LUND UNIVERSITY

# Quantum Optimization

Author: Luca Bernecker

Thesis submitted for the degree of
Bachelor of Science, 15 Ects
Project duration: Two months
Supervised by Andrea Idini

Department of Physics
Division of Mathematical Physics
May 2020

## Preface

I would like to begin by thanking all the people who helped me writing the thesis. Firstly, I would like to thank Emmanuel D'Costa and Ferdinand Gu for helping me understand the bundle adjustment code and Linda Erwe for peer reviewing the paper. Finally, I would like to show my gratitude to my supervisor Andrea Idini, who helped me with various problems, that I encountered and who not only advised me in the work.

## Abstract

This thesis concerns the implementation of quantum algorithms on a computer vision problem called bundle adjustment. Bundle adjustment can be simplified for illustration purposes by the following example. Two different cameras take a picture of the same object in different angles. Features of the pictures on the different cameras can be seen as points and, in a space sense, they are desired to be as close as possible to each other in a 3D space, because this minimizes the error on the 3D model in space. The bundle adjustment optimizes a cost function so that, distance between the points of the individual pictures can be minimized. The different quantum algorithms may be used to minimize the error between the points in the 3D space.

In order to understand the quantum algorithms one needs to understand the foundations of quantum computing. Quantum computers are fundamentally different compared to classical computers. Whereas a classical computer works with bits, which can be 0 or 1, a quantum computer is based on qubits, which are vectors $|+\rangle$ and $|-\rangle$ of a Hilbert space. The computations on qubits and between different qubits are performed via quantum gates.

A more precise picture of how bundle adjustment works and how gates are applied to form different quantum algorithms as well as different algorithms used in this thesis are given and explained in detail in section 3. The three quantum algorithms used in this paper are HHL, VQE and QAOA. The HHL algorithm is a quantum linear problem solver and is used to update the cost function. VQE and QAOA are quantum algorithms, which find the minimum eigenvalue and its corresponding eigenvector.

In the result section, we apply the VQE algorithm onto a linear problem with four points. Then we attempt to introduce VQE and HHL to the bundle adjustment, where computational difficulties and strategy problems arise.

# Contents

## 1 Abbreviations

| | |
|---|---|
| COBYLA | Constrained Optimization by Linear Approximation |
| HHL | Harrow-Hassidim-Lloyd |
| MEF | Modular Exponential Function |
| QAOA | Quantum Approximate Optimization Algorithm |
| QFT | Quantum Fourier Transform |
| SPSA | Simultaneous Perturbation Stochastic Approximation |
| VQE | Variational Quantum Eigensolver |

# 2 Introduction

## 2.1 Quantum Computing

Approximately two decades ago, quantum computing was introduced as a new computational paradigm. Quantum computing is mainly used within physics, due to the fact that qubits are based on spins of particles, e.g. photons or electrons[1, 2]. In a classical computer, information is presented with bits, which have the possibility to have binary values, i.e. either 0 or 1. Combining zeroes and ones allows for the presentation of complex information, to a level where complex programs are constructed and difficult calculations can be performed. Physically, qubits can be represented by the spins of photons, electrons or nuclei. The vertical or horizontal polarization can give a qubit the same binary representation as a bit. Moreover, the qubit can be represented as both binary values zero and one at the same time, each respectively with a certain probability, where the qubit is in a super position of the binary values. By combining the binary values, a vector space called Hilbert space can be constructed. A Hilbert space is an abstract vector space allowing the creation of infinite dimensions, in contrast to a Euclidean space which only spans over three dimensions. In quantum physics calculations are mainly within a Hilbert space. The probabilities of the respective binary values are always normalized to have the added probabilities to be equal to one. Once the qubits are measured, the super position collapses to either value. In the following, different qubit states and combinations of two qubits are listed with their vector representation in Hilbert space.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \ |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \ |00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \ |10\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{1}$$

For 4 bits of information in a classic computer, 4 bits are needed, whereas in a quantum computer theoretically only two qubits could be used[3]. There exist different methods of loading data of bits into qubits and utilizing the full capacities of qubits is complex and may be difficult to achieve, where in contrast loading one bit of information into one qubit is less convoluted, as one does not need to couple multiple qubits together via CNOT gates and reload qubits[4]. Another reason for not being able to utilize the full strength of the qubits and quantum computer can arise from hardware issues, where qubits are interacting with the environment or not precisely produced circuits[5].

Quantum computing can be performed with the programming language python, using the package Qiskit on a classical computer. Qiskit is created by IBM's research team and allows one to create a quantum circuit and perform calculations on a platform, which uses python. When using Qiskit, IBM offers, at the moment, up to 9 different quantum computers, which range between 1 to 32 qubits, with frequent updates. Firstly, on a quantum computer a circuit is created with qubits; secondly operations are implemented on the circuit through the use of gates.

Most gates apply rotations to qubits. One of the most important gates, which is used in most algorithms to prepare the state, is the Hadamard "H" gate. The H gate rotates a state from 0, 1 to $|+\rangle$,$|-\rangle$ separately, resulting in a super position of the binary values one and zero. $|+\rangle$ and $|-\rangle$ are superpositions of 0 and 1 according to $1/(\sqrt{2}\,|0\rangle + |1\rangle)$ and $1/\sqrt{2}(|0\rangle - |1\rangle)$ respectively. By applying multiple gates onto a quantum circuit, algorithms can be created [6].

## 2.2 Bundle Adjustment

The purpose of this thesis is to introduce quantum computing into bundle adjustment within the field of computer vision. Bundle adjustment is well described by Triggs, McLauchlan, Hartley and Fitzgibbon: "Bundle adjustment is the problem of refining a visual reconstruction to produce jointly optimal 3D structure and viewing parameter (camera pose and/or calibration) estimates."(Page 1-2)[7]. The term "jointly optimal" can be explained by an example. In bundle adjustment, different cameras on separate positions take a picture of the same object at various angles. On each camera, a specific characteristic of an object can be identified, which can be illustrated as roof top in Fig.(1). The contrast of the green above and brown below the roof can be easily identified, when analysing the picture as a matrix. Each color has a number and therefore a different matrix representation. The cameras from different angles may have the contrast in a slightly different shape and place because of the angles, but the contrast of the colors holds. For the different cameras the goal is to identify the roof to be at the exact place in space.

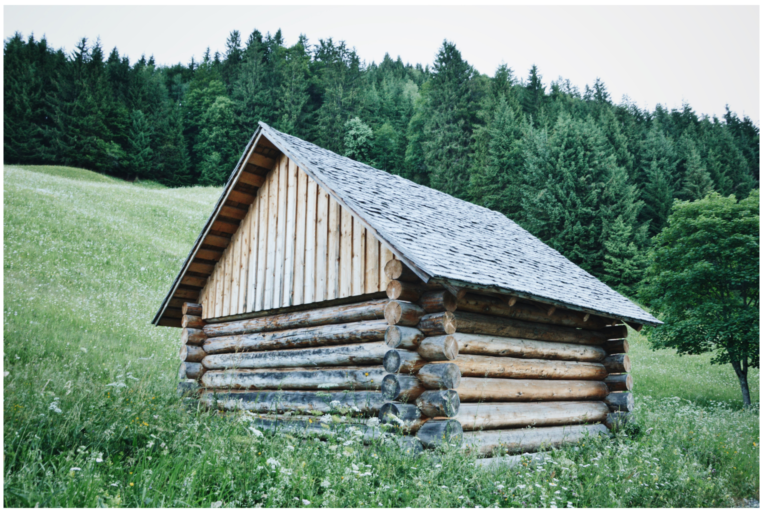

Figure 1: Rooftop by Linda Erwe

A cost function is required to minimize the distance of corresponding points from the different cameras, meaning that each roof top in the different cameras should be optimized in order to be recognized at the same place in space. In general, bundle adjustment requires a lot of data, time and computational power and can possibly map an entire city with reference points or to recognize a point corresponding to a database of millions of reference points[7, 8]. When mapping a city, as discussed in the article "Building Rome in a day" by Agrawal et al, often pictures need to be prepared for bundle adjustment, depending on if the pictures are preselected, or if a program has to find pictures of the same object in a database first, which leads to a significant computational increase. Therefore, one may introduce quantum computing to increase the speed of the processes. The problem tackled in this thesis is kept simple and the optimization is only performed on a simplification of a standard setup within 3D reconstruction[7].

