}, r_{l-1})$ 
15:   4. Apply the coarse grid correction by prolongating the error to the fine grid.
16:    $e_l = P_{l-1}^l e_{l-1}$ 
17:    $x_l^{k,2} = x_l^{k,1} + e_l$ 
18:   5. Post-Smoothing: Apply  $\nu_2$  smoothing steps using an iterative method  $S_2$ .
19:    $x_l^{k+1} = S_2^{\nu_2} x_l^{k,2}$ 
20:   return  $x_l^{k+1}$ 

```

---

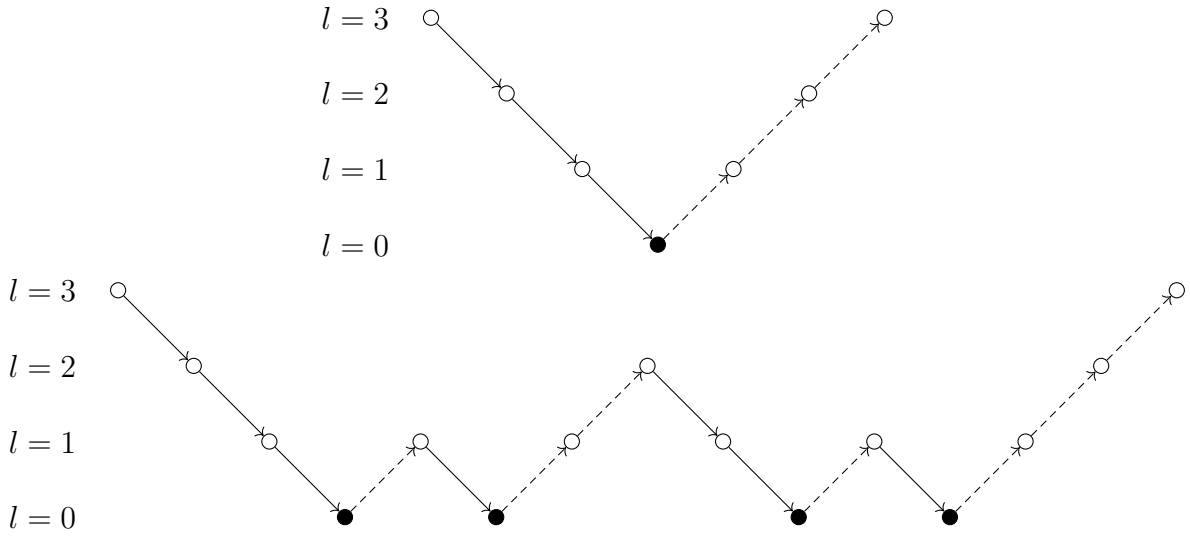


Figure 3.1.: Diagram depicting the V and W cycles corresponding to  $\mu = 1$  and  $\mu = 2$ . The empty and filled circles denote  $\nu$  applications of a smoother and direct solution, respectively. Moreover, the continuous and dashed arrows represent a restriction and prolongation, respectively.

see at the end of this chapter, one multigrid iteration needs  $O(n_L)$  operations. To obtain an overall error  $\varepsilon$  equal to the discretization error  $\varepsilon_{\text{disc}}$ , the algorithm needs  $O(1/h_L)$  iterations, which gives an overall run time of  $O(n_L \log(1/h_L))$  [Hac85]. The *F cycle* or *full multigrid method* uses a nested iteration and is not shown here since it cannot simply be obtained from Algorithm 2 using a different value of  $\mu$ . It has a lower computational cost than Algorithm 2 and achieves an approximation with error  $\varepsilon_{\text{disc}}$  using  $O(n_L)$  operations. Next let us consider the restriction and prolongation operators.

### 3.2.3. Grid Transfer Operations

The grid transfer is one of the key parts of the multigrid method and usually the most difficult to implement. It is needed to approximate a solution on grid  $l$  using a given solution on grid  $l - 1$  (prolongation) and vice versa (restriction). To explain the basic idea of the grid transfer operations, let us consider a simple example with two grids [Wic22], corresponding to function spaces

$$V_l = \{ \varphi_1^l, \dots, \varphi_{n_l}^l \}, \quad (3.20)$$

$$V_{l-1} = \{ \varphi_1^{l-1}, \dots, \varphi_{n_{l-1}}^{l-1} \} \quad (3.21)$$

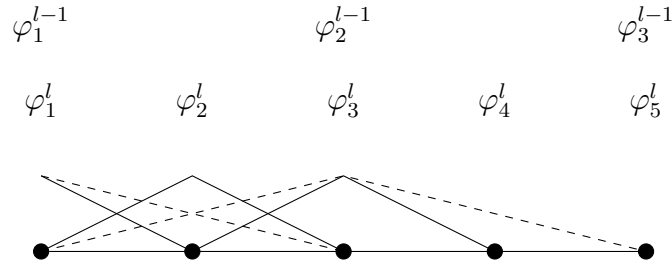


Figure 3.2.: Illustration of some exemplary basis functions on the levels  $l$  and  $l - 1$  which are depicted using continuous and dashed lines, respectively [Wic22].

with  $h_{l-1} = 2h_l$ . By assumption, we have  $n_{l-1} < n_l$  and  $V_{l-1} \subset V_l$ . Hence, there exist coefficients  $r_{ij}$  that restrict the function  $\varphi_i^{l-1}$  on the fine grid to the coarse grid:

$$\varphi_i^{l-1} = \sum_{j=1}^{n_l} r_{ij} \varphi_j^l. \quad (3.22)$$

Figure 3.2 shows an example of basis functions with  $n_l = 5$  grid points on the fine grid and  $n_{l-1} = 3$  grid points on the coarse grid. For instance, we can then decompose the first basis function on the coarse grid as follows:

$$\varphi_1^{l-1} = r_{11} \varphi_1^l + r_{12} \varphi_2^l + \dots + r_{15} \varphi_5^l, \quad (3.23)$$

$$= 1 \varphi_1^l + 0.5 \varphi_2^l. \quad (3.24)$$

Extending this to the remaining basis functions  $\varphi_2^{l-1}$  and  $\varphi_3^{l-1}$ , we obtain the restriction matrix

$$R_l^{l-1} = \begin{pmatrix} 1 & 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 1 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 & 1 \end{pmatrix} \in \mathbb{R}^{3 \times 5}. \quad (3.25)$$

A common choice is to choose the prolongation matrix as its transpose

$$P_{l-1}^l = c R_l^{l-1} \quad (3.26)$$

up to a constant  $c \in \mathbb{R}$ . Another option is to choose  $R_l^{l-1}$  and  $P_{l-1}^l$  independently. For instance, the prolongation can be achieved using piecewise linear interpolation or any other suitable interpolation method.

### 3.3. Cost Complexity

Finally, we show that the computational cost of a multigrid iteration is  $O(n) = O(n_L)$ , where  $n_l$  denotes the number of unknowns on level  $l \in [L]$ . For the proof [Beu20], let  $h_l$

be the discretization parameter on level  $l$ . We have

$$h_l = \alpha h_{l-1} \quad (3.27)$$

with  $\alpha = \frac{1}{2}$ . As a consequence,

$$n_{l-1} \leq \alpha^d n_l, \quad (3.28)$$

where  $d$  is the spatial dimension. The general approach is to bound the number of operations  $W_l$  needed for one iteration of the  $l$ -grid method. This can then be used on the finest grid and is recursively applied to  $W_L$ . One key assumption is the sparsity of the matrices  $A_l$ , due to which each non-recursive step of the multigrid iteration only needs  $O(n_l)$  operations. Moreover, we assume that the number of inner iterations  $\mu$  which determines the cycle type is bounded by

$$\mu < \alpha^{-d} = 2^d. \quad (3.29)$$

We derive the cost from the steps given in Algorithm 2. Let  $c_1 n_l$  and  $c'_1 n_l$  be the cost for one step of pre- and post-smoothing, which is  $O(n_l)$  due to the sparsity of the matrix. Moreover, let  $c_2 n_l$  be the cost to calculate the defect  $r_l = b_l - A_l x_l^{k,1}$  which is given by an  $O(n_l)$  addition and an  $O(n_l)$  sparse matrix vector product. Next, let  $c_3 n_l$  be the cost for a grid transfer operation. Finally, let  $c_4 n_0$  be the cost of the coarse grid solution for  $l = 1$ , and  $\mu W_{l-1}$  for  $l > 1$ . Applying the coarse grid correction is only an addition and requires  $n_l$  operations. Adding up all contributions for  $l > 1$ , we obtain the recursive estimate

$$W_l \leq (\nu_1 c_1 + \nu_2 c'_1 + c_2 + 2c_3 + c_4 + 1)n_l + \mu W_{l-1}. \quad (3.30)$$

Let  $C := \nu_1 c_1 + \nu_2 c'_1 + c_2 + 2c_3 + c_4 + 1 = O(1)$  and recursively apply the estimate to obtain

$$W_L \leq C n_L + \mu W_{L-1} \quad (3.31)$$

$$\leq C n_L + \mu C n_{L-1} + \mu^2 W_{L-2} \quad (3.32)$$

$\vdots$

$$\leq C \sum_{i=0}^{L-1} \mu^i n_{L-i} + \mu^L c_4 n_0 \quad (3.33)$$

$$\leq \max(C, c_4) \sum_{i=0}^L \mu^i n_{L-i} \quad (3.34)$$

$$\leq \max(C, c_4) \sum_{i=0}^L (\mu 2^{-d})^i n_L \quad (3.35)$$

$$\leq \max(C, c_4) n_L \frac{(\mu 2^{-d})^{L+1} - 1}{\mu 2^{-d} - 1} \quad (3.36)$$

$$= O(n_L). \quad (3.37)$$

### 3. Geometric Multigrid Methods

---

In Equation 3.35 we used  $n_{l-1} \leq \alpha^d n_l$  multiple times to obtain  $n_l$  in every summand. Then we used the assumption from Equation 3.29 in the form of  $\mu 2^{-d} < 1$  to simplify the geometric sum. Hence, the only optimal choice for  $d = 1$  is the V-cycle  $\mu = 1$ . For  $d = 2$ , we obtain  $\mu \leq 3$  and  $\mu \leq 7$  for  $d = 3$ .

# CHAPTER 4.

---

## Quantum Linear Systems Algorithms

---

In this chapter, we introduce the Quantum Linear Systems Problem (QLSP), which is the quantum counterpart of the classical linear systems problem. We study two Quantum Linear Systems Algorithms (QLSAs) that can be used to solve the QLSP, including the CKS algorithm which is the theoretical centerpiece of this thesis [CKS17].

In the classical linear systems problem, the task is to find the solution vector to a given system of equations:

**Definition 4.1 (Linear Systems Problem).** *Let  $A \in \mathbb{C}^{N \times N}$  be the coefficient matrix of a linear system and let  $b \in \mathbb{C}^N$ . The linear systems problem then consists of the task to find  $x \in \mathbb{C}^N$  such that*

$$A \cdot x = b. \quad (4.1)$$

Since exact inversion algorithms are relatively slow as seen in Table 3.1, we often relax the problem and instead try to find  $\tilde{x} \in \mathbb{C}^n$  such that

$$\|x - \tilde{x}\|_t < \varepsilon \quad (4.2)$$

for some given tolerance  $\varepsilon > 0$ , where  $\|\cdot\|_t$  is the target norm.

It is important to note that quantum linear systems algorithms are not able to solve the above problem. One of the biggest restrictions is that the state vector cannot efficiently be read out. It is thus necessary to modify the problem to only ask for the quantum state encoding the solution vector. This leads us to the quantum linear systems problem. Assume that  $A$  is  $d$ -sparse and can be accessed efficiently using a sparse access oracle  $\mathcal{P}_A$ . For sparse matrices, only the nonzero elements are stored. The representation used here corresponds to storing a list of nonzero elements for each column. Formally, let  $j \in [N] := \{1, \dots, N\}$ ,  $l \in [d]$  and assume that  $\nu : [N] \times [d] \rightarrow [N]$  is the map that returns entry  $l$  from the list of nonzero elements in column  $j$ . Then  $\mathcal{P}_A$  is the quantum oracle corresponding to the following map:

$$\mathcal{P}_A |j, l\rangle = |j, \nu(j, l)\rangle. \quad (4.3)$$

Furthermore,  $\mathcal{P}_A$  can be used to retrieve the matrix element at some given position  $(j, k) \in [N] \times [N]$ :

$$\mathcal{P}_A |j, k, z\rangle = |j, k, z \oplus A_{jk}\rangle. \quad (4.4)$$

The QLSP can then be defined as follows:

**Definition 4.2 (Quantum Linear Systems Problem).** Let  $A \in \mathbb{C}^{N \times N}$  be the coefficient matrix of a linear system and let  $b \in \mathbb{C}^N$ . Assume that  $A$  has spectral norm  $\|A\| = 1$ , is  $d$ -sparse and Hermitian with known condition number  $\kappa = |\lambda_{\max}|/|\lambda_{\min}|$ . Let  $x = A^{-1}b \in \mathbb{C}^N$ . The corresponding quantum states are given by

$$|b\rangle = \mathcal{N}\left(\sum_i b_i |i\rangle\right), \quad |x\rangle = \mathcal{N}\left(\sum_i x_i |i\rangle\right). \quad (4.5)$$

Furthermore, assume that we have an oracle  $\mathcal{P}_A$  allowing us to access entries of  $A$  as described above and an oracle  $\mathcal{P}_B$  which prepares the state  $|b\rangle$  in time  $O(\text{poly}(\log N))$ . The Quantum Linear Systems Problem then consists of the task to prepare a state  $|\tilde{x}\rangle$  such that

$$\| |\tilde{x}\rangle - |x\rangle \| \leq \varepsilon, \quad (4.6)$$

which succeeds with probability  $\Omega(1)$ .

Solving this problem gives us a quantum state which is an approximation of the exact solution with a constant success probability. This means that we only have to repeat the algorithm  $\Omega(1)$  times to obtain an approximate solution state  $|\tilde{x}\rangle$  with certainty. Since the solution is given as a quantum state which can only be read out using exponentially many repetitions of the algorithm, it is a non-trivial problem to obtain relevant information. The most natural way to circumvent this problem is to only consider quantities that can be efficiently calculated and read out, such as expectations of quantum observables.

In general, the goal is to find an efficient algorithm to solve the QLSP faster than the corresponding classical problem can be solved. This is the reason for the assumptions that  $A$  can be accessed efficiently and that  $|b\rangle$  can be prepared efficiently. The assumption on the sparsity of the matrix is also related to the efficiency and can be relaxed if there exists an efficient Hamiltonian simulation algorithm to simulate  $A$ . For non-Hermitian matrices, consider instead

$$A' = \begin{pmatrix} 0 & A \\ A^\dagger & 0 \end{pmatrix}, \quad (4.7)$$

which is Hermitian. Solving the system  $A'x' = \begin{pmatrix} b \\ 0 \end{pmatrix}$ , we obtain  $x' = \begin{pmatrix} 0 \\ x \end{pmatrix}$ . Note that the above formulation of the QLSP is due to Childs, Kothari and Somma [CKS17]. The conditions on  $A$  differ slightly in other algorithms such as the HHL algorithm which we consider next.

## 4.1. HHL Algorithm

In this section, we give a brief introduction to the HHL algorithm, which was given by Harrow, Hassidim and Lloyd in 2009 [HHL09]. It is the first proposed algorithm to solve

the QLSP. In this section, assume that  $\|A\| \leq 1$  instead of  $\|A\| = 1$  and  $\|A^{-1}\| \leq \kappa$ . The main result by Harrow, Hassidim and Lloyd can then be summarized as follows.

**Theorem 4.3 (HHL Algorithm).** *The QLSP can be solved by a gate-efficient algorithm that makes*

$$Q_{\mathcal{A}}^{\mathcal{P}_A} = O(d\kappa^2 \text{poly}(\log(d\kappa/\varepsilon))/\varepsilon) \quad (4.8)$$

queries to the oracle  $\mathcal{P}_A$  and makes

$$Q_{\mathcal{A}}^{\mathcal{P}_B} = O(d\kappa \text{poly}(\log(d\kappa/\varepsilon))) \quad (4.9)$$

uses of  $\mathcal{P}_B$ .

For consistency, we state this result as given by Childs, Kothari and Somma [CKS17]. The authors replaced the complexity of the Hamiltonian simulation algorithm [Ber+06] that was used in the original HHL algorithm by the best known algorithm [BCK15] at that time. This improves the dependence on the sparsity from  $O(d^2 \text{poly} \log(d))$  to  $O(d \text{poly} \log(d))$ . Note that the input size  $N$  does not appear in the query complexity, but the gate complexity is  $O(Q_{\mathcal{A}}^{\mathcal{P}_A} \text{poly}(\log(Q_{\mathcal{A}}^{\mathcal{P}_A})), \log(N))$ . Hence, the HHL algorithm is efficient only if both  $d$  and  $\kappa$  scale as  $O(\text{poly} \log(N))$ . Compared to the multigrid method which has a complexity of  $O(N \log(1/\varepsilon))$ , the HHL algorithm applied to the QLSP is exponentially faster in the system size but exponentially slower in the tolerance  $\varepsilon$ .

Note that the HHL algorithm can only give an exponential speedup if both  $\kappa$  and  $1/\varepsilon$  are  $O(\text{poly} \log(N))$ . However, for practical applications such as solving PDEs, the condition number of the resulting matrices normally scales as  $O(N)$  or even worse. Since the optimal scaling in  $\kappa$  is  $O(\kappa)$ , this is still an open problem.

Next, let us consider how the HHL algorithm works. The algorithm acts on three registers: The so-called clock register  $C$  is needed for the QPE and its size  $n_C$  determines the precision of the routine. The input register  $I$  is used to store the input  $|b\rangle \in \mathbb{C}^N$  and will also contain the output  $|x\rangle$ . Finally, the ancilla register  $S$  only consists of one qubit.

