Constructing Quantum Gates Using Optimization Techniques

Diedrik Leijenaar Oksnes

Thesis submitted for the degree of
Master in Applied Computer and Information Technology (ACIT)
- Phase III
60 credits

Department of Computer Science
Faculty of Technology, Art and Design

OSLO METROPOLITAN UNIVERSITY

Autumn 2023

Constructing Quantum Gates Using Optimization Techniques

Diedrik Leijenaar Oksnes

Constructing Quantum Gates Using Optimization Techniques

http://www.oslomet.no/

Printed: Oslo Metropolitan University

Abstract

Quantum computers show great promise, but current implementation struggles with decoherence. In this project, I construct a set of universal qubit gates for one qubit using a combination of dynamic magnetic fields and one static magnetic field, the model I use has one or two qubits and is symmetric when swapping the qubits. Moreover, I try to mitigate noise including a jump operator for amplitude damping for one qubit, jump operators found with tomography and a randomly generated jump operator for two using the Lindblad equation. Meanwhile, the mitigation of noise was minimal but showed a clear trend of what the best parameters were.

Acknowledgments

I would like to thank my supervisors Sølve Selstø and Kristian Wold and others like Sergiy Denysov, my fellow student Sebastian Testanière Overskott and my family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, I will start with the problem statement and some of the motivations for this study. With more background in what quantum computing is and some basic ideas in the next chapter.

1.1 Problem Statement

This project aims to study the effect dynamic magnetic fields have on the spin of spin $\frac{1}{2}$ electrons, the result of the study can also apply to other two level trapped quantum particles. The goal of the study is to see if the use of dynamic magnetic fields is able to improve the fidelity of quantum computers.
With this study I hope to construct several different gates with a focus on one qubit gates including the Pauli gates, single qubit gates acting on two particles and two qubit gates including the $SWAP$ gate. To construct the quantum gates I use the process of minimizing a selected cost function that optimizes the selected parameters. A cost function is a way to measure how close something is to a target, and an optimization to minimise this is the objective of this project. The simulation in this project takes into account things like the time spent manipulating spin and difference from the target gate, by varying the strength and direction of a magnetic field to manipulate the spin of an electron, or even overlapping magnetic fields used to manipulate the spin in different directions.
Having a cost function based on how different the effect of the magnetic fields on the spin $\frac{1}{2}$ particle is from a specific gate using different measurement techniques, depends on how the simulation is being done and what parameters are to be optimized. The landscape generated by the cost function when varying the parameters in a noise free case, also gives interesting information about how hard it is to get a gate that is resistant to decoherence and noise. This information can be used to expand the simulation to take into account realistic expectations of the noise

and decoherence current quantum computers are expected to be subject to. The goal from there is to see if it is possible to reduce the noise generated when manipulating quantum particles.
So in summary the goals of this project are:

- To find out what quantum gates can be constructed using a mixture of dynamic and static magnetic fields
- To find optimal parameters that create common gates
- To simulate exposure to noise and see if it is possible to optimize the parameters to lessen the effect

Part of the goal of this study is to increase my understanding of quantum mechanics. It has only been a minor part of the courses in my study, and to acquire a deeper understanding, I needed to study the relevant parts of quantum mechanics.

1.2 Motivation

Quantum computers have several theoretical applications where they will vastly outperform normal computers (E. K. Nielsen, 2011). E.K. Nielsen describes how in the 1980s it was found that a well controlled quantum mechanical system quantum computer could simulate quantum systems by Benioff (1980) and Feynman, 1982 (E. K. Nielsen, 2011). Since it is hard to have a perfect quantum computer, they are subject to interference from noise from the environment and decoherence as information is lost and the quantum state is destroyed leading to a wrong result when we try to measure the qubit later. Thus current computers are noisy intermediate-scale quantum (NISQ) computers that do not have the overwhelming advantage quantum computers are predicted to have when the fault tolerance is complete (Preskill, 2018). Later in 1990s, it was that several functions widely used in cryptography could be broken by using a quantum computer, and Shor (1994) created Shor's algorithm to speed up factorization of integers that would break many of currently used cryptography algorithms (M. Nielsen & Chuang, 2010). Meanwhile, L. Grover made Grover's algorithm that could speed up search, while not quite as big a speed upgrade as Shor's algorithm, searches are so prevalent that it would, if implemented, have large effects on many systems. Shor's and Grover's algorithms can, among other algorithms that could take advantage of the parallelism quantum computing offer, lead to large reductions in the time needed to do computations on many problems that take a long time on classical computers.
Quantum parallelism makes the quantum computer able to do many calculations in parallel for a function using the same qubits, and while the calculations are being

done the the qubits will interfere with each other. Using this property makes you able to find some global property for function. This makes you able to find properties for functions valid for all possible input states (M. Nielsen & Chuang, 2010). There are still questions of what kind of global states can be found and how to extract them, but for fields of computation where the parallelism shines, it can have an exponentially lower time to compute.

While quantum computers can simulate ordinary computers and do everything they can do, it is not expected that a quantum computer will take over the job of classical computers in all fields. The difficulty of creating quantum computers with many qubits means that they can do fewer individual computations that cannot benefit from parallelism. Since there are many computations that cannot be done using parallelism, where we are only after one ideal extracted from one input, these computations will have to be done on classical computers even in the future. The end result is expected to be a computer that has en extra module to do the quantum computations only for those fields where it can offer an advantage (Johansson, 2022).

But a large limitation is that current quantum computers are often quite noisy and even with noise correction, the progress of which was started by Shor and A. Steane (E. K. Nielsen, 2011), there are limitations on how much manipulation of quantum bits can be done before the noise generated lead to decoherence. Even with the shielded environments and extreme cooling used, current quantum computers still suffer from decoherence and interference from all kinds of environmental factors, like ionizing radiation and scattering. So the chance of noise and the resulting decoherence increase with time spent in quantum states.

There are several fields within quantum computing where there is limited knowledge of how quantum manipulation can be done. This project explores if it is possible to reduce coherence loss in quantum computing by constructing gates in ways that reduce the interference on the quantum system, by limiting the interaction with the system as much as possible and trying to counteract the coherence loss.

This study focuses on the time dependency of the magnetic fields, as most manipulation of qubits in the physical world is not quite as neat as modelled without time dependence in the Schrodinger equation. My model of a quantum system introduces dynamics in the quantum system, by varying some of the magnetic fields in strength with time. It is interesting to see the effect this has on how gates can be constructed. Another example of dynamics in quantum systems is that the magnetic pulses generated in quantum computers are not completely discrete, either full power or no power piecewise continuous pulses in the physical world. But a pulse varies with time, taking some time to ramp up to full strength and some time to turn off (Personal communication, S. Selstø, 2022). By varying the magnetic pulse strength I hope we will see how gates can be constructed, and create the effect of a gate on the

state that we desire after acting on the qubit with magnetic pulses for a set time. The simulations with the time dependent Schrödinger equation and the results from the cost function have shown that the landscape created has clearly defined minima, but those minima are in valleys that can be harder for gradient descent to follow. There are also a few points where there can be local minima that the gradient descent algorithm can become trapped in. This leads to the situation where a poor starting position can make the algorithm end up trapped and get a poorer result than desired. This will also influence my results as I will describe in Sec. 5.1.2.

Chapter 2

Background

In this chapter, I will give a light introduction to some parts of quantum mechanics, mostly the ways used to simulate the behaviour of particles. Following with some general knowledge of quantum computing and some common quantum gates.

2.1 Quantum mechanics

In the field of quantum mechanics, there know that in many situations particles can only be in certain states. Electrons in an atom can only be in a finite number of possible energy levels, or have a quantized number of possible states, therefore the name quantum mechanics.
Quantum mechanics where first constructed in early 1900 with most studies after 1920 (M. Nielsen & Chuang, 2010). In 1925 Samuel Goudsmit and George Uhlenbeck discovered that electrons had an intrinsic angular momentum or in other words a spin of $\frac{\hbar}{2}$ (Karlsson, 2022). Wolfgang Pauli came around to this idea, after first criticizing it, he went on to create the famous Pauli Principle:

$$\psi(r_1, r_2) = \pm\psi(r_2, r_1). \tag{2.1}$$

This formula states that for identical particles there is an exchange symmetry. For two particles that changes place the wave function is the same except for a $\pm$ sign in front of the wave function. It is anti symmetric (or $-$) for fermions and symmetric (or $+$) for bosons (Griffiths, 1995), where fermions are particles with half integer spin, and bosons have integer spins.
The Stern–Gerlach experiment, as mentioned in Griffiths (1995, pp. 174–175), is considered a prototype to the methods used to measure and prepare quantum particles by getting a definite state. Using a quantum computer you are dependent on actually knowing the initial state. The experiment is made by having a particle with spin moving in an inhomogeneous magnetic field, such as two magnets spaced

apart, where it will deflect either up or down with different strength dependent on it's angular momentum. Which for electrons that can only be in two states means it will deflect into two different directions dependent on it's state. And no matter how many different electrons are sent into the magnetic field it will always be deflected either up or down with the same strength.

2.1.1 Hamiltonian

The Hamiltonian is the energy operator in a quantum system and is often written as $\hat{H}$. It contains information about all interactions acting on the system described by the Schrödinger equation. If it does not interact with an external system, it will simply contain the sum of the kinetic and potential energy (Griffiths, 1995):

$$\hat{H} = -\frac{\hbar^2}{2m}\nabla^2 + V \tag{2.2}$$

V is the potential energy and $\frac{\hbar^2}{2m}\nabla^2$ is the kinetic energy. It contains two constants, one is $\hbar$ is the reduced Plancks constant and comes from $\frac{h}{2\pi}$ where h is the Plancks constant. The other is m the mass, where the mass m depends on what kind of particle you are working with, as the mass varies greatly. You can, by changing what unit of measurement is used in the equation, set these constants to 1, as I do in my implementation throughout the project.
For quantum systems in three dimension the ∇^2 operator decomposes to:

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \tag{2.3}$$

The Hamiltonian is used in equations that describe how a quantum system evolves showing the effect of the energies interacting with the quantum system. Examples of such equations are the Schrödinger equation and the Lindblad equation used in this project, where I use a variant of the Schrödinger equation where the Hamiltonian only describes the spin of particles. It does not contain the kinetic energy, as I focus on particles that are fixed in space.

2.1.2 The Schrödinger equation

The Schrödinger equation was formulated by Erwin Schrödinger in 1926 and described in it's original form the behaviour of a non relativistic spin less particle. The equation was later generalized by Wolfgang Pauli in 1927 to include the spin of non relativistic particles ('Schrödinger equation', 2011). It has received later generalizations to be able to apply it to relativistic particles. But as this project does not concern itself with relativistic particles, they are not relevant here.

The Schrödinger equation can be used to find the probability of a particle being in a place, the probability of how fast it moves or how likely it is to be in a certain spin state. By solving this differential equation you get a wave function that describes the state of the particle (Griffiths, 1995). This differential equation is easy to solve numerically for a simplified system with one particle, with the spin as the only degree of freedom, such as the ones I use for single qubit gates.

$$i\hbar\frac{d}{dt}\Psi(t) = \hat{H}\Psi(t) \tag{2.4}$$

In Eq. (2.4), the Schrödinger equation, Ψ is the state of the system and can represent many things, including the probability of a particle being in a particular state.
The equation is the most general version of the Schrödinger equation and by changing what the energy operator $\hat{H}$ from Eq. (2.2) is in the Schrödinger equation Eq. (2.4) and what the state Ψ represents, you can change what kind of system is being represented. $\hat{H}$ is the Hamiltonian and represents the effects influencing how the state Ψ change with time, this is the energy operator and it describes all the energies in the system.
The Schrödinger equation Eq. (2.4) is a partial differential equation that can be reasonable to be numerically solved for small systems, sometimes it's even possible to solve it analytically. But as soon as the system it describes gets bigger it reaches a level where it is currently impossible to numerically calculate it.

Position Schrödinger wave function

Depending on the nature of the Hamiltonian, the Schrödinger equation has many possible forms. For a one dimensional particle that can move in space it can be:

$$i\hbar\frac{\partial}{\partial t}\Psi = -\frac{\hbar^2}{2m}\frac{\partial^2\Psi}{\partial x^2} + V\Psi \tag{2.5}$$

The energies in the Hamiltonian $\hat{H}$, are here the one dimensional spatial case of Eq. (2.2). The Hamiltonian acts on Ψ the wave function, to give the Schrödinger Eq. (2.5). This variant of the Schrödinger equation gives the position for a particle in one dimension in space, the position on a line. To get the position in three dimension, the ∇^2 in Eq. (2.2) would be Eq. (2.3) .
For clarity you may for Eq. (2.5) include the position x and the time t, so that it is easy to see that the one dimensional wave function $\Psi = \Psi(x,t)$ is only dependent on position and time.
When you take the square of the absolute value of the wave function as in the following equation:

$$P(x,t) = |\Psi(x,t)|^2, \tag{2.6}$$

it gives you the probability density for the possible positions, and it describes the probability to be in a specific position the different wave functions describes. If you do a measurement to determine the state of the particle, we say that the wave function collapses as the state is measured. The wave function will in that case drastically change as the act of measuring a quantum particle forces it to be in specific state, whereas before measurement the particle was in a quantum state where it could be in different positions with a probability described by the wave function. So the measurement forced the particle to be in the particular state it was measured in, reducing all other probabilities to zero (Griffiths, 1995). This effect is called the collapse of the wave function.
As $\Psi(x,t)$ is an solution of the Schrödinger equation, it can be said that $A\Psi(x,t)$ is also a solution, for any complex A. The sum of the values of all possible states in the wave is required to be 1, it must also be 1 for the integral in a continuous system as seen in the following equation:

$$\int_{-\infty}^{+\infty} |\Psi(x,t)|^2 dx = 1 \tag{2.7}$$

As the wave function is a probability density describing the possible states of the system, the particle have to be somewhere. So we must pick an A that normalizes the value to 1. For equations where it is not possible to create an multiplicative factor that can set it to 1, as the value is either 0 or ∞, the states cannot represent particles and must be rejected as invalid (Griffiths, 1995). This normalization is a requirement for all types of Schrödinger equations, and for discrete systems it is the sum of all possible states that have to be 1.

2.1.3 Closed quantum systems

So far all the quantum systems described in this chapter have been closed quantum systems. Closed quantum systems are used to describe all systems where we can sufficiently isolate the systems from the environment, so that we no longer need to concern ourselves with how interference from the environment acts on the smaller system we study. This means that it's enough to use a unitary operator given by the Hamiltonian to find the end state of the system:

$$\Psi(t) = U(t)\Psi_0, \tag{2.8}$$

enabling a simple transformation of states. Especially when the Hamiltonian the unitary is derived from is not time dependent, they are the easiest types of quantum systems to model.

But for some quantum systems that we cannot sufficiently isolate, like with quantum computing, it is not sufficient to describe the behaviour of the quantum system. Because you have to sufficiently isolate a quantum system, and must know everything about the system and every interaction by the energies involved to be able to treat a quantum system as closed. For quantum computers, this is an ideal that does not exist in the physical world. But using closed quantum systems shows how a quantum computer behaves in an ideal world.

To find if you can treat a quantum system as closed, you have to look at the small quantum system you are interested in, and then see if you can isolate it sufficiently from the environment. So that you no longer need to take into account interactions with energies outside the smaller quantum system.

In this project, I only use one or two qubit quantum systems, based on spin, but I use both closed and open systems to describe and simulate them.

2.1.4 Open quantum systems

You get an open quantum system when you want to study a smaller quantum system, but are unable to sufficiently isolate it from its environment.

Since is very difficult to completely isolate a quantum computer and other sensitive quantum systems, you must take the energies from the environment and their interaction with the quantum system into account and model them. These models are simplifications. While a model of a closed system that describes every energy and its interaction with all particles would give a perfectly accurate model, there are simply too many interactions to take into account all possible interactions when talking about quantum computers. So describing the quantum system as being open enables modelling the interactions needed to accurately reflect reality.

Open quantum systems are often modelled as being smaller quantum systems in a bath that represents the environment (Breuer, Petruccione et al., 2002).

As seen in the Fig. 2.1 the quantum system represented by H_{sys} is being influenced by its environment H_{env} with the interaction represented by H_{int}. We model the simplified open system to try to find an equivalent solution as if we had modelled the whole system.

A density matrix such as ρ, is a way to describe the state of a quantum system in a more general way than the wave function. It is always $\rho \geq 0$ while $Tr(\rho) = 1$ described in more detail in Sec. 3.4.

The figure below is inspired by Breuer, Petruccione et al. (2002) and gives an overview of how the use of open quantum systems is viewed:

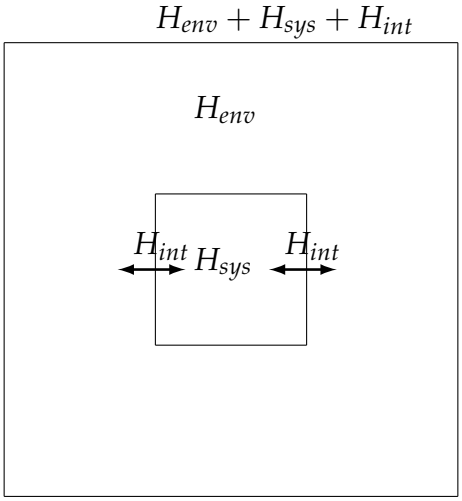

Figure 2.1: Model of open quantum system, using a smaller quantum system H_{sys} submerged in a larger quantum bath H_{env}, with the arrows indicating interactions H_{int}

$$\begin{array}{ccc} \rho(0) = \rho_{sys} \otimes \rho_{env} & \xrightarrow{unitary\ evolution} & \rho(t) = U(t,0)\left[\rho_{sys} \otimes \rho_{env}\right] U^{\dagger}(t,0) \\ \downarrow tr_{env} & & \downarrow tr_{env} \\ \rho_{sys}(0) & \xrightarrow{dynamic\ map} & \rho_{sys}(t) = V(t)[\rho_{sys}(0)] \end{array} \tag{2.9}$$

As you can see the idea is to trace out the environment ρ_{env} from the starting state $\rho(0)$, getting a simplified system $\rho_{sys}(0)$ and create a quantum map that will reach the same end state $\rho_{sys}(t)$ as if we had had a closed quantum system and only traced out the environment at the end. Finding a quantum map V that enables this is the goal of using theory for open quantum systems. This enables the modelling of quantum systems even without modelling the whole environment, as the number of variables are too high to account for them all.
The equations that do this are called master equations. Even though there exist several different master equations, in this project I only use the Lindblad Master equation. In Sec. 2.1.5 I will explain why I chose this equation. It can be used on any number of particles, but in my project, I used it to simulate the evolution of two qubit states when acted upon by magnetic fields while being exposed to noise.

No hiding theorem

When talking about open quantum systems we often talk about how the information is lost because of decoherence. But it's more accurate to say that the information

dissipates to the environment. This is called the no-hiding theorem, and was first published in 2007 (Braunstein & Pati, 2007). The no hiding theorem has massive implications for black holes and when there is information missing after quantum systems decohere. There have even been experiments done to prove this, among others, there was in 2011 an experiment to prove that the information is not lost, but could instead be found in the environment as described in (Samal et al., 2011).

2.1.5 Gorini–Kossakowski–Sudarshan–Lindblad master equation

The Gorini–Kossakowski–Sudarshan–Lindblad equation, mostly called the Lindblad equation, was published in 1976 by two different groups. (Lindblad, 1976) was published by Göran Lindblad and (Gorini et al., 1976) was published by a group consisting of Vittorio Gorini, Andrzej Kossakowski and George Sudarshanin (Chruściński & Pascazio, 2017). As they submitted their papers at the same time and the publishing date only differed by months they can both be considered the origin of the equation.
As the Schrödinger equation was not sufficient to describe open quantum systems these two groups created a method to model influence from the environment (Gorini et al., 1976; Lindblad, 1976). This master equation enabled a non unitary evolution where information dissipates to the environment. It is a way to model open quantum systems, and is the main equation used in this project.
The Lindblad master equation was chosen as the master equation to use with the open quantum systems, because it gives a completely positive trace preserving map (Gorini et al., 1976; Lindblad, 1976). When this map is applied to a state it will ensure that all possible states are positive semi definite: The probability for each state becomes $p \geq 0$, ensuring there is no possibility to have a state with negative probability. And the trace of the state stays at 1, ensuring that the state exists similarly to the normalization in Eq. (2.7) for the Schrödinger equation. This makes the end state reached by using the quantum maps generated by the Lindblad master equation always physically possible states as long at it starts with a physically possible state. I will come back to this with more detail in section 3.6.

2.2 Quantum computing

Nielsen and Chuang states how: "Quantum computation emerged in the 1980s when P. Benioff and R. Feynman realized that the apparent exponential complexity in simulating quantum physics could be overcome by using a sufficiently well controlled quantum mechanical system to perform a simulation" (E. K. Nielsen, 2011).

Quantum systems suffer from the "curse and blessing of dimensionality". The curse is especially prevalent when exploring such systems using classical computing. The complexity of computation increases exponentially as the number of values needed to represent a quantum state increases exponentially with the number of particles used, compounding this with the increased computational power required for every additional parameter. This quickly limits how many particles it is possible to use, and how many parameters you can optimize for. This consequently limits you to simulating and exploring quantum systems with few particles when using classical computing.
This is completely reversed when using a quantum computer. You can see the blessing of dimensionality in the fact that how many particles you can simulate, scales exponentially with how many qubits the quantum computer has. This is why there is such interest in finding ways to get error tolerant quantum computers.

2.2.1 Qubits

A qubit is the quantum computing equivalent to a bit in normal computing. The major problem with manipulating a qubit comes from the fact that it can suffer from decoherence and no longer be in a state where it gives correct information when measured. This comes from interference from the environment, and is a major limitation that reduces how useful current quantum computers are. Many studies in quantum computing are done to see what can be done to limit how fast a qubit decoheres, which is why many quantum computers are operated in as low temperature and perfect vacuum as possible to reduce how much they are affected by their environment.
A qubit can be seen as something that is in a state where it has a certain probability to be 0 and a certain probability to be 1 as in the equation:

$$\psi = \alpha \left|0\right\rangle + \beta \left|1\right\rangle = \alpha \left|\uparrow\right\rangle + \beta \left|\downarrow\right\rangle . \tag{2.10}$$

The sum of these probabilities is always 1 as the qubit will always have a state when measured as long as it exists (M. Nielsen & Chuang, 2010). When talking about spin systems we represent the state of spin up of a particle as $\left|\uparrow\right\rangle$ with the state $\left|0\right\rangle$ and spin up $\left|\downarrow\right\rangle$ with the state $\left|1\right\rangle$.
"Functional and scalable qubits must meet well-defined criteria. First, reliably initializing each qubit into one of its two levels must be possible. Second, the final state of each qubit must be knowable by a projective measurement that gives the correct answer with high probability. Third, qubit manipulation must be implementable using high-quality single- and two-qubit gates." (Vandersypen & Eriksson, 2019, page 39)

There are several different ways to construct qubits, the most famous are using superconducting materials and constructed by IBM Dial (2022) and Google. In this study we will see the effects of magnetic fields on the $1/2$ spin of a fermion, where the state of the qubit is set using the direction of the spin of the fermion. Quantum dots is one of the possible ways to implement a two state spin as a qubit, and can be constructed with silicon semiconductors to have a great potential to scale if the challenges with noise from imperfections in it's construction can be dealt with (Vandersypen & Eriksson, 2019). But it is also possible to create qubits from ion traps or by trapping a particle in an optical lattice as proposed in Deutsch et al. (2000). Another possibility is to create qubits without spin, for example by using the two lowest states of a quantum dot or having a double quantum dot and using being in a left or right state.