## 3 Method

### 3.1 Bundle Adjustment

The phrase bundle adjustment derives from "bundle", which is referring to the light rays, that are reflected by the points towards the camera and the word "adjustment" refers to the optimization of the two cameras and the points with the cost function[7]. Bundle adjustment can be seen as a large sparse geometric parameter estimation problem, in which case the parameters are then the combined points, coordinates, camera poses and calibrations. Bundle adjustment is a minimization of the reproduction error and in our case used for 3D reconstruction[7].

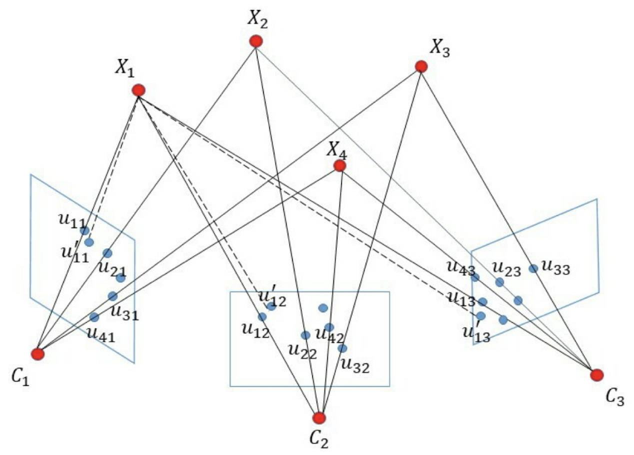

Figure 2: a representation of Bundle Adjustment from [9]

In Fig.(2) the different $u$´s are the observations and $X_1$, $X_2$, $X_3$ and $X_4$ are the points in the real world 3D frame. $C_1$, $C_2$ and $C_3$ are the cameras on different positions with different angles on the $X$ points and $u'$´s are the re-projected 2D points. The solid lines show the projection lines. The dotted lines represent the re-projection lines of the points with respect to the cameras[9].

By creating a 3D image with different pictures, information about the position of the cameras, orientations and parameters can additionally be retrieved. Bundle adjustment has the potential to not just structure simple 3D models, but also complex objects with various constraints. Therefore, bundle adjustment is the future method of choice for computer vision problems[7].

Bundle adjustment has the possibility to use different optimization algorithms. The most effective classical optimizer is to date the Levenberg-Marquardt[9]. This has been initially used in the small bundle adjustment version in this thesis. Therefore, we can implement the following cost function and the definition of Hessian. The explicit simplification in this thesis is to only include two cameras in comparison to Fig.(2) For this problem the cost function is a squared cost defined by:

$$S = \sum_{i=1}^{n} (\vec{y}_i - \vec{x}_i)^2, \tag{2}$$

where $\vec{y}$ are the observation vectors $(u)$, $\vec{x}$ are the re-productions $(u')$ in Fig.(2) and our parameter vectors. $n$ is the number of the points in total. We try to minimize function (2).

The idea is to find the Hessian and apply the quantum algorithm VQE and HHL the Hessian. As the Hessian is computationally intense to calculate, an approximation is made. One may consider a more general representation of the previous explained function to minimize:

$$S_i = \underset{x}{min}\frac{1}{2} \mid\mid \vec{y}_i - \vec{f}(\vec{x})_i \mid\mid^2 \tag{3}$$

Afterwards, looking at the Tailor approximation of a simple cost function one can write an approximation as follows:

$$\vec{f}(\vec{x}) = \vec{f}(\vec{x}_0) + \hat{J}(\vec{x} - \vec{x}_0) \cdot (\vec{x} - \vec{x}_0) + \text{higher orders} \tag{4}$$

$J(x - x_0)$ is the Jacobian with the distance from the equilibrium to the new point $x - x_0$. Additionally we assume that $x$ is close to $x_0$, where $x_0$ is the initial equilibrium. Then the Tailor approximation can be truncated and generalized for our case leading to:

$$S_i = \underset{x}{min}\frac{1}{2} \mid\mid \vec{y}_i - \vec{f}(\vec{x}_0)_i - \hat{J}(\vec{x} - \vec{x}_0) \cdot (\vec{x} - \vec{x}_0)_i \mid\mid^2 \tag{5}$$

Finally the square of eq.(5) results in $J^T J$ that is the linear approximation of Hessian.

## 3.2 Gates and Quantum Circuit

There are four major companies offering quantum programming platforms, three of them have their own programming languages. IBM, uses Qiskit[6], Microsoft, uses Q, Google, uses Cirq and finally Alibaba. Cirq is very similar to Qiskit, not simply because it creates a user friendly environment for creating highly complex quantum circuits, but likewise a few gates are exactly the same[6, 10]. For the following section, it is assumed that the code is written in Python and Qiskit is imported as:

```
from qiskit import *.
```

The first step when initializing a quantum circuit is to open a quantum and classical register and then load it into the circuit. This can be done by coding:

```
circuit.QuantumCircuit(qr, cr)
```

Where *qr* and *cr* are integers and express the length of the register. The register is initialized with $|0\rangle$. Afterwards, gates are applied onto the circuit. For example an H gate can be applied to the circuit by coding:

```
circuit.h(qr).
```

This applies the Hadamard gate to all quantum registers. Gates are mainly applied to the quantum register, with the exception of measuring[6].

The "H" gate also called Hadamard gate and transforms a $|0\rangle$ to $|+\rangle$ and $|1\rangle$ to $|-\rangle$ respectively. Most circuits are initialized with a Hadamard gate over the input qubits, which allows for a more complex treatment of the qubits as the superposition of the binary values of zero and one [6]. Another widely used gate is the CX gate. The CX gate acts on two qubits, namely a control and a target qubit. If the control qubit is in the state $|1\rangle$, it changes the target qubit from $|0\rangle$ to $|1\rangle$ or the other way around[6]. Those gates can be represented in a matrix form seen in table 3.2, where the gates are classified in one and multi qubit gates. One qubit gates are gates, which only apply to one qubit and the multi qubit gates are applied to multiple qubits.

| One qubit gates | Multi qubit gates |
|---|---|
| $\mathrm{H} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ | $\mathrm{CX} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$ |
| $\mathrm{X} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ | $\mathrm{CU} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & U_{00} & U_{01} \\ 0 & 0 & U_{10} & U_{11} \end{pmatrix}$ |
| $\mathrm{R}(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$ | $\mathrm{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ |
| $\mathrm{U3} = \begin{pmatrix} cos(\theta) & -e^{i\lambda}sin(\theta/2) \\ e^{i\phi}sin(\theta/2) & e^{i\lambda+i\phi}cos(\theta/2) \end{pmatrix}$ | |

Table 1: Common gates [6, 11]

There are three different gates, which can be described simultaneously, the U3, U2, U1. The U3 is a rotation on each, x-,y-,z- axes with respect to Euler angles $\theta$, $\phi$, $\lambda$ and the respective order. An example is U3($\pi/2$, $\pi/2$ and $\pi/2$), which transforms $|0\rangle$ to $|+\rangle$. The U2 transform is only operating in two dimensions and the U1 in one dimension. The U2 can achieve the same outcome as the U3 in the example, but the path taken is different. Finally the U1 is a rotation from x to y[6].