Let now  $|u_j\rangle$  denote the eigenvectors of  $A$  and let

$$|b\rangle = \sum_{i=1}^N b_i |i\rangle = \sum_{j=1}^N \beta_j |u_j\rangle \quad (4.10)$$

be the eigenbasis decomposition of  $|b\rangle$ , where the coefficients  $\beta_j$  are unknown. We want to obtain the state

$$|x\rangle = A^{-1} |b\rangle = \sum_{j=1}^N \beta_j \lambda_j^{-1} |u_j\rangle. \quad (4.11)$$

Figure 4.1 shows a high-level circuit diagram of the algorithm. The main idea is to use quantum phase estimation to approximately compute the eigenvalues of  $A$ , which would classically correspond to solving the linear system.

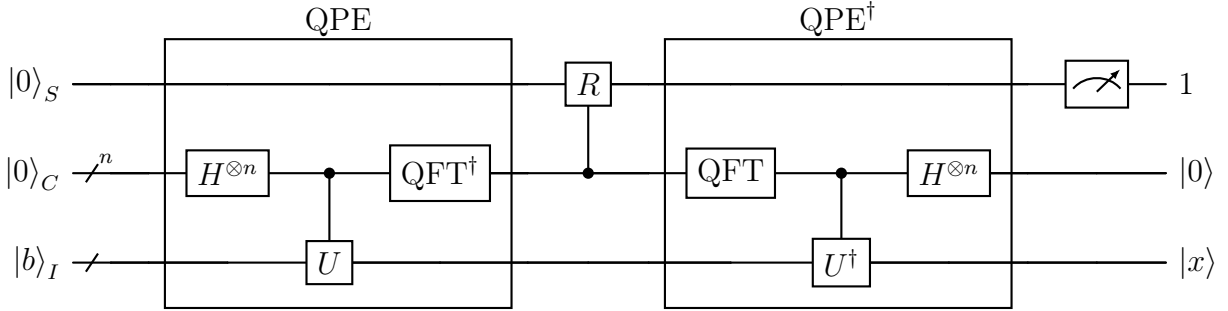


Figure 4.1.: High-level circuit diagram of the HHL algorithm. We first perform QPE with the operator  $U = e^{iAt}$  on the  $C$  and  $I$  registers, followed by a multicontrolled rotation and the uncomputation of the QPE routine. Finally, we measure the ancilla qubit. If the measurement outcome is 1, then the post-measurement state is  $|0\rangle_C |x\rangle_I$ . Here,  $n = n_C$  is the size of the  $C$  register.

The initial state is given by

$$|\psi_0\rangle = (I \otimes I \otimes \mathcal{P}_B) |0\rangle_S |0\rangle_C |0\rangle_I = |0\rangle |0\rangle |b\rangle = |0\rangle |0\rangle \sum_{j=1}^N \beta_j |u_j\rangle, \quad (4.12)$$

where we omit the register labels from now on. We then apply quantum phase estimation to the  $C$  and  $I$  registers. The authors use a slightly modified version of QPE, where the initial superposition is given by

$$|\Psi_0\rangle = \sqrt{\frac{2}{T}} \sum_{\tau=0}^{T-1} \sin\left(\frac{\pi(\tau + \frac{1}{2})}{T}\right) |\tau\rangle, \quad (4.13)$$

where  $T = 2^{n_C}$ . In our derivation, we use the standard QPE as given in Section 2.6 for simplicity. The first step is to prepare the uniform superposition in the  $C$  register:

$$|\psi_1\rangle = (I \otimes H^{\otimes n_C} \otimes I) |\psi_0\rangle = |0\rangle \frac{1}{\sqrt{T}} \sum_{\tau=0}^{T-1} \sum_{j=1}^N \beta_j |\tau\rangle |u_j\rangle. \quad (4.14)$$

The next step of the QPE routine is to apply  $\text{CU}_t^{2^i}$  for  $i \in \{0, \dots, n_C - 1\}$ , where the evolution time is given by  $t = t_0/T$  with  $t_0 = O(\kappa/\varepsilon)$ . In this case, the product of all  $\text{CU}_t^{2^i}$  operators can be written as the conditional Hamiltonian evolution

$$\text{CU}_H = \sum_{\tau=0}^{T-1} |\tau\rangle\langle\tau| \otimes e^{iA\tau t_0/T}. \quad (4.15)$$

Applying  $\text{CU}_H$  to the second and third register yields

$$|\psi_2\rangle = (I \otimes \text{CU}_H) |\psi_1\rangle \quad (4.16)$$

$$= |0\rangle \frac{1}{\sqrt{T}} \sum_{\tau=0}^{T-1} \sum_{j=1}^N \beta_j |\tau\rangle e^{iA\tau t_0/T} |u_j\rangle \quad (4.17)$$

$$= |0\rangle \frac{1}{\sqrt{T}} \sum_{\tau=0}^{T-1} \sum_{j=1}^N \beta_j |\tau\rangle e^{i\lambda_j \tau t_0/T} |u_j\rangle \quad (4.18)$$

$$= |0\rangle \frac{1}{\sqrt{T}} \sum_{j=1}^N \sum_{\tau=0}^{T-1} e^{2\pi i \frac{\lambda_j t_0}{2\pi} \tau/T} |\tau\rangle \beta_j |u_j\rangle, \quad (4.19)$$

where we applied the eigenvalue relation in Equation 4.18. In the last equation, the sum over  $\tau$  is essentially the result of the QFT applied to the basis state  $|\frac{\lambda_j t_0}{2\pi}\rangle$ , assuming that  $\frac{\lambda_j t_0}{2\pi} = k \in \mathbb{N}$ .

The last step of the QPE routine is to apply the inverse QFT to the  $C$  register:

$$|\psi_3\rangle = (I \otimes \text{QFT}^\dagger \otimes I) |\psi_2\rangle = |0\rangle \sum_{j=1}^N \sum_{k=0}^{T-1} \alpha_{k|j} \beta_j |k\rangle |u_j\rangle, \quad (4.20)$$

where  $|\alpha_{k|j}| = \left| \frac{1}{T} \sum_{\tau=0}^{T-1} e^{2\pi i \tau (\lambda_j t_0 / (2\pi) - k) / T} \right|$  is large if and only if

$$\lambda_j \approx \frac{2\pi k}{t_0}. \quad (4.21)$$

Assuming that  $|\alpha_{k|j}|$  is large, we can thus calculate an eigenvalue  $\lambda_j$  from a given value of  $k$ . Defining  $\tilde{\lambda}_k = \frac{2\pi k}{t_0}$ , we relabel the states in the  $C$  register to

$$|\psi_3\rangle = |0\rangle \sum_{j=1}^N \sum_{k=0}^{T-1} \alpha_{k|j} \beta_j |\tilde{\lambda}_k\rangle |u_j\rangle. \quad (4.22)$$

Note that this does not change the value of the ket  $|k\rangle = |\tilde{\lambda}_k\rangle$  but only the label. This is convenient, since the state  $|k\rangle$  essentially encodes  $\tilde{\lambda}_k$  by Equation 4.21. To obtain the state

$$|x\rangle = \mathcal{N}(A^{-1} |b\rangle) = \mathcal{N}\left(\sum_{j=1}^N \beta_j \lambda_j^{-1} |u_j\rangle\right), \quad (4.23)$$

we need to transfer the approximate eigenvalues  $\tilde{\lambda}_k$  to the amplitudes of  $|u_j\rangle$ . Since this operation is not unitary, we have to measure at least one qubit to obtain this result. The

#### 4. Quantum Linear Systems Algorithms

---

first step is to apply the controlled rotation

$$R = \sum_{\lambda \in \{\lambda_{\min}, \dots, \lambda_{\max}\}} R_y \left( 2 \arcsin \left( \frac{c}{\lambda} \right) \right) \otimes |\lambda\rangle\langle\lambda|, \quad (4.24)$$

where  $c$  is a normalization constant and  $|\lambda\rangle = |\bar{\lambda}_k\rangle = |k\rangle$  for some  $k \in \{0, \dots, T-1\}$  with

$$\bar{\lambda}_k := \lambda_{\min} + k \frac{\lambda_{\max} - \lambda_{\min}}{T-1}. \quad (4.25)$$

We thus only need to know the smallest and the largest eigenvalue. The rotation around the  $y$  axis is given by

$$R_y(\theta) = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix} \quad (4.26)$$

and maps

$$R_y(\theta) |0\rangle = \cos(\theta/2) |0\rangle + \sin(\theta/2) |1\rangle. \quad (4.27)$$

Hence, with  $\cos(\arcsin(x)) = \sqrt{1-x^2}$ , we obtain

$$|\psi_4\rangle = (R \otimes I) |0\rangle \sum_{j=1}^N \sum_{k=0}^{T-1} \alpha_{k|j} \beta_j |\tilde{\lambda}_k\rangle |u_j\rangle \quad (4.28)$$

$$= \left( \sqrt{1 - \frac{c^2}{\tilde{\lambda}_k^2}} |0\rangle + \frac{c}{\tilde{\lambda}_k} |1\rangle \right) \sum_{j=1}^N \sum_{k=0}^{T-1} \alpha_{k|j} \beta_j |\tilde{\lambda}_k\rangle |u_j\rangle. \quad (4.29)$$

Since the  $C$  register containing  $|\tilde{\lambda}_k\rangle$  is not needed anymore, the QPE can now be uncomputed. Assuming the phase estimation routine works perfectly, we have  $\alpha_{k|j} = 1$  if  $\tilde{\lambda}_k = \lambda_j$  and 0 otherwise. In this case

$$|\psi_5\rangle = (I \otimes \text{QPE}^\dagger) |\psi_4\rangle = \left( \sqrt{1 - \frac{c^2}{\tilde{\lambda}_k^2}} |0\rangle + \frac{c}{\tilde{\lambda}_k} |1\rangle \right) |0\rangle \sum_{j=1}^N \beta_j |u_j\rangle. \quad (4.30)$$

Finally, we measure the ancilla register. In the case that the measurement outcome is 1, the final state is

$$|\psi_6\rangle = \mathcal{N} \left( |0\rangle \sum_{j=1}^N \beta_j \lambda_j^{-1} |u_j\rangle \right), \quad (4.31)$$

as desired. Since we used a measurement for the last step, there is a certain probability that the inversion routine fails. This is the case if and only if we measure 0. Using this outcome, we can check if the routine was successful and hence employ amplitude amplification to increase the success probability of the algorithm.

## 4.2. CKS Algorithm

In this section, we introduce the QLSA by Childs, Kothari and Somma [CKS17] which is the main subject of our study. This algorithm uses a Linear Combination of Unitaries (LCU) to approximately implement the inverse of  $A$  as a Fourier series. We will also refer to it as the *CKS algorithm* and call the main paper the *CKS paper*. The main result is given in the following theorem.

**Theorem 4.4 (CKS Algorithm - Fourier Approach).** *The QLSP can be solved with  $O(\kappa\sqrt{\log(\kappa/\varepsilon)})$  uses of a Hamiltonian simulation algorithm that approximates  $\exp(-iAt)$  for  $t = O(\kappa \log(\kappa/\varepsilon))$  with precision  $O(\varepsilon/(\kappa\sqrt{\log(\kappa/\varepsilon)}))$ . Using the best known algorithm for Hamiltonian simulation [BCK15], this makes*

$$Q_{\mathcal{A}}^{\mathcal{P}_A} = O(d\kappa^2 \log^{2.5}(\kappa/\varepsilon)) \quad (4.32)$$

queries to  $\mathcal{P}_A$ , makes

$$Q_{\mathcal{A}}^{\mathcal{P}_B} = O(\kappa\sqrt{\log(\kappa/\varepsilon)}) \quad (4.33)$$

uses of  $\mathcal{P}_B$ , and has gate complexity

$$G_{\mathcal{A}} = O(d\kappa^2 \log^{2.5}(\kappa/\varepsilon)(\log(N) + \log^{2.5}(\kappa/\varepsilon))). \quad (4.34)$$

The authors also study a similar method based on Chebyshev series which has a better dependence on  $\varepsilon$  (see Theorem 4, [CKS17]). Moreover, there is an improvement to a gate-efficient algorithm with a query complexity of  $O(d\kappa \text{poly}(\log(d\kappa/\varepsilon)))$  that is nearly linear in  $\kappa$  (see Theorem 5, [CKS17]). In the following, we only consider the Fourier series approach. This approach is more general than the Chebyshev one, since it can also be used when  $A$  is not sparse but can be efficiently simulated.

Compared to the HHL algorithm, the CKS algorithm has an exponentially improved dependence on the tolerance  $\varepsilon$ . This is notable because the dependence on the remaining quantities is the same up to logarithmic factors. Hence, compared to the multigrid method, the CKS algorithm is also exponentially faster in the system size  $N$ . However, due to the scaling in the condition number, this speedup vanishes if we consider matrices where  $\kappa \neq O(\text{poly} \log(N))$ . While both the HHL algorithm and the CKS algorithm are based on Hamiltonian simulation, we will see that they work in fundamentally different ways. Let us first consider the mathematical background of the algorithm.

### 4.2.1. Derivation of the LCU approach

Assume in the following that  $A$  is Hermitian, has spectral norm  $\|A\| = |\lambda_{\max}| = 1$  and is  $d$ -sparse with known condition number  $\kappa$  as required by the QLSP. The main idea is to find a series representation of  $\text{Inv} : x \mapsto \frac{1}{x}$  on the domain

$$D_{\kappa} := [-1, -1/\kappa] \cup [1/\kappa, 1], \quad (4.35)$$

#### 4. Quantum Linear Systems Algorithms

---

where  $\kappa$  is the condition number of  $A$ . Since  $\sigma(A) \subset [-\|A\|, \|A\|]$  and

$$\frac{1}{\kappa} = \frac{|\lambda_{\min}|}{|\lambda_{\max}|} = |\lambda_{\min}|, \quad (4.36)$$

this domain contains the spectrum of  $A$ . With the continuity of  $\text{Inv}$  on  $D_\kappa$ , we can then use the functional calculus (Theorem A.1, Appendix) to approximate the inverse  $A^{-1} = \text{Inv}(A)$  using finitely many terms of the series. This greatly simplifies the convergence analysis since everything reduces to functions over  $\mathbb{C}$ .

We first have to find a suitable series representation of the function  $\text{Inv}$ . Let  $f : \mathbb{R} \rightarrow \mathbb{R}$  be any odd function with

$$\int_0^\infty f(x) dx = 1. \quad (4.37)$$

The first step is to observe that

$$\frac{1}{x} = \frac{1}{x} \lim_{t \rightarrow \infty} \int_{0x}^{tx} f(y) dy = \frac{1}{x} \int_0^\infty f(xy) x dy = \int_0^\infty f(xy) dy \quad (4.38)$$

for all  $x \neq 0$  by substitution. For best performance, choose a function  $f$  which decays quickly and which has a quickly decaying Fourier series: Let  $f(y) = ye^{-y^2/2}$  and  $\psi(x) = \frac{1}{\sqrt{2\pi}}e^{-x^2/2}$ . Using the well-known fact that  $\psi$  is an eigenfunction of the Fourier transform  $\mathcal{F} : L_1(\mathbb{R}) \rightarrow L_\infty(\mathbb{R})$  with eigenvalue 1 and

$$\mathcal{F}(\partial\psi)(\xi) = i\xi\mathcal{F}(\psi)(\xi) \quad (4.39)$$

for all  $\xi \in \mathbb{R}$ , we can show that  $f$  is an eigenfunction of the Fourier transform with eigenvalue  $-i$ :

$$-\frac{1}{\sqrt{2\pi}}\mathcal{F}(f)(\xi) = \mathcal{F}(\partial\psi)(\xi) = i\xi\mathcal{F}(\psi)(\xi) = i\xi\psi(\xi) = \frac{i}{\sqrt{2\pi}}f(\xi) \quad (4.40)$$

for all  $\xi \in \mathbb{R}$  and hence,

$$f(y) = i(\mathcal{F}f)(y) = \frac{i}{\sqrt{2\pi}} \int_{-\infty}^\infty ze^{-z^2/2} e^{-iyz} dy. \quad (4.41)$$

By combining Equations 4.38 and 4.41, we obtain the integral representation

$$\frac{1}{x} = \frac{i}{\sqrt{2\pi}} \int_0^\infty dy \int_{-\infty}^\infty dz ze^{-z^2/2} e^{-ixyz} \quad (4.42)$$

for all  $x \neq 0$ . Note that this representation only contains  $x$  in the exponent of  $e^{-ixyz}$ , which is the key to efficiently implement this series using Hamiltonian simulation. The next steps are the derivation of suitable cutoff values and the discretization of this double integral such that the sum efficiently approximates  $x \mapsto \frac{1}{x}$  to an arbitrary precision  $\varepsilon$ . We use the following notion as a measure of closeness:

**Definition 4.5** ( $\varepsilon$ -close). *Let  $f, g : D \rightarrow \mathbb{C}$  be two functions defined on the same domain  $D$ . We then say that  $f$  and  $g$  are  $\varepsilon$ -close on  $D$  given that*

$$|f(x) - g(x)| \leq \varepsilon \quad \forall x \in D. \quad (4.43)$$

The following lemma captures how we approximate  $1/x$  on  $D_\kappa$ .