State vector

Instead of describing each state separately as in Eq. (2.10), one can also use a state represent qubits, by setting each value into a vector and where each value represent the possibility to be in a specific state. For example for a one spin particle you have two states up and down:

$$|\uparrow\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad , \quad |\downarrow\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.11a,b}$$

That when using $|\psi\rangle$ to represent the state can be written as:

$$|\psi\rangle = \begin{bmatrix} a \\ b \end{bmatrix} \tag{2.12}$$

Where $a = \alpha\,|\uparrow\rangle$ and $b = \beta\,|\downarrow\rangle$ while the square of the value represents the probability to be in that specific state. A requirement for this is that the sum of the square of values is one, so that there always exist a state:

$$||\Psi\rangle|^2 = \sum_n |C_n\,|\phi_n\rangle|^2 = 1 \tag{2.13}$$

And more specifically for a single qubit we have:

$$||\psi\rangle|^2 = \left| \begin{bmatrix} a \\ b \end{bmatrix} \right|^2 = |a|^2 + |b|^2 = 1 \tag{2.14}$$

This will be the same when you have four states, where each of the four possible states are set as one of the values in a vector.

For example in a two particle spin system, you can have four possible spin states: up up, up down, down up, and down down:

$$|\uparrow\uparrow\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad , \quad |\uparrow\downarrow\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{2.15a,b}$$

$$|\downarrow\uparrow\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad , \quad |\downarrow\downarrow\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.15c,d}$$

When these states are manipulated you will almost never end up in a situation where any of the single values reach a 100% chance of occurring by having a value of 1 for one state, while all other states are 0, but it can for some operations be an ideal to strive for.
When using the state vector $|\psi\rangle$ to represent the state like this:

$$\psi = \alpha\,|00\rangle + \beta\,|01\rangle + \gamma\,|10\rangle + \delta\,|11\rangle = \alpha\,|\uparrow\uparrow\rangle + \beta\,|\uparrow\downarrow\rangle + \gamma\,|\downarrow\uparrow\rangle + \delta\,|\downarrow\downarrow\rangle \tag{2.16}$$

$$|\psi\rangle = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}, \tag{2.17}$$

the square of the absolute values represent the probability for the qubit to be in that state. To that end for all physical situations where the particles still exists in the described quantum system the sum of the squares of the absolute value of all possible states must be 1 as in Eq. (2.18). Otherwise ψ no longer represents a physical situation that can exist.

$$|a|^2 + |b|^2 + |c|^2 + |d|^2 = 1 \tag{2.18}$$

2.2.2 Gates

While a regular computer has logical gates in its CPU to manipulate a bit between its two states, a quantum computer has quantum gates to manipulate qubits, so that they can be changed into any possible state. A common way to describe a one qubit gate is by describing them as rotations on a Bloch sphere as seen in Fig. 2.2, where the different gates are a way to move the point at $\vec{a}$ to a different point on the sphere.

Unlike the gates in a regular computer, the quantum gates are reversible and there will always exist a combination of gates that can get you back to the previous state. This counts for an ideal quantum computer, where decoherence is not considered.

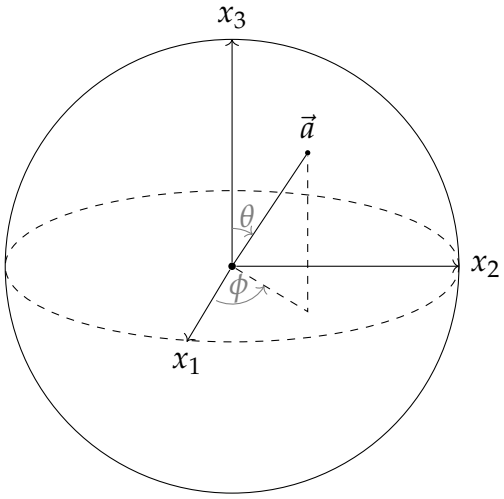

Figure 2.2: Example of a Bloch sphere by Janosh Riebesell(Riebesell, 2022)

I will in the following part I talk about different types of quantum gates, in the next sub sections I will describe these gates further.

In an earlier smaller study it was concluded to be feasible to construct Pauli gates and the Hadamard gate using magnetic fields (Papanikolaou, 2021). This study aims to go further than that work to see if there are ways to create multi qubit gates and create all Clifford gates. From there I will look at the feasibility to make high fidelity gates with a very low amount of noise. Mostly by seeing if there are ways to reduce the time spent subjecting the qubits to the magnetic fields, as a shorter time period and less interaction will reduce the amount of noise generated.

The Clifford gates consists of the CNOT gate, Hadamard gate and S gate (phase shift gate). The Pauli gates are also part of this group, as they can trivially be constructed with these gates (M. Nielsen & Chuang, 2010). When we include the T gate that is not part of the gates in the Clifford group, we get a universal set of gates, such that it can approximate any gate that can be used on the qubit by only using a combination of the gates in the universal set.

While it is possible to simulate gates efficiently using classical computing as described in the Gottesman–Knill theorem (Aaronson & Gottesman, 2004; M. A. Nielsen, 2002), this only hold for as long as we do not try to simulate a full universal set of gates. The gates in the Clifford group alone can be simulated on a timescale of $O(n^2)$ while used alone.

Phase P and S gate

The S gate is

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \tag{2.19}$$

The phase shift gate P in Eq. (2.20) is part of a universal set of gates when combined with the Clifford group. But it is not a part of the Clifford set like the S gate is.

$$P = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{bmatrix} \tag{2.20}$$

Pauli gates

Pauli matrices have the effect that if two of the same Pauli matrices are acting on each other, the effect of the two gates will become I the identity matrix $X^2 = Y^2 = Z^2 = I$. These gates are anti commutative as $ZX = iY = -XZ$.

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{2.21a}$$

$$\sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{2.21b}$$

$$\sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{2.21c}$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.22a}$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{2.22b}$$

One example of the Pauli gates is the X gate also called NOT gate shown in Eq. (2.21a). When the X gate is applied to a qubit, it will change it's state to it's opposite. If it is in the $|0\rangle$ state it will change it to the $|1\rangle$ state and vice versa.

SWAP gate

Swap as described is a gate that swaps the position of two qubits. As seen in equation here:

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.23}$$

When looking at it's effect we can see that the swap gate only has effect when it can swap the position of the two qubits in different states. There is no observable effect when the qubits is in the same state as seen in the following equation:

$$[SWAP] \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad [SWAP] \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{2.24a,b}$$

$$[SWAP] \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad [SWAP] \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.24c,d}$$

CNOT gate

The $CNOT$ gate, or Controlled NOT gate, works on two qubits, and will if the first qubit is in the state $|\downarrow\rangle$ swap the state of the second qubit, having the same effect as the x Pauli gate in Eq. (2.21a).

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.25}$$

Hadamard gate

The Hadamard gate is a gate that can be used to set the qubit in a superposition from a defined starting position such as $|0\rangle$ or $|1\rangle$. When used on the state $|0\rangle$, you will get the $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ state. The equation for the Hadamard gate is:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.26}$$

The goal of this study is to construct the different gates using dynamic magnetic fields that vary with time.

2.2.3 Noisy Intermediate Scale Quantum

All current quantum computers are Noisy Intermediate-Scale Quantum (NISQ) and suffers from a large error rate, limiting what current quantum computers can be used for (Preskill, 2018). But they can be used to explore the effect of entangled particles even while the noise limits other things it can be used for. While error free quantum computing has the potential to achieve quantum supremacy and outperform classical computers, there are fields where even noisy quantum computers are expected to vastly exceed classical computers, such as the field of quantum annealing (Finnila et al., 1994), and they enable us to simulate all natural processes. And other fields where the advantage is more limited like Non-Polynomial(NP) hard problems, where it is only expected to be a modest increase in the computation speed (Preskill, 2018).
For Polynomial(P) time problems there exists a polynomial of the form $O = (n^k)$ for any n with a constant k that is greater than the time taken to solve the problem of complexity given by n. While a NP hard problem means that it is not possible to solve the problem deterministic in polynomial time. This means that there does not exist a polynomial of the form $O = (n^k)$ for any n with a constant k that is greater than the time taken to solve the problem for a problem of complexity n. Unless $P = NP$ which is doubted to be true, but currently not proven (Cook, 2001).
A famous example is the travelling salesman problem, an NP-Complete problem, where the complete part of "NP complete" shows it is a subclass of NP problems. If an answer exists, it can be proven quickly. Here the problem is the travel time between cities and the shortest route. While such a thing can be brute forced for a small number of cities and a limited number of roads between them. It quickly becomes impractical to check all solutions for even with a limited number of cities. The time to compute increases exponentially as the number of cities increases. It is possible to approximate the solution, but it will not give an optimal answer.
Part of the problem with noise comes from the difficulty in perfectly isolating the qubit while still allowing it to interact with other qubits to allow entanglement. Perfect qubits are only available in a vacuum while qubits in the real world will experience interference from the environment and even the construction of the physical qubit can have imperfections that introduce errors (Vandersypen & Eriksson, 2019).

2.2.4 Implementation of quantum system

Any quantum system that can be set into a quantified two state, and reliably measured, can be thought of as a spin up and spin down system. This enables the use of several different types of quantum systems being used to construct qubits in

quantum computers.

Electron as Quantum dot

According to Vandersypen and Eriksson: "Most textbook about spin systems focus on electrons because The corresponding states, spin up spin down, form the prototypical quantum bit (qubit), and rotations of the spin state constitute the simplest quantum logic gates." (Vandersypen & Eriksson, 2019) By trapping an electron with voltage you can create a quantum dot that can be used as the basis for a qubit. But these electrons exist in a noisy environment that makes it too vulnerable to noise when used alone. A study by David Weiss and Mark Saffman made in 2017 suggests that with current techniques a fully fault tolerant quantum bit, a logical quantum bit, might require as much as 1000 NISQ bits to perform error correction (Vandersypen & Eriksson, 2019, as cited in). They suggest that it will be a requirement to have millions of qubits to reach quantum supremacy.
But they propose that using silicon semiconductors can take advantage of current knowledge in classical computing hardware to create a large number of quantum dots and they speculate on the possibility to create chips that combine classical computing and quantum computing on a single chip or circuit board.

Semiconductor quantum dot

New research has demonstrated over a 99% fidelity rate for silicon semiconductors, and opening the way for using common error correction algorithms to correct for the noise and resulting decoherence (Mills et al., 2022). These quantum dots are currently in the 100 nm size and are theorized to allow for further scaling in quantum computers in the future. With speed higher than trapped ions, a fidelity close to a superconductive quantum computer, together with a recent increase in the capability to construct quantum dots, they state that it will be possible to create large multi qubit processors.

2.3 Quantum control landscapes

A quantum control landscape is a landscape construed by a cost function. The cost function is a way to measure how close something is to a target. In this project I simulate how close the effect of a Hamiltonian is to an ideal gate. In the cost landscape the optimal solutions are the points where the cost function has either minimal or maximum value dependent on the configuration of the cost function. Looking at such a landscape can tell you much about how stable the different

solutions are, but they are often costly to compute, as you have to compute for every point in the landscape.
An important part of finding an optimal way of manipulation quantum dots are finding maxima or minima in a quantum control landscape (A. N. Pechen & Tannor, 2011). Most methods to find the maxima use an evolutionary approach and the gradient, as I do, another possibility is to use the first or second order Hessian matrix to find a better point, but it is not used in this study and therefore not described. When using an evolutionary approach, you repeat your method until a local critical point is found. This is done because as the systems become complex with more variables, the computing power needed becomes too high for a more throughout search. This need for search algorithms makes it vulnerable to traps, as local minima or maxima can appear when the Jacobian is not full rank at all points. It has been shown that there exist traps in many types of quantum problems, this has a large impact on a large class of studies on quantum control systems.
There are problems with proving if a system has traps or is trap free (A. Pechen & Il'in, 2012). There has been conflicting research on whether quantum systems are trap free or not while using numerical simulations. The question whether these simulations are running over all possible quantum control possibilities limits the value, and it remains hard to prove if a specific system is trap free or not.
It has been proven that there exist Landau-Zener systems that are possible to manipulate, that are trap free. This is only possible when the physical system is not subject to large disturbances and the laser pulse used to control the system are among the less energetic pulses from the best pulses possible (A. Pechen & Il'in, 2012). But the Landau-Zener systems are vulnerable to noise and decoherence, which can lead to the creation of traps that makes an evolutionary algorithm find local maxima instead of global.
There is research that suggests that in many systems you will in a system that is completely noise free and with no decoherence be able to find local maxima that correspond to the global maxima (Rabitz et al., 2004). It is noted that approximations made in the computations can drastically alter the landscape and make these findings inaccurate for simulations. There are also noted vulnerabilities to weak noise and decoherence that can give rise to results that do not correspond to observed behaviour in reality. As the noise will have an effect on the least robust solutions in the system, and the vulnerability to finding a weak solution with algorithms. The result, that there exists an optimal quantum control landscape, is not intuitive.
A. Pechen and Il'in (2012) contradicts earlier work and shows that care must be taken when studying quantum control landscapes, as for many systems there are no guarantee that it is trap free.

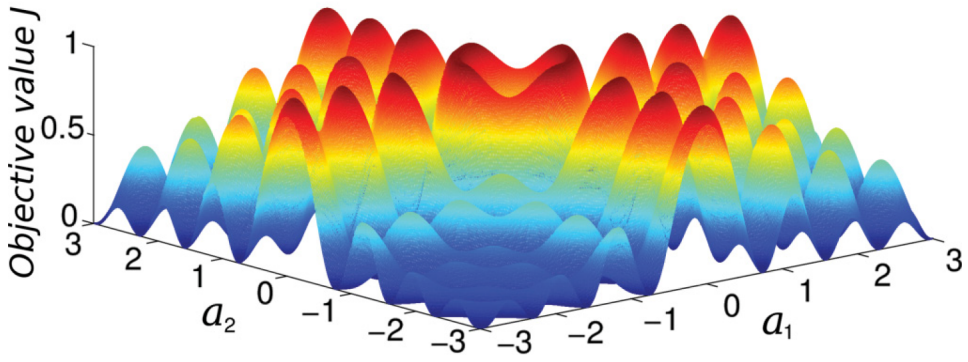

Figure 2.3: Example of an uneven quantum landscape with local maxima. Reprinted figure with permission from Pechen, A., & Il'in, N. (2012). Trap-free manipulation in the landau-zener system. *Phys. Rev. A*, *86*, 052117. https://doi.org/10.1103/PhysRevA.86.052117. Copyright 2012 American Physical Society.

2.4 Working with NISQ

There is research being done to work in a NISQ enviroment with algorithms that are error tolerant, like quantum approximate optimization algorithm (QAOA). (Xue et al., 2019) The limits of the current systems are too severe for most general problems. "For example, to factor 2048-bit RSA integers requires 20 billion noisy qubits." (Xue et al., 2019).
The result from Xue et al. (2019) shows that it is possible to use hybrid algorithms to solve several numerical problems. And the result of the landscape created by the cost function is merely flattened instead of distorted, so the results of using optimization algorithms are valid for use with the QAOA algorithm. But as the number of gates and the noise the system is exposed increases, the fidelity decreases exponentially.

Chapter 3

Theory

To give some insight into what I am trying to accomplish with my study I will give an overview of the theory behind my work.

3.1 Simulating

To understand the effect dynamic magnetic fields will have on particles, it is easier to start by simulating how the particles will behave under different circumstances. The Schrödinger equation and its wave function is my starting point. But to simulate decoherence and non unitary, non reversible effects, I have to use density matrices and a Master equation.

Since it is only possible to solve the Schrödinger equation analytically for a small subset of simple systems, I will use a numerical approach to solve the equation.

The Hamiltonian will be used to represent the energies acting upon the particle, with variables for the magnetic field while the Lindbladian will in addition have variables representing the interference from the environment.

The Crank-Nicolson or Runge-Kutta based solver is used to solve the resulting differential equation. Finally, I will be using a cost function to compare how well the solved differential equation corresponds to a desired quantum gate.

The main purpose of these simulations is, through the use of a Gradient descent algorithm to find out if there are parameters that for different conditions can enable me to create any desired gate.

I will here expand upon these methods of simulation, and how the measurement of the result is done.

3.2 The Schrödinger Equation

The Schrödinger equation is here used in conjunction with the Crank-Nicolson solver to find the effect of different interactions acting on the particles I simulate.
The Hamiltonian is the energy operator in the system and is often written as $\hat{H}$ as seen in Eq. (2.2). It contains information about all interactions acting on the system described by the Schrödinger equation and if it is not acted upon by outside energies, it will simply contain the sum of the kinetic and potential energy (Griffiths, 1995).

3.2.1 Time dependence of the Schrödinger equation

While the Schrödinger equation can take different forms depending on what the equation refers to. The general time dependent version of the Schrödinger equation is written like this:

$$i\hbar\frac{\partial}{\partial t}\Psi(t) = \hat{H}\Psi(t) \tag{3.1}$$

For this equation, the Hamiltonian can in many cases be thought of as a matrix $\hat{H}$ that work on the wave Ψ. The computation of Ψ then becomes the Hamiltonian operator acting on the state of the system ψ.
Eq. (3.1) is a very general way to write the Schrödinger equation, it covers every possible version that is time independent. As an example, you use the time independent Schrödinger equation to describe the energies of a quantum system, which can be written like this:

$$\hat{H}\Psi = E\Psi \tag{3.2}$$

The Eq. (3.2), the time independent Schrödinger equation, describes a stationary system where the probability density is static. From this equation, you can see that the eigenvalues of $\hat{H}$ give the energy levels of the system E. This requires that the Hamiltonian is time independent: it does not vary with time but is static.
You can describe the effect of the interactions from the Hamiltonian using a propagator $U(t)$ seen in Eq. (2.8).
When you have a time dependent Schrödinger equation where the Hamiltonian is time independent you can describe the effect of the interactions in the Hamiltonian acting on the system for a period of time, using a simple propagator:

$$U(t) = e^{-i\hat{H}t/\hbar}, \tag{3.3}$$

if you substitute this for $U(t)$ in Eq. (2.8), you get the simple propagator:

$$\Psi(t) = e^{-i\hat{H}t/\hbar}\Psi(t_0) \tag{3.4}$$

The simple propagator in Eq. (3.4) is only valid for a time independent Hamiltonian as seen in Eq. (3.2), the time independent Schrödinger equation applies when the Hamiltonian is not explicitly dependent on time, and the effect of the propagator is only changed by how long it acts on the particle.

3.2.2 Schrödinger equation for the spin of a particle

In this study, I focus on the use of the Schrödinger equation to simulate the effect of dynamic magnetic fields, while keeping one field at constant strength. To do so, I have to choose a Ψ and $\hat{H}$ that make the equation represent the spin of a particle.
To make Eq. (2.4) refer to the spin of a particle, Ψ will be a state vector as described in Sec. 2.2.1 where the square of each value in the vector represents the possibility for the particles being in that particular state. I use Eq. (3.1), the time dependent version of the Schrödinger equation, as the effect of the Hamiltonians I use are explicitly dependent on time.
The Hamiltonian $\hat{H}$ represents all interactions acting on the particle, in my example magnetic fields and interaction between particles. Details on the Hamiltonian will be discussed in Sec. 3.3.
By using the Schrödinger equation in combination with the cost function, I hope to get an understanding of the landscape the cost function describes and from there get an understanding of the feasibility of different directions of this study.
Since the Schrödinger equation only can be used to simulate closed systems, density matrices must be used to accurately simulate the effects of interference from the environment.

3.3 The Hamiltonian

The Eq. (2.21), the Pauli matrices, can form the basis for a Hamiltonian of a one qubit system. When creating a Hamiltonian, the Pauli matrices can be used to represent how strong the interactions acting on a particle are in a particular direction, with each gate being used to represent a magnetic field orthogonal to the other magnetic fields, multiplying each Pauli matrix with the strength of the magnetic field in the direction it represents. The constructed Hamiltonian can then be used to change the state of the system by using it as described in the equation Eq. (2.4).
A constructed Hamiltonian can be used to represent the interactions acting on the spin of a trapped particle with spin. This trapped spin particle can represent a one

qubit system and can be expanded to have more particles. The energy operator in Eq. (3.5) can be constructed so that σ represents the Pauli matrices and $\mathbf{B}$ is how strongly each of those matrices acts on the system. $\mathbf{B}$ represents the strength of the magnetic field in all directions, and for a particular direction, you have B_x being the strength of the field acting in the x-direction. By varying each B, to change how strong each magnetic field is, I hope to see how gates can be constructed and whether they have the effect on the state as desired, after acting on the qubit for a set time.

$$H_1 = \frac{e\hbar}{2m}\mathbf{B}(t)\cdot\boldsymbol{\sigma} = \frac{e\hbar}{2m}\left(B_x(t)\sigma_x + B_y(t)\sigma_y + B_z(t)\sigma_z\right). \tag{3.5}$$

The Φ in:

$$\Phi_1(t) = a(t)|\uparrow\rangle + b(t)|\downarrow\rangle, \tag{3.6}$$

can be talked about as the wave function.
Here it describes the state of the qubit, as the square of the absolute value to a describes the probability of the qubit spin direction being in the up position and the square of the absolute value to b being the probability of the particle being in the down position. As long as the particle stays in the system the norm should always be 1 so that $|a|^2 + |b|^2 = 1$.
Using the Hamiltonian and adding variables to it, it is possible to simulate different interactions acting on the particles, including the magnetic fields meant to manipulate them, and interference from other particles. However, the Hamiltonian is not enough to describe interference from the environment, to do that you need a more general equation that can describe an open quantum system, like the Lindblad equation described in Sec. 3.6.
When Eq. (3.5) is solved with $B(t)$ as $\sin(t)$ it gives:

$$H(t) = \begin{bmatrix} -\frac{\epsilon}{2} & \Omega^* \sin(t\omega) \\ \Omega \sin(t\omega) & \frac{\epsilon}{2} \end{bmatrix} \tag{3.7}$$

Where $\Omega = \Omega_x - \Omega_y i$, t is the time while ω is the frequency change of the magnetic fields.

Hermitian operator

To ensure that all observables are real, you must ensure that the Hamiltonian is Hermitian. That is that the Hamiltonian $H = H^\dagger$ or self adjoint, it can also be written (Gerald, 2014):

$$\langle Ax, y\rangle = \langle x, Ay\rangle \tag{3.8}$$

Hermicity ensures that it has real eigenvalues, so that when the Hamiltonian is used in the Time independent Schrödinger Eq. (3.2) to find the energy values, it does so by finding the eigenvalues. These eigenvalues have to be real to describe the energy levels of the wave function.