Another important group of gates are the Rx, Ry and Rz. The Rx gate is a rotation around the x axis. Ry around the y axis, and so forth. Coming from those rotations, simplified gates X, Y and Z are derived. Those gates are simply rotations with an angle of $\pi$[6].

There exist multiple different gates, which are combining the previously discussed gates, like the SWAP gate, which is a collection of CX gates. The last mentionable gate is the measurement gate, which measures the state of the qubits. Since the gate is intended to run on a quantum computer, the measurement is not reversible[6].

## 3.3 Algorithms

Algorithms as QAOA or HHL are composed of multiple different methods. The main methods are explained in the following section[6, 12].

### 3.3.1 Hamiltonian Simulation

In this thesis the Hamiltonian simulation is used in the HHL algorithm to find solutions to a linear equation. The evolution of a quantum state under the Schrödinger equation is defined by:

$$i\frac{d}{dt}|\psi\rangle = \hat{H}|\psi\rangle \tag{6}$$

In the evolution, one aims to find a quantum circuit, which implements an unitary operator $e^{-i\hat{H}t}$. Finding low run time evolutions is a current research field and there are different solution attempts. Low run time means that, if one finds a circuit, which is not highly complex and can be implemented in a low number of gates,then this circuit has a low run time. The following definition of quantum simulation is stated in [12]:

"We say that a Hamiltonian $\hat{H}$ that acts on n qubits can be efficiently simulated if for any $t > 0$, $\epsilon > 0$, there exists a quantum circuit $U_{\hat{H}}$ consisting of poly(n,t,1/$\epsilon$) gates such that $|| \ U_{\hat{H}} - \epsilon^{-i\hat{H}t} \ ||< \epsilon$. Since any quantum computation can be implemented by a sequence of Hamiltonian simulations, simulating Hamiltonians in general is BQP -hard, where BQP refers to the complexity class of decision problems efficiently solvable on a universal quantum computer." [12][13] Poly refers to polynomial and BQP is an abbreviation for bounded-error quantum polynomial time, where BQP- hard describes the bound of time taken for the algorithm.

Parameter $t$ is the time for that a minimum time of $\Omega(t)$ is needed, so that a certain time $t$ can be used to simulate a Hamiltonian $\hat{H}$. This is formally proven by the no fast-forwarding theorem[14].

### 3.3.2 Trotter-Suzuki Method

The Trotter-Suzuki method is used for Hamiltonian evolution, which is further clarified within this section and can widely be used in quantum machine learning and linear equation solvers[5]. Let us consider a unitary operator $U$, i.e. $UU^\dagger = U^\dagger U = 1$. Additionally, we assume that $U$ is self adjoint $U = U^\dagger$. Therefore, the Hamiltonian basis can be transformed by using $UHU^\dagger$. This allows us to rewrite $e^{-iUHU^\dagger t}$:

$$e^{-iUHU^\dagger t} = Ue^{-iHt}U^\dagger \tag{7}$$

Another important initial idea to simulate any efficiently diagonalisable Hamiltonian is, when having the possibility to reach the diagonal entries of the Hamiltonian by $\langle i| H |i\rangle = H_{ii}$, we can simulate the diagonal Hamiltonian by starting with an empty second register $|i, 0\rangle$, afterwards loading the Hamiltonian into it giving $|i, H_{ii}\rangle$. Then a gate can be applied onto the new state giving $e^{-iH_{ii}t} |i, H_{ii}\rangle$. Finally, one can unload the register again and use the super position principle to realize that, $e^{-iH_{ii}t} |i, 0\rangle = e^{-i\hat{H}t} |i\rangle \otimes |0\rangle$. Using Tailor expansion and the Cauchy formula we get the following expression, assuming that operators do commute and therefore do not give mixed terms when applying the Binomial theorem with the Cauchy formula:

$$e^{\hat{H}_1+\hat{H}_2} = e^{\hat{H}_1} \cdot e^{\hat{H}_2} \tag{8}$$

The derivation can be seen in[12]. For the general form for non commutators the Lie-product formula is needed[6].

$$e^{-i(\hat{H}_1+\hat{H}_2)t} = \lim_{m\to\infty} (e^{-i\hat{H}_1 t/m} e^{-i\hat{H}_2 t/m})^m \tag{9}$$

This simulation can be generalized up to $H_n$ and an error minimization for $m$ steps can be executed[12].

### 3.3.3 Quantum Fourier Transform

The Quantum Fourier Transform (QFT) is defined by the transformation from a state $|x\rangle$ to a new state $|k\rangle$ which is defined as follows:

$$|x\rangle \to \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega^{xk} |k\rangle \tag{10}$$

$\omega = e^{2i\pi/N}$ where N are the dimensions. Here it is assumed that the Fourier basis is orthogonal. The inverse transform is simply defined by adding a minus in the exponent of $\omega$. The transformation above can be visualized better assuming a computational basis state $|j_n\rangle$ transformed to a new basis state $|f_j\rangle$. Where the new basis can be written out:

$$|f_j\rangle = \frac{1}{\sqrt{N}}(|0\rangle + e^{i2\pi 0.j_n} |1\rangle)(|0\rangle + e^{i2\pi 0.j_{n-1}j_n} |1\rangle)...(|0\rangle + e^{i2\pi 0.j_1...j_n} |1\rangle) \tag{11}$$

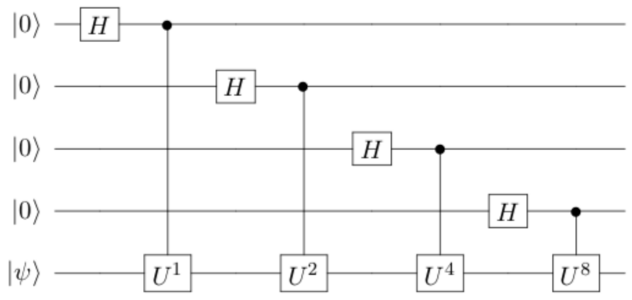

Figure 3: Circuit implementation[12].

When it comes to the implementation of QFT, one starts with $|0\rangle$ in quantum computing. As the equation indicates $|f_j\rangle$ is in form of either $|+\rangle$ or $|-\rangle$, depending on the $j_n$ state, which can be obtained by applying a Hadamard gate onto $j_n$, where $j_n$ is either 0 or 1. The complete procedure for the QFT is as follows. The Hadamard gate is applied onto the $n^{th}$ qubit, afterwards a rotation $R_k$ on the $|j_{n-k+1}\rangle$. Finally, a swap gate is implemented to bring the qubits into the right order[12].