**Lemma 4.6.** *Let the function  $h : D_\kappa \rightarrow \mathbb{C}$  be defined as*

$$h(x) = \frac{i}{\sqrt{2\pi}} \sum_{j=0}^{J-1} \Delta_y \sum_{k=-K}^K \Delta_z z_k e^{-z_k^2/2} e^{-ixy_j z_k}, \quad (4.44)$$

where  $y_j = j\Delta_y$ ,  $z_k = k\Delta_z$ , for some fixed  $J = \Theta(\frac{\kappa}{\varepsilon} \log(\kappa/\varepsilon))$ ,  $K = \Theta(\kappa \log(\kappa/\varepsilon))$ ,  $\Delta_y = \Theta(\varepsilon/\sqrt{\log(\kappa/\varepsilon)})$ , and  $\Delta_z = \Theta((\kappa\sqrt{\log(\kappa/\varepsilon)})^{-1})$ . Then  $h(x)$  is  $\varepsilon$ -close to  $1/x$  on the domain  $D_\kappa$ .

This is Lemma 11 in the CKS paper [CKS17]. The proof is rather technical and can be found in the appendix. Next, we consider how the corresponding operator  $h(A)$  can be implemented.

### 4.2.2. LCU Implementation

Recall that we can use Theorem 2.18 to implement a linear combination of unitaries. The following result captures how this theorem can be used to implement an operator such as  $h(A)$  where the summands are functions that map  $A$  to a unitary.

**Theorem 4.7.** *Let  $A$  be a Hermitian operator with eigenvalues in a domain  $D \subset \mathbb{R}$ . Suppose the function  $f : D \rightarrow \mathbb{R}$  satisfies  $|f(x)| \geq 1$  for all  $x \in D$  and is  $\varepsilon$ -close to  $\sum_i \alpha_i T_i$  on  $D$  for some  $\varepsilon \in (0, \frac{1}{2})$ , coefficients  $\alpha_i > 0$ , and functions  $T_i : D \rightarrow \mathbb{C}$  such that  $T_i(A)$  is unitary for all  $i \in I$ . Given an algorithm  $\mathcal{P}_B$  for preparing a state  $|b\rangle$ , there is a quantum algorithm that prepares a quantum state  $4\varepsilon$ -close to  $\mathcal{N}(f(A)|b\rangle)$ , succeeding with constant probability, that makes an expected  $O(\alpha/\|f(A)|b\rangle\|) = O(\alpha)$  uses of the unitaries  $\mathcal{P}_B$ ,  $U$ , and  $V$ , where*

$$U = \sum_i |i\rangle\langle i| \otimes U_i, \quad V|0^m\rangle = \frac{1}{\sqrt{\alpha}} \sum_i \sqrt{\alpha_i} |i\rangle, \quad \text{and} \quad \alpha = \sum_i \alpha_i, \quad (4.45)$$

and outputs a bit indicating whether it was successful. Furthermore, this algorithm can be modified to make  $O(\alpha)$  uses of  $\mathcal{P}_B$ ,  $U$ , and  $V$  in the worst case.

*Proof.* By Theorem 2.18, we can exactly prepare the state  $\mathcal{N}(\sum_{i \in I} \alpha_i T_i(A)|b\rangle)$  with constant success probability and with the stated resource requirements. Since the functions  $f$  and  $\sum_{i \in I} \alpha_i T_i$  are  $\varepsilon$ -close on a domain that includes the spectrum of  $A$ , we obtain

$$\left\| f(A) - \sum_{i \in I} \alpha_i T_i \right\| < \varepsilon \quad (4.46)$$

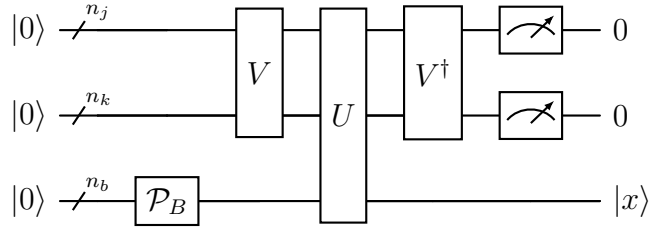


Figure 4.2.: Circuit diagram showing an overview of the CKS algorithm [CKS17]. We begin with preparing the state  $|b\rangle$  in the last register. Next, we apply the LCU operator  $W = V^\dagger UV$ . Finally, we measure the first two registers. If the outcome is  $0 \dots 0$ , the last register is in the state  $|x\rangle$ .

using the isometry property of the functional calculus (Appendix, Theorem A.1) in Equation A.5. We have now shown that  $f(A)$  can be approximated up to a given tolerance. To obtain an approximation of the target state, we apply Lemma A.2: Since  $f(x) \geq 1$  for all  $x \in D$ , the smallest eigenvalue of  $f(A)$  in absolute value is at least 1 and hence, the output state is  $4\varepsilon$ -close to  $\mathcal{N}(f(A)|b\rangle)$ , as claimed. Furthermore,  $f(x) \geq 1$  implies  $\|f(A)|b\rangle\| \geq 1$ , so

$$\frac{\alpha}{\|f(A)|b\rangle\|} = O(\alpha). \quad (4.47)$$

Since  $\alpha$  is known, by running the algorithm (say) 10 times longer than its expected running time, we obtain a bounded-error algorithm that makes  $O(\alpha)$  uses of  $\mathcal{P}_B$ ,  $U$  and  $V$  in the worst case.  $\square$

This is a restriction of Corollary 10 in the CKS paper [CKS17] to the unitary case. Figure 4.2 shows a high-level overview of the CKS algorithm excluding the amplitude amplification part. This is essentially the LCU procedure from Theorem 2.18.

Next, let us consider how to implement

$$h(A) = \frac{i}{\sqrt{2\pi}} \sum_{j=0}^{J-1} \Delta_y \sum_{k=-K}^K \Delta_z z_k e^{-z_k^2/2} e^{-iy_j z_k A}, \quad (4.48)$$

with  $J, K, \Delta_y, \Delta_z, y_j, z_k$  as given in Lemma 4.6. More specifically, we need to find finite index sets  $\mathcal{J}$  and  $\mathcal{K}$ , unitaries  $U_{jk}$  and coefficients  $\alpha_{jk} > 0$  such that  $h(A)$  can be written as an LCU:

$$h(A) = \frac{i}{\sqrt{2\pi}} \sum_{j=0}^{J-1} \Delta_y \sum_{k=-K}^K \Delta_z z_k e^{-z_k^2/2} e^{-iy_j z_k A} = \sum_{j \in \mathcal{J}} \sum_{k \in \mathcal{K}} \alpha_{jk} U_{jk}. \quad (4.49)$$

Here we need the assumption that  $A$  is Hermitian such that  $e^{-iy_j z_k A}$  is unitary. By comparison, we find

$$U_{jk} = i \operatorname{sgn}(k) e^{-iy_j z_k A}, \quad \alpha_{jk} = \frac{1}{\sqrt{\alpha}} \frac{\Delta_y \Delta_z}{\sqrt{2\pi}} |z_k| e^{-z_k^2/2}, \quad (4.50)$$

with  $\mathcal{J} = \{0, \dots, J-1\}$  and  $\mathcal{K} = \{-K, \dots, K\}$ .

The state

$$|\alpha\rangle = V|0\rangle = \sum_{j=0}^{J-1} \sum_{k=-K}^K \sqrt{\alpha_{jk}} |j\rangle |k\rangle \quad (4.51)$$

can be implemented using  $\log_2(J)$  Hadamard gates and  $O(K)$  gates for the  $|k\rangle$  superposition using the state preparation algorithm given by Shende et al. [SBM06]. This is shown in more detail in Section 4.2.3.

Let us now consider  $U$ . To implement  $U_{jk}$  efficiently, observe that

$$U_{jk} = i \operatorname{sgn}(k) e^{-ij\Delta_y k \Delta_z A} = i \operatorname{sgn}(k) (e^{-i\Delta_y \Delta_z A})^{jk}. \quad (4.52)$$

That is,

$$U_{jk} = i \operatorname{sgn}(k) B^{jk} \quad (4.53)$$

with  $B = e^{-i\Delta_y \Delta_z A}$ . The LCU operator  $U$  is given by

$$U = \sum_{j=0}^{J-1} \sum_{k=-K}^K |j\rangle\langle j| \otimes |k\rangle\langle k| \otimes i \operatorname{sgn}(k) B^{jk}. \quad (4.54)$$

To obtain an upper bound on the complexity, we wish to implement  $U_H$  as shown in the proof of the following lemma:

**Lemma 4.8.** *Let*

$$U = \sum_{i=0}^N |i\rangle\langle i| \otimes B^i, \quad (4.55)$$

where  $B$  is a unitary with gate complexity  $G$  and let the gate complexity of  $B^{2^j}$  be  $G_j \leq 2^j G$ . Then the gate complexity of  $U$  is

$$O\left(\sum_{i=1}^{\lfloor \log(N) \rfloor} G_i\right) = O(NG). \quad (4.56)$$

*Proof.* Let  $n := \lfloor \log(N) \rfloor$ , and consider the unitary  $B^{2^j}$  for  $j \in \{0, \dots, n\}$  which has gate complexity  $G_j \leq 2^j G$ . Hence the controlled version of this unitary,  $c\text{-}B^{2^j}$ , controlled by a single qubit, has gate complexity  $O(G_j)$ . The unitary  $U$  can then be implemented

#### 4. Quantum Linear Systems Algorithms

---

by a circuit that performs, for all  $j \in \{0, \dots, n\}$ , the controlled  $B^{2^j}$  operation on the second register controlled by the  $j$ -th qubit of the first register  $|i\rangle$ . If the first register is in state  $|i\rangle$ , the operation performed on the second register is exactly  $B^i$ , due to the binary encoding of the integer  $i$ . The gate complexity of this circuit is

$$O\left(\sum_{j=0}^{\lfloor \log(N) \rfloor} G_j\right) = O\left(\sum_{j=0}^{\lfloor \log(N) \rfloor} 2^j G\right) = O(NG). \quad (4.57)$$

□

This is Lemma 8 in the CKS paper [CKS17]. Note that this result also holds for the total query complexity of  $U$  when  $G$  is the query complexity of  $B$  and  $G_j$  is the query complexity of  $B^{2^j}$ . In our case,  $B^{2^j}$  corresponds to simulating  $A$  with evolution time  $\Delta_y \Delta_z 2^j$ . That is,  $U$  can be implemented with  $\log(JK)$  applications of the Hamiltonian simulation.

Comparing the formulation in Lemma 4.8 with  $U_H$ , there are three major differences: First, the phases  $i \operatorname{sgn}(k)$  need to be implemented separately: Let

$$U_p = \sum_{j=0}^{J-1} \sum_{k=-K}^K |j\rangle\langle j| \otimes |k\rangle\langle k| \otimes i \operatorname{sgn}(k) I, \quad (4.58)$$

$$U_H = \sum_{j=0}^{J-1} \sum_{k=-K}^K |j\rangle\langle j| \otimes |k\rangle\langle k| \otimes (e^{-iA\Delta_y\Delta_z})^{jk}, \quad (4.59)$$

then  $U = U_p U_H$ , where  $U_p$  is a multi-controlled phase gate.

Moreover, the exponent  $jk$  in  $U_H$  is not directly given by the index, but by the product of two indices. To solve this problem, we first have to calculate the product  $jk$  and store it in an ancilla register. We can then use Lemma 4.8 by simply controlling on the individual bits of the product instead. An efficient quantum multiplier based on the QFT is given by Ruiz-Perez et al. [RG17].

Finally, Lemma 4.8 only works for positive powers  $i \geq 0$ . In our case, we have  $jk \leq 0$  for  $k \leq 0$ , such that we also need the inverse Hamiltonian simulation  $B_+$ , where

$$B_{\pm} = e^{\pm iA\Delta_y\Delta_z}. \quad (4.60)$$

We thus have two different operators and cannot simply apply the lemma. One possibility to circumvent this is to use an ancilla qubit that stores if  $k > 0$  and additionally control on this qubit to apply the correct operator  $B_{\pm}$  for each value of  $k$ . In practice, this could be done using a quantum comparator such as the QFT based comparator given by Yuan et al. [Yua+23]. This specific comparator has the advantage that it does not need any ancilla qubits except for the output qubit.

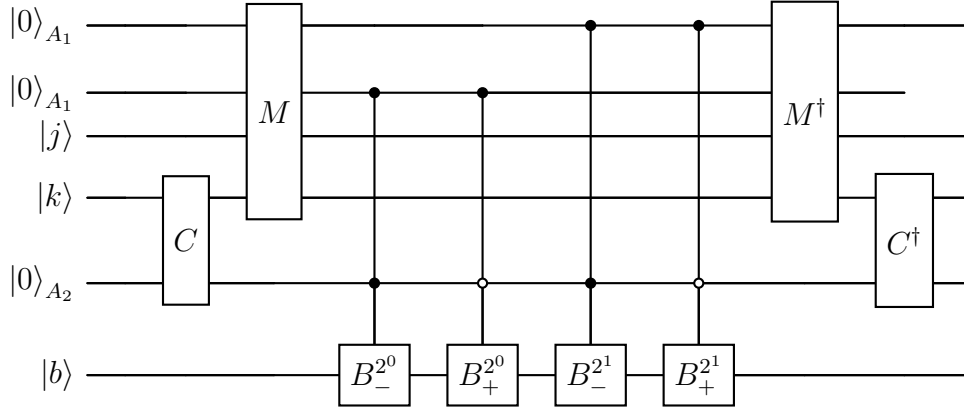
Figure 4.3.: Circuit diagram showing an exemplary implementation of  $U_H$ .

Figure 4.3 exemplifies how  $U_H$  can be implemented for the case that the product  $j|k|$  can be stored in two qubits. The ancilla register  $A_1$  consisting of two qubits stores the product  $j|k|$ , which is calculated using the subroutine  $M$ . The comparator  $C$  stores the result of the comparison  $k > 0$  in the  $A_2$  register for each  $k$ . We then use the binary exponentiation approach from Lemma 4.8: By controlling on the  $i$ -th bit of the product stored in  $A_1$ , and then applying the operators  $B_{\pm}^{2^i}$ , the overall routine implements  $B_{\pm}^{j|k|}$  on the last register. To apply  $B_-$  only for the positive values of  $k$ , we additionally control on the  $A_2$  register. Finally, the  $A_1$  and  $A_2$  registers are uncomputed by applying the inverse operations. Note that these steps are not given in the CKS paper. Instead, the authors assume that  $U$  can be implemented with the complexity given by Lemma 4.8 using the splitting in Equations 4.58 and 4.59.

### 4.2.3. Complexity Analysis

We finish this chapter with the proof of Theorem 4.4, which is the main result stating the complexity for the Fourier series approach. The proof given here is a restructured version of the proof given in the CKS paper, where we fixed some minor mistakes and added more explanations for clarity. In this section, we only present the most important parts of the proof, whereas the necessary technical results are given in the appendix.

The proof is structured as follows. Theorem 4.7 is used to implement the full algorithm  $\mathcal{A}$  and states the query complexities  $Q_{\mathcal{A}}^U = Q_{\mathcal{A}}^V = Q_{\mathcal{A}}^{\mathcal{P}_B} = O(\alpha)$  of the algorithm with respect to  $U$ ,  $V$  and  $\mathcal{P}_B$ , respectively. Since only  $U$  depends on  $\mathcal{P}_A$ , it remains to derive the query complexity  $Q_{\mathcal{A}}^{\mathcal{P}_A}$  to obtain  $Q_{\mathcal{A}}^{\mathcal{P}_A} = Q_{\mathcal{A}}^U Q_{\mathcal{A}}^{\mathcal{P}_A}$ . Finally, the gate complexity  $G_{\mathcal{A}}$  of the algorithm is given by

$$G_{\mathcal{A}} = Q_{\mathcal{A}}^U G_U + Q_{\mathcal{A}}^V G_V = O(\alpha(G_U + G_V)), \quad (4.61)$$

where the gate complexity of  $\mathcal{P}_B$  is not considered in the paper. By assumption and with Theorem 4.7, this amounts to an additive term of  $G_{\mathcal{P}_B} = O(\alpha \text{ poly } \log(N))$ . The gate and query complexities of  $U$  are related by

$$G_U = Q_U^H G_H + G_{U_p}, \quad (4.62)$$

where  $Q_U^H$  is the number of times that  $U$  uses Hamiltonian simulation,  $G_H$  is the gate complexity of the Hamiltonian simulation algorithm for the maximum evolution time occurring in  $U$  and  $G_{U_p}$  is the gate complexity of  $U_p$  as defined in Equation 4.58. The latter complexity is not considered in the paper, presumably because it is dominated by another term. For the proof of Theorem 4.4, we need the following lemma.

**Lemma 4.9.** *The function*

$$g(x) := \frac{i}{\sqrt{2\pi}} \int_0^{y_J} dy \int_{-z_K}^{z_K} dz z e^{-z^2/2} e^{-ixyz} \quad (4.63)$$

is  $\varepsilon$ -close to  $\frac{1}{x}$  on the domain  $D_\kappa$  for  $y_J = \Theta(\kappa \sqrt{\log(\kappa/\varepsilon)})$  and  $z_K = \Theta(\sqrt{\log(\kappa/\varepsilon)})$ .

This is Lemma 12 in the CKS paper [CKS17]. It allows us to understand how the error behaves if we truncate the integral representation of  $\text{Inv}$  in Equation 4.42 to a finite range of integration. The proof can be found in the appendix.

*Proof of Theorem 4.4.* By Theorem 4.7, the query complexity of the algorithm with respect to  $U$  is given by  $Q_{\mathcal{A}}^U = O(\alpha)$ , where  $\alpha$  is the  $L_1$  norm of the LCU coefficients. To express this as a function of the quantities of interest, we need Lemma 4.6 and Lemma 4.9. To apply Theorem 4.7, we discretize the integral from Equation 4.63, approximating  $g(x)$  by the function  $h(x)$  as defined in Lemma 4.6. In particular,  $\Delta_y$  and  $\Delta_z$  in Lemma 4.6 are chosen according to  $y_J$  and  $z_K$  in Lemma 4.9:

$$\Delta_y = \frac{y_J}{J} = \Theta\left(\frac{\varepsilon}{\sqrt{\log(\kappa/\varepsilon)}}\right), \quad (4.64)$$

$$\Delta_z = \frac{z_K}{K} = \Theta\left(\frac{1}{\kappa \sqrt{\log(\kappa/\varepsilon)}}\right). \quad (4.65)$$

By taking  $\Delta_y$  and  $\Delta_z$  sufficiently small,  $h(x)$  can approximate  $g(x)$  arbitrarily closely. Since the query complexity resulting from Theorem 4.7 does not depend on the number of terms in the linear combination, we can make the discretization error arbitrarily small and neglect its contribution.