3.3.1 Spin spin interactions

When there are two or more qubits there will be an interaction between them, and in the target system for qubits with spin, it will be a spin spin interaction. This means that any Hamiltonian will have to account for the interaction between the two particles as seen below with u as the strength of interaction:

$$\begin{aligned} H_2 &= \frac{e\hbar}{2m}\mathbf{B}(t)\cdot\boldsymbol{\sigma}\otimes I + I\otimes\frac{e\hbar}{2m}\mathbf{B}(t)\cdot\boldsymbol{\sigma} + \sum_{j\in\{x,y,z\}} u_j\sigma_j\otimes\sigma_j \\ &= \frac{e\hbar}{2m}\big(B_x(t)(\sigma_x\otimes I + I\otimes\sigma_x + B_y(t)(\sigma_y\otimes I + I\otimes\sigma_y) \\ &\qquad + B_z(t)(\sigma_z\otimes I + I\otimes\sigma_z)) + \sum_{j\in\{x,y,z\}} u_j\sigma_j\otimes\sigma_j \end{aligned} \tag{3.9}$$

When this is solved, it gives a Hamiltonian that can show the effect of interactions between two qubits with different strengths given by u. When e, $\hbar$ and m are set to 1 and u is one value it gives two matrices, one for the external magnetic fields and the other for the interactions:

$$H = \frac{1}{2}\begin{bmatrix} 2B_z & B_x - iB_y & B_x - iB_y & 0 \\ B_x + iB_y & 0 & 0 & B_x - iB_y \\ B_x + iB_y & 0 & 0 & B_x - iB_y \\ 0 & B_x + iB_y & B_x + iB_y & 2B_z \end{bmatrix} \tag{3.10}$$

With the interactions being:

$$H_{inter} = u\cdot\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 \\ 0 & 2 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

Which combined gives:

$$H_2 = \frac{1}{2}\begin{bmatrix} 2B_z + 2u & B_x - iB_y & B_x - iB_y & 0 \\ B_x + iB_y & -2u & 4u & B_x - iB_y \\ B_x + iB_y & 4u & -2u & B_x - iB_y \\ 0 & B_x + iB_y & B_x + iB_y & 2B_z + 2u \end{bmatrix} \tag{3.12}$$

For my project where as earlier $B(t)$ is chosen as $\sin(t)$ and for brevity $\Omega = \Omega_x - \Omega_y i$, the Hamiltonian can be written:

$$H_2(t) = \frac{1}{2}\begin{bmatrix} 2\epsilon + 2u & \Omega\sin(t) & \Omega\sin(t) & 0 \\ \Omega^*\sin(t) & -2u & 4u & \Omega\sin(t) \\ \Omega^*\sin(t) & 4u & -2u & \Omega\sin(t) \\ 0 & \Omega^*\sin(t) & \Omega^*\sin(t) & 2\epsilon + 2u \end{bmatrix} \tag{3.13}$$

But when the material the qubits are contained in gives different strengths to the interactions in the different directions, u is no longer only one constant repeated with a linear multiplier in the matrix. The interactions u are instead added to the matrix while being on the form:

$$u \cdot \sigma \otimes \sigma = u_x\sigma_x \otimes \sigma_x + u_y\sigma_y \otimes \sigma_y + u_z\sigma_z \otimes \sigma_z \tag{3.14}$$

This enables the simulation of materials that have different strengths on the interactions in different directions, at the cost of the Hamiltonian becoming more complicated and having additional parameters:

$$H_2(t) = \frac{1}{2}\begin{bmatrix} 2\epsilon + 2u_z & \Omega\sin(t) & \Omega\sin(t) & 2u_x - 2u_y \\ \Omega^*\sin(t) & -2u_z & 2u_x + 2u_y & \Omega\sin(t) \\ \Omega^*\sin(t) & 2u_x + 2u_y & -2u_z & \Omega\sin(t) \\ 2u_x - 2u_y & \Omega^*\sin(t) & \Omega^*\sin(t) & 2\epsilon + 2u_z \end{bmatrix} \tag{3.15}$$

Where the interactions alone are:

$$u \cdot \sigma \otimes \sigma = \begin{bmatrix} u_z & 0 & 0 & u_x - u_y \\ 0 & -u_z & u_x + u_y & 0 \\ 0 & u_x + u_y & -u_z & 0 \\ u_x - u_y & 0 & 0 & u_z \end{bmatrix} \tag{3.16}$$

Because these Hamiltonians are set up symmetrical and Hermitian, so that $H = H^\dagger$, I cannot distinguish between the two qubit states $|\uparrow\downarrow\rangle$ $|\downarrow\uparrow\rangle$, and I cannot affect the qubits individually, removing the possibility to implement the CNOT gate without creating a new Hamiltonian that distinguishes the effect it has on the qubits.

3.3.2 Constant pulse quantum gates

While I simulated a quantum system, I tested how the shape of the magnetic pulses could affect the result of the Hamiltonian. Discrete pulses are not that interesting to simulate as they can be perfectly analysed analytically.
For example, for the Hamiltonian:

$$H = B\sigma_x, \tag{3.17}$$

you can use the fact it is time independent on the same form as Eq. (3.2), we can use the simple propagator from Eq. (3.4) and expand the propagator using the power series of the exponential function e^x:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \tag{3.18}$$

This can then be used to get a simpler end result:

$$\begin{aligned}
U &= e^{\frac{iB\sigma_x}{\hbar}} && \text{(3.19a)}\\
&= \sum_{n=0}^{\text{inf}} \frac{1}{n!} \left(\frac{-iB\sigma_x t}{\hbar}\right)^n && \text{(3.19b)}\\
&= \sum_{n=0}^{\text{inf}} \frac{1}{(2n)!} \left(\frac{-iBt}{\hbar}\right)^{2n} I + \sum_{n=0}^{\text{inf}} \frac{1}{(2n+1)!} \left(\frac{-iBt}{\hbar}\right)^{2n+1} \sigma_x && \text{(3.19c)}\\
&= \sum_{n=0}^{\text{inf}} \frac{-1}{(2n)!} \left(\frac{Bt}{\hbar}\right)^{2n} I - i \sum_{n=0}^{\text{inf}} \frac{-1}{(2n+1)!} \left(\frac{Bt}{\hbar}\right)^{2n+1} \sigma_x && \text{(3.19d)}\\
&= \cos\left(\frac{Bt}{\hbar}\right) I - i \sin\left(\frac{Bt}{\hbar}\right) \sigma_x && \text{(3.19e)}
\end{aligned}$$

Where at Eq. (3.19c) we split the even and odd number of steps in the power series, and by rearranging Eq. (3.19c) to Eq. (3.19d) you get the power series form of sinus and cosinus. This makes the end result the simple expression in Eq. (3.19e).
Using Eq. (3.19e) you can easily find the strength required to get the X gate, as it only requires that the expression must be equal σ_x. And since the global phase does not matter it only requires that the expression inside sinus is equal to 1 and the expression inside cosinus is equal to 0. Using the fact that this is true for $\frac{\pi}{2}$ you get:

$$\frac{Bt}{\hbar} = \frac{\pi}{2} \tag{3.20}$$

An easily solved equation, as an example for $\hbar$ set to 1 and a time period of 2π you get $B = \frac{1}{4}$. This means it is not an interesting case to simulate on one qubit, since you can always repeat this for σ_y and σ_z, and get as shown with σ_x in Eq. (3.19e), a simple expression. This is because the Hamiltonian can always be decomposed into the interactions from the individual magnetic fields.
This is why the focus of this study was on the use of dynamic magnetic fields, to give the system some more dynamic changes that I hoped would give a possibility to counteract the effect of noise.

3.3.3 Examples of quantum gates

Some commonly appearing gates, that can be simulated by my program, are the Hadamard gate and the Swap gate.

Hadamard

The Hadamard gate is a gate commonly used to set a qubit in a superposition, and when used in combination with the CNOT gate, it can set a pair of qubits in a bell state. The equation is written earlier in Eq. (2.26) found in Sec. 2.2.2. When using constant pulse it can be constructed out of the Z and Y Pauli gates used consecutively:

$$H = ZY^{-\frac{1}{2}} = Y^{\frac{1}{2}}Z \tag{3.21}$$

So it should be possible to get close even if we use the dynamic magnetic pulses for the Ω_x and Ω_y direction with a static field in the ϵ direction.

Square root of SWAP

The square root of SWAP gate can in combination with single qubit gates be used to construct the CNOT gate, and can therefore in combination with other single qubit gates, be used to show that it is possible to have a universal set of gates.
The swap gate in Eq. (2.23) has a simple equation, while the square root of it is slightly more complicated:

$$\sqrt{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1+i}{2} & \frac{1-i}{2} & 0 \\ 0 & \frac{1-i}{2} & \frac{12i}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.22}$$

After starting to implement it, one of my supervisors found a paper that referenced how it could be constructed using only interactions between qubits (Fan et al., 2005). This is reasonable because when you look at Eq. (3.11) you can see the similarities with Eq. (3.22).

3.4 Density Matrix

A density matrix is a matrix that can be used to describe the open quantum system, where parameters in the matrix describe the effect the environment has on the smaller system being modelled.

It can be more complicated to use, which is why the start of the project focuses on the Schrödinger equation to get an overview of what ideas are worth focusing on.
The purpose of a density matrix is to be able to show more states of the system (E. K. Nielsen, 2011). While the Schrödinger equation can be said to be a single pure state in the density matrix, the density matrix is an ensemble of pure states, and is used if all information about the quantum system is not known. When working with density matrices the diagonal gives the probability to be in a given state while the off diagonal elements hold the quantum information about the system.

$$\rho = \sum_j p_j |\psi_j\rangle \langle \psi_j| \tag{3.23}$$

Since the density matrix is an ensemble of states as seen in Eq. (3.23), the coefficient p_j gives the probability of the quantum system represented by the density matrix being in the state j given by ψ_j. If only one of the states is different to zero you have a pure state, and then you know everything about its current state. But if you have multiple possible states, it's a mixed state and you may not be able to separate out every possible state, as for each density matrix, there may be multiple possible ensembles of states that give the same density operator (M. Nielsen & Chuang, 2010). You may use a density matrix to represent the state instead of the state vector described in Sec 2.2.1, but it cannot be used with the Schrödinger equation. But a density matrix requires a master equation, the Lindblad equation described in Sec. 3.6 is an example of such a master equation.

3.5 Quantum map, Unitary and non Unitary

A quantum map is a very general way of referring to any of the ways you can manipulate a density matrix to another density matrix, mapping the states of one density matrix to another. The U from Eq. (2.8) is an example of a quantum map.

3.5.1 Unitary

A unitary operator is an operator that preserves the inner product of the matrix it is applied to. When its inverse is applied to it you get the identity matrix (Sen, 2013):

$$A^\dagger A = A A^\dagger = I \tag{3.24}$$

When working with the Schrödinger equation and closed quantum system all the operators U, often called a propagator, from Eq. (2.8), will be unitary. It will therefore have an inverse operator $U^\dagger$ that can get back to the starting point after applying the inverse of the operator on $\Psi(t)$ from Eq. (2.8):

$$\Psi_0 = U(t)^\dagger \Psi(t) \tag{3.25}$$

This way we can recover the starting state Ψ_0. To ensure that the propagator U is unitary, one of the requirements is that the Hamiltonian that you propagate is Hermitian, as described in Sec. 3.3.

3.5.2 Non unitary

In contrast to the unitary quantum maps, non unitary quantum maps cannot guarantee that there exists a unitary operator and there is no inverse operator. So that when you get the end state $\Psi(t)$, there is no longer any way to recover the starting state using the inverse U^{-1}. As while it may exist it is no longer unitary and self-adjoint and Eq. (3.24) is no longer valid. The inverse may not exist or may lead to states that are not physically possible.
This is one of the drawbacks of the Lindblad equation and limits what states are easy to get back to when using a master equation to manipulate an open quantum system.

3.6 The Lindbladian

To expand on what is written in Sec. 2.1.5 I will explain the Lindblad master equation. While the Schrödinger Eq. (2.4) is perfectly able to describe the evolution of a quantum system, it has great limitations when it comes to simulating interactions from outside the system. The Lindblad master equation can be used to show how interaction with the system has a tendency to return to its ground state. It is a general purpose equation to simulate Markovian interactions, systems whose future state changes only depend on what the state of the system was after it last changed.
We are interested in quantum sub systems in the whole quantum system that is the universe, but interactions within this system can have a large impact on how the behaviour of a subsystem we want to study. While simulating every interaction would be impossible, simulating a subsystem with some approximations is possible (Manzano, 2020). For my project, the subsystem is a one or two qubit system including the noise from the environment and the tendency for particles to fall to their ground states.
Since I am now working in an open quantum system, the state vector is no longer enough to contain all information about this system. The Lindbladian instead uses a density matrix as described in Sec. 3.4. This enables an ensemble of mixed states that often arise when working with open quantum systems.
The equation is trace preserving, meaning the trace of the density matrix will always be 1. The Lindblad equation is completely positive semi definite, so that each pure

state in the ensemble of states in the density matrix always have has a positive or zero percent probability of existing, guaranteeing the state is physically possible (Gorini et al., 1976; Lindblad, 1976).

3.6.1 Lindblad master equation

The general Lindblad equation as found in Manzano (2020) here in Eq. (3.26), describes a quantum system where there are a number of jump operators L that each describes unwanted interactions that can lead to decoherence.

$$\dot{\rho} = -i\left[H, \rho(t)\right] + \sum_j \left(L_j \rho(t) L_j^\dagger - \frac{1}{2}\left\{ L_j^\dagger L_j, \rho(t) \right\} \right) \tag{3.26}$$

The jump operator can here be used to describe many types of interactions with the environment outside the quantum system. The density matrix is represented by $\rho(t)$ and the internal interactions of the system are represented by the Hamiltonian H.

3.6.2 Decay to ground state

For a Lindblad equation that only concerns itself with the chance of decay where the state falls from $|1\rangle$ *to* $|0\rangle$ it's possible to use:

$$i\dot{\rho} = H\rho - \rho H - i\gamma(a^\dagger a\rho + \rho a^\dagger a - 2a\rho a^\dagger) \tag{3.27a}$$

$$\dot{\rho} = -i(H\rho - \rho H) - \gamma(a^\dagger a\rho + \rho a^\dagger a - 2a\rho a^\dagger) \tag{3.27b}$$

The gamma in the Lindblad equation represents how fast the particle being simulated is likely to fall down into its ground state. Part of the project is to see if there are ways to mitigate how fast the particle falls into the ground state and suffers from decoherence. This decoherence can be expressed as a jump operator:

$$a = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{3.28}$$

$$a^\dagger a = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.29}$$

These jump operators say something about what kind of noise is present in this case amplitude dampening. This is a simple model of noise where a says something about how the state tends towards $|0\rangle$, a is representing $|0\rangle\langle 1|$ or Eq. (3.28). While $a^\dagger a$ represent Eq. (3.29) in the Lindblad equation and is a way to remove value from one state adding it to another state. In Eq. (3.27) $2a\rho a^\dagger$ is used to subtract from one state to

add to another state in the resulting density matrix, to represent the decay towards its ground state. Which can be expressed as the differential equation:

$$i\dot{\rho} = \begin{bmatrix} 2i\gamma\rho_{11} - \rho_{01}\Omega + \rho_{10}\Omega^* & -\epsilon\rho_{01} - i\gamma\rho_{01} - \rho_{00}\Omega^* + \rho_{11}\Omega^* \\ \epsilon\rho_{10} - i\gamma\rho_{10} + \rho_{00}\Omega - \rho_{11}\Omega & -2i\gamma\rho_{11} + \rho_{01}\Omega - \rho_{10}\Omega^* \end{bmatrix}, \tag{3.30}$$

where $\Omega = \Omega_x - \Omega_y i$ with Ω_x, Ω_y and ϵ as the magnetic fields placed orthogonal with respect to each other, ρ is a matrix that starts with one value set to 1 at a time and γ tells how fast the particle falls to zero.
An example of the four ρ matrices for one particle:

$$\rho_{00} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \rho_{01} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \rho_{10} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \rho_{11} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.31}$$

This is every individual state of ρ with only one value of 1 with the rest zero, while only the ρ_{00} and ρ_{11} states, the states with diagonal elements, are valid states. As only those states have a trace of one and are Hermitian, but all states have to be used to build the complete quantum map. The matrix ρ will for n number of particles have each side the size of 2^n, so it's a $2^n \times 2^n$ matrix, making it grow fast as the number of particles increases.
This is a great limitation on the use of the Lindblad equation. The number of times it must be solved, as needed for all possible values of ρ, makes the computation take exponentially longer as the number of particles increases. The Lindblad equation increases computational time even faster than the Schrödinger equation seen in Sec. 3.2. This necessitates changes to keep the computation time manageable for any implementation with more than one particle.
The Lindblad equation is trace preserving, as the trace of the density matrix should always be 1. Making the diagonal of the matrix represent the possibility of being measured in a given state while the off diagonal elements represents the quantum information.

3.6.3 Lindblad on ODE form

For one particle it is possible with the Lindblad equation to write out the equation as a $4x4$ matrix, consisting of four differential equations with one for each state of ρ, instead of having a $2x2$ matrix with values of ρ as input.

$$\dot{\rho} = \begin{bmatrix} 0 & -\Omega_x + i\Omega_y & \Omega_x + i\Omega_y & 2i\gamma \\ -\Omega_x - i\Omega_y & -\epsilon - i\gamma & 0 & \Omega_x + i\Omega_y \\ \Omega_x - i\Omega_y & 0 & -\epsilon - i\gamma & -\Omega_x - i\Omega_y \\ 0 & \Omega_x - i\Omega_y & -\Omega_x - i\Omega_y & -2i\gamma \end{bmatrix} \begin{bmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \\ \rho_4 \end{bmatrix} \tag{3.32}$$

This makes the matrix derived from the Lindblad equation solvable by the Crank-Nicolson Sec. 3.7.1 solver using a flattened rho to get a column vector of size $4x1$. I would then solve it for all 4 possible start states of ρ, and get a quantum map. This quantum map can then be used to compare the effect of the Lindbladian with a desired gate, using an appropriate measurement method as discussed in Sec. 3.8.

3.6.4 Lindblad two qubits

When using the Lindblad equation on two qubits you have to compute it for all possible variations of ρ as for one qubit as in Eq. (3.31). This leads to having 16 starting states, where only 4 are physically possible states, located on the diagonal. The Lindblad equation itself is a 4×4 matrix with the Hamiltonian H in the Lindblad Eq. (3.26) being equal to the Hamiltonian for two particles in Eq. (3.13). The four jump operators are also in the form of 4×4 matrices, with the effect of the jump operators changing with the state of the density matrix ρ.
If you want to have more granular control of the strength of the individual jump operators, you can give each jump operator L_j a multiplier γ_j to control their strength independently:

$$\dot{\rho} = -i\left[H, \rho(t)\right] + \sum_j \left(\gamma_j L_j \rho(t) L_j^\dagger - \gamma_j \frac{1}{2}\left\{L_j^\dagger L_j, \rho(t)\right\}\right) \tag{3.33}$$

To simplify the calculation of the jump operators described in Eq. (3.26) and taking the γ seen in Eq. (3.33), you can precalculate part of the equation so that the jump operator in the anti-commutator can be written:

$$\sum_j \frac{1}{2}\left\{L_j^\dagger L_j, \rho(t)\right\} = \sum_j \left(\frac{1}{2}\gamma_j L_j^\dagger L_j\right)\rho + \rho \sum_j \left(\frac{1}{2}\gamma_j L_j^\dagger L_j\right) = D\rho + \rho D, \tag{3.34}$$

this way D is only calculated once for each set of jump operators.

3.6.5 Generating jump operators

In order to have a jump operator suitable for testing, it is constructive to have a consistent way of getting traceless jump operators. I begin this process with creating a 4×4 matrix with normally distributed values, with mean 0, standard deviation of 1 and using the same process for an imaginary part:

$$L_{step} = \begin{bmatrix} \mathcal{N}(0,1) & \mathcal{N}(0,1) \\ \mathcal{N}(0,1) & \mathcal{N}(0,1) \end{bmatrix} + i \begin{bmatrix} \mathcal{N}(0,1) & \mathcal{N}(0,1) \\ \mathcal{N}(0,1) & \mathcal{N}(0,1) \end{bmatrix} \tag{3.35}$$

To ensure it is traceless I remove the product of the trace multiplied with the identity matrix and divided with the dimension of the matrix d:

$$L_{final} = L_{step} - \frac{Tr(L_{step})I}{d} \tag{3.36}$$

This enables easy generation of multiple possible jump operators to simulate the noise. Depending on the need you can create an individual jump operator or use several for a set of jump operators.

3.7 Solving Differential Equations

Since solving the Schrödinger equation gets exponentially harder with each additional particle simulated, solving it analytically is basically impossible for any more complicated systems. The method used here is instead a numerical approximation that gives a close enough result to go further with.
The equations only have to be on a differentiable form and while there are many possible ways to solve differential equations I have used a Crank-Nicolson solver based on Sec. 3.7.1 and a Runge Kutta solver Sec. 3.7.2.

3.7.1 Crank-Nicolson solver

The crank-Nicolson method is a numerical equation to solve Partial Differential Equations(PDE). It uses both the forward Euler and backwards Euler to get a good approximation and to reduce the errors. The forward Euler is explicit and requires a large number of timesteps to be accurate, while the backwards Euler method is implicit and more accurate for larger timesteps to solve for. It is based on a research paper written by John Crank and Phyllis Nicolson (Crank & Nicolson, 1947).
The version in my project is a simplification that I use to solve Ordinary Differential Equations(ODE) instead of the Partial Differential Equations(PDE) the original version is used for.
It uses a suitable timestep dt to take a step forward:

$$\phi_{new} = \left(I + \frac{i \cdot dt}{2} H(t + dt)\right)^{-1} \cdot \left(I - \frac{i \cdot dt}{2} H(t)\right) \phi_{old} \tag{3.37}$$

This is repeated until the propagation reaches the desired end time as you increment the time for each step $t_{new} = t_{old} + dt$.

3.7.2 Runge–Kutta solver

As the time used to solve the Lindblad equation for even one particle is high with the Crank-Nicolson solver, I use an alternative solver based on the Runge-Kutta method. It's a fourth order solver with an error on the scale of $\mathcal{O}(dt^4)$ and even with bigger

time steps it is projected to be more accurate and faster than the Crank-Nicolson solver.
It is accurate up to the fourth order of the size of the steps and is an explicit method dependent only on the current step and not the next step (Süli & Mayers, 2003, p. 328-329). This makes it easier to write the algorithm and implement it.
I use the four step Runge-Kutta version where it gradually computes the next four steps, once for the current value and time, twice for halfway to the next timestep and lastly once at the next timestep. Always using the state got from the previous step to calculate the next before summing it and multiplying with the size of the timestep h and dividing by 6 before adding it to the previous state:

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h \\ k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + \frac{h}{2}, y_n + h\frac{k_1}{2}) \\ k_3 &= f(t_n + \frac{h}{2}, y_n + h\frac{k_2}{2}) \\ k_4 &= f(t_n + h, y_n + hk_3) \end{aligned} \tag{3.38}$$

This gives great accuracy even while having larger timesteps and can be used to solve any ODE.