### 3.3.4 Quantum Phase Estimation

Quantum phase estimation is an application to find the eigenvalue to a corresponding eigenstate. In this method, an operator is applied to a known eigenstate $|\psi\rangle$. The eigenvalue is complex and can be written as $\lambda = e^{i2\pi\phi}$, where $\phi$ is a phase in-between 0-1. Therefore, it is possible to approximate $\phi$. Mathematically, this can be generalized and described with $U^j |\psi\rangle = \lambda^j |\psi\rangle = e^{i2\pi\phi j} |\psi\rangle$. One may start as usual in quantum computing with$|0\rangle^{\otimes n}$ qubits and m quantum states in $|\psi\rangle$. $n$ qubits are initalized to zero $|0\rangle^{\otimes n}$ and $m$ qubits inizalized in $|\psi\rangle$. Then, a Hadamard gate is applied to the n qubits, giving the state $\frac{1}{\sqrt{N}} |x\rangle |\psi\rangle$, where $N = 2^n$. Afterwards, the operator $U^{2^k}$ is applied, where $k$ is the control qubit. For example, k=0 is the first qubit and k=n-1 the last. Then the operators $U^{2^k}$ are implemented. Figure (3) shows how the quantum phase estimation is implemented for the preparation of four qubits. Quantum phase estimation prepares n qubites, which are under superposition. The procedure can be simplified by using quantum Fourier transformations. Finally, a $U$ gate is applied and finally the Quantum Fourier transformation and a measurement[12].

### 3.3.5 VQE

The VQE is used to find the minimum eigenvalue by a parametrized circuit ( i.e. variational form). For the VQE to run, $H$ is needed to be Hermitian,

$$H = H^\dagger \tag{12}$$

According to the spectral theorem the eigenvalues of a Hermitian matrix must be real. Therefore, any eigenvalue of $H$ has the property of $\lambda_i^* = \lambda_i$. Additionally we can write H as the following,

$$H = \sum_{i=1}^{N} \lambda_i \left|\psi_i\right\rangle \left\langle\psi_i\right| , \tag{13}$$

where each eigenvalue is connected to an eigenvector. The expectation value of the observable Hamiltonian on any quantum state is:

$$\langle H \rangle_{\psi} = \langle\psi| H |\psi\rangle \tag{14}$$

This can be combined to:

$$\langle H \rangle_{\psi} = \langle\psi| H |\psi\rangle = \langle\psi| \sum_{i=1}^{N} |\psi_i\rangle \lambda_i \langle\psi_i \mid \psi\rangle = \sum_{i=1}^{N} \lambda_i \mid \langle\psi_i \mid \psi\rangle \mid^2 \tag{15}$$

Which can be used to write:

$$\lambda_{min} \leq \langle H \rangle_{\psi} = \sum_{i=1}^{N} \lambda_i \mid \langle\psi_i \mid \psi\rangle \mid^2 \tag{16}$$

Eq.(16) implies that the expectation value of a Hamiltonian over a wave function is always bigger or equal to the smallest eigenvalue of a Hamiltonian. Additionally, the expectation value of $|\psi_{min}\rangle$ is $\langle H \rangle_{\psi_{min}} = \lambda_{min}$, where $|\psi_{min}\rangle$ is the ground state. The variation of $\psi$ to find $\psi_{min}$ is known as variational method. A pratical approach to vary the ansatz is needed to implement the variational method on a quantum computer. VQE does so through the use of a parameterized circuit with a fixed form. The action of a variational method is described by a linear transformation $U(\theta)$. One applies a variational form to a state $|\psi\rangle$, resulting in $U(\theta)|\psi\rangle = |\psi(\theta)\rangle$. Afterwards, iterations, which intend to optimize $\theta$, result in an expectation value $\langle\psi(\theta)| H |\psi(\theta)\rangle \approx E_{gs} = \lambda_{min}$.

The theoretical approach of optimizing the parameters would allow consistent measurements of the lowest eigenvalues. Nevertheless, the hardware of a quantum computer may have defects and noise resulting in lower accuracy of the objective function. Some optimizers also use cardinality of the parameter set, which is the size of the parameter set, to evaluate the objective functions. These different aspects must be considered, when deciding on the right optimizer for the VQE or for the QAOA. There are different methods to optimize the parameters $\theta$, one of the most common parametric optimizations is gradient descent, which updates variables in iterations. Gradient descent is updating the variables by using derivatives until they result in being at the minimum value of the cost function. Although gradient descent could be used to optimize the variables, it is not used in the VQE. Using gradient descent could result in a local optimum instead of the desired global optimum. The environment in question also fluctuates between being noise-free and noisy, which is a problem, because it could result in a local optimum instead of the desired global optimum and the environment varies from being noisy and noise-free environment.

Therefore, other techniques are used for VQE. When VQE is implemented in a noisy environment, Simultaneous Perturbation Stochastic Approximation optimizer (SPSA) should be used[6]. The main feature of the SPSA is, that it approximating the gradient with two measurements of the objective function. The gradient approximation has no dependence on dimensions, therefore the algorithm tends to be used for problems which involve high dimensionality[15]. In a noise-free environment, Constrained Optimization by Linear Approximation optimizer (COBYLA) can be used with greater success compared to derivative depended optimizers. COBYLA is a derivative free optimizer, which uses linear algebraic manipulation to iterate to the global maximum[16].

Variational forms can be categorized into two categories. The first category is using pre-knowledge and therefore, a domain. The second has no pre-knowledge and therefore uses a heuristic circuit. Heuristic circuits are the most abstract kind of circuits without an initial domain, which leads to an increase of gates and complexity[17].

An example of the first category is a certain molecule at ground state. The domain and pre-knownledge are the numbers of particles, which are known. Using the knowledge of particle number one may be able to reduce the number of parameters, by limiting the variational form for particle preserving transformations. In the second category, gates are layered so that an approximation of a wide spectrum of states can be made. Such variational forms are for example RyRz and Ry. The variational form can be configurated by three main points, the depth, the number of qubits in the system and the entanglement setting[6]. The depth is the number of total gates. This setup does not require any domain, but the complexity of the circuit is increased.

### 3.3.6 QAOA

QAOA is an optimization algorithm, which enables to optimize different variables such as in cost functions, distances of lengths, etc.. The QAOA algorithm aims to find a state $|\psi_P(\bar{\gamma}, \bar{\beta})\rangle$, where $\bar{\gamma}$ and $\bar{\beta}$ maximize or minimize the expectation value of the Hamiltonian $F_P = \langle\psi_P(\bar{\gamma}, \bar{\beta})| H |\psi_P(\bar{\alpha}, \bar{\beta})\rangle$. The final state can be denoted by $|\psi_P(\bar{\gamma^*}, \bar{\beta^*})\rangle$, where $\gamma^*$, $\beta^* \in \mathbb{R}$. When the desired state is achieved, it can be measured in the basis Z $|\bar{x}\rangle$ to achieve an output $x^*$. The resulting state is close to the expectation value $M_P = F_P(\bar{\gamma^*}, \bar{\beta^*})$.