The  $L_1$  norm of the LCU coefficients is given by

$$\alpha = \frac{1}{\sqrt{2\pi}} \sum_{j=0}^{J-1} \Delta_y \sum_{k=-K}^K \Delta_z |z_k| e^{-z_k^2/2}, \quad (4.66)$$

where, for  $\Delta_z \ll 1$ ,

$$\sum_{k=-K}^K \Delta_z |z_k| e^{-z_k^2/2} \approx \int_{-z_K}^{z_K} |z| e^{-z^2/2} dz \leq 2 \int_0^\infty z e^{-z^2/2} dz = 2 = O(1). \quad (4.67)$$

Here, the error in the approximation is negligible since we can take  $\Delta_z$  arbitrarily small without affecting the query complexity. By Lemma 4.9, we have  $y_J = \Theta(\kappa \sqrt{\log(\kappa/\varepsilon)})$ . Hence, we obtain

$$\sum_{j=0}^{J-1} \Delta_y = \Theta(J\Delta_J) = \Theta(y_J) = \Theta(\kappa \sqrt{\log(\kappa/\varepsilon)}), \quad (4.68)$$

and thus, by Equation 4.66,

$$Q_{\mathcal{A}}^U = O(\alpha) = O(\kappa \sqrt{\log(\kappa/\varepsilon)}). \quad (4.69)$$

Next, consider the query complexity of

$$U = \sum_{j=0}^{J-1} \sum_{k=-K}^K |j, k\rangle \langle j, k| \otimes i \operatorname{sgn}(k) e^{-iA y_j z_k}. \quad (4.70)$$

with respect to the Hamiltonian simulation algorithm. Since the number of applications of  $U$  in the algorithm is  $O(\alpha)$ , the unitary  $U$  must be implemented with a tolerance of  $\varepsilon' = O(\varepsilon/\alpha)$  to obtain an overall tolerance of  $\varepsilon$ . While  $U$  uses Hamiltonian simulation with  $O(JK)$  different evolution times, the query complexity can be reduced to

$$Q_U^H = O(\log(JK)) \quad (4.71)$$

using the binary exponentiation approach shown in the proof of Lemma 4.8. To implement  $U$  with a tolerance of  $\varepsilon'$ , each application of Hamiltonian simulation needs to be implemented with a tolerance of  $\bar{\varepsilon} = O(\varepsilon'/\log(JK))$ .

For the overall query complexity  $Q_{\mathcal{A}}^{\mathcal{P}_A}$ , consider first the query complexity of the Hamiltonian simulation algorithm with respect to  $\mathcal{P}_A$ :

$$Q_H^{\mathcal{P}_A} = O\left(\frac{d\|A\|_{\max} t \log(\|A\|t/\bar{\varepsilon})}{\log(\log(\|A\|t/\bar{\varepsilon}))}\right) = O(dt \log(t/\bar{\varepsilon})), \quad (4.72)$$

since for the QLSP we have  $\|A\|_{\max} \leq \|A\| \leq 1$  and  $\log(x)/\log(\log(x)) = O(\log(x))$ . Lemma 10 [BCK15] does not explicitly handle the case where the evolution time  $t$  is so short that some of these expressions are smaller than 1. In our application, we need to simulate powers of the unitary  $B = e^{-iA\Delta_y\Delta_z}$ , where  $\Delta_y\Delta_z = \Theta(\varepsilon/\kappa)$ , which is indeed a very short evolution time. For such short times, the expression in Equation 4.72 should

have the term first term  $dt$  replaced by  $(dt+1)$ , and all logarithms should be prevented from dropping below 1; i.e., we treat every logarithm as a maximum of its original expression and 1. The largest power involved is  $B^{JK}$ , so the longest evolution time that appears in  $U$  is

$$\tau := \Delta_z J \Delta_z K = O(y_J z_K) = O(\kappa \log(\kappa/\varepsilon)), \quad (4.73)$$

by Lemma 4.9. Using  $\tau$  as an upper bound for  $t$ , the number of queries to simulate  $A$  for any time  $t \leq \tau$  to error  $\bar{\varepsilon}$  is at most

$$Q_H^{\mathcal{P}A}(t) = O((dt + 1) \log(\tau/\bar{\varepsilon})). \quad (4.74)$$

Hence the sum of the query complexities of  $B^{2^j}$  for  $r = 0$  to  $r = \log(JK) = O(\log(\kappa/\varepsilon))$  is given by the sum of  $Q_H^{\mathcal{P}A}(t)$  for the times  $t_j = \Delta_y \Delta_z 2^j$ :

$$Q_U^{\mathcal{P}A} = O\left(\sum_{j=0}^{\log(JK)} (d\Delta_y \Delta_z 2^j + 1) \log(\tau/\bar{\varepsilon})\right) \quad (4.75)$$

$$= O((dy_J z_K + \log(JK)) \log(\tau/\bar{\varepsilon})) \quad (4.76)$$

$$= O(d\kappa \log^2(\kappa/\varepsilon')), \quad (4.77)$$

where we used that  $\log(JK) = O(dy_J z_K)$  and  $\varepsilon' \leq \varepsilon$ . Hence, the overall query complexity is given by

$$Q_{\mathcal{A}}^{\mathcal{P}A} = O(\alpha Q_U^{\mathcal{P}A}) = O(d\kappa^2 \log^{2.5}(\kappa/\varepsilon)). \quad (4.78)$$

Next, consider the gate complexity  $G_U$ . The number of gates needed to simulate  $A$  for time  $t$  with error  $\bar{\varepsilon}$  is

$$G_H = O((d\|A\|_{\max} t) \log(\|A\|t/\bar{\varepsilon})(\log(N) + \log^{2.5}(\|A\|t/\bar{\varepsilon}))), \quad (4.79)$$

As before, we obtain

$$G_H(t) = O((dt + 1)(\log(N) + \log^{2.5}(\tau/\bar{\varepsilon})) \log(\tau/\bar{\varepsilon})) \quad (4.80)$$

$$= O(Q_H^{\mathcal{P}A}(t)(\log(N) + \log^{2.5}(\tau/\bar{\varepsilon}))), \quad (4.81)$$

for any time  $t \leq \tau$ . Since this only differs from  $Q_H^{\mathcal{P}A}(t)$  by a factor, we can reuse the previous calculation to obtain

$$G_U = O(Q_U^{\mathcal{P}A}(\log(N) + \log^{2.5}(\tau/\bar{\varepsilon}))) \quad (4.82)$$

$$= O(d\kappa \log^2(\kappa/\varepsilon')(\log(N) + \log^{2.5}(\tau/\varepsilon'))). \quad (4.83)$$

Finally, consider the gate complexity  $G_V$ . The unitary  $V$  maps the  $m$ -qubit state  $|0^m\rangle$  to

$$\frac{1}{\sqrt{J}} \sum_{j=0}^{J-1} \sum_{k=-K}^K c\sqrt{|z_k|} e^{-z_k^2/2} |j, k\rangle = \left(\frac{1}{\sqrt{J}} \sum_{j=0}^{J-1} |j\rangle\right) \otimes \left(\sum_{k=-K}^K c\sqrt{|z_k|} e^{-z_k^2/2} |k\rangle\right), \quad (4.84)$$

where  $m = O(\log(JK))$  and

$$c = \sqrt{\frac{\Delta_y \Delta_z J}{\sqrt{2\pi\alpha}}}. \quad (4.85)$$

The operation  $V$  can be implemented in two steps, preparing the superpositions over  $|j\rangle$  and  $|k\rangle$  independently since the state in Equation 4.84 is a product state. First, we use  $O(\log(J))$  Hadamard gates to prepare the uniform superposition  $J^{-1/2} \sum_{j=0}^{J-1} |j\rangle$ , assuming for simplicity that  $J$  is a power of 2. Then we use  $O(K)$  gates to prepare the corresponding superposition over  $|k\rangle$  [SBM06]. Using the values of  $J$  and  $K$  from Lemma 4.6, the gate complexity of  $V$  is

$$G_V = O(\log(J) + K) \quad (4.86)$$

$$= O(\log(\kappa \log(\kappa/\varepsilon)/\varepsilon) + \kappa \log(\kappa/\varepsilon)) \quad (4.87)$$

$$= O(\log(\kappa/\varepsilon) + \log(\log(\kappa/\varepsilon)) + \kappa \log(\kappa/\varepsilon)) \quad (4.88)$$

$$= O(\kappa \log(\kappa/\varepsilon)). \quad (4.89)$$

Note that  $G_U = O(d\kappa \log^2(\kappa/\varepsilon')(\log(N) + \log^{2.5}(\tau/\varepsilon')))$  dominates  $G_V$ . Hence, with  $\alpha = O(\kappa \sqrt{\log(\kappa/\varepsilon)})$ , the overall gate complexity is given by

$$G_{\mathcal{A}} = O(\alpha G_U) = O(d\kappa^2 \log^{2.5}(\kappa/\varepsilon)(\log(N) + \log^{2.5}(\tau/\varepsilon))). \quad (4.90)$$

□



# CHAPTER 5.

---

## Numerics

---

In this chapter, we apply the CKS algorithm [CKS17] to approximately solve the two-dimensional Poisson equation which serves as a simple example for an elliptic PDE. The main goal is to study how the algorithm performs in this case with respect to the approximation error and qubit usage. To obtain the results, we use a full state vector simulation of a simplified version of the algorithm. The simulations are run on a node of the IFAM compute server which has 128 cores and almost 1 TB of physical memory. This allows us to run complex simulations with up to 36 qubits in single precision or 35 qubits in double precision without needing to communicate between nodes, using 512 GiB of memory to store the state vector. We begin with the precise problem formulation.

### 5.1. Problem Formulation

Let  $\Omega \subset \mathbb{R}^n$  be an open set,  $f : \Omega \rightarrow \mathbb{R}$  and  $g : \partial\Omega \rightarrow \mathbb{R}$ . We consider the Poisson problem with a Dirichlet boundary condition

$$\begin{cases} \Delta u = f & \text{in } \Omega, \\ u = g & \text{on } \partial\Omega. \end{cases} \quad (5.1)$$

In the following, we consider  $\Omega = (0, 1) \times (0, 1)$ ,  $f \equiv 1$  and  $g \equiv 0$ . To discretize Equation 5.1, we use the finite difference method. The basic idea is to exploit the fact that the difference quotient can be discretized:

$$u'(x) = \lim_{h \rightarrow 0} \frac{u(x+h) - u(x)}{h} \approx \frac{u(x+h) - u(x)}{h} \quad (5.2)$$

for  $h \ll 1$ . The term on the right-hand side is the first-order *forward finite difference*. In our case, we apply the *central finite difference*

$$u'(x) \approx \frac{u(x + \frac{h}{2}) - u(x - \frac{h}{2})}{h} \quad (5.3)$$

twice to obtain

$$u''(x) \approx \frac{\frac{u(x+h)-u(x)}{h} - \frac{u(x)-u(x-h)}{h}}{h} = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2}, \quad (5.4)$$

assuming that  $h \ll 1$ . Coming back to our two-dimensional case, let the mesh  $T_h = \{(x_j, y_j)\}_{j=0, \dots, n-1}$  be given by  $x_j = y_j = jh$  with  $n \in \mathbb{N}$  such that  $h = \frac{1}{n-1} \ll 1$ . We then write

$$u_{i,j} = u(x_i, y_j). \quad (5.5)$$

Applying this to the Poisson problem (5.1), we obtain the discrete formulation

$$(\Delta u)_{i,j} = (\partial_{xx} u)_{i,j} + (\partial_{yy} u)_{i,j} \quad (5.6)$$

$$= \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} \quad (5.7)$$

$$= \frac{1}{h^2} (u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}) \quad (5.8)$$

$$= f_{i,j}. \quad (5.9)$$

Since the boundary values are prescribed, we are only interested in the values on the inner  $m \times m$  grid points, where  $m = n - 2$ . We need to solve  $M = m^2$  equations, one for each grid point. The solution vector is ordered as follows

$$u = (u_{1,1}, u_{2,1}, \dots, u_{m,1}, u_{1,2}, u_{2,2}, \dots, u_{m,2}, \dots, u_{m,m}), \quad (5.10)$$

to obtain a linear system  $Au = b$ , where  $A \in \mathbb{R}^{M \times M}$  and

$$b = -h^2(f_{1,1}, f_{2,1}, \dots, f_{m,1}, f_{1,2}, f_{2,2}, \dots, f_{m,2}, \dots, f_{m,m}) = -h^2(1, \dots, 1) \in \mathbb{R}^M. \quad (5.11)$$

In our case,  $b$  has this simple form since we chose the boundary values to be zero. For the system matrix, let

$$D = \begin{pmatrix} 4 & -1 & 0 & 0 & 0 & \dots & 0 \\ -1 & 4 & -1 & 0 & 0 & \dots & 0 \\ 0 & -1 & 4 & -1 & 0 & \dots & 0 \\ \vdots & & & \ddots & & & \vdots \\ 0 & \dots & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & \dots & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & \dots & 0 & -1 & 4 \end{pmatrix} \in \mathbb{R}^{m \times m}, \quad (5.12)$$

where the structure of the discretized differential operator can be seen. The system matrix is then given by

$$A = \begin{pmatrix} D & -I & 0 & 0 & 0 & \dots & 0 \\ -I & D & -I & 0 & 0 & \dots & 0 \\ 0 & -I & D & -I & 0 & \dots & 0 \\ \vdots & & & \ddots & & & \vdots \\ 0 & \dots & 0 & -I & D & -I & 0 \\ 0 & 0 & \dots & 0 & -I & D & -I \\ 0 & 0 & 0 & \dots & 0 & -I & D \end{pmatrix} \in \mathbb{R}^{M \times M}. \quad (5.13)$$

We should also consider if and how this problem formulation matches the assumptions of the CKS algorithm in Theorem 4.4. It can easily be seen that  $A$  is Hermitian and  $d$ -sparse with  $d = 5$ . Due to the simple structure of  $A$ , there exists an efficient quantum oracle  $\mathcal{P}_A$  that allows us to compute the entries of  $A$  as required. Moreover, we can efficiently prepare  $|b\rangle$ . In particular, this is true in our case where we chose  $f \equiv 1$  but also when the boundary values are given by an efficiently computable distribution. Since  $\|A\| \neq 1$ , we first need to normalize  $A$  to apply the algorithm. Hence, we need to know the absolute value of the largest eigenvalue of  $A$ . Moreover, the algorithm assumes that we know the condition number or at least an upper bound, since the construction of the algorithm explicitly depends on it. Hence, we also need to know the absolute value of the smallest eigenvalue or at least a lower bound on it. Since we are only interested in the performance here, we calculate the condition number beforehand to obtain the best possible precision. However, due to the gate complexity  $O(d\kappa^2 \log^{2.5}(\kappa/\varepsilon)(\log(M) + \log^{2.5}(\kappa/\varepsilon)))$ , we also need that  $\kappa = O(\text{poly}(\log(M)))$  to maintain the speedup in the matrix size  $M$ . As can be seen in Figure A.2, this is not the case for the matrix obtained when discretizing Poisson's problem, where  $\kappa = O(M)$ . This is a problem that has to be considered for any QLSA since the optimal scaling in  $\kappa$  is  $O(\kappa)$ . One way to solve this problem could be quantum preconditioning, as proposed by Clader et al. [CJS13].

## 5.2. Implementation

In this section, we describe how the CKS algorithm can be simulated to solve the Poisson equation. The core of the algorithm is the LCU procedure which is given by the operator  $W = (V^\dagger \otimes I)U(V \otimes I)$  followed by a measurement. Let us first consider how to implement the LCU operators  $V$  and  $U$ . These operators largely depend on the chosen discretization and the cutoff values  $J = \Theta(\kappa \log(\kappa/\varepsilon)/\varepsilon)$  and  $K = \Theta(\kappa \log(\kappa/\varepsilon))$  which were given in Lemma 4.6. Let  $\varepsilon > 0$  be the desired approximation precision and  $\kappa$  the condition number of the matrix to invert. We then choose the cutoff values

$$J = \max\left(\left\lfloor \frac{\kappa}{2\varepsilon} \log(\kappa/\varepsilon) \right\rfloor, 1\right), \quad (5.14)$$

$$K = \max(\lfloor 4\kappa \log(\kappa/\varepsilon) \rfloor, 1), \quad (5.15)$$

according to the given asymptotic behavior, where  $\lfloor \cdot \rfloor : \mathbb{R} \rightarrow \mathbb{Z}$  rounds to the nearest integer. In our case, we had to ensure that  $J, K \geq 1$ . The factors  $\frac{1}{2}$  and 4 were chosen because  $J$  grows too fast in comparison to  $K$  which does not give satisfying results in the studied regime. These values give good results in our case but might still be improved. Then  $\Delta_y$  and  $\Delta_z$  are given as follows:

$$\Delta_y = \sqrt{\kappa/J}, \quad (5.16)$$

$$\Delta_z = \left(\kappa \sqrt{K/\kappa}\right)^{-1}. \quad (5.17)$$

---

**Listing 1** Code snippet that calculates the unitary mapping the first basis vector  $|0\dots 0\rangle$  to the vector  $|v\rangle$  [Gro21].