3.8 Measuring the Fidelity of Gates

As this is a simulation I need to have some way to measure how close I am to an ideal result; the target gates. For this, I have used some different measurement methods, depending on what I am simulating. Starting with the norm measure, a measure that is accurate but inflexible in regard to the global phases. When testing it gives very uneven cost landscapes with poorer optimizations compared to using the average gate measure. For the norm measure, a difference in phase is taken to be a difference in distance, while global phases are not measurable for particles, including the half spin qubit that the project focuses on, so the global phase does not matter when doing a measure of a qubit. Therefore the norm measure requires some method to resolve the effect of the global phase.
The average gate fidelity gives a more forgiving cost landscape, and like the channel fidelity, it is not sensitive to differences in the global phase. It measures the difference between two pure states.
The channel fidelity is a way to measure the difference between two quantum states or a quantum state and a quantum map. When I use channel fidelity, I create a Choi

matrix from a quantum map, that comes from solving the Lindblad equation with its density matrix. This is compared to a Choi matrix created from the target state or gate, but using a Choi matrix enables comparing two quantum maps, expanding the scope of what you can compare using channel fidelity.
In my measurements, 0 is a perfect match and 1 is the worst possible result. I have enforced a positive value for different results and 0 for a perfect match in this project.

3.8.1 Norm measure

When looking at the trace distance, it can, as demonstrated in M. Nielsen and Chuang (2010), be thought of as half the euclidean distance between vectors or the 2-norm. This is one of the methods I use to measure the difference from the target gate I want.
The norm measure gives the difference for the result of the simulation from the particular gate I want to target. By subtracting the target gate from the result $C = A - B$, and getting the euclidean norm of the result:

$$||C|| \equiv \max_{\langle u|u\rangle=1} |\langle u|C|u\rangle| \tag{3.39}$$

I can say something about how different it is from the gate I want.
To resolve the fact that the norm measure does not ignore the global phase I use a little trick, and change the matrix to match the phase of one of the elements in the target gate, selecting an element that is non zero (Personal Communication, K. Wold, 2022). This way the difference in the global phase is found by getting the angel of one element in the quantum map I measure, an angle from an element with the same placement in the target gate:

$$\omega = -i\phi + i\theta \tag{3.40}$$

Where ϕ is the phase of the element in the quantum map and θ is the phase of the element from the target gate. This is then used to match the global phase for the quantum map created and the target gate:

$$U_{new} = e^{\omega} U_{old} \tag{3.41}$$

For better accuracy, it might have been better to use the average phase of all elements, but I swapped to Average gate fidelity described below, a method known to ignore the global phase.

3.8.2 Average gate fidelity

After testing with the norm measure Eq. (3.39), it showed a quite complicated cost landscape, so I decided to look at alternative ways to measure the difference from the target gate, to see if it was possible to find a measure that made it easier for gradient descent to do a search.
The average gate fidelity is taken from the quantum fidelity measure:

$$F(\rho,\sigma) = \left(Tr\left[\sqrt{\sqrt{\rho}\sigma\sqrt{\rho}} \right] \right)^2, \tag{3.42}$$

and derived as described in (Magesan et al., 2011) to get:

$$F_{avg} = \sum_i \frac{\left(Tr[K_i]Tr[K_i^\dagger]\right) + d}{d^2 + d} \tag{3.43}$$

Where d is the dimension of the matrix and K_i is the i'th possible Kraus-operator K. Kraus-operators are unitary operators, that map a pure state to another pure state while, being a completely positive and trace preserving operation as described in Sec. 3.5.1. The Kraus representation of the linear super operator T working on the state ρ can be written as (Magesan et al., 2011):

$$T(\rho) = \sum_n K_n \rho K_n^\dagger \tag{3.44}$$

Since the Kraus operators in Eq. (3.44) are completely positive and trace preserving it leads to the Kraus operators having the trait:

$$\sum_n K_n^\dagger K_n = \hat{I}, \tag{3.45}$$

this means that the sum of the multiplication of all complex conjugated Kraus operators with its corresponding Kraus operator gives the identity operator (Hellwig & Kraus, 1969; Magesan et al., 2011).
What I use in my project is a special occurrence, that can be used when only measuring on one pure state, so that there is only one unitary Kraus operator, instead of the more general method that can be used on quantum maps.
And since I only use this measure on one pure state, I can set $K_i = U^\dagger U_{\text{Gate}}$ and get the equation:

$$F = \frac{|Tr[U^\dagger U_{\text{Gate}}]|^2 + d}{d^2 + d} \tag{3.46}$$

With U as the result of the simulation and U_{Gate} as the target. This is simpler to implement than the full equation and works well to find the fidelity on pure states, and is only a minor adjustment to get the infidelity:

$$IF = 1 - F, \tag{3.47}$$

The result, as seen in Eq. (3.46) is that by using the trace of the unitary created and a target gate and squaring the result, it is possible to get the fidelity with a high degree of accuracy.
Preliminary testing showed a smoother cost landscape and less local minima, when using this method to measure the difference from a quantum gate. It is expected to be a little easier to find good alternatives using gradient descent, which is described in 3.9.

3.8.3 Channel fidelity with Choi matrix measure

Given that the Lindbladian Sec. 3.6 used in my project are not directly comparable to a target gate, so I have to use a different method to compare them. As doing a comparison using Choi matrices with channel fidelity is easier than using Kraus operators, it also enables comparing two open quantum maps instead of the limit of at most one open quantum map that channel fidelity has by itself.
I create a Choi matrix from the solution of the Lindblad equation, with result that the Choi matrix has more dimensions than the gate the lindbladian is compared to. While the gate has the dimensions $d \times d$, the Choi matrix created by the lindbladian has $d^2 \times d^2$. To do the comparison I have to create a Choi matrix from the target gate, so that I compare two Choi matrices and can measure the difference.
The channel fidelity is a way to do this kind of measure. It is done by taking the tensor product of $U\rho U^\dagger$ with ρ. Where U is the gate to target and ρ is a square matrix with one value of one and the rest 0. Iterating this over every possible value for ρ summing them and dividing by d^2 (Magesan et al., 2011):

$$\chi = \sum_i \frac{U\rho U^\dagger \otimes \rho}{d^2} \tag{3.48}$$

This is then compared to the effect of the Lindblad using channel fidelity.
This is less accurate than both average gate fidelity and norm measure, but it is capable of measuring the non-unitary quantum map created by the Lindbladian.
It is, like the Average gate fidelity, also based on Eq. (3.42), but used in its full form, where *rho* is set to the Choi matrix χ from Eq. (3.48) and is a quantum state, while σ is a Choi matrix created from a quantum map that you want to compare to.
Using a Choi matrix had the advantage that finding the state fidelity of a Choi matrix is the same as finding the fidelity of the quantum channel:

$$F_{qc}(T_1, T_2) = F_{qs}(\chi_1, \chi_2) \tag{3.49}$$

Enabling the use of the channel fidelity to measure the fidelity of the quantum state, even in cases where both T_1 and T_2 are quantum maps.
This process gives the channel fidelity F, that for a one particle system, without other modifications will a value in the domain $[0.25, 1]$, and more generally in the interval $\left[\frac{1}{2^n}, 1\right]$. Because the measurement used in this project is the infidelity, that is Eq. (3.47) as a measure of how close the result is to the target gate I have the interval $\left[0, 1 - \frac{1}{2^n}\right]$.

3.9 Gradient Descent

Gradient descent is a well known optimization algorithm to find local minima for a function. It does this by taking the partial derivative for each variable before multiplying it with a learning rate to set how large step each iteration of the algorithm takes.
Due to the limits of gradient descent, when it comes to local minima, I have taken some steps to reduce the run time of using the algorithm, to be able to look at alternative start points. One of the things I did to reduce the run time, was to stop iterating when the difference was too small to matter, or when the starting point was so bad it was unlikely to reach a minima.
Gradient descent works by taking the derivatives of the variables in the vector $\vec{x}$ using the ∇ and its derivation operators to get all the partial derivatives. When placing all these derivatives together in a vector, you get the gradient, hence the name of the algorithm. By subtracting the gradient multiplied by a learning rate as below you can use gradient descent to find a minima, while adding the gradient can be used to find maxima for the function.

$$\vec{x}_{n+1} = \vec{x}_n - \mu \nabla Q(x) \tag{3.50}$$

The learning rate μ influences how fast it can converge to a result and how good that result is.

3.9.1 Randomized start points

Given the many observed local minima in the cost landscape, it has sometimes been a challenge to find the global minima. This requires either a good starting point or many runs.
To simplify the way forward, I have created code that runs many iterations with gradient descent with random start parameters and with fewer steps in the gradient descent algorithm, throwing out the worst ones as early as possible. While looking

closer at the best ones with longer runs of the optimization algorithm, to find good minima close to the global minima.

3.9.2 Momentum

Having momentum in the gradient descent algorithm makes you retain some of the earlier direction and can help the algorithm better follow a "valley" in the cost landscape.
Gradient descent have problems when used on landscapes with large differences in values with small differences in parameters. Using momentum is an attempt to smooth out its tendency to jump back and forth, when as an example if it is trapped in a "valley" with high values on both sides. When a path down is in a valley, the algorithm has a tendency to jump between the sides of the valley towards the lower value, greatly increasing the time to reach the bottom. So the momentum is a way to make it keep going in a direction, smoothing out the path and giving the algorithm a way to get out of smaller local minima. It does this by remembering the earlier steps and multiplying by a variable β that can be a constant or be modified depending on the need. Thereafter the new gradient has to be multiplied by 1 minus β:

$$\nabla Q(x)_{total} = \beta \nabla Q(x)_{old} + (1 - \beta) \nabla Q(x)_{new} \tag{3.51}$$

With this momentum the steps are keeping some momentum from earlier steps, enabling it to be smoother if in a "valley" or to get into and out of smaller local minima.

3.9.3 Adam-stochastic gradient

The Adam algorithm is a variant of Gradient Descent based on AdaGrad and RMSProp, two different variants that are good with sparse gradients and non stationary object respectively (Kingma & Ba, 2014).
Adam is an algorithm that modifies gradient descent to use different learning rates for the different parameters, and with this increases the computational efficiency and increase accuracy. "The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients" (Kingma & Ba, 2014, p. 1). This reduces and can even remove the need to adjust the learning rate while using the algorithm, simplifying its implementation. The fact that the learning rate is adjusted for each parameter individually, is a great boon when dealing with landscapes that are uneven or have saddle points.
Using the Adam algorithm can make it easier and faster to get a good result, especially in a cost landscape as complicated as here, where there in some directions

are large differences and minor in others, and many local minima.
I have mostly used the default parameters recommended, but there are several hyperparameters, including the learning rate, that can be tuned to optimize the algorithm for a specific problem.

Chapter 4

Method

I will in this chapter go through some of the methods used in the code of the project, and write with more detail about how I implement the theories described in Chapter 3. After this I will describe the approach I used in this study.

4.1 Solving differential equations

To solve the Schrödinger equation described in Sec. 3.2 and Lindblad equations in Sec. 3.6 I use numerical approximation to reach a solution. These calculations will simulate how a real quantum system behaves, with the accuracy to reality depending on how accurate the Hamiltonians used describes the energies involved in a real system and the accuracy of the solvers used.

4.1.1 Numerical implementation

Implementing tools for simulation, like those used in this project, is a laborious process. To conduct the simulations of the quantum system in my project, I implemented the code in the script language Matlab. For compiling I'm using Matlab versions R2022a and R2022b. A link to the repository with code is in the appendix.
I have strived to make several classes and functions as generic as possible, in order to more easily allow expanding what type of Hamiltonians can be used and simplify the addition of different measures and methods to solve the differential equations.
Part of this is using object oriented programming, with inheritance to guarantee the behaviour of the subclasses, and simplifying the program structure so that you just have to implement the required behaviour, while the program will handle the details.
I have used an enum, a common object oriented construct for linking a name to a number of constants. It keeps track of all possible differential equation solvers, and makes it easy to add and implement additional ones. But there are limits to

the checks that the solver can do, to check that it is appropriate for the type of Hamiltonian or Lindbladian used. Expanding the program to include a setting for each solver to identify the appropriate solver method for a Hamiltonian, is left for future users. So when you add a new Hamiltonian, you have to make sure it has a valid default solver and caution users of the program to not change the solver unless they know what they are doing.

4.1.2 Representing gates

To represent the gates I use class files that inherit from the interface Lst. 4.1. The abstract `rotate` function is used for counteracting the effect of the global phase by setting the phase of the state to be compared so that the phase of the input state is equal to the phase of the gate.

```
classdef (Abstract) GateInterface
    methods(Abstract, Static)
        rotate = rotation(U)
    end

    properties(Abstract, Constant)
        gate
        Psi0
    end
end
```

Listing 4.1: Interface for a Gate

This `GateInterface`, is a simple interface containing only one abstract method `rotate` ↪ and some abstract constant properties. The method is used to enable a function that will rotate the phase of a matrix input to match the global phase of the gate. Some gates do not have values different from 0 in all positions, which means that the 0 element does not have a phase, therefore which value to match the phase for, changes for different gates. This is only required by functions that use the Norm measure described in Sec. 3.8.1, as it is the only solver I use that requires correction for the global phase of the gate and quantum map.
The constant `gate` and `Psi0` are properties that will not change after I create the gate class, requiring that each gate will have a class to represent it with its own descriptive class name. The `gate` is simply the gate on matrix form, while `Psi0` is an identity matrix that was created to match the size of the gate so that each column represents one of the possible start states. Where each start state is solved, and then combined to create a quantum map.
This enables each gate to be a simple class with only a few values and makes adding, doing simulations and optimizations on new Gates easy to accomplish.

Time settings

I use the class `TimeOptions` to keep track of the time variables. It is a very simple construction that only contains some default values, with the addendum that all values must be real.

```
classdef TimeOptions
    properties
       Tstart double;
       Tpulse double;
       Tsize double {mustBeGreaterThan(Tsize,0)};
    end


    methods
        function Time = TimeOptions(options)
            arguments
               options.Tstart(1,1) double = 0;
               options.Tpulse(1,1) double = 2*pi;
               options.Tsize(1,1) double = 1000;
            end

            Time.Tstart = options.Tstart;
            Time.Tpulse = options.Tpulse;
            Time.Tsize = options.Tsize;
        end
    end
end
```

Listing 4.2: Class for time settings

`Tpulse` is the size of the pulse. `Tstart` is the start point, if you want to start the pulse at some other point than the default 0. `Tsize` is the number of timesteps used when solving the Hamiltonian, which contains an instance of the `TimeOptions` class.

4.1.3 Hamiltonian and Lindbladian

The Hamiltonian and Lindbladian are based on the theory described in Sec. 3.3 and Sec. 3.6 respectively, they use an interface such as Lst. 4.3 and Lst. 4.8.

```
classdef (Abstract) HamiltonianInterface

    properties(Abstract, Constant, GetAccess=public)
       matrixSize int8 {mustBePositive}
    end
```

Listing 4.3: Interface for a Hamiltonian

Having the matrix size available, makes several steps in the solving easier, especially when there is a mismatch between the matrix size of the `Hamiltonian` and the `Gate` it is compared to.

```
    properties(Access=public)
        Parameters double {mustBeReal}
        Measure Measure = Measure.AvgFidelity;
        Solver HamiltSettings.Solvers = HamiltSettings.Solvers.
    ↪ Crank_Nicolson;
        Time TimeOptions
    end
```

Listing 4.4: Interface for a Hamiltonian

The `Parameters` are the variable that are to be optimized, with a default requirement only that they are real numbers. For `Measure` and `Solver` the default values work with most Hamiltonians, while the `HamiltSettings.Solvers.Crank_Nicolson` solver only works for Hamiltonians acting on state vectors and the measure `Measure.AvgFidelity` ↪ is working on any pure state that gives a matrix the same size at the target gate. `TimeOptions` contains all settings regarding the time, and modifications of the output should use the dependent properties.

```
    properties(Dependent)
        TimeStep double
        TimeVector(1,:) double
        Period double
    end


    methods(Abstract)
        createHamiltonian(this)
    end
```

Listing 4.5: Interface for a Hamiltonian

These dependent variables enable us to modify the behaviour of the `TimeVecor` and the timesteps and their sizes. Moreover, the abstract method `createHamiltonian(this)` ↪ is used to create a function handle that takes the time and outputs the state of the Hamiltonian at the given time.

```
    methods

        % Get functions for dependent variabels
        function TimeStep = get.TimeStep(this)
            arguments
                this Hamiltonians.Interfaces.HamiltonianInterface
            end

```

```
36              TimeStep = this.Period/this.Time.Tsize;
37          end
38
39          function TimeVector = get.TimeVector(this)
40              arguments
41                  this Hamiltonians.Interfaces.HamiltonianInterface
42              end
43
44              TimeStart = this.Time.Tstart;
45              TimeEnd = this.Period + this.Time.Tstart;
46
47
48              TimeVector = (TimeStart):this.TimeStep:(TimeEnd);
49              TimeVector(end) = [];
50          end
51
52
53
54          function this = set.Time(this, Time)
55              % Set the TimeOptions used by this hamiltonian
56              arguments
57                  this Hamiltonians.Interfaces.HamiltonianInterface
58                  Time TimeOptions
59              end
60
61              this.Time = Time;
62
63          end
```

Listing 4.6: Interface for a Hamiltonian

The function for the dependent variable `TimeStep` uses a normal get function `get.Period ↪ (this)` found in Lst. 4.7, that itself uses the custom function `periodGet`, that can be overridden by a subclass.
The get function `get.TimeVector(this)` for the dependent variable `TimeVector` uses the get function `get.Period(this)`, found in Lst. 4.7 below. It uses the values in an instance of the class `TimeOptions` and the dependent variable `Period`, to create a vector with the number of timesteps set in the `TimeOptions` modified for alternate start times. The use of a custom function in the get function is necessary as you cannot override a set or get function, but methods they call can be overridden.
By comparison `set.Time(this,Time)` is a simple method, that just validates that it is a subclass that attempts to set the `Time` variable to a valid `TimeOptions`.

```
65          % Get function for dependent variable linking to protected
    ↪ function
66          function Period = get.Period(this)
```

```
67              Period = this.periodGet;
68          end
69
70          % set function that links to custom validation that can be
71          % overridden
72          function this = set.Parameters(this, para)
73              arguments
74                  this Hamiltonians.Interfaces.HamiltonianInterface
75                  para
76              end
77              test = this.parameterValidate(para);
78
79              this.Parameters = test;
80          end
81      end
82
83      methods(Access=protected)
84          % Default parameter validation for set function
85          function valid = parameterValidate(this,para)
86              arguments
87                  this
88                  para(1,3) double {mustBeReal}
89              end
90              valid = para;
91          end
92
93
94          % Default get function for period
95          function Period = periodGet(this)
96              arguments
97                  this Hamiltonians.Interfaces.HamiltonianInterface
98              end
99
100             Period = this.Time.Tpulse;
101         end
102     end
```

Listing 4.7: Interface for a Hamiltonian

For the method `get.Period(this)` it simply calls the custom protected function `periodGet`, this function for the most basic cases just returns the `Tpulse` variable stored in the `Time` variable. But it can be overridden if you need to include time before or after the time set for the pulse, for instance when using ramp on or off, as described later in Sec. 5.3.2.

A custom validation function also exists for the set function of the variable `Parameters` ↪ . It has the set function `set.Parameters(this, para)` that uses the custom overridable

protected function `parameterValidate(para)`. As set and get functions cannot be overridden, it enables subclasses to override the validation of parameters and customize what time domain is used.
This makes creating new Hamiltonians easy as only the code to construct it from the parameters with `createHamiltonian(this)` is needed. To have more control of what time to use you have the option to override the `timeVector` if the effect of a Hamiltonian has an offset in time, the energies involved are not applied simultaneously or the effect of the Hamiltonian stretches outside the size given in `Tpulse`. In addition, the validation function `parameterValidate` can be overridden to have custom validation of the parameters, as the default only takes a vector with three values.
As seen in Fig. 4.1 the extension of the interfaces gives an easy way to build additional Hamiltonians extending `HamiltonianInterface` that can refer to completely different energies as long as it is still hermitian, so that the unitary it can create still exists as described in Sec. 3.5.1. More complex operators as described in Sec. 3.6 must use the `LindbladInterface`.

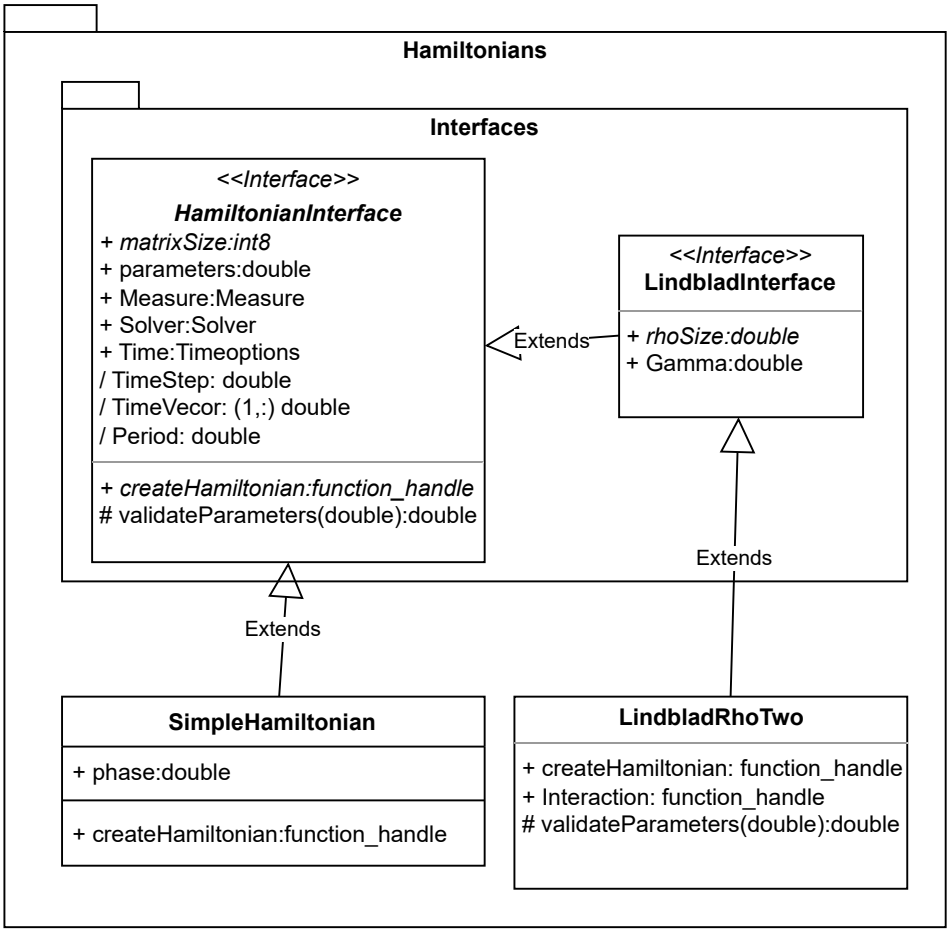

Figure 4.1: Class diagram of a Hamiltonian and a Lindbladian with interfaces, the example classes `SimpleHamiltonian` and `LindbladRhoTwo` inherit from their respective interfaces `HamiltonianInterface` and its subclass `LindbadInterface`

```
classdef (Abstract) LindbladInterface < Hamiltonians.Interfaces.
    ↪ HamiltonianInterface

    properties(Abstract, Constant, GetAccess=public)
       rhoSize int8 {mustBePositive}
    end

    properties(Access=public)
        Gamma double {mustBeReal} = 0;
    end

    methods
        function this = LindbladInterface(options)
            arguments
                options.Measure Measure = Measure.ChoiFidelity;
                options.Gamma double = 0;

            end

            this.Measure = options.Measure;
            this.Gamma = options.Gamma;
        end


        function this = set.Gamma(this, Gamma)
            % Set the lambda used by this hamiltonian
            arguments
                this Hamiltonians.Interfaces.LindbladInterface
                Gamma(1,1) double {mustBeReal}
            end

            this.Gamma = Gamma;
        end
    end
end
```

Listing 4.8: Interface for a Lindbladian

As the `LindbladInterface` inherits from the `HamiltonianInterface` as seen in Lst. 4.3. It may be much simpler, and only requires the additional parameter for rhoSize, and comes with a default constructor that changes the measure to use Choi fidelity. The optional parameter `Gamma` is used to modify how strongly the jump operators act on the qubits.