The state $|\psi_P(\bar{\gamma}, \bar{\beta})\rangle$ can be derived as follows. The initial state is constructed by a Hamiltonian given in equation (17) and a single Pauli rotation:

$$H = \sum_{k=1}^{m} C_k(x) \tag{17}$$

$C_k(x)$ is the cost function with the respective index $k$, which iterates from 0 to $m$ and has as input the parameter $x$. The Hamiltonian of equation (17) can be concluded by using the cost function $C$, which is in binary combinatorial optimization problems canonically presented as:

$$C(x) = \sum_{Q,\bar{Q}\subset n} \omega_{(Q,\bar{Q})} \prod_{i\in Q} x_i \prod_{j\in\bar{Q}} (1 - x_j) \tag{18}$$

Where $\omega_{(Q,\bar{Q})}$ are the weights, Q are the subsets of a total set n. Using the cost function $C$ of eq. 18 and map it to a diagonalized basis Hamiltonian it results in:

$$H = \sum_{x\in 0,1^n} C_k |x\rangle \langle x| \tag{19}$$

$x \in 0, 1^n$ are describing the computational basis states $|x\rangle$. If $Q \leq l$, where l is a number of weights, then the Hamiltonian can be represented as the sum of Pauli operators $Z$. Where Z can be represented in matrix form:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{20}$$

This leads to a Hamiltonian:

$$H = \sum_{(Q,\bar{Q})\subset[n]} \omega_{(Q,\bar{Q})} \frac{1}{2^{|Q|+|\bar{Q}|}} \prod_{i\in Q}(1 - Z_i) \prod_{j\in \bar{Q}}(1 + Z_j) \tag{21}$$

Assuming that only polynomially in n of $\omega_{(Q,\bar{Q})}$ are non-zero and $Q$ as well as $\bar{Q}$ are not very large, equation (17) holds.

Corresponding to eq.(17) a transverse field Hamiltonian with computational basis is described as follows:

$$B = \sum_{i=1}^{n} X_i \tag{22}$$

where $X$ are qubit Pauli rotations. Then the initial state is prepared by alternating unitarians with p iterations:

$$|\psi_P(\bar{\gamma}, \bar{\beta})\rangle = e^{i\beta_P B} e^{-i\gamma_P H} ... e^{-i\beta_1 B} e^{-i\gamma_1 H} |+\rangle^n \tag{23}$$

Equation (23) can be used onto $|+\rangle^n$, with $X|+\rangle = |+\rangle$. This allows us to get the specific $\bar{\gamma^*}, \bar{\beta^*}$, which has been the main interest and concludes the QAOA, because as mentioned previously $M_P = F_P(\hat{\alpha^*}, \hat{\beta^*})$, when the limit $lim_{p\to\infty} M_p = C_{max}$[6].

### 3.3.7 HHL

The issue of solving linear equations is a common problem, widely spread through different areas in science. A common area, where linear equations are being solved and has a big data amount is machine learning. A problem of particular interest in quantum computing, since it has been demonstrated that, it is possible to achieve a new speed-up compared to available the classical algorithms. One of the algorithms has a significant speedup of $log(N)$, compared to $N$ from the best classical algorithm[12].

The linear equation problem in quantum mechanics has been defined by Childs, Kothari, and Somma as follows: "(QSLP) Let A be an N x N Hermitian matrix with unit determinant. Also, let $\boldsymbol{b}$ and $\boldsymbol{x}$ be N-dimensional vectors such that $\boldsymbol{x} := A^{-1}\boldsymbol{b}$. Let the quantum state in [log N] qubits $|b\rangle$ be given by

$$|b\rangle := \frac{\sum_i b_i |i\rangle}{|| \sum_i b_i |i\rangle ||} \tag{24}$$

and $|x\rangle$ by

$$|x\rangle := \frac{\sum_i x_i |i\rangle}{|| \sum_i x_i |i\rangle ||} \tag{25}$$

where $b_i$, $x_i$ are respectively the $i^{th}$ component of vectors $\boldsymbol{b}$ and $\boldsymbol{x}$ and $|| \boldsymbol{b} ||$ and $|| \boldsymbol{x} ||$ are their relative Euclidean norms. Given the matrix A (whose elements are accessed by an oracle) and the state $|b\rangle$; output a state $|\tilde{x}\rangle$ such that $|| |\tilde{x}\rangle - |x\rangle || \leq \epsilon$ with some probability larger than $\frac{1}{2}$. Note that in practice, we will introduce a 'flag' qubit which will determine whether or not this process has been successful." [12]

For a QLSP to be efficient, two main criteria have to be fulfilled. Firstly, the preparation of the state $|b\rangle$ needs to be efficient, which in the case of quantum computing, has the concrete meaning 'polylogarithmic in the system size N'. Lastly, the reading of the state $|x\rangle$ needs to be efficient as well[12]. The HHL algorithms can be approximately described in three phases. The first step is the phase estimation, where an H, U gates and Fourier transform (FT) are implemented. The second step is the rotation, where an R gate is implemented. Lastly, the third step is to uncompute the register by simply reversing the phase estimation. The detailed procedure starts with considering an operator $A = \sum_j \lambda_j |u_j\rangle \langle u_j|$. One may assume for simplicity that the eigenvectors of A may simply be $|u\rangle$ and using the above QFT the eigenvalue of A is $e^{i\psi_j}$. The method of quantum phase estimation can be used to map $|0\rangle |u_j\rangle \rightarrow |\tilde{\psi}\rangle |u_j\rangle$ , where $\tilde{\psi}$ is a binary representation of $\psi$. Previously the eigenvector of A has been defined as $|u_j\rangle$, which gives means that it has corresponding eigenvectors $e^{i\lambda_j t}$. Therefore, if the phase estimation is applied to the matrix $e^{iAt}$ the following mapping can be produced[12].

$$|0\rangle |u_j\rangle \rightarrow |\tilde{\lambda}\rangle |u_j\rangle \tag{26}$$

The tilde sign indicates the binary representation to a certain error. Then the rotation is applied onto $|\tilde{\lambda_j}\rangle$. For the rotation a new ancilla register in $|0\rangle$ is added and the rotation $\sigma_y$ with the operator form of $e^{-i\theta\sigma_y}$:

$$\sqrt{1 - C^2/\tilde{\lambda_j^2}}\, |\tilde{\lambda_j}\rangle |u_j\rangle |0\rangle + C/\tilde{\lambda_j}\, |\tilde{\lambda_j}\rangle |u_j\rangle |1\rangle , \tag{27}$$

where C is the normalization constant and $\lambda_j$ the eigenvalue of the eigenvectors $|u_j\rangle$ The operator $e^{-i\theta\sigma_y}$ is the rotation:

$$e^{-i\theta\sigma_y} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} , \tag{28}$$

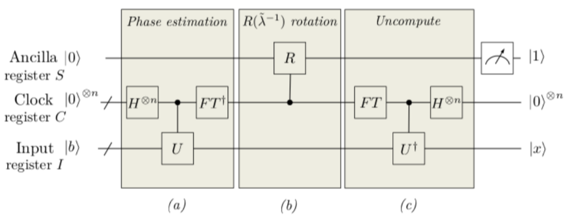
Phase estimation
$R(\tilde{\lambda}^{-1})$ rotation
Uncompute
Ancilla $|0\rangle$ register $S$
Clock $|0\rangle^{\otimes n}$ register $C$
Input $|b\rangle$ register $I$
$H^{\otimes n}$
$FT^{\dagger}$
$U$
$R$
$FT$
$U^{\dagger}$
$|1\rangle$
$|0\rangle^{\otimes n}$
$|x\rangle$
(a)
(b)
(c)