---

```

1 import numpy as np
2 import scipy.linalg as la
3
4 def create_unitary(v):
5     return np.hstack((v, la.null_space(v.T)))

```

---

Moreover, let  $n_j$  and  $n_k$  be the sizes of the  $|j\rangle$  and  $|k\rangle$  registers, respectively, and let  $\alpha$  be the  $L_1$  norm of the LCU coefficients as given in Equation 4.66. Then  $V$  maps  $|0^{n_j+n_k}\rangle$  to

$$V |0^{n_j+n_k}\rangle = \frac{1}{(2\pi)^{\frac{1}{4}}\sqrt{\alpha}} \sum_{j=0}^{J-1} \sqrt{\Delta_y} \sum_{k=-K}^K \sqrt{\Delta_z|z_k|} e^{-z_k^2/4} |j\rangle |k\rangle \quad (5.18)$$

$$= \left( \frac{1}{2^{n_j/2}} \sum_{j=0}^{J-1} |j\rangle \right) \otimes \left( \sum_{k=-K}^K \frac{\sqrt{\Delta_y \Delta_z} 2^{n_j}}{(2\pi)^{\frac{1}{4}}\sqrt{\alpha}} \sqrt{|z_k|} e^{-z_k^2/4} |k\rangle \right), \quad (5.19)$$

and

$$U = i \sum_{j=0}^{J-1} \sum_{k=-K}^K |j\rangle\langle j| \otimes |k\rangle\langle k| \otimes \text{sgn}(k) e^{-iA y_j z_k} \quad (5.20)$$

$$= \sum_{j=0}^{J-1} \sum_{k=-K}^K |j\rangle\langle j| \otimes |k\rangle\langle k| \otimes i \text{sgn}(k) (e^{-iA \Delta_y \Delta_z})^{j k}. \quad (5.21)$$

In Chapter 4, we have already seen that  $V$  can be implemented by  $n_j$  Hadamard gates and a state preparation algorithm [SBM06]. To simplify the implementation, we replaced the state preparation algorithm by the matrix that fulfills this action. Since  $V_2$  must be unitary it suffices to prescribe the action on the  $|0^{n_k}\rangle$  state to find the full matrix representation. This can be achieved in python using the code snippet given in Listing 1.

The next step is then to implement  $U$  which is a general multi-controlled unitary up to

some phases. Let

$$U_p = \sum_{j=0}^{J-1} \sum_{k=-K}^K |j\rangle\langle j| \otimes |k\rangle\langle k| \otimes i \operatorname{sgn}(k) I, \quad (5.22)$$

$$U_+ = \sum_{j=0}^{J-1} \sum_{k=-K}^{-1} |j\rangle\langle j| \otimes |k\rangle\langle k| \otimes (e^{iA\Delta_y\Delta_z})^{j|k|}, \quad (5.23)$$

$$U_- = \sum_{j=0}^{J-1} \sum_{k=1}^K |j\rangle\langle j| \otimes |k\rangle\langle k| \otimes (e^{-iA\Delta_y\Delta_z})^{j|k|}. \quad (5.24)$$

Then  $U = U_p U_+ U_-$ , where  $U_p$  can be implemented using controlled phase gates and  $U_{\pm}$  are controlled gates that apply the operators  $B_{\pm} = e^{\pm iA\Delta_y\Delta_z}$  to the power of  $j|k|$  for each term in the sum, respectively. Since all the powers are positive integers now, we can apply Lemma 4.8 to implement  $U_+$  and  $U_-$  efficiently using binary exponentiation. To control on the single bits of  $j|k|$ , however, we first need to calculate the product  $j|k|$  and store the result in an ancilla register. We will later specify how this can be done in our simulation.

The last part of the LCU procedure is to measure the  $|j\rangle$  and  $|k\rangle$  registers. Since we simulate the algorithm by storing the full statevector and then applying gate matrices one after another, we can simply read out the result from the state vector without relying on a probabilistic measurement approach. This allows us to omit the amplitude amplification so we have already described the full algorithm now except for the state preparation of  $|b\rangle$ , which can again be done using the snippet in Listing 1. Note that we also tried to use amplitude amplification instead of a direct state readout. However, despite the high probability of success of  $p \geq 0.9$ , this method was too slow to fully read out larger output vectors. There were some issues that occurred during the implementation of the LCU which can broadly be categorised into runtime and memory usage. We will take a closer look at some of these issues which influenced the final implementation to a large extent.

### 5.2.1. Runtime Improvements

One of the most important challenges that is that due to the the simulation runtime increases exponentially in the number of qubits. The first approach was to use Qiskit, a python library that can be used to build algorithms based on the quantum circuit model and simulate them using different simulators such as the statevector simulator [Qis23]. Qiskit works well for small circuits of up to 15 qubits, but is very limited when it comes to larger circuits of 20 qubits or more, since it does not support sparse matrices. Moreover, applying custom gates given as a unitary matrix is slow due to many internal sanity checks which lead to drastically increased runtimes of days or even weeks. Other libraries such as qutip [JNN13] support sparse matrices and efficiently apply smaller gates using the `scipy.sparse.kron` method and the sparse matrix-vector product from the `scipy` python





## 5. Numerics

---

grid size	$M$	$\kappa$	$\varepsilon$	$T$	$t_{V_2}/T$
4	4	3.0	$1.624 \times 10^{-7}$	13 min	0.29
6	16	9.5	$3.623 \times 10^{-6}$	24 min	0.58
10	64	32.2	$1.624 \times 10^{-4}$	63 min	0.79
18	256	116.5	$4.652 \times 10^{-3}$	6 h	0.89
34	1024	440.7	$1.820 \times 10^{-1}$	46 h	0.91

Table 5.1.: Exemplary run times  $T$  for solving the two-dimensional Poisson equation using the best possible value of  $\varepsilon$  with 36 qubits in single precision, where  $M$  is the matrix size,  $\kappa$  the condition number of the matrix and  $t_{V_2}/T$  the percentage of the time needed for applying the two dense  $V_2$  matrices.

around is the memory limit. Since the size of the state vector grows exponentially in the number of qubits, we reach the maximum memory very quickly. To simplify the algorithm, we replaced the Hamiltonian simulation algorithm by the sparse matrix exponential using `scipy.sparse.expm` from the `scipy` python library. This also has the advantage that we do not need the qubits for the output of the sparsity access oracle as required for the QLSP from Definition 4.2.

Another part that cannot be replaced as easily is the multiplication circuit that performs the operation  $|j\rangle |k\rangle |0\rangle \rightarrow |j\rangle |k\rangle |j|k\rangle$  which is needed for the LCU implementation. An algorithm that could be used for this purpose is the QFT multiplier [RG17] which efficiently computes the product of two integers and has almost optimal qubit usage. However, it assumes that the  $|j\rangle$  and  $|k\rangle$  registers have the same sizes  $n_j = n_k$ , so we would need  $4 \max(n_j, n_k)$  qubits for the  $|j\rangle$  and  $|k\rangle$  register and the output register of the multiplier instead of  $n_j + n_k$ .

To see why this is problematic, let us consider an example. Assume that we want to solve the Poisson equation on an  $18 \times 18$  grid. The boundary values are prescribed, so the remaining grid points to solve for are given by the inner  $16 \times 16$  grid. This means that  $A \in \mathbb{R}^{256 \times 256}$  with  $\kappa \approx 116$  and we would need  $n_b = \log_2(256) = 8$  qubits to store  $b$ . Table 5.2 shows the memory requirements for different values of  $\varepsilon$ . The first half of the table shows the number of qubits  $n_q = n_b + n_j + n_k$  without the multiplier. This can be used to calculate the resulting memory usage  $n_{\text{bytes}} = 2^{n_q+4}$ , since we need  $2 \cdot 8 = 2^4$  bytes to store a complex double precision number. In the second half, we first calculate  $n'_j = \max(n_j, n_k)$  to obtain the number of qubits  $n_d = 4n'_j$  needed to store the padded  $|j\rangle$  and  $|k\rangle$  registers and the multiplier output. This gives us the number of qubits  $n_d = 4n'_j - (n_j + n_k)$  that are additionally needed for the multiplier and the total number of qubits  $n'_q = n_b + 4n'_j$  including the multiplier padding and output. It can be seen that while the memory requirements are still somewhat feasible without the multiplier, it is already impossible in the first case when including the multiplier, since the largest supercomputers as of today

$\varepsilon$	0.2	0.1	0.05	0.025
$n_j$	11	12	14	15
$n_k$	13	13	13	13
$n_q$	32	33	35	36
memory usage [GB]	69	137	550	1100
$n'_j$	13	13	14	15
$n_d$	28	27	29	32
$n'_q$	60	60	64	68
memory usage [TB]	$10^7$	$10^7$	$10^8$	$10^9$

Table 5.2.: Exemplary memory usage for solving the Poisson equation on an  $18 \times 18$  grid, illustrating why the quantum multiplier circuit has to be replaced in our implementation. Here,  $n_q$  is the number of qubits of our simulated implementation and  $n'_q$  the number of qubits including the multiplier.

only have physical memory of around  $10^4$  TB.

To solve this problem, the first step is to observe that for the binary exponentiation, we actually do not need the whole product  $j|k|$ , but only one single bit of its binary representation at a time. We can thus calculate a single bit  $(j|k|)_i$  of the binary representation of  $j|k|$ , store it in an ancilla qubit and control on that ancilla to apply the Hamiltonian simulation. Then, we uncompute the ancilla qubit and proceed with the next bit  $(j|k|)_{i+1}$ . This procedure is exemplified in Figure 5.1, assuming again that the product  $j|k|$  can be stored in two bits. The action on the last register is the same as in Figure 4.3. Since there is no simple quantum circuit to calculate only the  $i$ -th bit of  $j|k|$  without calculating  $j|k|$  before, we classically calculate the product and build the corresponding gate. It is essentially a multicontrolled NOT gate.

$$M_i = \sum_{j=0}^{J-1} \sum_{k=-K}^K |j, k\rangle\langle j, k| \otimes X_{i,j,k}, \quad (5.30)$$

where

$$X_{i,j,k} = \begin{cases} X, & \text{if } (j|k|)_i = 1 \\ I, & \text{else} \end{cases}. \quad (5.31)$$

That is, we set the ancilla qubit to the  $i$ -th bit of  $j|k|$ . This allows us to reduce the number of qubits including the multiplier to  $n_q = n_b + n_j + n_k + 1$ . Using the same idea, we can even omit this ancilla qubit by directly controlling the Hamiltonian simulation gate on the  $|j\rangle|k\rangle$  registers. This makes it possible to simulate much larger problems or smaller values of  $\varepsilon$ , essentially trading memory for runtime. Since the matrices  $M_i$  need a

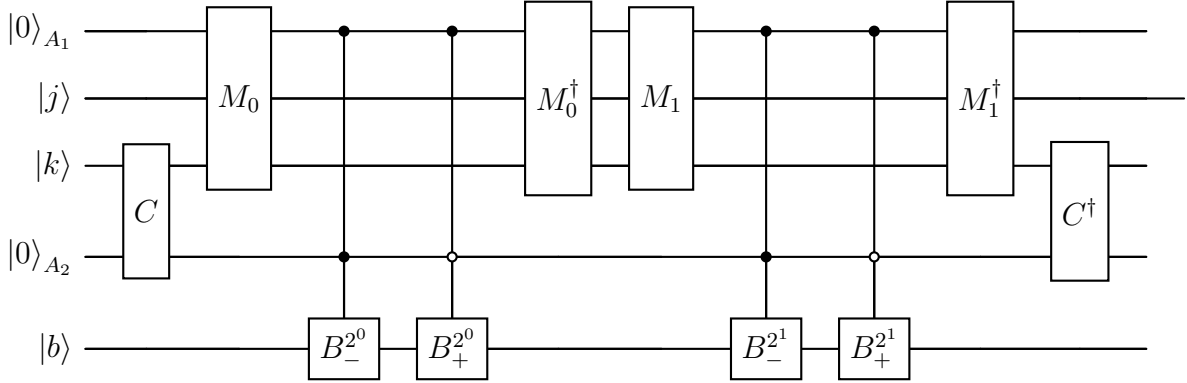


Figure 5.1.: Circuit diagram showing the implementation of  $U_H$  in our simulation. The ancilla register  $A_1$  consisting of one qubit stores one bit  $(j|k|)_i$  of the product, which is set using the subroutine  $M_i$ . By uncomputing the  $A_1$  register for each application of  $B_{\pm}$ , we can reuse the  $A_1$  by increasing the run time. The comparator  $C$  stores the result of  $k > 0$  in the  $A_2$  register for each  $k$ . The binary exponentiation is applied bit by bit, as seen before in Figure 4.3.

significant amount of memory even when stored in a sparse format, we use a matrix-free approach for a faster implementation with a negligible memory overhead.

### 5.2.3. Space Complexity

Let us now consider the qubit usage for our simulated implementation and an implementation on an actual quantum computer. For the simulation, we need at least

$$n_j = \lceil \log_2(J) \rceil, \quad (5.32)$$

$$n_k = \lceil \log_2(2K + 1) \rceil, \quad (5.33)$$

$$n_b = \lceil \log_2(M) \rceil, \quad (5.34)$$

qubits for the  $|j\rangle$  and  $|k\rangle$  superpositions and the output state. Altogether, we need

$$n_q = n_j + n_k + n_b \quad (5.35)$$

$$= \left\lceil \log_2 \left( \max \left( \left\lfloor \frac{\kappa}{2\varepsilon} \log(\kappa/\varepsilon) \right\rfloor, 1 \right) \right) \right\rceil + \lceil \log_2(2 \max(\lfloor 4\kappa \log(\kappa/\varepsilon) \rfloor, 1) + 1) \rceil + \lceil \log_2(M) \rceil \quad (5.36)$$

$$= O \left( \log \left( \frac{\kappa}{\varepsilon} \log(\kappa/\varepsilon) \right) + \log(\kappa \log(\kappa/\varepsilon)) + \log(M) \right) \quad (5.37)$$

$$= O \left( \log \left( \frac{\kappa}{\varepsilon} \log(\kappa/\varepsilon) \right) + \log(M) \right) \quad (5.38)$$

qubits for the simulation.

For the implementation on an actual quantum computer, we would need

$$n'_j = n'_k = 2 \max(n_j, n_k) \quad (5.39)$$

qubits each for the  $|j\rangle$  and  $|k\rangle$  registers and another  $2 \max(n_j, n_k)$  qubits to store the in- and output of the multiplier. To prepare the ancilla qubit which allows us to apply the two Hamiltonian simulation operators, we can use a quantum comparator such as the QFT based comparator proposed by Yuan et al. [Yua+23]. This comparator allows for comparison of a given integer with a quantum register and only needs one ancilla qubit to store the output. For the Hamiltonian simulation algorithm [BCK15] that has been used in the CKS paper [CKS17], we need another

$$n_H = 2 \lceil \log_2(M) \rceil + \lceil \log_2(2k + 1) \rceil + n_{\text{dtype}} + 4 \quad (5.40)$$

qubits, where  $n_{\text{dtype}}$  is the number of bits needed for the data type of the elements of  $A$ . Moreover,  $k = O(\log(\tau/\varepsilon'))$ ,  $\tau = d \|A\|_{\max} t = O(dt)$  with the evolution time  $t = O(\kappa \log(\kappa/\varepsilon))$  and tolerance  $\varepsilon' = O(\varepsilon/(\kappa \sqrt{\log(\kappa/\varepsilon)}))$  of the Hamiltonian simulation algorithm such that

$$n_H = O(\log(M) + \log \log(d\kappa^2 \log^{3/2}(\kappa/\varepsilon)/\varepsilon)). \quad (5.41)$$

For an actual implementation, we would thus need

$$n'_q = n'_j + n'_k + 2n'_j + n_H + n_c \quad (5.42)$$

$$= O\left(\log\left(\frac{\kappa}{\varepsilon} \log(\kappa/\varepsilon)\right) + \log(M) + \log \log(d\kappa^2 \log^{3/2}(\kappa/\varepsilon)/\varepsilon)\right) \quad (5.43)$$

qubits, where  $n_c = 1$  is the number of ancilla qubits needed for the quantum comparator [Yua+23]. Note that the asymptotic space complexities of the simulated implementation and an actual implementation are the same up to the slowly growing  $\log \log$  term. However, by replacing the multiplier, we save a significant amount of qubits which allows us to successfully simulate the algorithm.

Let us now estimate the qubit usage for solving the QLSP corresponding to the Poisson problem on an actual quantum computer. This will mostly depend on the condition number, which grows as  $O(h^{-2})$ . Figure A.2 in the appendix shows this behavior with explicit values for the inner grid size  $m$  up to  $10^3$ . To obtain the values for  $n'_q$ , we use Equation 5.42 and set all constant factors in the asymptotic notation to 1. In Figure 5.2 it can be seen that we need more than 300 qubits even for the smallest systems. However, with twice the amount we can already handle grid sizes of  $m = 10^6$  which corresponds to matrices of size  $M = 10^{12}$ .

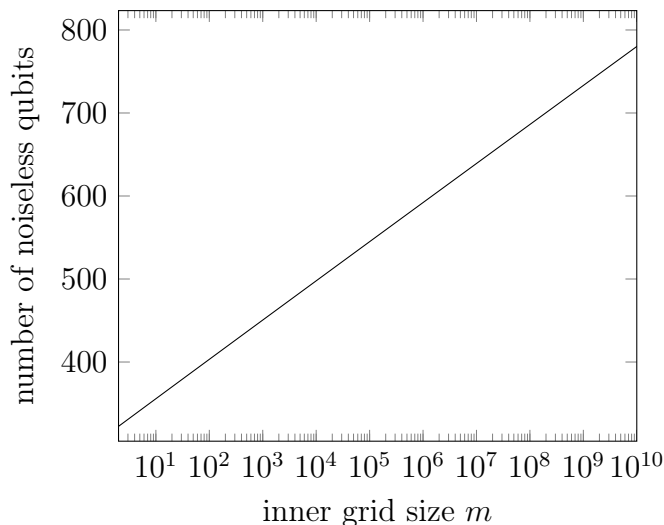


Figure 5.2.: Plot showing the approximate number of noiseless qubits  $n'_q$  needed for solving the Poisson equation on a grid of size  $(m + 2) \times (m + 2)$  with a target precision of  $\varepsilon = 1 \times 10^{-16}$ , assuming that all constants in the asymptotic notation are equal to one.