Example Hamiltonian

A straight forward implementation of the `HamiltonianInterface` seen in Lst. 4.3 is the class `SimpleHamiltonian`:

```
classdef SimpleHamiltonian < Hamiltonians.Interfaces.
    ↪ HamiltonianInterface
    properties
        Phase double {mustBeReal}
    end

    properties(Constant)
        matrixSize = 2;
    end


    methods
        function this = SimpleHamiltonian(options)
            % A simple hamiltonian, of the form:
            % H =   [-epsilon/2                 omega*sin(t*Phase)
            %         conj(omega)*sin(t*Phase)   epsilon/2]
            %
            % Takes the optional name value parameters Time and Phase
            %
            % Time is the TimeOptions used by this hamiltonian
            %
            % Phase is the Phase used by this hamiltonian to shift the
            % frequency the magnetic fields x and y oscillates

            % Input validation and default values
            arguments
                options.Time TimeOptions = TimeOptions;
                options.Phase double {mustBeNonzero} = 1;
                options.Parameters(1,3) double {mustBeReal} = zeros(1,3)
    ↪ ;
            end

            this.Time = options.Time;
            this.Phase = options.Phase;
            this.Parameters = options.Parameters;
        end
```

Listing 4.9: A Hamiltonian for one qubit

It contains an additional optional parameter `Phase` as seen in Eq. (3.7). This is used if you want to change how fast you move in the sinus cycle, as it multiplies the time `t` as seen in Lst. 4.10 thereby changing the frequency of the sinus cycle. The result of running a simulation, after changing the frequency, will change and be

harder to physically implement, if the sinus cycle does not end with a difference from its starting time, given in `TimeOptions`, of $n \cdot 2\pi$ for $n \in \mathbf{N}$. The only other variable it contains is an implementation of the `matrixSize`, a mandatory variable that tells the rest of the program the size of the Hamiltonian created with the function `createHamiltonian` shown in Lst. 4.10 below.

The constructor then does some simple validation of the given parameters and provides a default if they are not provided.

```
36          function H = createHamiltonian(this)
37              % Creates a Hamiltonian with the parameters provided and the
38              % values stored in this instance of the class
39              %
40              % parameters must be in the form of a (1,3) double vector,
    ↪ with
41              % real numbers. It contain epsilon, omegaX and omegaY.
42              %
43
44
45              % Input validation
46              arguments
47                  this Hamiltonians.SimpleHamiltonian
48              end
49
50              epsilon = this.Parameters(1,1);
51              omegaX = this.Parameters(1,2);
52              omegaY = this.Parameters(1,3);
53              phase = this.Phase;
54              omega = omegaX + 1i*omegaY;
55
56  %             % Setup parameters
57  %             B1 = @(t) 2*omegaX*sin(Phase*t);
58  %             B2 = @(t) -2*omegaY*sin(Phase*t);
59  %             B3 = @(t) -epsilon;
60  %
61  %             % Creating the hamiltonian
62  %             H = this.pauliRotations(B1,B2,B3);
63              H = @(t)  [-epsilon/2                  omega*sin(t*phase);...
64                      conj(omega)*sin(t*phase)   epsilon/2];
65
66          end
67
68
69
70          function this = set.Phase(this, Phase)
71              % Set the Phase used by this hamiltonian
72              arguments
```

```
                this Hamiltonians.SimpleHamiltonian
                Phase(1,1) double {mustBeReal}
            end

            this.Phase = Phase;
        end

    end
end
```

Listing 4.10: A Hamiltonian for one qubit

`createHamiltonian` is the main function of the class. It takes the provided parameters and uses them to create a Hamiltonian returned as a function handle. This function handle will take the time parameter `t` and return the state of the Hamiltonian at that time.
`set.Phase(this,Phase)` is simply a small validation when setting the phase in an instance of the class.

Example Lindbladian

An example of an implementation of the interface `LindbladInterface` is the class `LindbladRhoOne` found below in Lst. 4.11. It is an implementation of Eq. (3.30), the decay to ground state for one qubit.

```
classdef LindbladRhoOne < Hamiltonians.Interfaces.LindbladInterface
    properties(Constant)
        matrixSize = 2;
        rhoSize = 2;
    end


    methods
        function this = LindbladRhoOne(options)
            % A matrix derived from the lindblad equation
            % Takes the optional name value parameters Rho, Time and
    ↪ Gamma
            %
            % Time is the TimeOptions used by this hamiltonian
            %
            % Gamma is an optional parameter with default value of 0. It
            % scales how large the effect reducing the state of the
    ↪ system
            % towards basis state [1;0], it is the interference
    ↪ experienced
            % by the system.

```

```
            % Input validation and default values
            arguments
                options.Time TimeOptions = TimeOptions;
                options.Gamma double {mustBeReal} = 0;
                options.Parameters(1,3) double {mustBeReal} = ones(1,3);
                options.Measure = Measure.ChoiFidelity;
                options.Solver = HamiltSettings.Solvers.Runge_Kutta_Rho;
            end
            this.Time = options.Time;
            this.Gamma = options.Gamma;
            this.Parameters = options.Parameters;
            this.Measure = options.Measure;
            this.Solver = options.Solver;
        end

        function lindblad = createHamiltonian(this)
            % Creates a Hamiltonian with the parameters provided and the
            % values stored in this instance of the class
            %
            % parameters must be in the form of a (1,3) double vector,
    ↪ with
            % real numbers. It contain epsilon, omegaX and omegaY.
            %



            % Input validation
            arguments
                this Hamiltonians.Interfaces.LindbladInterface
            end

            epsilon = this.Parameters(1,1);
            omegaX = this.Parameters(1,2);
            omegaY = this.Parameters(1,3);
            Gamma = this.Gamma;
%             rho = sparse(2,2);
%             rho(this.Rho) = 1;

            % Setup parameters
            B1 = @(t) omegaX*sin(t);
            B2 = @(t) 1i*omegaY*sin(t);
            Omega = @(t) B1(t)+B2(t);


            % Creating hamiltonian
            H = @(t) [-epsilon/2 Omega(t); conj(Omega(t)) epsilon/2];
            a = [1; 0]*[0 1];
```

```
            A = a'*a;

            % creatin components of the Lindbladian
            lindblad = @(t,rho) -1i*(H(t)*rho - rho*H(t))...
                -Gamma*(A*rho + rho*A -2.*a*rho*a');
        end


    end
end
```

Listing 4.11: A Lindbladian for one qubit

For properties, it only implements the two mandatory abstract properties of its superclass `matrixSize` and `rhoSize`.
Most of the work for the class is done in the method `createHamiltonian`, where a regular Hamiltonian dependent on time `t` is created as seen described in Sec. 3.3. This Hamiltonian is used with `rho` and the jump operators modified by `Gamma` as seen in the Lindblad Eq. (3.26), in this specific case Eq. (3.27) both described in Sec. 3.6. `createHamiltonian` creates a function handle that takes a state `rho` and time `t` to return the state of the Lindbladian for a given time `t` and ρ.

4.1.4 Implementing noise

To ensure the noise behaves in a predictable manner, I created a class that contains the needed methods to prepare the noise. In the constructor, it takes sub-classes to the abstract class Lst. 4.12. It only contains a few abstract methods that declare how to access the jump operators, some precalculated values from the jump operators as seen in Eq. (3.34), and the individual gamma values that describe how strong the different jump operators are in relation to each other.

```
classdef(Abstract) TwoParticleNoiseInterface
    methods(Abstract)
        getL
        getD
        getGamma
    end
end
```

Listing 4.12: Abstract class for the value of jump operators in a two qubit Lindblad equation

The values from subclasses that inherit from Lst. 4.12 are then used in Lst. 4.13. It calculates the effect of the jump operator using the precalculated value of `this.D` with a provided `rho`, to find the effect of the jump operators for a given *rho*, and it gives the effect of all jump operators seen in Eq. (3.26).

```
classdef LindbladNoise
    % Class with methods to construct a two particle noise with some pre
    % calculated values, for a noise source specified during
    ↪ construction
    properties(SetAccess = immutable)
        L (4,4,4) double
        D (4,4) double
        Gamma (4,1) double
    end


    methods(Access = public)
        function this = LindbladNoise(NoiseType)
            % Constructor that takes the noise from the noise class and
            % saves some pre calculated output for later use
            arguments
                NoiseType LindbladNoise.TwoParticleNoiseInterface
            end
            this.L = NoiseType.getL;
            this.D = NoiseType.getD;
            this.Gamma = NoiseType.getGamma;
        end

        function noise = getNoise(this, rho)
            %Returns the noise for a given rho

            arguments
                this LindbladNoise.LindbladNoise
                rho(4,4) double
            end

            L_sum = zeros(4,4);

            for n = 1:4
                L_n = this.L(:,:,n);
                gamma_n = this.Gamma(n);
                L_sum = L_sum + gamma_n .* L_n * rho * L_n';

            end

            noise = this.D*rho+rho*this.D - 2*L_sum;
        end
    end

end
```

Listing 4.13: Class for jump operators for a two qubit Lindbladian

Some subclasses to Lst. 4.12 are Lst. 4.14, containing noise captured with tomography in a real quantum computer (Samach et al., 2022). While Lst. 4.15 is an example of randomly generated noise created in my program using Lst. 4.16.

```
classdef TomographyNoise < LindbladNoise.TwoParticleNoiseInterface
    % Class for noise levels extracted from the research paper Lindblad
    % Tomography of a Superconducting Quantum Processor, as jump
    ↪ operators
    % and their weights
    properties(Constant)
        L1 = [ 0.501+0.001i 0.000+0.001i -0.0002+0.002i -0.002+0.000i;
0.000-0.001i 0.499-0.001i -0.001+0.001i -0.002+0.002i;
-0.001-0.001i 0.001-0.001i -0.499+0.000i 0.000+0.001i;
0.002+0.000i -0.001-0.001i 0.000+0.000i -0.501+0.000i];
        L2 = [0.448-0.001i 0.109+0.246i 0.001-0.002i 0.002-0.002i;
0.061-0.125i -0.451+0.002i -0.001+0.002i 0.002+0.005i;
-0.003-0.002i -0.003+0.001i 0.454+0.001i 0.109+0.249i;
-0.004+0.001i 0.000+0.002i 0.064-0.131i -0.451-0.002i];
        L3 = [ -0.072+0.161i 0.639-0.031i -0.003-0.001i 0.007+0.00i;
0.114+0.039i 0.072-0.161i -0.001+0.001i -0.005-0.001i;
0.002+0.003i 0.001+0.002i -0.071+0.162i 0.655-0.044i;
0.002-0.003i -0.001-0.002i 0.134+0.035i 0.071-0.163i]
        L4 = [ 0.004+0.000i -0.001-0.003i 0.000-0.703i -0.003+0.00i;
0.000-0.001i 0.000-0.002i -0.004-0.002i 0.000-0.703i;
0.001+0.078i 0.002+0.001i 0.000+0.002i -0.002-0.003i;
0.000-0.001i -0.001+0.079i 0.000-0.001i -0.004+0.000i];
        gamma1 = 0.071;
        gamma2 = 0.097;
        gamma3 = 0.042;
        gamma4 = 0.055;
    end

    methods(Static)
        function L = getL
            L = cat(3, LindbladNoise.TomographyNoise.L1,...
                LindbladNoise.TomographyNoise.L2,...
                LindbladNoise.TomographyNoise.L3,...
                LindbladNoise.TomographyNoise.L4);
        end

        function D = getD
            D = LindbladNoise.TomographyNoise.gamma1*LindbladNoise.
    ↪ TomographyNoise.L1'*LindbladNoise.TomographyNoise.L1 +...
                LindbladNoise.TomographyNoise.gamma2*LindbladNoise.
    ↪ TomographyNoise.L2'*LindbladNoise.TomographyNoise.L2 +...
                LindbladNoise.TomographyNoise.gamma3*LindbladNoise.
    ↪ TomographyNoise.L3'*LindbladNoise.TomographyNoise.L3 +...
```

```
                LindbladNoise.TomographyNoise.gamma4*LindbladNoise.
    ↪ TomographyNoise.L4'*LindbladNoise.TomographyNoise.L4;
        end

        function Gamma = getGamma
            Gamma = [LindbladNoise.TomographyNoise.gamma1,...
                LindbladNoise.TomographyNoise.gamma2,...
                LindbladNoise.TomographyNoise.gamma3,...
                LindbladNoise.TomographyNoise.gamma4];
        end
    end
end
```

Listing 4.14: Jump operators created by tomography

```
classdef GeneratedNoiseTraceless < LindbladNoise.
    ↪ TwoParticleNoiseInterface
    % A noise generated using the GenerateRandomNoise script using
    ↪ default
    % parameters
    properties(Constant)
        L = [0.0452220476344547 + 0.389719909673081i...
          -0.996094096614954 - 0.409057845790514i...
          0.176601628617548 + 1.00038318883897i...
          1.81692224917996 - 2.13346755941796i;...

            0.867232157629573 - 0.0595506104590645i...
        0.0602519096772833 - 0.906304239609380i...
        0.755179935677488 - 0.801329298311214i...
        -0.123833350279246 + 0.403977280549421i;...

            0.975986463472540 - 0.661011014460339i...
        -1.29744747716739 - 0.284902829463087i...
        -0.00803401840821616 + 0.467350598527234i...
        -1.11060135506064 - 0.958499409348658i;...

            0.337390252379212 + 0.305950915068987i...
        0.917388589145936 - 0.0647868558912201i...
        1.84438963700050 + 0.273804791937703i...
        -0.0974399389035219 + 0.0492337314090649i]
        Gamma = 1;
    end

    methods(Static)
        function L = getL
            % The jump operators, using repmat to replicate the L to
    ↪ return
            % a 4x4x4 matrix to simulate four independent noises
```

```
            L = repmat(LindbladNoise.GeneratedNoiseTraceless.L,1,1,4);
        end

        function D = getD
            % Pre calculates the state independent noise, multiplies the
    ↪  D
            % value to get same effect as four independents noises
            tmp = LindbladNoise.GeneratedNoiseTraceless.Gamma...
                *LindbladNoise.GeneratedNoiseTraceless.L'...
                *LindbladNoise.GeneratedNoiseTraceless.L;
            D = tmp*4;
        end

        function Gamma = getGamma
            Gamma = repmat(LindbladNoise.GeneratedNoiseTraceless.Gamma,
    ↪ 4,1);
        end
    end

end
```

Listing 4.15: Jump operators created randomly

These simple classes could be streamlined to inherit `getL` and `getD` from an abstract superclass, but as there are only two in use, it was not seen as a priority.

```
function noise = GenerateRandomNoise(options)
% Function to generate random traceless noise with optional random seed
    ↪ and
% size of the noise
arguments
    options.Seed = 42;
    options.MatrixSize = 4;
end

% initialize variables
seed = options.Seed;
matrixSize = options.MatrixSize;

% set the random number generator
s = rng(seed);

% Create random noise
L = randn(matrixSize) + 1i*randn(matrixSize);

% Make the noise traceless and return it
```

```
20 noise = L - trace(L)*eye(matrixSize)/matrixSize;
```

Listing 4.16: Creating random jump operators

In Sec. 3.6.5 I discussed how to create a randomly generated traceless noise. To generate the jump operators in Lst.4.15 I used a normally distributed random noise for each value. Then by creating a 4×4 matrix and a random seed to guarantee replicability of the creation of the jump operators, mirroring what is described in Eq. (3.35). Finally, I removed the trace by using the procedure in Eq. (3.36) and I removed the sum of the trace divided by the matrix dimension.
This then gives one jump operator, this procedure can be repeated to get any number of jump operators for a two qubit system, or can be used as a single jump operator.

4.1.5 Solvers

There are several possible solvers, and to keep track of all of them I've been using an enum, linking the name of the method to the method itself. This makes it easier to give the Hamiltonian class the appropriate function to solve itself. This enables easy addition of more solve methods as it only needs to be added to the enum and implement the appropriate interface.

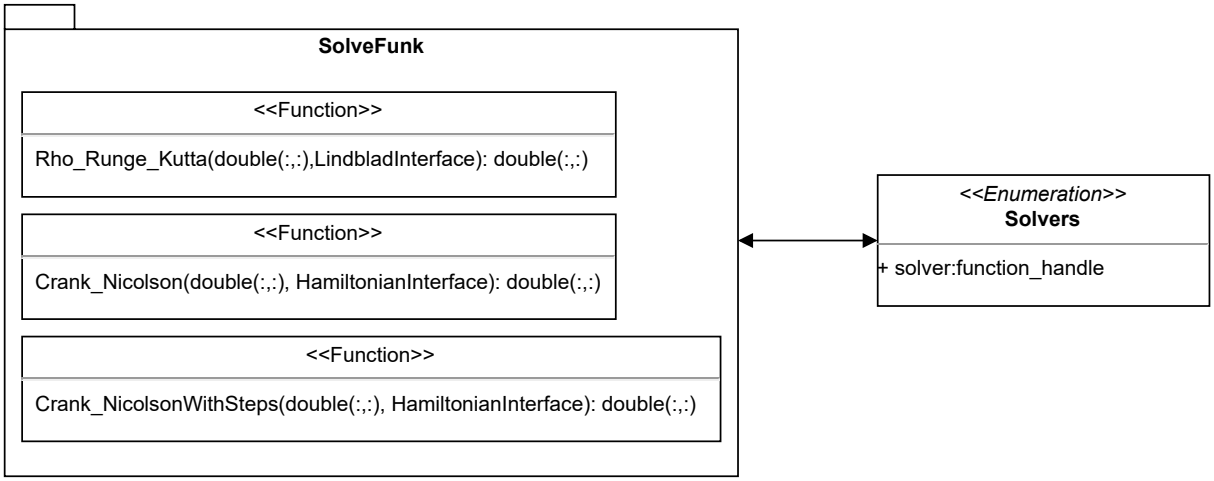

Figure 4.2: Class diagram of the solvers in the enum

This enables ease of use of different solvers using the same files and the same functions. The solvers must only be called to solve a specific Hamiltonian that it is compatible with, as there is no compatibility check, otherwise, unexpected behaviour may occur. All functions take two parameters: A state to propagate and what Hamiltonian to simulate. These functions are called each time a position is to be evaluated, and are stored in the Hamiltonian class.

```
1 classdef Solvers
```

```
    % Enum to hold all solvers in project
    properties
        solver function_handle
    end

    methods
        function con = Solvers(solver)
            con.solver = solver;
        end
    end

    enumeration
        Crank_Nicolson(@(psi,hamilt) SolveFunk.Crank_Nicolson(psi,hamilt
    ↪ ))
        Crank_Nicolson_with_steps(...
            @(psi,hamilt) SolveFunk.Crank_NicolsonWithSteps(psi,hamilt))
        Runge_Kutta(@(rho, hamilt) SolveFunk.Runge_Kutta(rho, hamilt))
        Runge_Kutta_Rho(@(rho, hamilt) SolveFunk.Rho_Runge_Kutta(rho,
    ↪ hamilt))
    end
end
```

Listing 4.17: Enum of all solvers

It is important to make sure that the solving functions are compatible with the `Hamiltonian` they are used to solve. Most Lindblad equations can not be solved the same way as a Hamiltonian, because most have a square matrix ρ instead of an array Ψ to show the states.

The Crank-Nicolson solver

The Crank-Nicolson solver is here used to solve the differential equation that is created when using the chosen Hamiltonian in the Schrödinger equation.

```
function [Solution] = Crank_Nicolson(Psi, Hamiltonian)
% Solves the schrödinger equation with Crank Nicolson for a specified
% Hamiltonian and startvalues Psi.


% Input validation
arguments
    Psi (:,1) double
    Hamiltonian Hamiltonians.Interfaces.HamiltonianInterface;
end

% Get parameterized hamiltonian
H = Hamiltonian.createHamiltonian;
```

```
14
15 % Set timelength
16 dt = Hamiltonian.TimeStep;
17 tVector = Hamiltonian.TimeVector;
18
19 % Find sizes
20 leng = length(tVector);
21 psiHeight = size(Psi,1);
22
23
24 % Crank Nicolson solver
25 I = eye(psiHeight);
26 Phi = @(t,y) (I + 1i*dt/2*H(t+dt))^(-1) * (I - 1i*dt/2*H(t))*y;
27
28 for n = 1:leng
29     Psi = Phi(tVector(n),Psi);
30 end
31
32 Solution = Psi;
```

Listing 4.18: Crank-Nicolson solver

The solver based on Crank-Nicolson that is used here is dependent on two parameters t time and y the state of the system, dt is the size of the time steps and the Hamiltonian H is dependent on time, while I is an identity matrix the same size as the Hamiltonian. It will go through all the timesteps in the dependent property `TimeSteps` found in the `HamiltonianInterface` in Lst. 4.5. For each timestep, it will use the Crank-Nicolson solver in the function `Phi` to propagate the state `Psi` to its new value at time `t`. This new state `Psi` is then used for the next step which is propagated by the effect of the Hamiltonian to its next value at a new point in time. `psiHeight` will in most cases be the Hilbert dimension of the quantum system 2^n, it will for example in a one qubit system be 2, therefore `Psi` will in that case be a $2x1$ matrix, and is the end state of the system with each value representing the probability density for a different state. Here `Psi(1)` is used to get the probability, by taking the square of the absolute value of it as the probability for up, and the square of the absolute value of `Psi(2)` as down, after the system has been transformed by the dynamics of the Hamiltonian, as described in Sec. 2.2.1.