Figure 4: Circuit of HHL.[12]

where $\theta = arccos(C/\tilde{\lambda})$. It can be assumed to have a quantum state $|b\rangle = \sum_i b\,|i\rangle$. $|b\rangle$ can be written as the eigenbasis $\sum_j |u_j\rangle$. Keeping in mind that $\sum_j |u_j\rangle$ are the eigenbasis of A, therefore $|b\rangle$ can be expressed by $\sum_j \beta_j\,|u_j\rangle$. This can be used to write the state [12],

$$\sum_{j=1}^{N} \beta_j\,|\tilde{\lambda_j}\rangle\,|u_j\rangle\,(\sqrt{1 - C^2/\tilde{\lambda}_j^2}\,|0\rangle + C/\tilde{\lambda_j}\,|1\rangle) \tag{29}$$

If this state is uncomputed, then the $|\tilde{\lambda}\rangle$, can be replaced with a superpositioned 0 state $|0\rangle$. Therefore, one is able to compute the linear equation $|x\rangle = A^{-1}\,|b\rangle$. This is shown in the figure below. In the figure one can see, like previously discussed, the three different steps it takes. The $U$ gate can be defined to be $U = \sum_{k=0}^{T-1} |k\rangle\,\langle k|^C \otimes e^{iAt_0/T}$ where C is the clock register in this case, T = 2t and $t_0 = 2\pi$. Finally, $\otimes$ indicates that this is an additional part of the operator. One may notice that in Fig.(4) a new register was added. The clock register is used for the implementation of the quantum evolution. It can simulate the time steps taken and is prepared as followed:[12]

$$|\psi_0\rangle = \sqrt{2/T} \sum_{\tau=0}^{T-1} \frac{\pi(\tau + 1/2)}{T}\,|\tau\rangle^C\,, \tag{30}$$

where a certain error can be achieved and $T$ indicates the number of steps to achieve the simulation of $e^{iAt}$. In general, $t_0$ gives the error bounds[12].

In the HHL algorithm, an important part of the simulation of the Hamiltonian is the filter function. This will not be discussed in detail in this work. Nevertheless Childs, Kothari and Somma cover the topic more specifically in [12]. To be noted that a simulation is not necessarily error-free. Therefore, over time, an error in the eigenvalue may appear. The problem may be significant for the inverted $A$, as the error of $1/\lambda$ can strongly affect the invertion. This leads to the filter function, which only inverts $A$ on well-conditioned subspace[12].

## 4 Results

### 4.1 Shor Algorithm

The Shor algorithm is a good example of an algorithm which can be implemented on a quantum computer and runs faster than any classical approach, such as Bernstein–Vazirani algorithm or Grover's algorithm. The Shor algorithm takes an integer number and calculates its prime factors[18]. We implement a discrete example of the Shor algorithm to factorize N=15 with the input of a=11. The function of the Shor algorithm is called modular exponential function (MEF) and is defined by $f = a^x mod(N)$ [19]. Additionally, the smallest positive function satisfying MEF $a^r mod(N) = 1$ must be satisfied[18]. Using the period (r) and the fact that one factor of N is established by the greatest common denominator of $a^{r/2} \pm 1$ assuming that the factor $N$ has higher probability than 1/2 the factor can be established. Parts of Shor´s algorithm provides the solutions to the period, leading to finding the solutions for the factors. In the case of this thesis, it turns out to be a period of two [19]. This simplifies to one multiplication step $a^2 mod(N) = 1$[18].

The procedure of the algorithm is highly simplified compared to the general Shor algorithm, because a specific example is taken. Firstly, a Hadamard gate is applied onto an empty register, resulting into a transformation, $|0\rangle^{\otimes n} \rightarrow 2^{-n/2} \sum_{x=0}^{2^n-1} |x\rangle$. Then MEF is applying $a^r mod(N)$ onto the register, resulting in the equation (31). In the special case two CX gates implement the MEF[19].

$$1/\sqrt{2} \sum_{x=0}^{2^n-1} |x\rangle \, |a^x mod(N)\rangle \tag{31}$$

Equation (31) is highly entangled all $2^n$ inputs in general. Then, a QFT is applied. In this special case, the QFT is simply implemented by two Hadamard gates and one rotation between the first and second qubit. [18]

$$1/2^n \sum_{x}^{2^n-1} \sum_{y}^{2^n-1} e^{2\pi yx/2^n} |y\rangle \, |a^x modN\rangle \tag{32}$$

with $y = c\ 2^n/r$, where r is the period, c is an integer and $n = 2log_2 N$[19]. The implementation of the circuit of the specific example is done in this paper and can be seen in Figure 4.1. QFT and MEF can be implemented in general for any number N. However, MEF is rapidly becoming increasingly complex.

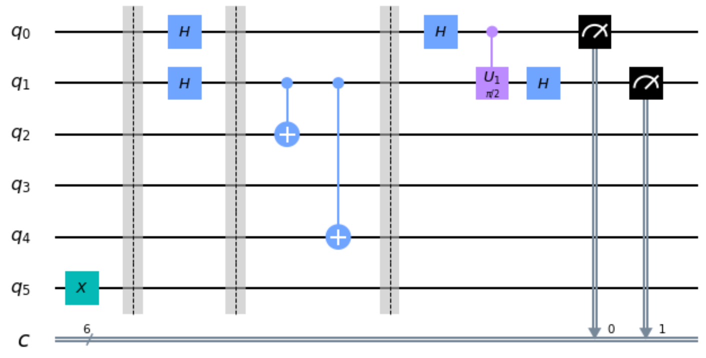

Figure 5: Shor circuit for N=15 and a=11.

There are many different implementations of a QFT and many different modifications. For instance some use a different QFT for the Shor algorithm, which minimizes the depth of the algorithm, compared to a general QFT.[6, 18, 19]

## 4.2 Environment

In order to compare different algorithms and to switch between them in an elegant manner, a launcher was created, which facilitates a change change between different optimizers and different initial points for the bundle adjustment. Without the created environment the changes of algorithms like Levenberg-Marquardt Algorithm, VQE and HHL, would take longer time and changes in different python files.

## 4.3 Linear Problem

The linear problem considered in this thesis is an introduction to the actual bundle adjustment problem. The main idea of solving the linear problem can be considered to be similar to the bundle adjustment problem. Each of the systems utilizes a VQE, which is updating the steps. The results of the linear problem will be discussed quantitatively, as the main principle of the problem is to see that VQE can be implemented to update steps resulting in a reasonable answer. Another feature to mention is that the quality of the result can differ when stopping updating steps at different tolerance ”tol”, where the updating stops and by the constant in the cost function.

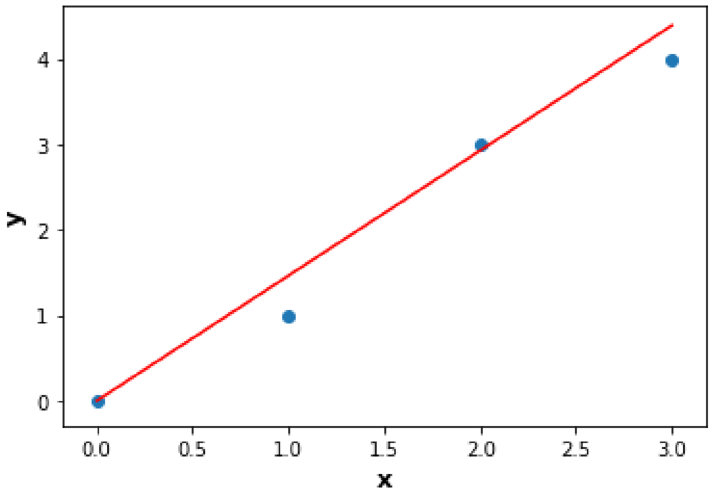

Figure 6: Linear fit between four points using VQE.