### 5.3. Solution Convergence

In this section, we study the convergence of our implementation applied to Poisson's problem. For reference, we calculate the normalized numerical solution  $x_n$  using the sparse solver `scipy.sparse.linalg.spsolve` from the `scipy` library. Note that in this section, we always compare the results obtained using the simulation and the `scipy` solver on the same grid size instead of comparing with a reference solution on a finer grid.

Given some simulated solution  $|x_s\rangle$ , the error is given by

$$\delta = \||x_n\rangle - |x_s\rangle\|_2. \quad (5.44)$$

Note that  $\delta \leq 2$  since both  $|x_n\rangle$  and  $|x_s\rangle$  are normalized. The general strategy is to prescribe the number of qubits  $n_{\text{sim}}$  and then calculate the lowest possible  $\varepsilon$  such that we do not waste any resources. This was realized using Brent's method `scipy.optimize.brentq` to find an approximate root of the function

$$f(\varepsilon) = n_{\text{sim}} - n_q + \varepsilon \quad (5.45)$$

in the interval  $[10^{-10}, 0.49]$  with  $n_q$  as given in Equation 5.36. This minimizes  $\varepsilon$  with the correct value of  $n_{\text{sim}}$  since  $|f(\varepsilon)| > 1$  for  $n_{\text{sim}} \neq n_q$ . The algorithm works well in our case and gives reliable results.

As an example, consider the solution on a  $34 \times 34$  grid which is the largest grid that we were able to simulate. Figure 5.3 shows the surface plot and heat map of the solution.

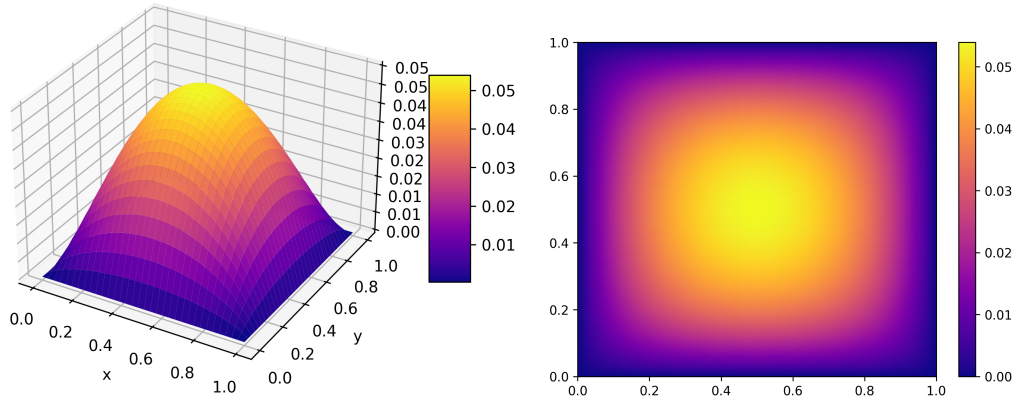


Figure 5.3.: Surface plot and heat map of the solution to the Poisson equation on a  $34 \times 34$  grid using the CKS algorithm with  $\varepsilon = 1.82 \times 10^{-1}$ . We obtain an error of  $\delta \approx 9.32 \times 10^{-4}$ .

grid size	$\varepsilon_{\max}$	$\varepsilon_{\min}$
$6 \times 6$	$2.615 \times 10^{-1}$	$2.869 \times 10^{-5}$
$10 \times 10$	$2.938 \times 10^{-1}$	$1.246 \times 10^{-3}$
$18 \times 18$	$1.834 \times 10^{-1}$	$1.764 \times 10^{-2}$

Table 5.3.: Largest and smallest values for the target precision  $\varepsilon$  in Figure 5.4.

The maximum grid size was mostly limited due to the scaling of the condition number. It should be noted that while the input precision was only  $\varepsilon = 1.82 \times 10^{-1}$ , the error  $\delta = 9.32 \times 10^{-4}$  is approximately two magnitudes smaller. One possible explanation is that the upper bound of  $\delta \leq 4\varepsilon$  on the error given by Childs, Kothari and Somma [CKS17] is not tight so we can in fact get much better values. However, the large deviation is most probably due to the fact that the replacements for Hamiltonian simulation and state preparation introduce almost no error. In any case, the overall error  $\delta$  would be higher when using a proper Hamiltonian simulation and state preparation algorithm.

For the convergence analysis, we vary the input precision  $\varepsilon$  and study how it affects the error  $\delta$ . As described above, we simply achieve this by running one simulation for each possible value of  $n_{\text{sim}}$ , where the minimum value is due to the requirement that  $\varepsilon < \frac{1}{2}$ . Since we need more qubits for a smaller input precision  $\varepsilon$ , we mostly used small grids for the convergence plots to allow for a higher range of  $\varepsilon$ .

Figure 5.4 shows the error  $\delta$  as a function of the number of qubits  $n_q$  for different grid sizes. On the  $34 \times 34$  grid, we were not able to obtain sufficiently many data points. It can be seen that the error quickly decreases, where it is notable that we can achieve values of  $\delta \approx 10^{-8}$  with only 35 qubits on the  $6 \times 6$  grid. In fact, approximately five

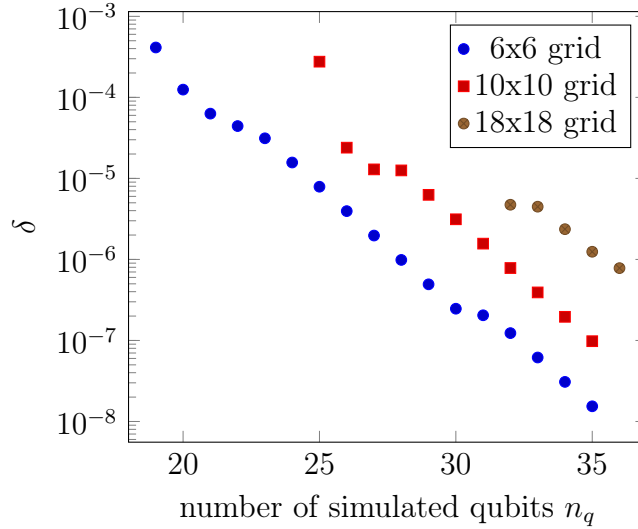


Figure 5.4.: Plot showing the error  $\delta$  with respect to the numerical solution using scipy. The simulations on the  $6 \times 6$  grid and on the  $10 \times 10$  grid have been run in double precision. For the  $18 \times 18$  grid, we used single precision to obtain one additional data point. The input precision  $\varepsilon$  ranges from  $\varepsilon_{\max}$  to  $\varepsilon_{\min}$  with values given in Table 5.3.

additional simulated qubits lead to a reduction of the error by an order of magnitude, so the error is indeed decreasing exponentially in the number of qubits. It is also interesting that the worst result on the  $18 \times 18$  grid is much better than the worst result on the  $6 \times 6$  grid, despite similar values of  $\varepsilon_{\min,6 \times 6} = 2.615 \times 10^{-1}$  and  $\varepsilon_{\min,18 \times 18} = 1.834 \times 10^{-1}$ . This is probably again due to the theoretical error of the Hamiltonian simulation routine. The graph has a distinct appearance for each grid size, where parts decrease with the same exponent, having a single step with a lower decrease in between. It should however be noted that the specific appearance of the graph depends on the implementation.

Next, we want to estimate how the error might behave as a function of the number of noiseless qubits  $n'_q$  on an actual quantum computer. As before, we use Equation 5.42 and set each constant to 1 in the asymptotic notation. We then use the results from Figure 5.4 to find the map  $n'_q \mapsto \delta$ . As can also be seen in Table 5.2, decreasing  $\varepsilon$  does not always lead to an increase in the necessary number of qubits. Hence, we keep only one data point with the lowest error  $\delta$  for each value of  $n'_q$ .

These results can be seen in Figure 5.5 for the  $6 \times 6$  grid and in Figure 5.6 for the  $10 \times 10$  grid. Unfortunately, all data points on the  $18 \times 18$  grid correspond to the same value of  $n'_q$  so we were not able to create a plot for this grid size. We then used a fit function

$$\delta(n'_q) = a \exp(bn'_q) \quad (5.46)$$

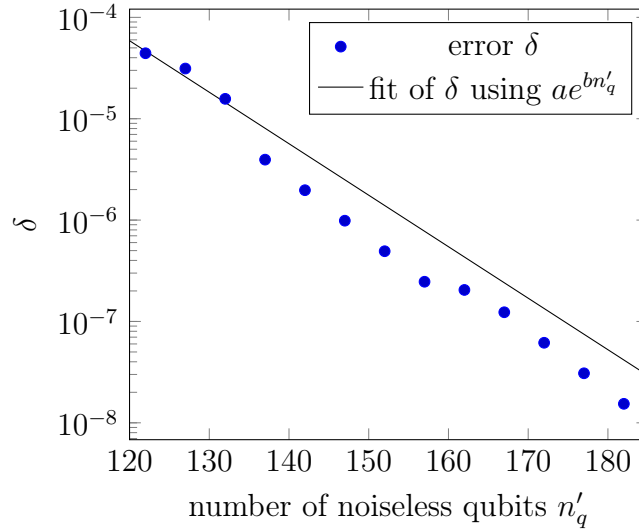


Figure 5.5.: Plot of the expected number of noiseless qubits needed to solve the Poisson equation on a  $6 \times 6$  grid using the CKS algorithm with error  $\delta$ . Using a fit function  $\delta(n'_q) = ae^{bn'_q}$ , we obtain  $a = 73.409$  and  $b = -0.116976$  with  $r^2 = 0.973049$ .

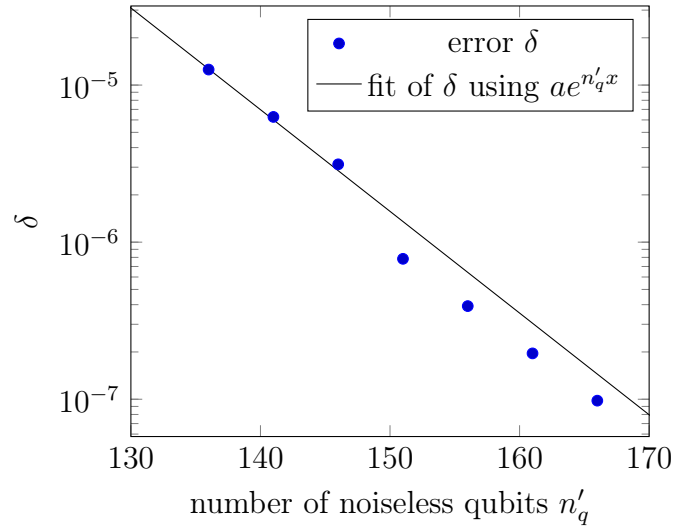


Figure 5.6.: Plot of the expected number of noiseless qubits needed to solve the Poisson equation on a  $10 \times 10$  grid using the CKS algorithm with error  $\delta$ . Using a fit function  $\delta(n'_q) = ae^{bn'_q}$ , we obtain  $a = 8087.641$  and  $b = -0.149080$  with  $r^2 = 0.995518$ .

for  $\delta$  which is exponentially decreasing. The high coefficients of determination  $r_{6 \times 6}^2 = 0.973\,049$  and  $r_{10 \times 10}^2 = 0.995\,518$  confirm the suitability of the model for the given range. Note that leaving out only the first data point on the  $6 \times 6$  grid, the value increases to  $r_{6 \times 6}^2 = 0.991\,290$ . Assuming that the model is correct even for larger values of  $n'_q$ , we can use the fit to extrapolate and calculate how many qubits are needed for a given precision. Due to the different values of  $b$ , this might only be possible for the  $6 \times 6$  and  $10 \times 10$  grids.

However, note that there are always parts of the plot where the values seem to be in a very straight line. For instance, this is the case for the last four data points in both plots. Using only these data points for the fit, we obtain

$$b_{6 \times 6} = -0.138\,631, \quad (5.47)$$

$$b_{10 \times 10} = -0.138\,595, \quad (5.48)$$

which is a lot closer. We obtain similar results for the data points 4 through 8 on the  $6 \times 6$  grid and the first three points on the  $10 \times 10$  grid:

$$b_{6 \times 6} = -0.138\,743, \quad (5.49)$$

$$b_{10 \times 10} = -0.138\,845. \quad (5.50)$$

Thus, using one of these values might be a better estimate for extrapolating to larger grid sizes.

## 5.4. Functional Values

Since the full solution vector cannot be read out without destroying the speedup, we have to restrict the problem to calculate a single scalar value. As an exemplary application that could be efficiently realised on a quantum computer, we consider four goal functionals:

$$F_1[u] = \delta_{0.5,0.5}[u] = u(0.5, 0.5) \quad (5.51)$$

$$F_2[u] = \int_0^1 u(x, 0.5) \, dx \quad (5.52)$$

$$F_3[u] = \int_0^1 \int_{0.5}^1 u(x, y) \, dx dy \quad (5.53)$$

$$F_4[u] = - \int_{\partial\Omega} (\nabla u) \cdot n \, ds, \quad (5.54)$$

where  $\delta_{x,y}$  is the evaluation functional,  $n$  is the outer normal vector and  $\int_{\gamma} \cdot ds$  denotes a curve integral.

The authors of the HHL algorithm [HHL09] recognized this problem and proposed the following approach. Let  $M$  be a linear operator. By mapping  $M$  to a quantum-mechanical

operator, we can then estimate the expectation value

$$\langle x | M | x \rangle = x^T M x \quad (5.55)$$

by performing the quantum measurement corresponding to  $M$ . For instance, by measuring the Hermitian observable

$$M = \sum_{i,j} M_{i,j} |i\rangle \langle j|, \quad (5.56)$$

we can estimate the quadratic form  $\sum_{i,j} M_{i,j} \bar{x}_i x_j$  [HHL09]. If we have enough additional qubits to store  $k$  copies of the output state  $|x\rangle$ , we can estimate degree- $2k$  polynomials in the entries of  $x$  by measuring the  $nk$ -qubit observable

$$M = \sum_{i_1, \dots, i_k, j_1, \dots, j_k} M_{i_1, \dots, i_k, j_1, \dots, j_k} |i_1, \dots, i_k\rangle \langle j_1, \dots, j_k|. \quad (5.57)$$

Clader et al. describe in more detail how to access single values of the solution vector such as the first functional value  $u(0.5, 0.5)$  and the overlap  $|\langle R | x \rangle|^2$  of the solution vector with an arbitrary vector  $|R\rangle$  [CJS13]. Moreover, they propose to use the quantum algorithm for data fitting given by Wiebe, Braun and Lloyd [WBL12] to extract other features of the solution vector, which is not specified further.

In 2016, Montanaro and Pallister studied how the CKS algorithm can be applied to solve PDEs discretized using the finite element method [MP16]. The authors consider the complexity of approximating the value of

$$\langle r, u \rangle = \int_{\Omega} r(x) u(x) dx, \quad (5.58)$$

where  $u$  is the solution to the PDE and  $r : \Omega \rightarrow \mathbb{R}$  with  $\Omega \subset \mathbb{R}^d$  is an arbitrary function: They find that it is not always possible to achieve an exponential speedup over any classical algorithm. It is claimed that the complexity analysis given by Clader et al. is incomplete, such that the exponential speedup stated in their paper [CJS13] might vanish.

As can be seen from the above examples, it is a non-trivial problem to determine the end-to-end speedup for a given problem. In this thesis, we do not consider the specific implementation of the given functionals, which requires further investigation. Instead, we read out the full solution vector and then calculate the functional values using bilinear interpolation and Simpson's rule for integration.

In the previous section, we have always kept the grid size constant and varied the number of qubits. We now want to consider the case where the number of qubits is constant and the grid size varies. To obtain the best precision, all simulations are run with  $n_q = 35$  qubits. Figure 5.7 then shows the convergence of the functional values as a function of the inner grid size  $m$ . While we were not able to obtain values as good as seen in Figure 5.4, the functional values still seem to converge. Hence, in this regime,

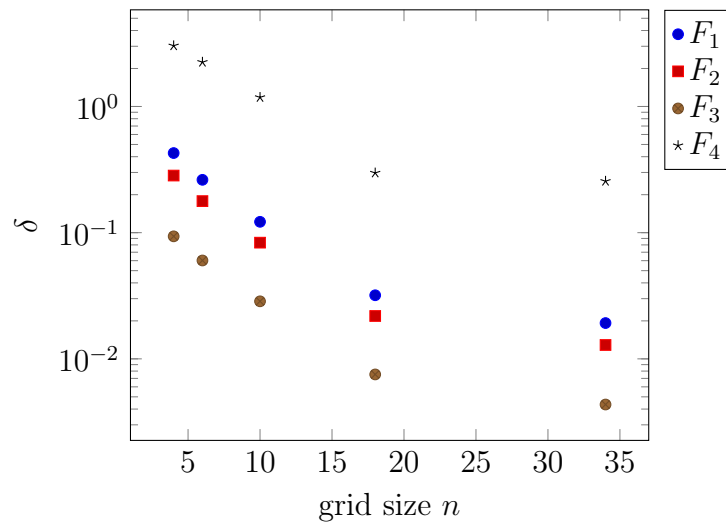


Figure 5.7.: Plot showing the error  $\delta = |F_i[u_s^{(m)}] - F_i[u_n^{(128)}]|$  of the functional values for different grid sizes with respect to the numerical solution  $u_n^{(128)}$  on the  $128 \times 128$  grid using scipy. All data points were simulated with  $n_q = 35$  qubits in double precision.