Crank-Nicolson with steps

This is the same function as Lst. 4.18 but keeping every step of `Psi` and returning all of them.

```
1 for n = 1:leng
2     Psi = Phi(tVector(n),Psi);
```

```
    Y(:,n) = Psi;
end

Solution = Y;
```

Listing 4.19: Cranc Nicolson solver with steps

The solution will be the states `Y(1, end)` and `Y(2, end)`, while `Y` contains all steps to see the evolution of the state with time. The evolution of `Y(1, :)` and `Y(2, :)` enables you to see how the probability for each state evolves with time, thereby giving a better view of how the state is propagated by the Hamiltonian.

Runge Kutta Rho

`Rho_Runge_kutta` is based on the theory described in Sec. 3.7.2 is of fourth order so it should be as accurate as Crank-Nicolson, even with larger time steps, enabling faster calculations to find a solution.

```
function [Solution] = Rho_Runge_Kutta(Rho, Hamiltonian)
    % Solves the schrödinger equation with the Runge Kutta method for a
    % specified Hamiltonian and a given startvalue Rho.


    % Input validation
    arguments
        Rho (:,:) double
        Hamiltonian Hamiltonians.Interfaces.LindbladInterface;
    end


    % Function for step in Runge Kutta
    function step = rk_algorithm(t, rho)
        k1 = H(t, rho);

        rho2 = rho+dt*k1/2;
        k2 = H(t+dt/2, rho2);

        rho3 = rho+dt*k2/2;
        k3 = H(t+dt/2, rho3);

        rho4 = rho+dt*k3;
        k4 = H(t+dt, rho4);

        step = rho + 1/6*(k1 + 2*k2 + 2*k3 + k4)*dt;
    end

    % Get parameterized hamiltonian
```

```
30      H = Hamiltonian.createHamiltonian;
31
32
33      % Set timelength
34      dt = Hamiltonian.TimeStep;
35      tVector = Hamiltonian.TimeVector;
36
37      % Find sizes
38      leng = length(tVector);
39
40      for n = 1:leng
41          Rho = rk_algorithm(tVector(n),Rho);
42      end
43      Solution = Rho;
44  end
```

Listing 4.20: Runge-Kutta solver for rho

Runge Kutta takes as input to the function a square matrix `Rho`, an initial density matrix, to evolve for and a `Hamiltonian` based on the Lindblad equation described in Sec. 3.6 with the energies in the system, and extracts a vector with the time steps after creating a function handle with `Hamiltonian.createHamiltonian`. The algorithm makes `k2` ↪ `, k3, k4` based on the previous `k` starting with `k1`, and combines it with the initial state `Rho`. The result of using the completed algorithm gives the evolution of `Rho` after one timestep, and is repeated for all timesteps, using the result of the previous step as input in the next until it reaches the end state, after having simulated the effect of the `Hamiltonian` working on the state `Rho` for a period of time, defined by the timesteps given in `Hamiltonian.TimeVector`.

4.1.6 Solving for all basis states

When you solve a Hamiltonian or a Lindbladian, you want to solve it for all possible basis states, which for a Hamiltonian is a state vector described in Sec. 2.2.1. You solve for all possible basis states by solving for a state vector starting in each state, and combine the solution for the different basis states to recreate the whole unitary matrix. It is the same for a Lindbladian, but as it uses a density matrix for the basis states as discussed in Sec. 3.4, it must be solved for all possible basis states in the density matrix, to capture all of the effects the energies of the Lindbladian have on the state given in the density matrix. For a given `Hamiltonian` I can find the solution for all given start values in `Psi0` using:

```
1  function U = SolveForStartValue(Hamiltonian, Psi0)
2  % Solves a Hamiltonian for all inputs in Psi. Where Psi0 is all possible
3  % states as column vectors in a matrix format
```

```
    arguments
        Hamiltonian Hamiltonians.Interfaces.HamiltonianInterface
        Psi0 double
    end

    % Setup matrices
    Hsize = Hamiltonian.matrixSize;
    index = size(Psi0, 2);
    U = zeros(Hsize,index);

    % Getting the solution for the start positions
    for n = 1:index
        [Psi] = UseSolver(Psi0(:,n), Hamiltonian);
        U(:,n) = Psi;
    end
end
```

Listing 4.21: Solving for all psi values for a given gate

Lst. 4.21 uses a solver from the enum in Lst. 4.17 inside the function `UseSolver`, a simple function that only uses the included differential equation solver.
`Psi0` is given as input and can be a column vector or a square matrix, where each column in the matrix is a possible start state for the `Hamiltonian`. To be a valid state the column has to be the same height as the Hamiltonian.

4.2 Implementing the Measure

In this project, I have three different ways to measure the difference between the effect a Hamiltonian has on possible states and an ideal gate or the effect a Lindbladian has on the possible states and an ideal gate.

4.2.1 Norm measure

This is the implementation of the norm measure discussed in Sec. 3.8.1.

```
% Compare distance for a gate U with a given gate

% Start positions
Psi0 = Gate.Psi0;

% Setup target gate
targetGate = Gate.gate;


% Solve with startvalues in Psi0
```

```
U = SolveForStartValue(Hamiltonian, Psi0);

% Rotate to reduce effect of global phase
U = Gate.rotation(U);

% Measuring distance
Diff = norm(U-targetGate);
```

Listing 4.22: Gate comparison in Matlab

When measuring the effect a one qubit Hamiltonian has on a state, I solve the Schrödinger equation twice, once for each basis state, to create the unitary propagator by combing the solutions in `U`. This propagator shows the effect the Hamiltonian has on one qubit, and is then compared to the effect of a gate. `Gate.rotation(U)` from the `GateInterface` in Lst. 4.1 is used to change the phase of the unitary matrix, so that it has the same global phase as the gate it is compared, as described in Sec. 3.8.1. After this, the target gate is subtracted from the unitary matrix and the euclidean norm is extracted from the difference between them. It will reach a value up to 2 for different gates and the value 0 for gates with identical effects on the particles.
The Matlab implementation of the norm for matrices, is given by the largest singular value decomposition, and it is also called the 2-norm.

```
gate = [0 1;1 0];
```

Listing 4.23: X gate in Matlab

The above is an example of a target gate in Matlab, here the x gate as seen in Eq. (2.21a).

Accuracy

As the norm measures the difference between the target gate and the created quantum map, it is as good as the machine precision in your simulation program. This means it can get as low as 0 for identical gates, but is limited to 10^{-15} for any case where you do not end up with a zero matrix after subtracting the matrices. The norm measure can be accurate up to the machine precision of the simulations. This can for some gates, have an error with size of 10^{-15}. The measurement will be in the interval $[0, 2]$, with 0 being perfect match. A problem is that a global phase of the solution can distort how accurate it is, but by changing the phase of all elements, so that one of the elements matches the target gate, the effect can be reduced.

4.2.2 Average gate fidelity

The implementation of the average gate fidelity measurement method, which is discussed in Sec. 3.8.2.

```
% Compare distance for a gate U with a given gate

% Start positions
Psi0 = Gate.Psi0;
index = double(Hamiltonian.matrixSize);


% Solving the shrödinger equation
U = SolveForStartValue(Hamiltonian, Psi0);

% loading the gate matrix
targetGate = Gate.gate;

% Average fidelity
Diff = 1-(abs(trace(U'*targetGate))^2 +index)/(index^2 +index);
```

Listing 4.24: Average gate fidelity

I use the `Psi0` from the gate to get a matrix with all basis states as described in Sec. 4.1.6, I take individual array slices of the matrix to get all the basis states of the qubit. Then as in Sec. 4.2.1, I solve the Schrödinger equation with the Hamiltonian for all the basis states of the qubit, using the function `SolveForStartValu` found in Lst. 4.21, where all the solutions are combined to the propagator `U`. Then I use the average gate fidelity as discussed in Sec. 3.8.2, to find the fidelity. Thereafter I return the infidelity by subtracting the fidelity from 1. The value of the measure will be in the interval $[0, 1]$, with 1 for perfect fidelity, but since I use infidelity as a measurement for the cost function I have $1 - fidelity$.
When I used this measure it gave smoother cost landscapes than when using the norm to measure the difference. Therefore the measurement I focused on using for the Hamiltonians, was the average gate fidelity.

Accuracy

The average gate fidelity is an accurate method to find the difference between two quantum states. It can for normal use have an error of 10^{-13}, but when comparing a one particle Hadamard gate with itself it can get as low as $3.3307 \cdot 10^{-16}$. The measurement method uses matrix multiplication, so it can only be completely accurate for some gates with integer values, but it gets close enough for the purpose of my project.

4.2.3 Channel fidelity with Choi matrix

This is the implementation of the channel fidelity for a Choi matrix, discussed in Sec. 3.8.3.

```
function Diff = ChoiMatrixMeasure(Lindbladian, Gate)
% Compares the difference of a Lindbladian with a given gate using a
    ↪ Choi
% matrix and returns the fidelity

    arguments
        Lindbladian Hamiltonians.Interfaces.HamiltonianInterface
        Gate Gates.GateInterface
    end

    % Extract gate
    targetGate = Gate.gate;

    % Get matrix size from gate
    index = size(targetGate, 2);

    % Setup values
    matrixSize = index^2;
    scale = 1/index;


    % initialize matrix
    ChoiPart = zeros(index, index, matrixSize, matrixSize);
    targetPart = zeros(index, index, matrixSize, matrixSize);

    % Solve for all rho values
    for n = 1:index
        for m = 1:index
            if(n>m)
                % Use the fact the parts are symmetrical around the
                % diagonal to reduce runtime with the upper triangular
                % matrix to fill in the lower triangular
                Rho = sparse(m, n, 1, index, index);
                lindbladSolution = SolveFunk.SolveForRho(Lindbladian,
    ↪ Rho, index);
                part = kron(lindbladSolution, Rho);
                ChoiPart(m,n,:,:) = part;
                ChoiPart(n,m,:,:) = ctranspose(part);

                tPart = kron(targetGate*Rho*targetGate', Rho);
                targetPart(m,n,:,:) = tPart;
                targetPart(n,m,:,:) = ctranspose(tPart);
            elseif(n==m)
                % Adding the diagonal parts directly
                Rho = sparse(m, n, 1, index, index);
                lindbladSolution = SolveFunk.SolveForRho(Lindbladian,
    ↪ Rho, index);
```

```
                ChoiPart(m,n,:,:) = kron(lindbladSolution, Rho);

                targetPart(m,n,:,:) = kron(targetGate*Rho*targetGate',
    ↪ Rho);
            end
        end
    end

    % Sums and reduces the parts of the choi matrix
    Choi = sum(ChoiPart, [1, 2]);
    Choi = squeeze(Choi);
    Choi = (Choi.*scale);
    sqrChoi = sqrtm(Choi);

    % Sums and reduces the kron products of the target gate
    TargetChoi = sum(targetPart,[1, 2]);
    TargetChoi = squeeze(TargetChoi);
    TargetChoi = TargetChoi.*scale;

    content = sqrtm(sqrChoi*TargetChoi*sqrChoi);


    % Finds fidelity
    Diff = abs(trace(content)^2);

end
```

Listing 4.25: Calculating channel fidelity with Choi matrix

To try and reduce the amount of computations that are needed, I took advantage of the fact that the Lindbladians I use are Hermitian, the Hermitian is discussed in Sec. 3.3. This enables only computing the upper triangular part of the matrix, and placing the conjugate of the elements above the diagonal in the lower triangular part of the matrix. This could reduce the time needed to solve the Lindblad equation, as for a 2 qubit Lindbladian the Runge-Kutta solver would have to be used 16 times. But with this optimization, it would only need to be used 10 times.

Accuracy

Channel fidelity suffers more from inaccuracy than the other measurement types used in the project, as it computes the matrix square root. This inaccuracy means that the best possible measurements will have an error of at least 10^{-8}, no matter what method is used, as long as the datatype double is used, and default precision computations are used.

4.2.4 Tautological gates and benchmark accuracy

To have an indication of the accuracy of the simulation, I have in some cases used tautological gates as the target gates. The name comes from its self referencing nature, as I construct it by using a Hamiltonian I created, with set parameters, and then use it as a target gate and compare it to the effect of the Hamiltonian.
The tautological gates in this project are constructed by running a Hamiltonian with all input parameters set to 1. This is then used and compared with itself to see how good this measurement is when I compare the result of the measurement with different measurement methods. While it is not a gate realistically used, it can give a good indication of where to start looking. But I still have the added burden of finding a good minima with gradient descent for a gate that is actually used. It guarantees that a solution exists when optimizing using an optimization algorithm. By giving an indication of how hard it is to search for all the possible solutions, it will also see how often gradient descent has a tendency to get stuck in local minima.
For creating a one qubit tautological gate I use Eq. (3.7) setting all input parameters, $(\epsilon, \Omega_x, \Omega_y)$ to one. Meanwhile, for the two qubit tautological gate, I used Eq. (3.12) setting $(\epsilon, \Omega_x, \Omega_y, u)$ to one.

4.3 Gradient descent

I use a variant of gradient descent called the Adam solver described in 3.9.3, with my implementation as follows:

```
arguments
   Hamiltonian Hamiltonians.Interfaces.HamiltonianInterface
   Gate Gates.GateInterface

   options.learning double = 1e-3;
   options.Beta1 double {mustBeInRange(options.Beta1,0,1,"exclude-upper
   ↪ ")} = 0.9;
   options.Beta2 double {mustBeInRange(options.Beta2,0,1,"exclude-upper
   ↪ ")} = 0.999;
   options.epsilon double {mustBePositive} = 1e-8;
   options.minDiff double {mustBePositive} = 1e-6;
   options.maxIter = 1000;
end

% setting initial values
learning = options.learning;
Beta1 = options.Beta1;
Beta2 = options.Beta2;
epsilon = options.epsilon;
```

```
cutoff = Hamiltonian.Measure.cutoff;
minDiff = options.minDiff;
last = MeasureDiffGeneral(Hamiltonian, Gate=Gate);


% Max intervals
maxIt = options.maxIter;

% Initialise momentum vector
para = Hamiltonian.Parameters;
leng = length(para);
M = zeros(1,leng);
V = zeros(1,leng);
```

Listing 4.26: Adam solver

The algorithm has two required parameters, `Hamiltonian` and `Gate` the Hamiltonian to solve for and the target gate. It has some optional parameters: `learning` the learning rate, `Beta1` and `Beta2` used for calculating the moment and second moment, `epsilon` a small value with the default recommended value to be 10^-8, a `minDiff` that denotes the minimum required difference after 100 iterations and finally `maxIter` to denote the maximum number of iterations. From the `Measure` to the `Hamiltonian` it extracts a `cutoff` ↪ that is set based on the measurement used, this cutoff stops the gradient descent if the infidelity is more than halfway of the maximum after a hundred steps. As it may take a long time to get to a minima and as the runtime of the program is already high, I decided to focus on the start points closer to a local minima.

```
for iter = 1:maxIt
    grads = CalculateGradients(Hamiltonian, Gate);
    M = Beta1*M + (1-Beta1)*grads;
    V = Beta2*V + (1-Beta2)*grads.^2;

    % Compute bias-corrected first and second moment estimate
    M_bias = M./(1-Beta1^iter);
    U_bias = V./(1-Beta2^iter);


    para = para - learning.*M_bias./(sqrt(U_bias)+epsilon);


    Hamiltonian.Parameters = para;
```

Listing 4.27: Adam solver

Here is the algorithm itself, starting with calculating `M` used for the first moment and `V` for the second. The bias corrections `M_bias` and `U_bias` is used to counteract the fact the algorithm is dominated by the starting values of the momentum `M` and `V` that are zero.

```
    if mod(iter,100) == 0
        test = MeasureDiffGeneral(Hamiltonian, Gate=Gate)
        if(test > cutoff || abs(test-last) < minDiff)
            disp(iter)
            break
        else
            last = test;
        end
    end
end


distance = MeasureDiffGeneral(Hamiltonian, Gate=Gate);
```

Listing 4.28: Adam solver

Here are the early stop conditions. They trigger every hundred iterations and stop the optimization if the difference in measurement from the last test, a hundred iterations ago, is less than the `minDiff`, or if the value is higher than the cutoff.

```
    % Validate input
    arguments
        Hamiltonian Hamiltonians.Interfaces.HamiltonianInterface
        Gate Gates.GateInterface
        options.h(1,1) double = 1e-3;
    end

    h = options.h;

    % Extract parameters
    para = Hamiltonian.Parameters;

    % initialize vector
    vectorLength = length(para);
    Gradients = zeros(1,vectorLength);

    % For each parameter, find it's gradient with middle finite
    % difference scheme
    for n = 1:vectorLength
        [plussVector, minusVector] = CalculateVectors(para, h, n);
        Hamiltonian.Parameters = plussVector;
        Hpluss = MeasureDiffGeneral(Hamiltonian, Gate=Gate);

        Hamiltonian.Parameters = minusVector;
        Hminus = MeasureDiffGeneral(Hamiltonian, Gate=Gate);

        Gradients(n) = (Hpluss - Hminus)/(2*h);
    end
```

```
end
```

Listing 4.29: Midpoint approximation for all possible parameters

This function uses the midpoint approximation to differentiate and a small sub function to setup the parameters that are to be differentiated.
It can also take an optional scalar `h` that describes how big the difference used in the final difference method should be.

```
    % Sets up vectors
    plusVector = parameters;
    plusVector(location) = plusVector(location) + h;

    minusVector = parameters;
    minusVector(location) = minusVector(location) - h;
end
```

Listing 4.30: Adding and subtracting the value h from the parameter to use in the midpoint approximation

Lst. 4.30 is a small script that only prepares the parameters by adding and subtracting the small value `h` on the chosen parameter location in the vector.

4.4 Experiments

To test my program and work towards a result I started simple, before expanding to using the Lindblad equation as described in Sec. 3.6. First I ran tests on one qubit gates before using two qubit gates with interactions.

4.4.1 One qubit gates

I started the project by constructing some known gates and trying to optimize the simple one qubit Hamiltonians with gradient descent. I worked to understand the influence of the parameters before expanding the project. The first gates used as targets were the X gate and Hadamard gate.

Hadamard

Optimizing for the Hadamard gate was difficult with my program, it was too dependent on the starting position for me to be sure that I had a good global minima. So most of the testing ended up being done with tautological gates.

Tautological gate

Even with a large number of random start points, I only got a few good approximations for the Hadamard gate. These could have an infidelity of 10^{-5} a reasonably good result, but not the theoretical best that the average gate fidelity measurement described in Sec. 3.8.2 could have. So, to be certain of my test, I started using tautological gates. I constructed gates by entering known parameters, most often by setting all parameters to 1, and seeing how good the program was to get back to these points with minor changes in parameters using gradient descent.

This gave me a good idea of how good the gradient descent algorithm was for finding minima and was a starting point where I could look at more realistic gates if I found a method that showed promising testing results.

4.4.2 Lindblad with noise

To see how a quantum system exposed to noise behaved, I started with the Lindblad equation for the state $|1\rangle$ falling to $|0\rangle$ as described in Sec. 3.6.2. It was done by increasing the γ value, changing how fast it falls to its ground state, while for several points in an interval, I ran the Adam algorithm to see how much optimizing the parameters could help in counteracting the effect of the noise. This difference would then show how good the use of dynamic magnetic fields would be for mitigating this source of noise. As the Lindblad equation is more complicated and has to be computed for more states to get the full quantum map, efforts were made to optimize this.

To ensure a good overview of the optimization using the Lindblad equation, I set a tautological gate as the target. Thereby being able to always start at the most optimal parameters before applying noise.

4.4.3 Two qubit gates

For two qubits I started adding the interaction between the qubits into the Hamiltonian, as described in Sec. 3.3.1, to see how that affected the optimal parameters and cost landscape. The strength of the interaction added another parameter in combination with the larger size of the Hamiltonian, which had double the size and four times the number of elements in each matrix, which increased the time it took to run gradient descent.

Hadamard

I started with the Hadamard gate on two qubits simultaneously the $H \otimes H$ gate, but the effort to get a reasonably good local minima made it apparent that I would need another algorithm to give a better start position to the gradient descent algorithm. Moreover, testing by allowing the interaction between qubits to have different effects in different directions, massively increased the time used with gradient descent, as it increased the needed parameters from four to six. A double Hadamard gate was chosen as a starting point, to see how adding interaction between the parameters affected how well the Adam optimizer was able to find a good minima.

Tautological gate

As with one qubit gates, when you use a tautological gate, it is easier to know if you have a good starting point, when the gate is created using known starting values. The additional parameter increases the difficulty of finding good minima, but having a starting point makes gradient descent better able to get back to the optimal parameters as long as it starts close enough. Tautological gates also guarantee that the solution actually can exist. This enables focusing on whether it is possible to mitigate the effect of noise, while for the gates I use it may be uncertain if it is even possible to recreate the gate with the Hamiltonians I use.

Lindblad with noise

The main focus of this study, is the use of the Lindblad equation, here for two qubits with two different jump operators. One came from a Lindblad tomography of a superconducting quantum computer (Samach et al., 2022), and the other was randomly generated with Lst. 4.16.

For all of these different jump operators I started with a baseline of no optimization and a value of zero for γ the strength of the jump operators. And as I increased the value of γ I tried to use gradient descent to find better parameters and compared the effect of this with unoptimized parameters.

The jump operators found with tomography are:

$$L_1 = \begin{bmatrix} 0.501 + 0.001i & 0.000 + 0.001i & -0.0002 + 0.002i & -0.002 + 0.000i \\ 0.000 - 0.001i & 0.499 - 0.001i & -0.001 + 0.001i & -0.002 + 0.002i \\ -0.001 - 0.001i & 0.001 - 0.001i & -0.499 + 0.000i & 0.000 + 0.001i \\ 0.002 + 0.000i & -0.001 - 0.001i & 0.000 + 0.000i & -0.501 + 0.000i \end{bmatrix} \tag{4.1}$$

$$L_2 = \begin{bmatrix} 0.448 - 0.001i & 0.109 + 0.246i & 0.001 - 0.002i & 0.002 - 0.002i \\ 0.061 - 0.125i & -0.451 + 0.002i & -0.001 + 0.002i & 0.002 + 0.005i \\ -0.003 - 0.002i & -0.003 + 0.001i & 0.454 + 0.001i & 0.109 + 0.249i \\ -0.004 + 0.001i & 0.000 + 0.002i & 0.064 - 0.131i & -0.451 - 0.002i \end{bmatrix} \tag{4.2}$$

$$L_3 = \begin{bmatrix} -0.072 + 0.161i & 0.639 - 0.031i & -0.003 - 0.001i & 0.007 + 0.00i \\ 0.114 + 0.039i & 0.072 - 0.161i & -0.001 + 0.001i & -0.005 - 0.001i \\ 0.002 + 0.003i & 0.001 + 0.002i & -0.071 + 0.162i & 0.655 - 0.044i \\ 0.002 - 0.003i & -0.001 - 0.002i & 0.134 + 0.035i & 0.071 - 0.163i \end{bmatrix} \tag{4.3}$$

$$L_4 = \begin{bmatrix} 0.004 + 0.000i & -0.001 - 0.003i & 0.000 - 0.703i & -0.003 + 0.00i \\ 0.000 - 0.001i & 0.000 - 0.002i & -0.004 - 0.002i & 0.000 - 0.703i \\ 0.001 + 0.078i & 0.002 + 0.001i & 0.000 + 0.002i & -0.002 - 0.003i \\ 0.000 - 0.001i & -0.001 + 0.079i & 0.000 - 0.001i & -0.004 + 0.000i \end{bmatrix}, \tag{4.4}$$

and additionally have values for γ_j:

$$\gamma_1 = 0.071 \tag{4.5}$$
$$\gamma_2 = 0.097 \tag{4.6}$$
$$\gamma_3 = 0.042 \tag{4.7}$$
$$\gamma_4 = 0.055 \tag{4.8}$$
$$\tag{4.9}$$

These small γ_j values reduce how much impact the different jump operators have and modify how large effect they have in comparison to each other.
The jump operator I constructed is:

$$L = \begin{bmatrix} 0.0452 + 0.3897i & -0.9961 - 0.4091i & 0.1766 + 1.0004i & 1.8169 - 2.1335i \\ 0.8672 - 0.0596i & 0.0603 - 0.9063i & 0.7552 - 0.8013i & -0.1238 + 0.4040i \\ 0.9756 - 0.6610i & -1.2974 - 0.2849i & -0.0080 + 0.4674i & -1.1106 - 0.9585i \\ 0.3374 + 0.3060i & 0.9174 - 0.0648i & 1.8444 + 0.2738i & -0.0974 + 0.0492i \end{bmatrix} \tag{4.10}$$

and created by running Lst. 4.16 with the default parameters, random seed 42 and dimension of the jump operators 4.