The linear problem is including four points, one at (0,0) and three points at random in space. The (0,0) point is initialized, because we want to push the interception point (b) to zero, which leads to, that only the slope needs to be updated and the interception can be held at zero, as the interception is not being updated. The cost function for the problem is the mean square error being $1/n\sum_i^n (y_i - x_i \cdot k)^2$, where $k$ is the slope. The jacobian of the cost function is taken by using the package autograd. Then, the input for the VQE is the jacobian, which is a 4x4 matrix. The VQE and QAOA gives results of the lowest eigenvalue and corresponding eigenfunction. The eigenvalue resulting from the VQE is used to be added to the initialized slope $k$. In Fig.(6), one can see the estimated slope by the VQE. The slope given by the VQE is 1.464, where a standard classical result is about 1.400. The simulation takes about 5 minutes compared to 10 seconds from the classical algorithm. The code for the linear problem is detailed in the Appendix.

## 4.4 VQE and QAOA in Bundle Adjustment

The VQE and QAOA algorithms are attempted to be implemented into the simple version of bundle adjustment. The code was provided by Andrea Idini. Initially the program ran with the Levenberg–Marquardt algorithm, which in this thesis, is replaced by the VQE. The idea is to find the Hessian and use VQE on parts of the Hessian. Then the resulting eigenvalues are used to update the steps to minimize the function for the different inputs. The VQE and QAOA result both in the same shape of result, returning the lowest eigenvalue of the input matrix. Nevertheless, when loading the data into a quantum computer, differences as the variational form may change the result.

The VQE and QAOA are constructed for physics applications, because of that fact one may expect difficulties on the use outside of physics. As expected, several issues arose from the attempt of implementation of the QAOA and VQE, because the QAOA and VQE only calculate the ground state eigenvalues of a system the possibility is to use only one of the functions and calculate the eigenvalue of one function as in the linear problem. Due to the fact that the gradient is (2,1), one multiplies it with its transpose as when creating a Hessian out of the jacobian. This is because the VQE needs a square matrix of order $2^n$, where n is the bits number, to run, due to the fact of the superposition of the qubits. To now, the issues arising are that the eigenvalues are in too low order of magnitude to update the initial positions successfully, when using the techniques of splitting up the Hessian or creating smaller Hessian. Nevertheless, the eigenvalues converge.

The main issue of the VQE is that it can only calculate one eigenvalue at a time. In the linear problem, the partial derivatives of the function could be considered as whole jacobian and the transpose of the jacobian times the jacobian as hessian ($J^T J = H$). Therefore, the problem can be considered to be different, compared to the final implementation. In the final setup the Hessian can be seen to be split up to different smaller sections and the eigenvalues calculated upon it.

Different approaches were considered, another unsuccessful try was to fill up the missing values of the gradient with zeroes to a 2x2 matrix instead of multiplying the transpose of the matrix with itself. Another approach was to cut the Hessian into smaller pieces and evaluate on them, instead of creating smaller Hessians and add zeroes to create a $2^n$ Matrix. Neither of those approaches gave a proper step size and all of the different approaches returned very small eigenvalues in order of $10^{-8}$.

The issue with the VQE is that, when running on jupyter notebook it shows no errors and shows signs of calculating, while it is executing large simulations. On spyder the problem at hand was, that VQE simulation gave errors on correct code and only an update of the macos could solve the issue. The implementation of the VQE is at a stage where the VQE runs on a one value problem, as in the linear problem. Nevertheless, for the complex system, the setup with multiple parameters and functions it is not working currently. This is due to the fact that one eigenvalue would have to update multiple parameters. The Hessian is in shape of 60x60, where the VQE has the possibility to find the lowest eigenvalue for the iterations. Compared to the Levenberg-Marquardt, which updates up to 24 parameters each iteration. We worked with sparse matrix form "csr" in the bundle adjustment problem. Nevertheless, for the quantum optimizer, a list form is needed. Therefore, one has to get from csr to list, which is computationally expensive. The best solution, which was also attempted, was to never create sparse matrices and do the manipulations in classic array form. This is simpler done for the VQE and QAOA as it only needs the Hessian. Compared to the HHL, which has the Hessian and the vector, which in the Levenberg-Marquardt optimizer would be $y - f(\beta)$, where $y$ are the initial values and $f(\hat{\beta})$ are the function results with respect to parameters $\hat{\beta}$

## 4.5 HHL in Bundle Adjustment

Parallel to the VQE, the HHL algorithm was researched to be used within the bundle adjustment problem. The problem is very similar to the VQE. The research between those two implementations went jointly together. The idea behind the HHL in bundle adjustment is that one calculates $X$ in the linear problem $A\vec{X} = \vec{B}$. In the case of Gaussian-Newton algorithm, $A$ is the Hessian and $B$ is the preknown data minus the residual. The resulting vector $X$ is supposed to be the step to update the parameters for the minimization[20].

The issue with the HHL implementation is as follows. The jacobian, where the hessian is created of is of size 60x60 and the gradient is of size 60x1. In the program the distribution of the parameters are in matrices of 20x3x3 for the Jacobian and 20x3 for the gradient. The system already ran once by simply slicing the last matrix entries down to 20x3x1 and 20x1x1. The third matrix dimension appear from the three dimensions x, -y, -z. The issue arises when iterating over the different last entries of the matrix within the HHL. Finally, the placement of the HHL algorithm needs to be adjusted correctly. The idea is to update the cost by the HHL and therefore the steps.

Another remark to the HHL algorithm implementation is that it was attempted to instead of cutting the matrix down, to sum up the 30x3x3 to 60x3 and 60x1 matrices. This increased the computation on an intel i5 dual core processor, so that the jupyter notebook did not execute. This issue arises from i) that the classical computer needs to create fake qubits and execute the program over a simulation of a quantum computer, which needs more computation power than simply executing a classical algorithm, ii) the 60x3 matrix and the 60x1 matrix needs to be reshaped into the shapes of $2^n$, so that the qubits can function properly upon it.

That the shape of a matrix needs $2^n$ input shapes has been discussed in the introduction. This can be clarified further. For processing four states, four bits are the minimum requirement of describing the state, but in a quantum computer only two qubits may be are necessary. Therefore if one has a size of 5 states, then the minimum qubit requirement would lay between two and three qubits. Therefore those states would be expanded to eight states, which requires at least three qubits.

Updating a square cost function by a linear equation solver is possible[20]. Therefore we are optimistic towards the possibility of the successful implementation of the HHL. Nevertheless, for a real world application the HHL is not the best algorithm, as it requires a high amount of qubits to execute with big data amounts.

## 5 Outlook

Firstly, quantum mechanical computations were introduced and a simple example of quantum computing was done. More specifically a special case of the quantum algorithm called "Shor algorithm" was introduced. After presenting the quantum mechanical and computational background, we introduced the VQE into a linear problem to see if it is possible to update steps successfully with the VQE. For the HHL this was not necessary, because the application of the HHL of the aqua Qiskit library is simplified for the solution of a linear system.