the discretization error dominates the error of the algorithm. The boundary integral  $F_4$  has the largest error, which is probably due to the additional discretization error of the gradient.

---

# Conclusion

---

In this thesis, we applied the CKS algorithm [CKS17] to solve the QLSP for the two-dimensional Poisson equation. The main objective was to study how the algorithm performs in terms of error and qubit usage. We have also seen how the number of required qubits could be drastically reduced, allowing us to simulate the algorithm on the available hardware.

The study showed that the algorithm performs as desired, which is not immediately clear since the theoretical result only states the asymptotic behavior. We obtained error values as low as  $10^{-8}$  on the  $6 \times 6$  grid with 35 simulated qubits. The space complexity analysis showed that it could be possible to solve the Poisson equation with machine precision on grids of up to  $10^{10} \times 10^{10}$  using less than 800 qubits. Finally, we have seen that even at a fixed number of simulated qubits, the convergence of the goal function is dominated by the discretization error up to the largest grid that can be simulated.

While the results seem promising, it should be noted that we have made various assumptions and simplifications. In practice, one of the biggest limitations would be the scaling of the condition number which grows as  $O(N)$  in our case, where  $N$  is the matrix size. This essentially eliminates any speedup with respect to the system size. We have also simplified the algorithm for our implementation by replacing the state preparation and Hamiltonian simulation subroutines. In addition, we did not specifically consider how to implement the observables corresponding to the goal functionals. Finally, we assumed noiseless qubits and gates, whereas we would have to use an error correction method in practice.

Since a linear scaling in the condition number is optimal, even the best QLSAs cannot asymptotically outperform classical algorithms in most practical applications. Hence, quantum preconditioning needs to be further investigated. For example, it might be possible to use a future quantum multigrid method as a preconditioner, which is similar to a common approach in the classical case. With respect to our implementation, the next steps would be to simulate the full algorithm including the subroutines. We should also consider in detail how to implement the observables in order to perform an end-to-end complexity analysis. It would also be interesting to introduce noise into the simulation to see how stable the algorithm is in a more realistic scenario.



# CHAPTER A.

## Appendix

For the sake of completeness, we give the quantum circuit diagram of the general  $n$ -qubit QFT in Figure A.1.

The remainder of this appendix covers the steps of the proof of Theorem 4.4 that have been left out.

**Theorem A.1 (Functional Calculus for Hermitian Operators).** *Let  $\mathcal{H}$  be a Hilbert space over  $\mathbb{C}$ , and  $A = A^\dagger \in \mathcal{L}(\mathcal{H})$  a Hermitian linear operator and  $\mathcal{P}$  the set of all polynomials. We then define the map  $\psi_A : \mathcal{P} \rightarrow \mathcal{L}(\mathcal{H})$  by*

$$p = \sum_{j=0}^n c_j z^j \mapsto \sum_{j=0}^n c_j A^j =: \psi_A(p). \quad (\text{A.1})$$

*This map defines an algebra homomorphism, which fulfills*

$$\psi_A(\bar{p}) = \sum_{j=0}^n \bar{c}_j A^j = \sum_{j=0}^n (c_j A^j)^\dagger = \psi_A(p)^\dagger. \quad (\text{A.2})$$

*This map can be extended to an algebra homomorphism*

$$\varphi_A : C(\sigma(A), \mathbb{C}) \rightarrow \mathcal{L}(\mathcal{H}), \quad (\text{A.3})$$

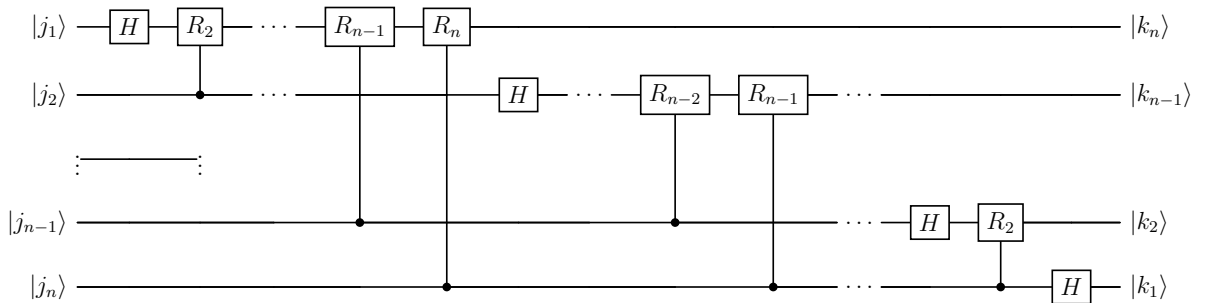


Figure A.1.: Circuit diagram of the  $n$ -qubit quantum Fourier transform. Here we omit the final SWAP gates which reverse the order of the output to obtain  $|k_1 \dots k_n\rangle$ .

which fulfills

$$\varphi_A(z \mapsto 1) = I \tag{A.4}$$

$$\|\varphi_A(f)\|_{\mathcal{L}(\mathcal{H})} = \|f\|_{sup} \tag{A.5}$$

$$\varphi_A(\bar{f}) = \varphi_A(f)^\dagger \tag{A.6}$$

$$f \geq 0 \rightarrow \varphi_A(f) \geq 0. \tag{A.7}$$

We normally write  $f(A)$  instead of  $\varphi_A(f)$ .

The following lemma shows how approximating an operator  $A$  allows us to approximate  $A|\psi\rangle$  for any state  $|\psi\rangle$ .

**Lemma A.2.** *Let  $C$  be a Hermitian operator with  $\|C^{-1}\| \leq 1$ , and let  $D$  be an operator that satisfies  $\|C - D\| \leq \varepsilon < \frac{1}{2}$ . Then the states*

$$|x\rangle = \frac{C|\psi\rangle}{\|C|\psi\rangle\|} \tag{A.8}$$

and

$$|\tilde{x}\rangle = \frac{D|\psi\rangle}{\|D|\psi\rangle\|} \tag{A.9}$$

satisfy  $\| |x\rangle - |\tilde{x}\rangle \| \leq 4\varepsilon$ .

This is Proposition 9 in the CKS paper [CKS17].

Next, we state a technical result which is needed for the next proof. This is Lemma 13 in the CKS paper [CKS17].

**Lemma A.3 (Lemma 13).** *Let  $\omega \in \mathbb{R}$  and  $\Delta_z > 0$ . Then*

$$\sum_{k=-\infty}^{\infty} e^{-(\omega+2\pi k/\Delta_z)^2/2} = \frac{1}{\sqrt{2\pi}} \sum_{k=-\infty}^{\infty} \Delta_z e^{-z_k^2} e^{i\omega z_k}, \tag{A.10}$$

where  $z_k := k\Delta_z$

*Proof.* Equation A.10 is a case of the Poisson summation formula (see, for example, [HN01]), which states that if  $f : \mathbb{R} \rightarrow \mathbb{C}$  is a Schwartz function with Fourier transform  $\hat{f} : \mathbb{R} \rightarrow \mathbb{C}$ , then

$$\sum_{k=-\infty}^{\infty} f(k) = \sqrt{2\pi} \sum_{k=-\infty}^{\infty} \hat{f}(2\pi k). \tag{A.11}$$

Here  $f$  is called a Schwartz function if, for all nonnegative integers  $m, n \in \mathbb{N}_{>0}$ ,

$$\sup_{x \in \mathbb{R}} \left| x^m \frac{d^n}{dx^n} f(x) \right| \tag{A.12}$$

is bounded by some constant that might depend on  $m$  and  $n$ .

In our case,  $f(x) = e^{-(\omega+2\pi x/\Delta z)^2/2}$  has the Fourier transform

$$\hat{f}(y) = \frac{\Delta z}{2\pi} e^{-(y\Delta z)^2/(8\pi^2)} e^{-iy\Delta z\omega/(2\pi)}. \quad (\text{A.13})$$

Then the left-hand side of Equation A.11 coincides with the right-hand side of Equation A.10, and it is easy to check that the right-hand side of Equation A.11 coincides with the right-hand side of Equation A.10. It is well known that a Gaussian is a Schwartz function, so the identity follows.  $\square$

We can now give the proof of Lemma 4.9, which can be found as Lemma 12 in the CKS paper [CKS17].

*Proof of Lemma 4.9.* Applying Fubini's Theorem, we first perform the integral over  $y$ :

$$g(x) = \frac{1}{\sqrt{2\pi x}} \int_{-z_K}^{z_K} dz e^{-z^2/2} (1 - e^{-ixy_J z}). \quad (\text{A.14})$$

Let  $C = [-z_K, z_K]$ . We then estimate the approximation error as follows for any  $x \neq 0$ :

$$\left| g(x) - \frac{1}{x} \right| = \left| g(x) - \frac{1}{\sqrt{2\pi x}} \int_{-\infty}^{\infty} dz e^{-z^2/2} \right| \quad (\text{A.15})$$

$$= \frac{1}{\sqrt{2\pi|x|}} \left| \int_C e^{-z^2/2} (1 - e^{-ixy_J z}) dz - \left( \int_C e^{-z^2/2} dz + \int_{C^c} e^{-z^2/2} dz \right) \right| \quad (\text{A.16})$$

$$= \frac{1}{\sqrt{2\pi|x|}} \left| - \int_C e^{-z^2/2} e^{-ixy_J z} dz - \int_{C^c} e^{-z^2/2} dz \right| \quad (\text{A.17})$$

$$= \frac{1}{\sqrt{2\pi|x|}} \left| - \int_{C \cup C^c} e^{-z^2/2} e^{-ixy_J z} dz - \int_{C^c} e^{-z^2/2} (e^{-ixy_J z} - 1) dz \right| \quad (\text{A.18})$$

$$\leq \frac{1}{|x|} \left( \left| \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} e^{-z^2/2} e^{-ixy_J z} dz \right| + \left| \int_{C^c} e^{-z^2/2} (e^{-ixy_J z} - 1) dz \right| \right) \quad (\text{A.19})$$

$$\leq \frac{1}{|x|} \left( e^{-(xy_J)^2/2} + \int_{[-z_K, -\infty] \cup [z_K, \infty]} e^{-z^2/2} |e^{-ixy_J z} - 1| dz \right) \quad (\text{A.20})$$

$$\leq \frac{1}{|x|} \left( e^{-(xy_J)^2/2} + 2 \int_{[z_K, \infty]} e^{-z^2/2} dz \right) \quad (\text{A.21})$$

$$\leq \frac{1}{|x|} e^{-(xy_J)^2/2} + \frac{2}{|x|} e^{-z_K^2/2} \quad (\text{A.22})$$

$$\leq \kappa e^{-(y_J/\kappa)^2/2} + 2\kappa e^{-z_K^2/2} \quad (\text{A.23})$$

## A. Appendix

---

In Equation A.18, we added a zero by subtracting and then adding  $\int_{C^c} e^{-z^2/2} e^{-ixy_J z} dz$ . Note that in the paper, there is a sign error in this equation. We then applied the triangle inequality and calculated the Fourier transform in the first summand which is known since it is the eigenfunction  $\psi$  from above. Using the estimate  $|e^{-ixy_J z} - 1| \leq 2$ , we obtain an even integrand so we can instead calculate twice the integral over the positive part  $[z_K, \infty]$ . In Equation A.22, we use the estimate

$$\frac{1}{\sqrt{2\pi}} \int_{z_K}^{\infty} e^{-z^2/2} dz \leq \frac{1}{2} e^{-z_K^2/2}, \quad (\text{A.24})$$

which follows from the upper bound

$$\int_x^{\infty} e^{-t^2/2} dt \leq \frac{\sqrt{2}}{2} e^{-x^2} \quad (\text{A.25})$$

given in the Handbook of Mathematical Functions [AS64]. Finally, we used the assumption that  $|x| \geq \frac{1}{\kappa}$ . Hence, there exist  $y_J = \Theta(\kappa \sqrt{\log(\kappa/\varepsilon)})$  and  $z_K = \Theta(\sqrt{\log(\kappa/\varepsilon)})$  such that

$$\left| g(x) - \frac{1}{x} \right| \leq \kappa e^{-(c_1 \kappa \sqrt{\log(\kappa/\varepsilon)/\kappa})^2/2} + 2\kappa e^{-(c_2 \sqrt{\log(\kappa/\varepsilon)})^2/2} \quad (\text{A.26})$$

$$= \kappa e^{-c_1^2 \log(\kappa/\varepsilon)/2} + 2\kappa e^{-c_2^2 \log(\kappa/\varepsilon)/2} \quad (\text{A.27})$$

$$= \kappa \left( \frac{\varepsilon}{\kappa} \right)^{c_1^2/2} + 2\kappa \left( \frac{\varepsilon}{\kappa} \right)^{c_2^2/2} \quad (\text{A.28})$$

$$\leq \varepsilon. \quad (\text{A.29})$$

□

Finally, we show how the integral representation can be discretized using a double sum. This is Lemma 11 in the CKS paper [CKS17].

*Proof of Lemma 4.6.* We have  $\Delta_y = y_J/J = \Theta(\varepsilon/\sqrt{\log(\kappa/\varepsilon)})$  and  $\Delta_z = z_K/K = \Theta(1/\kappa \sqrt{\log(\kappa/\varepsilon)})$ . Performing the geometric sum over  $j$ , we have

$$h(x) = \frac{i\Delta_y}{\sqrt{2\pi}} \sum_{k=-K}^K \Delta_z z_k e^{-z_k^2/2} \frac{1 - e^{-ixy_J z_k}}{1 - e^{-ix\Delta_y z_k}}. \quad (\text{A.30})$$

Using the triangle equality, we have the bound

$$\left| h(x) - \frac{1}{x} \right| \leq |h(x) - A| + |A - B| + |B - C| + \left| C - \frac{1}{x} \right|, \quad (\text{A.31})$$

where

$$A = \frac{1}{\sqrt{2\pi x}} \sum_{k=-K}^K \Delta_z e^{-z_k^2/2} (1 - e^{-ixy_J z_k}) \quad (\text{A.32})$$

$$B = \frac{1}{\sqrt{2\pi x}} \sum_{k=-\infty}^{\infty} \Delta_z e^{-z_k^2/2} (1 - e^{-ixy_J z_k}) \quad (\text{A.33})$$

$$C = \frac{1}{\sqrt{2\pi x}} \sum_{k=-\infty}^{\infty} \Delta_z e^{-z_k^2/2} \quad (\text{A.34})$$

We show that each term on the right-hand side of Equation A.31 is  $O(\varepsilon)$ . Since  $\left| \frac{1}{1-e^{-ix}} - \frac{1}{ix} \right| < 1$  for all  $x \in [-1, 1]$  (as is easily verified by plotting the left-hand side) and since  $x\Delta_y z_K = O(\varepsilon) < 1$  for sufficiently small  $\varepsilon$ , the first term is bounded by

$$|h(x) - A| \leq \sqrt{\frac{2}{\pi}} \Delta_y \sum_{k=-K}^K \Delta_z |z_k| e^{-z_k^2/2} \quad (\text{A.35})$$

$$\leq \sqrt{\frac{2}{\pi}} \Delta_y \sum_{k=-K}^K \Delta_z e^{-z_k^2/4} \quad (\text{A.36})$$

$$\leq 2\sqrt{\frac{2}{\pi}} \Delta_y \int_0^{\infty} e^{-z^2/4} dz \quad (\text{A.37})$$

$$= 2\sqrt{2}\Delta_y \quad (\text{A.38})$$

$$= O(\varepsilon), \quad (\text{A.39})$$

where we used the fact that  $|x| < e^{x^2/4}$  for all  $x \in \mathbb{R}$ . The second term is

$$|A - B| = \left| \frac{1}{\sqrt{2\pi x}} \sum_{|k|>K} \Delta_z e^{-z_k^2/2} (1 - e^{-ixy_J z_k}) \right| \quad (\text{A.40})$$

$$\leq \sqrt{\frac{2}{\pi}} \frac{2}{|x|} \sum_{k=K+1}^{\infty} \Delta_z e^{-z_k^2/2} \quad (\text{A.41})$$

$$\leq \sqrt{\frac{2}{\pi}} \frac{2}{|x|} \int_{z_K}^{\infty} e^{-z^2/2} dz \quad (\text{A.42})$$

$$\leq \frac{2}{|x|} e^{-z_K^2/2} \quad (\text{A.43})$$

$$= O(\varepsilon), \quad (\text{A.44})$$

## A. Appendix

---

where we upper bounded the sum of a decreasing sequence by the corresponding integral and applied the bound

$$\frac{1}{\sqrt{2\pi}} \int_{z_K}^{\infty} e^{-z^2/2} dz \leq \frac{1}{2} e^{-z_K^2/2} \quad (\text{A.45})$$

from Equation A.24 in the proof of Lemma 4.9.