Chapter 5

Results

This chapter will describe the results found by running simulations of different Hamiltonians and Lindbladians. I start by explaining how I did optimization, followed by some details about changes in the cost landscapes for different measures. After that I will describe the result of different optimizations, and end the chapter with optimizations using the Lindblad equation, to see how the effect of noise could be reduced.

5.1 Optimizing For a Gate

I started my project by optimizing with gradient descent for a given gate, starting with the X gate and Hadamard gate for one qubit. I found that it was possible to optimize the parameters to get close to the ideal gate, often on the scale of 10^{-5} from a perfect recreation of the target gate. I also found that gradient descent was very dependent on the start point and could get stuck in local minima. But the test did prove that dynamical magnetic fields could be used to create the Pauli gates and the Hadamard gate. This made the idea that magnetic fields could possibly have an effect on reducing the impact of noise feasible to test.

5.1.1 Target gates

When testing the effect of different Hamiltonians and Lindbladians I needed to have a target to compare it to. As the quantum map created by using a Hamiltonian or Lindbladian does not give insight on its own, the goal of the project is to reduce the effect noise has on a qubit. In order to have something to compare it to, I used either ideal gates that perfectly show how a known gate should be, or tautological gates, gates created with known parameters by my model.

Tautological vs. universal set of gates

In order to enable rapid testing and have a target with completely known parameters, I have created some tautological gates, that are created by setting the parameters of a Hamiltonian and reading the quantum map that emerges as the target gate.
This enables rapid testing and simplifies the running of gradient descent by ensuring I can always start arbitrarily close to the end goal. While it does not give a complete picture, it enables rapid testing of different solutions without being dependent on that the minima found by my program are good enough and completely correct.
It also gave me a chance to see how good the gradient descent algorithms were, as I knew the optimal parameters, and could test how good the optimizer was at rediscovering the parameters for a gate.
This could then be transferred to use on arbitrary gates to verify that the tautological gate gave a good enough result even while having a gate from a universal set of gates. While the tautological gate is not something that is used in a quantum computer it is possible to reach the state created by it using a combination of gates from a universal set of gates, as a universal set of gates can approximate any gate that can be used on the qubit, only by using a combination of the gates in the universal set.

One qubit gate

I focused mostly on two gates, the Hadamard gate used to set the qubit in a superposition found in Eq (2.26) and a tautological gate for testing purposes. The tautological gate was constructed by using Eq. (3.7) and setting all input parameters $(\epsilon, \Omega_x, \Omega_y)$ to one, and then solving it with the Crank-Nicolson solver described in Sec. 3.7.1, for the two starting state. The effect of the Hamiltonian then becomes:

$$Taut = \begin{bmatrix} 0.6585 + 0.7472i & -0.0634 - 0.0634i \\ 0.0634 - 0.0634i & 0.6585 - 0.7472i \end{bmatrix} \tag{5.1}$$

I did tests with other known gates, including the X gate, to ensure that the program could work as intended, but the main focus was on the two mentioned gates.

Two qubit gate

For two gates I started with the swap gate, but later I found a research paper that stated that one could construct any quantum circuit using three $(SWAP)^\alpha$ and six single qubit gates (Fan et al., 2005). As they could find $(SWAP)^\alpha$ using only exchange interactions it is therefore now clear that $SWAP$ and $\sqrt{SWAP}$ could be found using only interactions while the strength of the magnetic fields are set to 0. This simplified

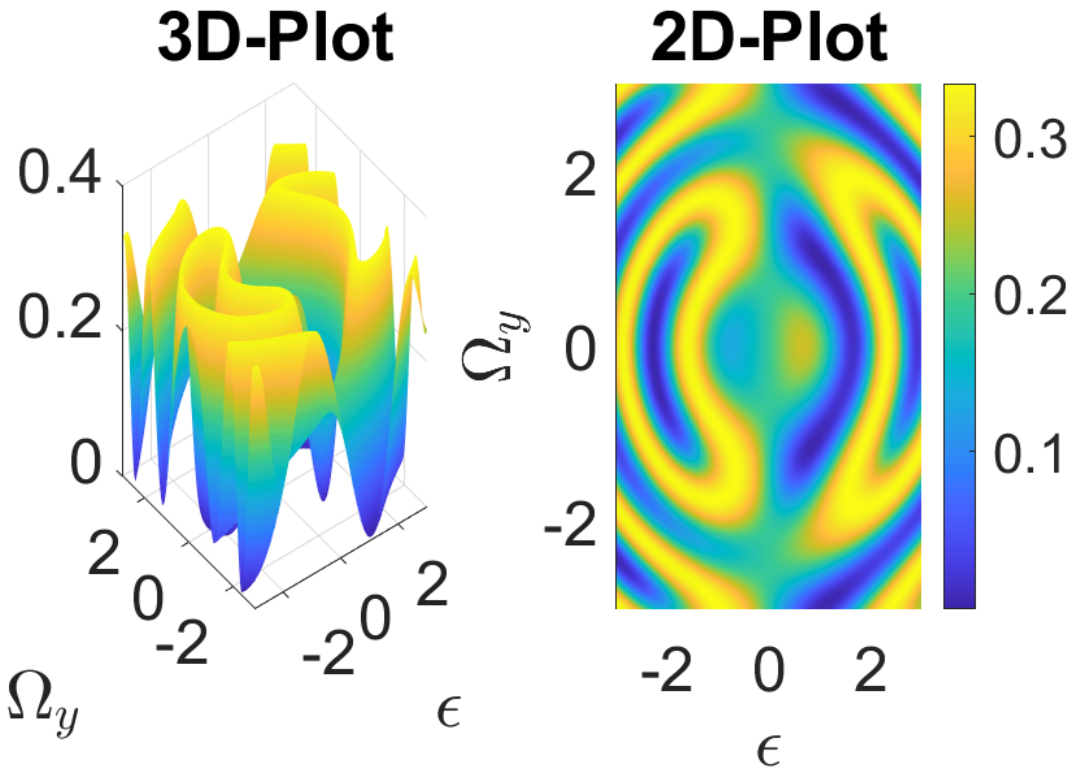

Figure 5.1: Cost landscape of tautological gate, varying ϵ and Ω_x with Ω_y

how hard it was to find the gates, and quick tests showed that my program could recreate this result with the effect of $SWAP$ being repeated in constant intervals as seen in Fig 5.2. The effect of a SWAP gate is repeated while varying u for specific ϵ values including $\epsilon = 0$ in Eq. (3.13).

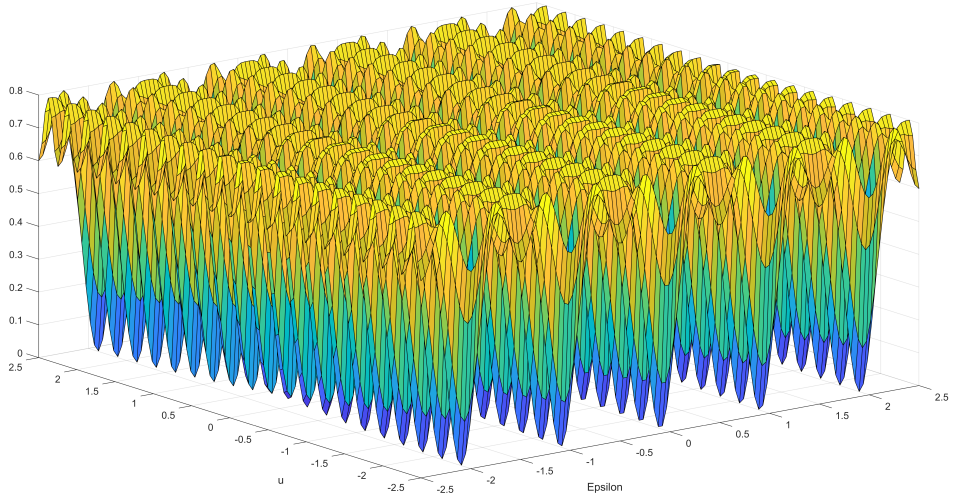

Figure 5.2: Cost landscape of SWAP gate varying ϵ and u with the other parameters set to 0

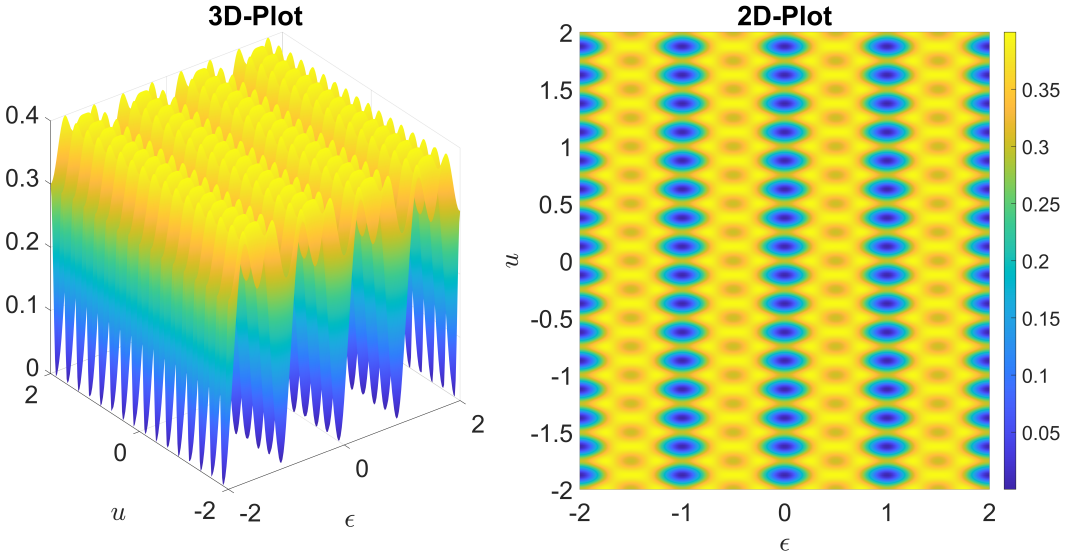

Figure 5.3: Cost landscape of SWAP gate varying ϵ and u with the other parameters set to 0

I attempted to optimize for a randomly generated gate, but the results were not promising. When using a Hamiltonian with six parameters, $\epsilon, \Omega_x, \Omega_y, u_z, u_x, u_y$ as described in Eq. (3.15) the best result where an infidelity of $5.56 \cdot 10^{-02}$ described in

Tab 5.1. It showed poor results after 600 runs of the Adam optimizer with Average gate fidelity used as the measure, with random start parameters in the interval $(0, 10)$. Only 3 of the attempts to optimize the parameter values got an infidelity below 0.1.

Variable	Value
ϵ	4.07
Ω_x	1.33
Ω_y	9.93
U_z	$2.20 \cdot 10^{-1}$
U_x	1.26
U_y	3.54
Infidelity	$5.56 \cdot 10^{-2}$

Table 5.1: Lowest infidelity found with Adam optimizer from random start points for a two qubit randomly generated gate

This poor performance in finding a good local minima when using the Adam optimizer for six parameters, made it impractical to use when the parameters were not known, and it greatly expanded the amount of parameter space to explore. It was therefore not the method of choice to use when starting with the Lindblad equations. The results gave me the incentive to start using gates where I knew what parameters are the optimal ones, leading to the creation of the tautological gates. Test with a double Hadamard gate on two qubits gave better results, so it was used early on before the focus moved to the tautological gates for testing.

The tautological gates gave me a better starting point, without having to search for a minima when I was planning to add noise using jump operators in the Lindblad equation. It was the main target when optimizing for two qubits, removing the long period of optimization needed to get a good starting point. The tautological gate created for two qubits using only parameter values of 1 ended up being:

$$\begin{bmatrix} 0.1367 + 0.5048i & -0.4951 + 0.0657i & -0.4951 + 0.0657i & 0.0002 - 0.4770i \\ 0.0657 + 0.4952i & 0.5230 - 0.0011i & -0.47701 + 0.0012i & -0.0661 + 0.4950i \\ 0.0657 + 0.4952i & -0.4770 + 0.0012i & 0.5230 - 0.0011i & -0.0661 + 0.4950i \\ -0.0002 + 0.4770i & 0.4951 + 0.0661i & 0.4951 + 0.0661i & 0.1370 - 0.5048i \end{bmatrix}, \tag{5.2}$$

all work using the Lindblad equation ended up being done on this tautological gate.

5.1.2 Gradient descent

There were $\frac{35}{800}$ runs with the Adam optimizer that were below 0.1, but only where only $\frac{4}{800}$ results that had an infidelity below 0.01. This was when using random parameters in the interval $(0, 10)$, using the Adam variant of gradient descent when optimizing over double Hadamard gate and having a Hamiltonian with 6 parameters. The parameters were $\epsilon, \Omega_x, \Omega_y$ and three interaction parameters u_z, u_x, u_y, one for each direction. This shows that gradient descent is extremely dependent on its start point when finding minima, here with the measure Average gate fidelity, but both the Channel fidelity and especially the Norm measure showed the same tendencies. For a better fit, another algorithm should be used to get the start points, so that gradient descent can be used in situations where It is not required to find the minima by itself. It will often get stuck in the many local minima before a good result is found. While there can be some connection between the local minima there are often many more local minima where the gradient descent can get stuck than there are good minima that can be used to construct a desired gate.

When creating cost landscapes for a non-uniform quantum map Sec. 3.6 gradient descent gave good smooth evolution of the parameters as the γ value that signifies the strength of noise increased, even as the infidelity grew toward 1 with increasing γ from it's starting value of 0. This shows that as long as you have a good start point the gradient descent algorithm gives a good way to optimize for disturbances. So that when you optimize to counteract changing circumstances, like the occurrence of noise, you have a good point to start, but that start point may no longer have the optimal parameters.

5.2 Cost Landscapes Hamiltonian

As a way to get a general overview of how difficult it is to optimize the parameters and the effect different parameters had, I created cost landscapes using the different cost functions described in section Sec. 3.8. This section covers cost landscapes for Hamiltonians and does not concern itself with noise, still, the knowledge from these attempts made further work easier.

The landscapes were constructed by freezing all except two parameters and thus showing the effect the combination of these two parameters had on the cost of the function, taken in the form of the infidelity.

5.2.1 Measure difference

The different measures had some effect on how easy it was to reach a good minima when using gradient descent. There were significant differences in how easy it was for gradient descent to find minima for the different measures, as I will describe in the following sections.

Norm measure

While the initial measure used, the norm measure Sec. 4.2.1, was easy to implement it suffered from differences in global phases, and numerical instability. These global phases are irrelevant when you measure gates, as they cannot be detected when doing a measurement. I attempted to correct for differences in the global phase, by targeting one value of the matrix U and synchronising that to a single value at the same place in the target gate, so that at least some of it was in the correct phase. This could be improved if I chose another element of the matrix to synchronize with, or if I had taken the average of the difference in phase so that the global phase difference would have been the smallest possible value for all elements instead of eliminating the phase difference for one element. The norm measure is made by finding the maximum singular value, this can make it more unstable as using any maximum value functions makes the measure more prone to having non-convex landscapes.

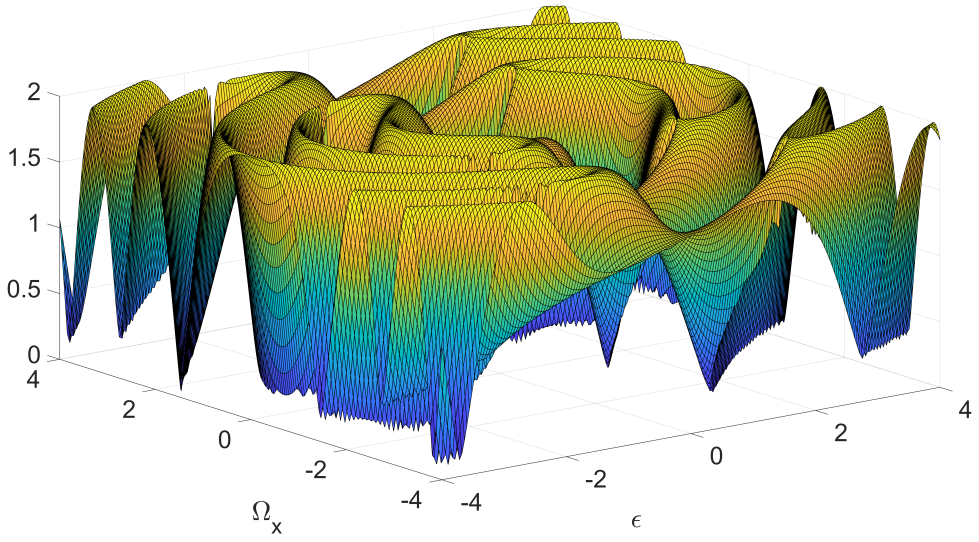

Figure 5.4: Norm measure for a tautological gate created with 1 as parameters while varying the ϵ and Ω_x parameters, with Ω_y locked to 1

As seen in Fig. 5.4 the measure gives a highly non-convex cost landscape that can have numerical instability, the non-convex landscape is caused partly by phase differences. My method to eliminate the difference in the global phase could be

improved by using an average of the phase difference or some other method instead of the current where I select one element. This has made it even harder to get a good result using gradient descent and its derived methods.

Average gate fidelity

Average gate fidelity(AVG) described in Sec. 4.2.2 does not concern itself with difference in the global phase, instead, it shows the average difference whereas the norm measure shown in Sec. 4.2.1 uses the largest singular value. This leads to AVG giving a smoother plot as seen in figure Fig. 5.5, with an example zoomed in Fig. 5.6, showing the difference for the Hadamard gate. These figures show some of the additional non-convexity the norm measure gives compared to the average gate measure.

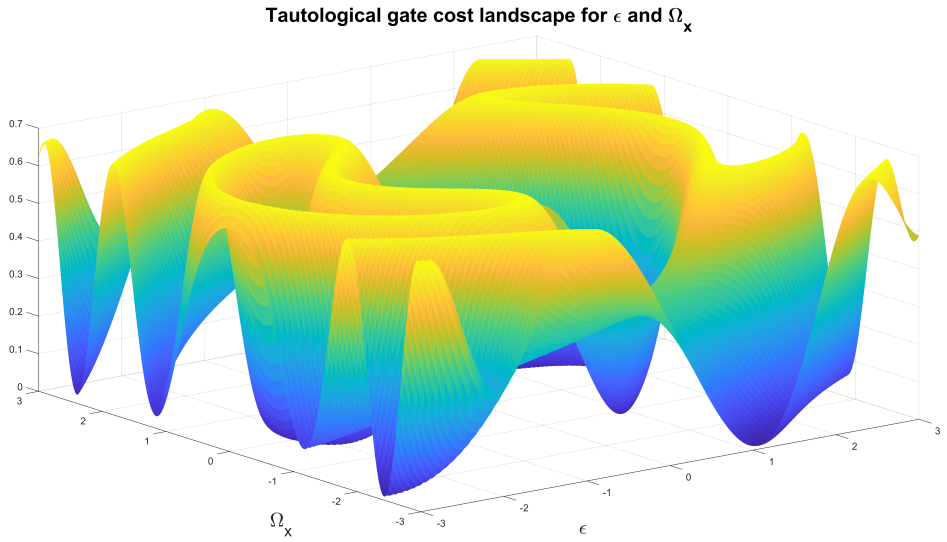

Figure 5.5: Cost landscape showing the same case as Fig. 5.4, but using Average Gate Fidelity instead of the norm to measure the difference

While it is still not easy to find global minima, the cost landscape is smoother and some local minima are closer to the global minima. When changing the phase to match phases for the norm measure, is focused on one element, the phase of the chosen element may be different than the global phase and it may increase the phase difference for another element. But finding the best phase by taking a global average was deemed not worthwhile when the average gate fidelity had broader use and was insensitive to the global phase difference.
The cost landscape is more convex, but still highly non-convex. Even though it is easier to find a good local minima, the landscape is still non-convex with many local minima and is difficult to search. But as you can see in Fig. 5.6 there is numerical

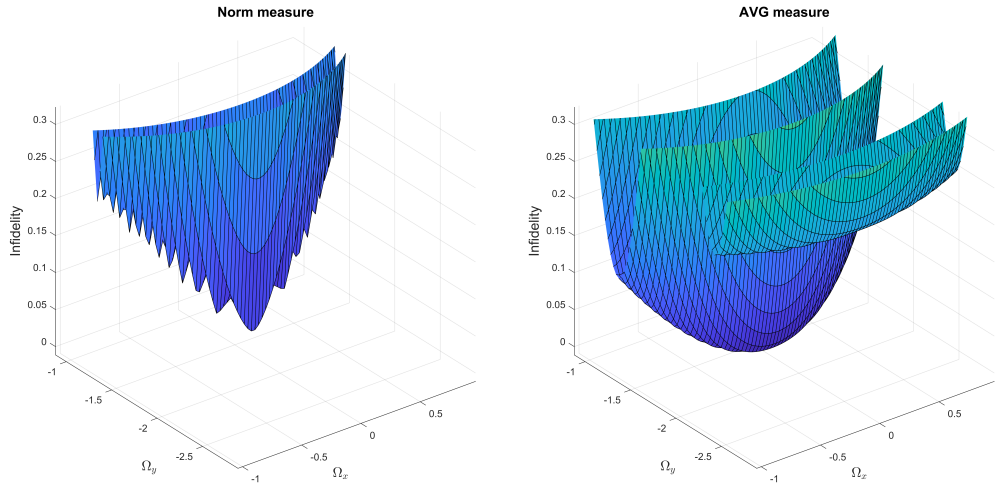

Figure 5.6: Zoomed in version of Hadamard gate with the ϵ parameter locked at 2.421, comparing the norm measure and the average gate measure, both scaled to 1