After assuring that the VQE is able to update a linear system with two points, the VQE and HHL algorithm were tried to be implemented into the bundle adjustment problem. Due to the lack of time, it was not possible to fully implement neither of the algorithms.

The outlook for further research is, to instead of using VQE to update the steps. It should be easier to use QPE to find all eigenvalues of the entire hessian at once, which does not fullfill the role of finding the lowest eigenvalue for minimizing the cost function, but the minimum eigenvalue is still calculated and additional eigenvalues, which may update further parameters. QPE is only working for Hermitian matrices and therefore we suggest to use the second algorithms decribed by Changpeng in his paper "Computing Eigenvalues of Matrices in a Quantum Computer" [21]. Changpeng introduces more general algorithms to implement different matrices, which do not need to be Hermitian and can have either only real eigenvalues or complex eigenvalues. This could be a possibility as in specific cases we have a symmetric hessian and indications on multiple eigenvalue optimization is given in [22, 23]. The next step would be to analyse if

Quantum algorithms are possible to improve the run time of bundle adjustment, and if not, how to compose them to improve the run time.

Another more ambitious outlook is to implement quantum computations into other computer science problems. Quantum computers are used a lot for physics problems and recently their way into other fields. It can be believed that quantum computing could have the possibility to change parts of science. Especially, the quantum computing in artificial intelligence could allow to process the immense amount of data in an exponential faster time. A main factor is the lacking hardware of quantum computers, which is constantly under improvement.

Quantum computing has potential to revolutionise parts of science, as the speed up is exponential, this could solve the problem of large getting data basis. Quantum computing can improve neural networks and machine learning, which can lead to improvements all over the industry. Nevertheless, one is still far away from producing a proper quantum computer. There are many difficulties with quantum computers, as their coherence error, gate errors, and general noise do not allow any useful results to this day.

## References

[1] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5:4213, 2014.

[2] Eugene F Dumitrescu, Alex J McCaskey, Gaute Hagen, Gustav R Jansen, Titus D Morris, T Papenbrock, Raphael C Pooser, David Jarvis Dean, and Pavel Lougovski. Cloud quantum computing of an atomic nucleus. *Physical review letters*, 120(21):210501, 2018.

[3] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information.* Cambridge Univ. Press, Cambridge, 2000.

[4] John A Cortese and Timothy M Braje. Loading classical data into a quantum computer. *arXiv preprint arXiv:1803.01958*, 2018.

[5] Subhash Kak. A quantum computing future is unlikely, due to random hardware errors, 2019.

[6] Abraham Asfaw, Luciano Bello, Yael Ben-Haim, Sergey Bravyi, Lauren Capelluto, Almudena Carrera Vazquez, Jack Ceroni, Frank Harkins, Jay Gambetta, Shelly Garion, Leron Gil, Salvador De La Puente Gonzalez, David McKay, Zlatko Minev, Paul Nation, Anna Phan, Arthur Rattew, Joachim Schaefer, Javad Shabani, John Smolin, Kristan Temme, Madeleine Tod, and James Wootton. Learn quantum computation using qiskit, 2020.

[7] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. 1999.

[8] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M. Seitz, and Richard Szeliski. Building rome in a day. *Commun. ACM*, 54(10):105–112, October 2011.

[9] Yu Chen, Yisong Chen, and Guoping Wang. Bundle adjustment revisited. *arXiv preprint arXiv:1912.03858*, 2019.

[10] Jesus Rodriguez. Google cirq and the new world of quantum programming, 2018.

[11] Patrick J Coles, Stephan Eidenbenz, Scott Pakin, Adetokunbo Adedoyin, John Ambrosiano, Petr Anisimov, William Casper, Gopinath Chennupati, Carleton Coffrin, Hristo Djidjev, et al. Quantum algorithm implementations for beginners. *arXiv preprint arXiv:1804.03719*, 2018.

[12] Danial Dervovic, Mark Herbster, Peter Mountney, Simone Severini, Naïri Usher, and Leonard Wossnig. Quantum linear systems algorithms: a primer. *arXiv preprint arXiv:1802.08227*, 2018.

[13] Alexei Yu Kitaev, Alexander Shen, Mikhail N Vyalyi, and Mikhail N Vyalyi. *Classical and quantum computation*. Number 47. American Mathematical Soc., 2002.

[14] Dominic W Berry, Graeme Ahokas, Richard Cleve, and Barry C Sanders. Efficient quantum algorithms for simulating sparse hamiltonians. *Communications in Mathematical Physics*, 270(2):359–371, 2007.

[15] Harold Kushner and G George Yin. *Stochastic approximation and recursive algorithms and applications*, volume 35. Springer Science & Business Media, 2003.

[16] Andrew R Conn, Katya Scheinberg, and Ph L Toint. On the convergence of derivative-free methods for unconstrained optimization. *Approximation theory and optimization: tributes to MJD Powell*, 1997.

[17] E Tielo-Cuautle and Alejandro Diaz-Sanchez. An heuristic circuit-generation technique for the design-automation of analog circuits. 1, 2003.

[18] Thomas Monz, Daniel Nigg, Esteban A Martinez, Matthias F Brandl, Philipp Schindler, Richard Rines, Shannon X Wang, Isaac L Chuang, and Rainer Blatt. Realization of a scalable shor algorithm. *Science*, 351(6277):1068–1070, 2016.

[19] Chao-Yang Lu, Daniel E Browne, Tao Yang, and Jian-Wei Pan. Demonstration of a compiled version of shor's quantum factoring algorithm using photonic qubits. *Physical Review Letters*, 99(25), 2007.

[20] Anna-Karin Tornberg. Linear algebra, part 2 eigenvalues, eigenvectors and least squares solutions, 2013.

[21] Changpeng Shao. Computing eigenvalues of matrices in a quantum computer. *arXiv preprint arXiv:1912.08015*, 2019.

[22] Adrian S Lewis. The mathematics of eigenvalue optimization. *Mathematical Programming*, 97(1-2):155–176, 2003.

[23] Bindel Spring. Notes for 2016-04-27, 2016.

## 6 Appendix

### 6.1 Linear Problem

```
x = np.array([0,1,2,3], dtype = float)
y = np.array([0,1,3,4],dtype = float)
slope = 0.1
i = 1
tol = 0.005
while i < 50:
    def ver(x,y,a):
        return(y-a*x)**2/50
    spaceholder = jacobian(ver)
    jaci =spaceholder(x,y,slope)
    qubitOp = MatrixOperator(jaci)
    backend = Aer.get_backend('statevector_simulator')
    var_form = RYRZ(qubitOp.num_qubits)
    optim = COBYLA()
    vqe = VQE(qubitOp,var_form,optim)
    result_vqe = vqe.run(backend)
    print(result_vqe)
    slope -= result_vqe['energy']
    if -tol < result_vqe['energy'] < tol:
        break
    print(slope)
y_pred = slope*x
plt.xlabel("x")
plt.ylabel("y")
plt.plot(x,y_pred, "r")
plt.scatter(x,y,marker = "o")
```

The code of the linear problem is given above. First the x, y values and the slope are initialized. Then a while loop is created with maximum of 50 iterations. In the while loop a function called "ver" is introduced, this function determines the distance of the slope too the point and tries to minimize it. Then the jacobian of this function is taken and afterwards the eigenvalues with the VQE of the jacobian are computed and added to the slope. Finally if the step size is below "tol" it breaks the loop and the graph with the slope and the two points are printed.