Using Lemma A.3 with  $\omega = xy_J$ , we obtain

$$|B - C| = \frac{1}{\sqrt{2\pi}x} \sum_{k=-\infty}^{\infty} \Delta_z e^{-z_k^2/2} e^{-ixy_J z_k} \quad (\text{A.46})$$

$$= \frac{1}{|x|} \sum_{k=-\infty}^{\infty} e^{-(xy_J + 2\pi k/\Delta_z)^2/2}. \quad (\text{A.47})$$

For  $k \neq 0$ , we have  $|xy_J + 2\pi k/\Delta_z| \geq |k|(2\pi/\Delta_z - y_J)$ . By choosing  $K$  sufficiently large, we can ensure that  $2\pi/\Delta_z \geq y_J$ . Thus we see that

$$|B - C| \leq \frac{1}{|x|} \left( e^{-(xy_J)^2/2} + 2 \sum_{k=1}^{\infty} e^{-k(2\pi/\Delta_z - y_J)^2} \right) \quad (\text{A.48})$$

$$= \frac{1}{|x|} \left( e^{-(xy_J)^2/2} + \frac{2}{e^{(2\pi/\Delta_z - y_J)^2} - 1} \right) \quad (\text{A.49})$$

$$= O(\varepsilon). \quad (\text{A.50})$$

Finally, by Lemma A.3 with  $\omega = 0$ , we have

$$\frac{1}{\sqrt{2\pi}} \sum_{k=-\infty}^{\infty} \Delta_z e^{-z_k^2/2} = \sum_{k=-\infty}^{\infty} e^{-(2\pi k/\Delta_z)^2/2} \quad (\text{A.51})$$

$$\leq 1 + 2 \sum_{k=1}^{\infty} e^{-2\pi^2 k/\Delta_z} \quad (\text{A.52})$$

$$= 1 + \frac{2}{e^{2\pi^2/\Delta_z^2} - 1} \quad (\text{A.53})$$

$$= 1 + O(\varepsilon e^{-\kappa^2 \log(\kappa)}), \quad (\text{A.54})$$

so  $C - \frac{1}{x}$  is also  $O(\varepsilon)$ , completing the proof.  $\square$

This is Lemma 11 in the CKS paper [CKS17].

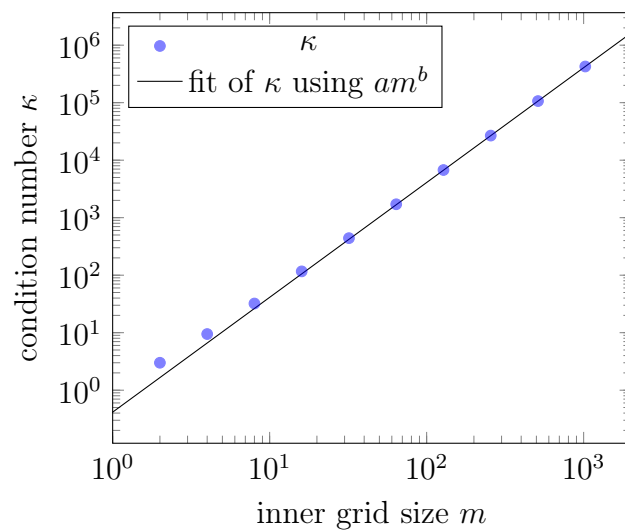


Figure A.2.: Plot of some numerically obtained values of the condition number  $\kappa$  for the Poisson system matrix of size  $M = m^2$ , where  $m = n - 2$  is the size of the inner grid, excluding the boundary. Using a fit function  $\kappa(m) = am^b$ , we obtain  $a = 0.415239$  and  $b = 1.996779$  with a coefficient of determination  $r^2 = 0.99999996$ . This matches the theoretical result  $\kappa = O(h^{-2}) = O(m^2)$  [Wic22].



---

# Bibliography

---

- [Aha+08] D. Aharonov et al. “Adiabatic Quantum Computation Is Equivalent to Standard Quantum Computation”. In: *SIAM Review* 50.4 (Jan. 2008), pp. 755–787. ISSN: 1095-7200. DOI: 10.1137/080734479.
- [AK23] M. Ali and M. Kabel. “Performance Study of Variational Quantum Algorithms for Solving the Poisson Equation on a Quantum Computer”. In: *Physical Review Applied* 20.1 (July 2023), p. 014054. ISSN: 2331-7019. DOI: 10.1103/physrevapplied.20.014054.
- [Amb12] A. Ambainis. “Variable time amplitude amplification and quantum algorithms for linear algebra problems”. en. In: (2012). DOI: 10.4230/LIPICSLIPIICS.STACS.2012.636.
- [AS64] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions*. National Bureau of Standards, Washington, D.C., 1964.
- [BCK15] D. W. Berry, A. M. Childs, and R. 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.
- [Ber+06] D. W. Berry et al. “Efficient Quantum Algorithms for Simulating Sparse Hamiltonians”. In: *Communications in Mathematical Physics* 270.2 (Dec. 2006), pp. 359–371. ISSN: 1432-0916. DOI: 10.1007/s00220-006-0150-x.
- [Ber+14] D. 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.
- [Beu20] S. Beuchler. “Multigrid and domain decomposition”. Lecture Notes. Apr. 2020.
- [BF28] M. Born and V. Fock. “Beweis des Adiabatenatzes”. In: *Zeitschrift für Physik* 51.3–4 (Mar. 1928), pp. 165–180. ISSN: 1434-601X. DOI: 10.1007/bf01343193.
- [Bra+02] G. Brassard et al. *Quantum amplitude amplification and estimation*. 2002. DOI: 10.1090/conm/305/05215.
- [Bra07] D. Braess. *Finite Elemente. Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*. Springer Berlin Heidelberg, 2007. ISBN: 9783540724490. DOI: 10.1007/978-3-540-72450-6.

- [CJS13] B. D. Clader, B. C. Jacobs, and C. R. Sprouse. “Preconditioned Quantum Linear System Algorithm”. In: *Physical Review Letters* 110.25 (June 2013), p. 250504. ISSN: 1079-7114. DOI: 10.1103/physrevlett.110.250504.
- [CKS17] A. M. Childs, R. Kothari, and R. D. Somma. “Quantum Algorithm for Systems of Linear Equations with Exponentially Improved Dependence on Precision”. In: *SIAM Journal on Computing* 46.6 (Jan. 2017), pp. 1920–1950. ISSN: 1095-7111. DOI: 10.1137/16m1087072.
- [CL19] A. M. Childs and J.-P. Liu. “Quantum spectral methods for differential equations”. In: *Communications in Mathematical Physics* 375, 1427–1457 (2020) (Jan. 4, 2019). DOI: 10.1007/s00220-020-03699-z. arXiv: 1901.00961 [quant-ph].
- [CLO20] A. M. Childs, J.-P. Liu, and A. Ostrander. “High-precision quantum algorithms for partial differential equations”. In: *Quantum* 5, 574 (2021) (Feb. 18, 2020). DOI: 10.22331/q-2021-11-10-574. arXiv: 2002.07868 [quant-ph].
- [CLO21] A. M. Childs, J.-P. Liu, and A. Ostrander. “High-precision quantum algorithms for partial differential equations”. In: *Quantum* 5 (Nov. 2021), p. 574. ISSN: 2521-327X. DOI: 10.22331/q-2021-11-10-574.
- [Cos+22] P. C. Costa et al. “Optimal Scaling Quantum Linear-Systems Solver via Discrete Adiabatic Theorem”. In: *PRX Quantum* 3.4 (Oct. 2022), p. 040303. ISSN: 2691-3399. DOI: 10.1103/prxquantum.3.040303.
- [CXS14] H.-T. Chiang, G. Xu, and R. D. Somma. “Improved bounds for eigenpath traversal”. In: *Physical Review A* 89.1 (Jan. 2014), p. 012314. DOI: 10.1103/PhysRevA.89.012314. URL: <https://link.aps.org/doi/10.1103/PhysRevA.89.012314> (visited on 12/25/2023).
- [DM98] L. Dagum and R. Menon. “OpenMP: an industry standard API for shared-memory programming”. In: *Computational Science & Engineering, IEEE* 5.1 (1998), pp. 46–55.
- [DO15] T. Dayar and M. C. Orhan. “On Vector-Kronecker Product Multiplication with Rectangular Factors”. In: *SIAM Journal on Scientific Computing* 37.5 (Jan. 2015), S526–S543. DOI: 10.1137/140980326.
- [Far+00] E. Farhi et al. *Quantum Computation by Adiabatic Evolution*. 2000. DOI: 10.48550/ARXIV.QUANT-PH/0001106.
- [Fey82] R. P. Feynman. “Simulating physics with computers”. In: *International Journal of Theoretical Physics* 21.6–7 (June 1982), pp. 467–488. ISSN: 1572-9575. DOI: 10.1007/bf02650179.

- [Gil+19] A. Gilyén et al. “Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics”. In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. STOC ’19. ACM, June 2019. DOI: 10.1145/3313276.3316366.
- [GLJ18] A. Ghai, C. Lu, and X. Jiao. “A comparison of preconditioned Krylov subspace methods for large-scale nonsymmetric linear systems”. In: *Numerical Linear Algebra with Applications* 26.1 (Oct. 2018). ISSN: 1099-1506. DOI: 10.1002/nla.2215.
- [GR02] L. Grover and T. Rudolph. “Creating superpositions that correspond to efficiently integrable probability distributions”. In: (Aug. 15, 2002). arXiv: quant-ph/0208112 [quant-ph].
- [Gro21] B. Grossmann. *Create a unitary matrix out of a column vector*. June 2021. URL: <https://math.stackexchange.com/a/4160071>.
- [Gue+20] G. G. Guerreschi et al. “Intel Quantum Simulator: a cloud-ready high-performance simulator of quantum circuits”. In: *Quantum Science and Technology* 5.3 (May 2020), p. 034007. DOI: 10.1088/2058-9565/ab8505.
- [Hac78] W. Hackbusch. “A fast iterative method for solving poisson’s equation in a general region”. In: *Lecture Notes in Mathematics*. Springer Berlin Heidelberg, 1978, pp. 51–62. ISBN: 9783540359708. DOI: 10.1007/bfb0067463.
- [Hac85] W. Hackbusch. *Multi-Grid Methods and Applications*. Springer Berlin Heidelberg, 1985. ISBN: 9783662024270. DOI: 10.1007/978-3-662-02427-0.
- [HHL09] A. W. Harrow, A. Hassidim, and S. Lloyd. “Quantum Algorithm for Linear Systems of Equations”. In: *Physical Review Letters* 103.15 (Oct. 2009), p. 150502. ISSN: 1079-7114. DOI: 10.1103/physrevlett.103.150502.
- [HN01] J. K. Hunter and B. Nachtergaele. *Applied Analysis*. World Scientific, Feb. 2001. ISBN: 9789812810670. DOI: 10.1142/4319.
- [JL23] S. Jin and N. Liu. “Analog quantum simulation of partial differential equations”. In: (Aug. 2023). arXiv: 2308.00646 [quant-ph].
- [JLY22] S. Jin, N. Liu, and Y. Yu. “Quantum simulation of partial differential equations via Schrodingerisation”. In: (Dec. 2022). DOI: 10.48550/ARXIV.2212.13969. arXiv: 2212.13969 [quant-ph].
- [JNN13] J. Johansson, P. Nation, and F. Nori. “QuTiP 2: A Python framework for the dynamics of open quantum systems”. In: *Computer Physics Communications* 184.4 (Apr. 2013), pp. 1234–1240. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2012.11.019.
- [JRM17] W. Jakob, J. Rhineland, and D. Moldovan. *pybind11 – Seamless operability between C++11 and Python*. <https://github.com/pybind/pybind11>. 2017.

- [Kit95] A. Y. Kitaev. “Quantum measurements and the Abelian Stabilizer Problem”. In: (Nov. 20, 1995). DOI: 10.48550/ARXIV.QUANT-PH/9511026. arXiv: quant-ph/9511026 [quant-ph].
- [Kot14] Kothari, Robin. “Efficient algorithms in quantum query complexity”. PhD thesis. 2014. URL: <http://hdl.handle.net/10012/8625>.
- [KP20] I. Kerenidis and A. Prakash. “Quantum gradient descent for linear systems and least squares”. In: *Physical Review A* 101.2 (Feb. 2020), p. 022316. ISSN: 2469-9934. DOI: 10.1103/physreva.101.022316.
- [LC17] G. H. Low and I. L. Chuang. “Hamiltonian Simulation by Uniform Spectral Amplification”. In: (July 2017). DOI: 10.48550/ARXIV.1707.05391. arXiv: 1707.05391 [quant-ph].
- [LC19] G. H. Low and I. L. Chuang. “Hamiltonian Simulation by Qubitization”. In: *Quantum* 3 (July 2019), p. 163. ISSN: 2521-327X. DOI: 10.22331/q-2019-07-12-163.
- [Lea+23] C. Leadbeater et al. “Non-unitary Trotter circuits for imaginary time evolution”. In: (Apr. 2023). DOI: 10.48550/ARXIV.2304.07917. arXiv: 2304.07917 [quant-ph].
- [Liu+22] X. Liu et al. “Survey on the Improvement and Application of HHL Algorithm”. In: *Journal of Physics: Conference Series* 2333.1 (Aug. 2022), p. 012023. ISSN: 1742-6596. DOI: 10.1088/1742-6596/2333/1/012023.
- [Llo96] S. Lloyd. “Universal Quantum Simulators”. In: *Science* 273.5278 (Aug. 1996), pp. 1073–1078. ISSN: 1095-9203. DOI: 10.1126/science.273.5278.1073.
- [LMS22] N. Linden, A. Montanaro, and C. Shao. “Quantum vs. Classical Algorithms for Solving the Heat Equation”. In: *Communications in Mathematical Physics* 395.2 (Aug. 2022), pp. 601–641. ISSN: 1432-0916. DOI: 10.1007/s00220-022-04442-6.
- [Low19] G. H. Low. “Hamiltonian simulation with nearly optimal dependence on spectral norm”. In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. STOC ’19. ACM, June 2019. DOI: 10.1145/3313276.3316386.
- [MP16] A. Montanaro and S. Pallister. “Quantum algorithms and the finite element method”. In: *Physical Review A* 93.3 (Mar. 2016), p. 032324. ISSN: 2469-9934. DOI: 10.1103/physreva.93.032324.
- [NC09] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2009. DOI: 10.1017/cbo9780511976667.

- 
- [NRT92] N. M. Nachtigal, S. C. Reddy, and L. N. Trefethen. “How Fast are Nonsymmetric Matrix Iterations?” In: *SIAM Journal on Matrix Analysis and Applications* 13.3 (July 1992), pp. 778–795. ISSN: 1095-7162. DOI: 10.1137/0613049.
- [Qis23] Qiskit contributors. *Qiskit: An Open-source Framework for Quantum Computing*. 2023. DOI: 10.5281/zenodo.2573505.
- [Reb+19] P. Rebentrost et al. “Quantum gradient descent and Newton’s method for constrained polynomial optimization”. In: *New Journal of Physics* 21.7 (July 2019), p. 073023. ISSN: 1367-2630. DOI: 10.1088/1367-2630/ab2a9e.
- [RG17] L. Ruiz-Perez and J. C. Garcia-Escartin. “Quantum arithmetic with the quantum Fourier transform”. In: *Quantum Information Processing* 16.6 (Apr. 2017). ISSN: 1573-1332. DOI: 10.1007/s11128-017-1603-1.
- [RLR23] D. Ramacciotti, A.-I. Lefterovici, and A. F. Rotundo. “A simple quantum algorithm to efficiently prepare sparse states”. In: (Oct. 30, 2023). DOI: 10.48550/ARXIV.2310.19309. arXiv: 2310.19309 [quant-ph].
- [Saa03] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Jan. 2003. ISBN: 9780898718003. DOI: 10.1137/1.9780898718003.
- [SBM06] V. Shende, S. Bullock, and I. Markov. “Synthesis of quantum-logic circuits”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25.6 (June 2006), pp. 1000–1010. ISSN: 1937-4151. DOI: 10.1109/tcad.2005.855930.
- [Sog22] T. Sogabe. *Krylov Subspace Methods for Linear Systems: Principles of Algorithms*. Springer Nature Singapore, 2022. ISBN: 9789811985324. DOI: 10.1007/978-981-19-8532-4.
- [Som+08] R. D. Somma et al. “Quantum Simulations of Classical Annealing Processes”. In: *Physical Review Letters* 101.13 (Sept. 2008), p. 130504. DOI: 10.1103/PhysRevLett.101.130504. URL: <https://link.aps.org/doi/10.1103/PhysRevLett.101.130504> (visited on 12/25/2023).
- [SSO19] Y. Subaşı, R. D. Somma, and D. Orsucci. “Quantum Algorithms for Systems of Linear Equations Inspired by Adiabatic Quantum Computing”. In: *Physical Review Letters* 122.6 (Feb. 2019), p. 060504. ISSN: 1079-7114. DOI: 10.1103/physrevlett.122.060504.
- [SX20] C. Shao and H. Xiang. “Row and column iteration methods to solve linear systems on a quantum computer”. In: *Physical Review A* 101.2 (Feb. 2020), p. 022322. ISSN: 2469-9934. DOI: 10.1103/physreva.101.022322.

- [Vir+20] P. Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.
- [WBL12] N. Wiebe, D. Braun, and S. Lloyd. “Quantum Algorithm for Data Fitting”. In: *Physical Review Letters* 109.5 (Aug. 2012), p. 050505. ISSN: 1079-7114. DOI: 10.1103/physrevlett.109.050505.
- [Weg05] I. Wegener. *Complexity Theory*. Springer-Verlag, 2005. ISBN: 3540210458. DOI: 10.1007/3-540-27477-4.
- [Wic22] T. Wick. *Numerical Methods for Partial Differential Equations*. en. Hannover : Institutionelles Repositorium der Leibniz Universität Hannover, 2022. DOI: 10.15488/11709.
- [Yua+23] Y. Yuan et al. “An improved QFT-based quantum comparator and extended modular arithmetic using one ancilla qubit”. In: *New Journal of Physics* 25.10 (Oct. 2023), p. 103011. ISSN: 1367-2630. DOI: 10.1088/1367-2630/acfd52.
- [Zuo+21] Q. Zuo et al. “An Extended Row and Column Method for Solving Linear Systems on a Quantum Computer”. In: *International Journal of Theoretical Physics* 60.7 (Feb. 2021), pp. 2592–2603. ISSN: 1572-9575. DOI: 10.1007/s10773-020-04685-w.