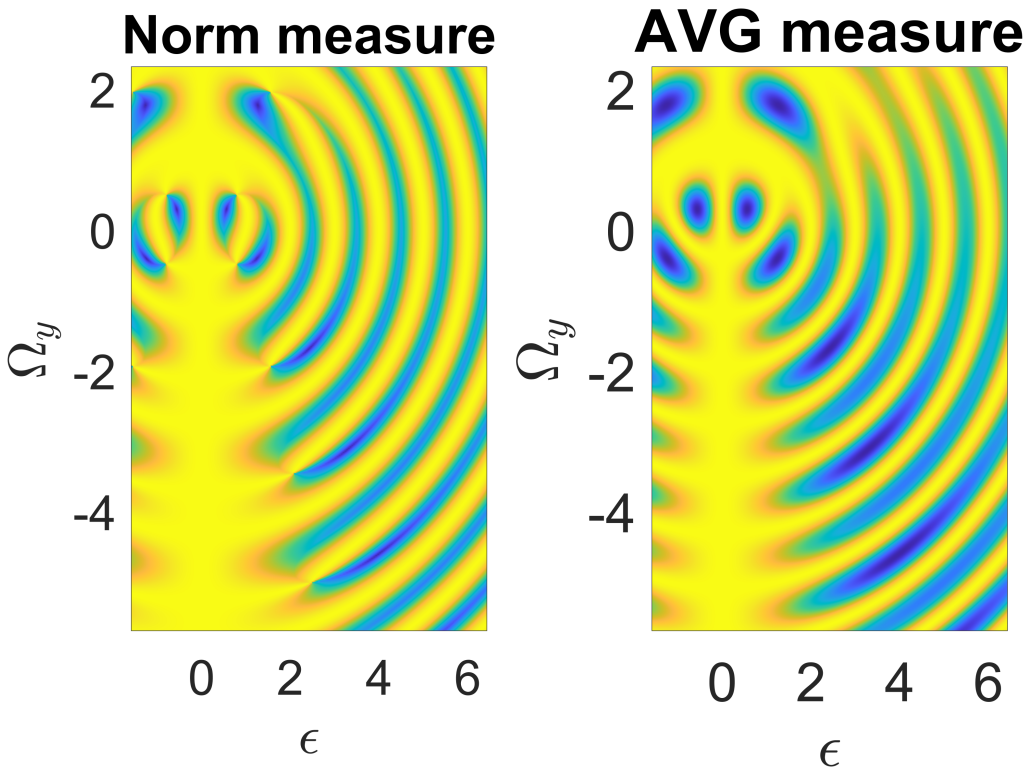

Figure 5.7: Cost landscape of the Hadamard gate with ϵ locked to 2.421, and with two different measures, comparing the norm measure and the average gate measure, norm scaled to 1

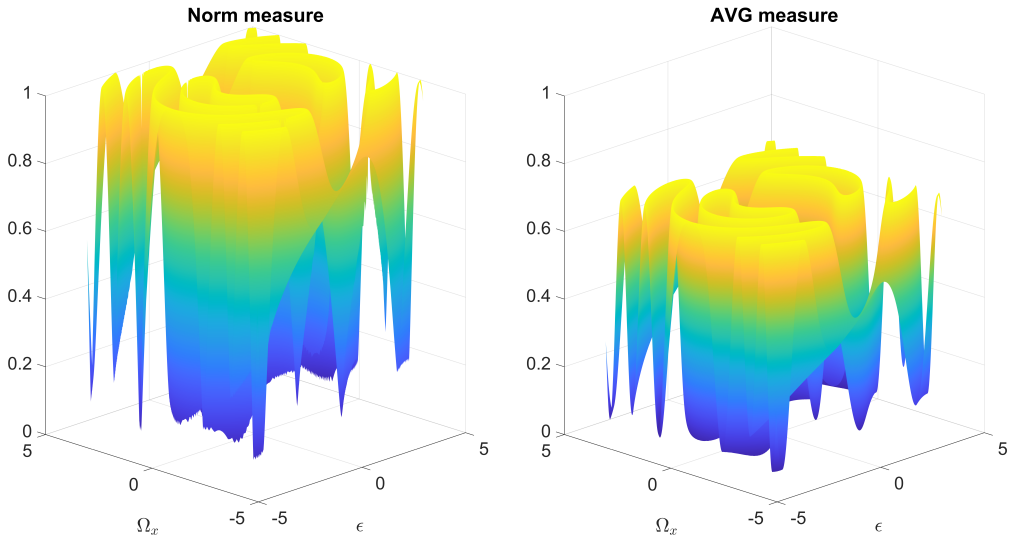

Figure 5.8: Cost landscape of a Tautological gate gate with ϵ locked to 1, and with two different measures, comparing the norm measure and the average gate measure, norm scaled to 1

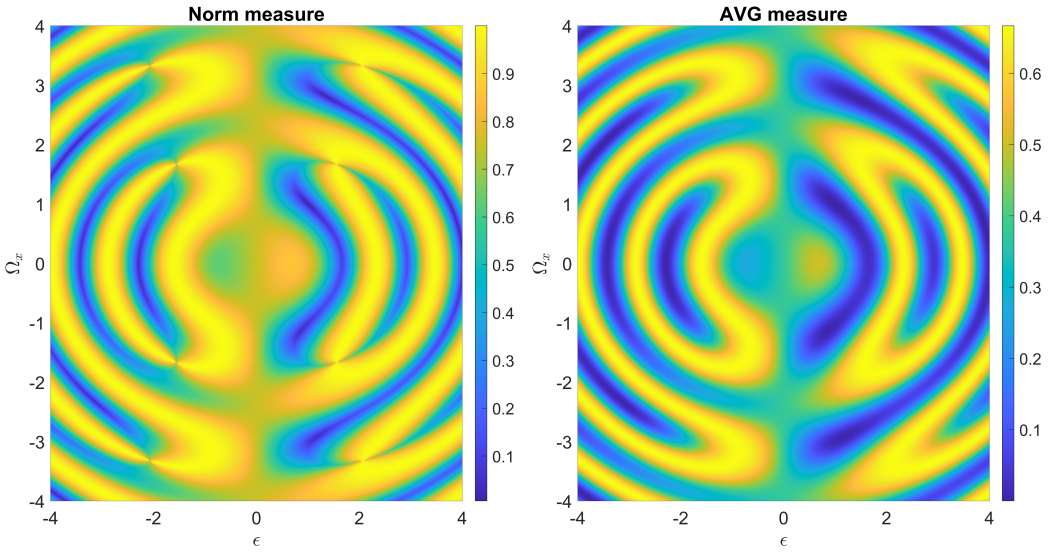

Figure 5.9: 2-D plot of a Tautological gate gate with ϵ locked to 1, and with two different measures, comparing the norm measure and the average gate measure, norm scaled to 1

instability, as the uneven spikes in the figure using the norm, are only an artefact of the resolution I used to plot the room. But it may have an effect on how effective a gradient descent algorithm is. While many of the minima found can be equal to the global minima or close, any 3d plot only sees the effect of varying two parameters. It will therefore not show any minima that depend on a change in any other parameter that is set to be constant for the plot. This makes it harder to get an overview of how many good local minima exist, but from the behaviour of the Adam algorithm, it seems like there are many more local minima that exist than there are global minima or local minima that are close to the global minima.

5.3 One Qubit

Even for one particle, the cost landscape can be quite complicated. But for many gates, it regularly repeats itself as the strength of the parameters increases, as seen in Fig. 5.10. This gives the possibility that a one qubit Hamiltonian, is less dependent on the start parameters than a two qubit Hamiltonian, as long as you avoid some of the poor start points.

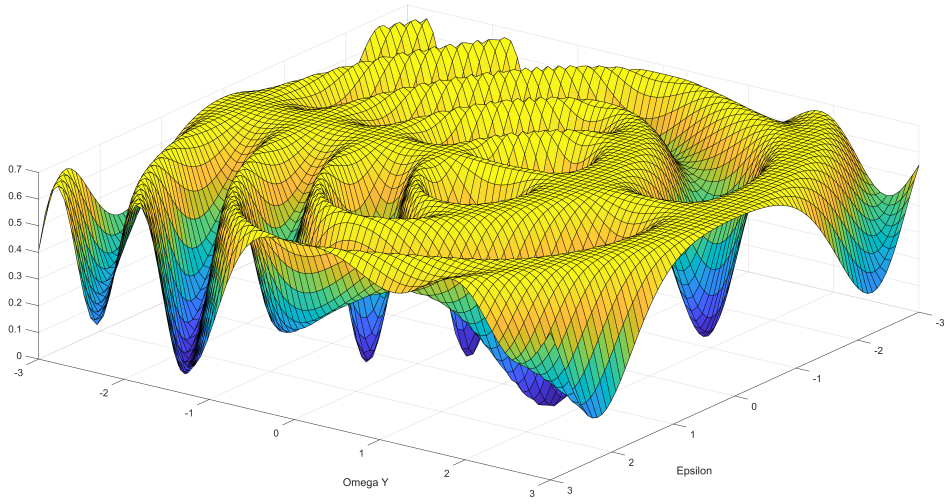

Figure 5.10: Difference from the Hadamard gate varying the ϵ and Ω_y parameters with Ω_x parameter set to 0

Hadamard gate

The Hadamard gate required many runs with gradient descent, to get parameters that enabled a low infidelity. The best result was an infidelity of $1.5349 \cdot 10^{-9}$ when using the parameters in Tab 5.2 and measuring with the Average gate fidelity. The Hadamard gate had a complicated cost landscape despite the fact that it is only using one qubit. As seen in Fig. 5.11 it seems like it has several global minima and many more local minima that can easily trap gradient descent.
For the Hadamard gate the most optimal parameters I could find with the Adam optimizer are seen in Tab. 5.2, I used the Adam optimizer with 400 different random staring parameters in the interval $(-5, 5)$. The best set of parameters gave the infidelity $1.5349 \cdot 10^{-9}$, a very small value that approaches zero. While the theoretical infidelity with average gate fidelity is 10^{-13}, it is close enough that machine precision may limit it from getting better results while using my code.
There seems to be one optimal value for Ω_x consistently in the domain D :

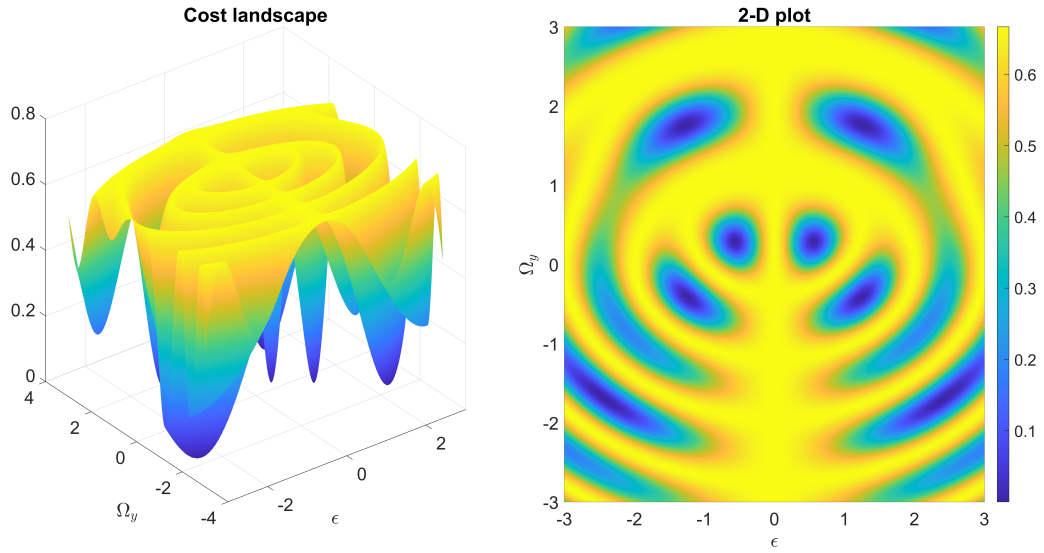

Figure 5.11: Cost landscape for a Hamiltonian with respect to the Hadamard gate, with static Ω_x set to 0 varying ϵ and Ω_y, with a version seen from above

Variable	Value
ϵ	2.4210
Ω_x	0
Ω_y	-1.6971
Infidelity	$1.5349 \cdot 10^{-9}$

Table 5.2: Optimized variables found for the Hadamard gate using Adam optimizer

$(-3, 3)$, which I used in these plots as seen in Fig. 5.12, while for ϵ and Ω_y there is a dependence on each other.

X gate

As the X gate is only dependent on the magnetic field in the X-direction it is much easier to optimize over this gate. Unless disturbed, the only parameter that should not be 0 is the strength of Ω_x. This drastically reduces runtime and the possible number of local minima, at least when using discrete pulses or ramp on and off effects on the discrete pulses as described below in Sec. 5.3.2.
When using dynamic magnetic pulses it was not that simple, as I used an optical cycle of length 2π with the dynamic magnetic field strength dependent on $\sin(x)$, which in testing was seen to have no effect when used alone. This led to me look into the effect of the sinus pulse:

$$\int_0^{2\pi} \Omega_x \sin(t) dt = \int_0^{2\pi} \Omega_y \sin(t) dt = 0, \tag{5.3}$$

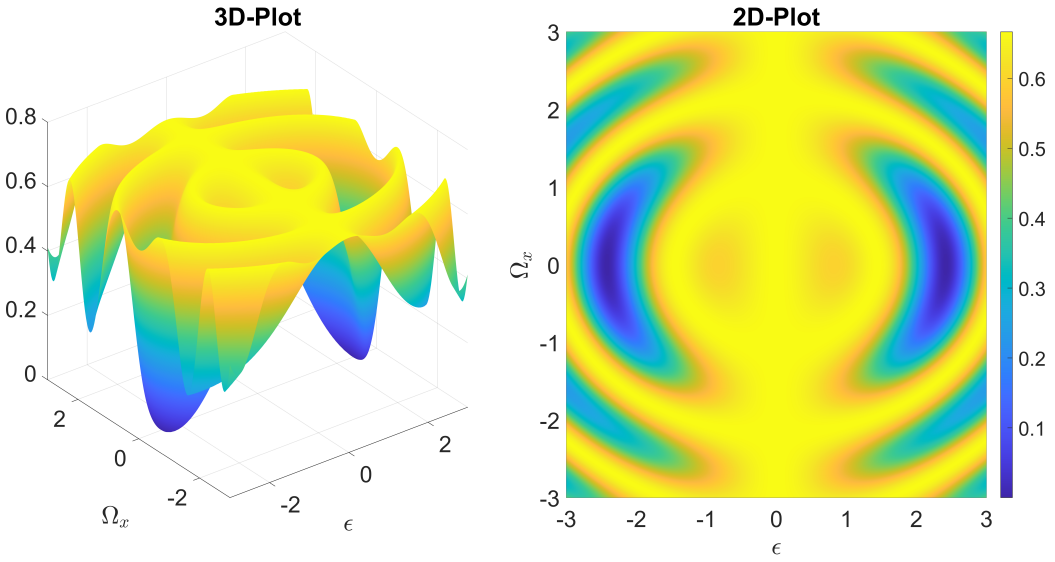

Figure 5.12: Cost landscape for a Hamiltonian with respect to the Hadamard gate, with static Ω_y set to -1.6971 varying ϵ and Ω_x

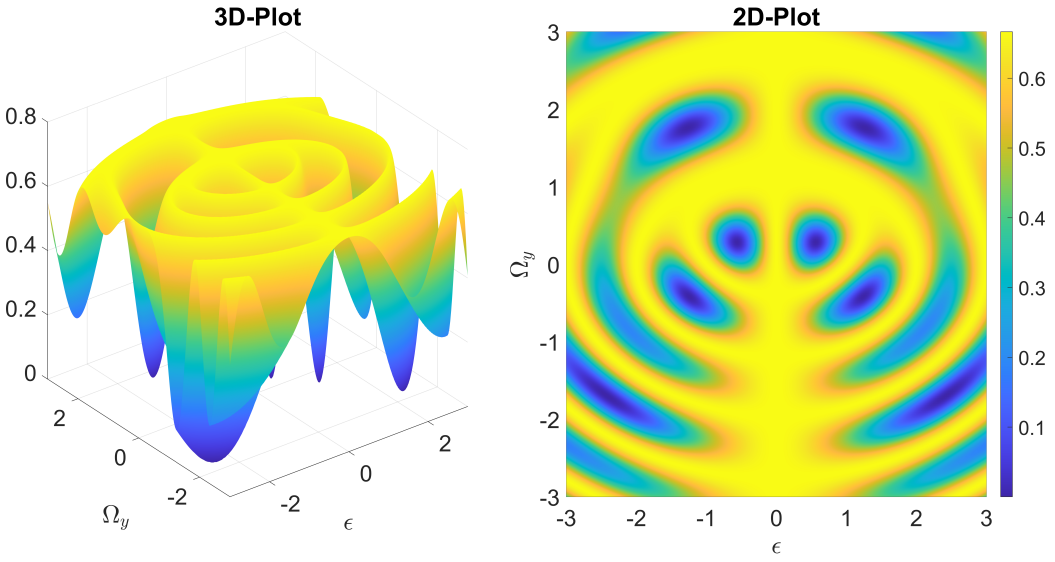

Figure 5.13: Cost landscape for a Hamiltonian with respect to the Hadamard gate, with static ϵ set to 2.4210 varying Ω_x and Ω_y

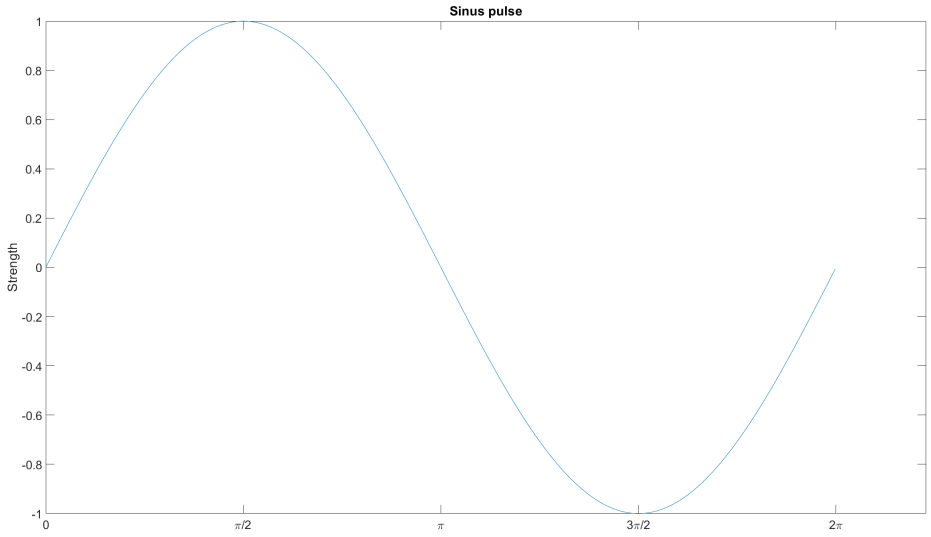

Figure 5.14: Strength of the sinus pulse in the interval $[0, 2\pi]$

ϵ	2.4610
Ω_x	0
Ω_y	5.0179
Infidelity	$1.9526 \cdot 10^{-04}$

Table 5.3: Best parameters and infidelity found with Adam optimizer for the X-gate

valid for any Ω_x or Ω_y. It means that for a dynamic magnetic field in a single direction, that contains a whole cycle, here 2π, integrates to zero. I decided to look more into how the strength of the magnet field was distributed. This led to the discovery that the symmetry given by:

$$\sin\phi = -\sin\phi + \pi \tag{5.4}$$

means that the Hamiltonian has one effect in the time period $[0, \pi]$ and the opposite effect in the interval $[\pi, 2\pi]$, as you can also see in Fig. 5.14. This symmetry means it will counteract itself as long as all energies involved follow the strength of the sinus pulse.

After 200 iterations of the Adam optimizer with three random parameters, in the interval $(0, 10)$ using a uniform random number generator, I got 2 results lower than 10^{-3}, and 10 lower than 10^{-1} with the best infidelity and the parameters belonging to it in Tab 5.3.

You can in Fig. 5.15 see how there seem to be repeating minima where the values of ϵ and Ω_y seem to have a consistent ratio, and any of these minima is close to the global minima. With the lowest infidelity found being in Tab. 5.4

ϵ	-3.4716
Ω_x	0
Ω_y	-2.0669
Infidelity	$6.0603 \cdot 10^{-6}$

Table 5.4: Lowest infidelity found in a plot by only manipulating ϵ and Ω_y

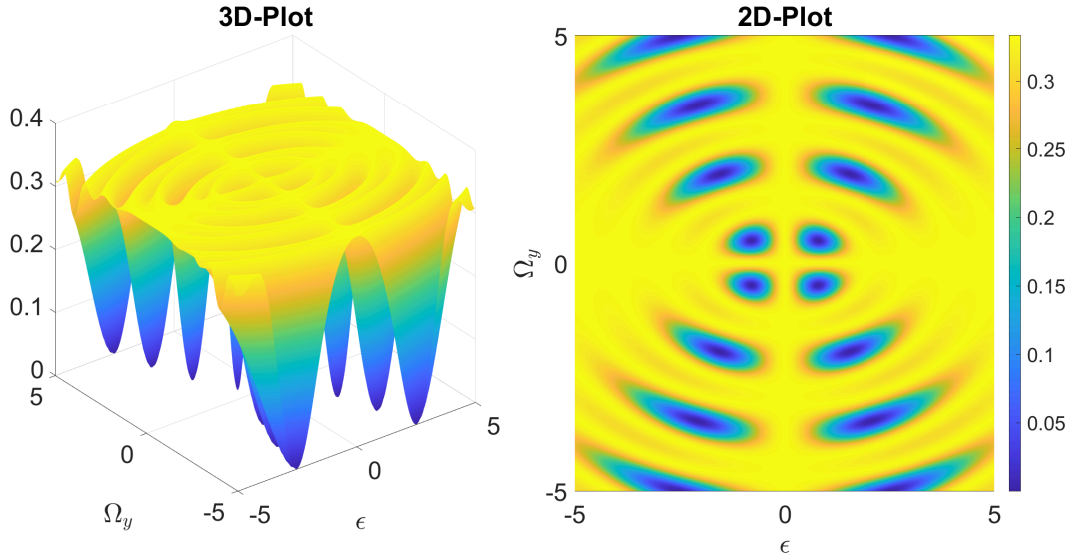

Figure 5.15: Cost landscape for the X gate, varying the ϵ and $Omega_y$ with the Ω_x parameter set to 0

The fact that this is lower than what the gradient descent algorithms found, confirms that starting from a random point in non-convex landscapes such as this, is not very good and that I may have used a too low a maximum number of iterations or been too strict with stopping, if the difference after 100 iterations was less than 10^{-6}. But the use of the Adam algorithm may improve many of the local minima found in the plot if used as the starting point.

5.3.1 T gate

To see what gates I could create, I did a small optimization for the T gate:

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix} \tag{5.5}$$

It is one of the gates needed in addition to the Clifford gates(Hadamard, CNOT, S gate) to form a universal set of gates. As $T = \sqrt{S}$, it is trivial to construct the S gate from the T gate. When optimizing with the Adam solver from 40 different random start points in the interval $(0, 10)$, I got a good optimization with an infidelity of only

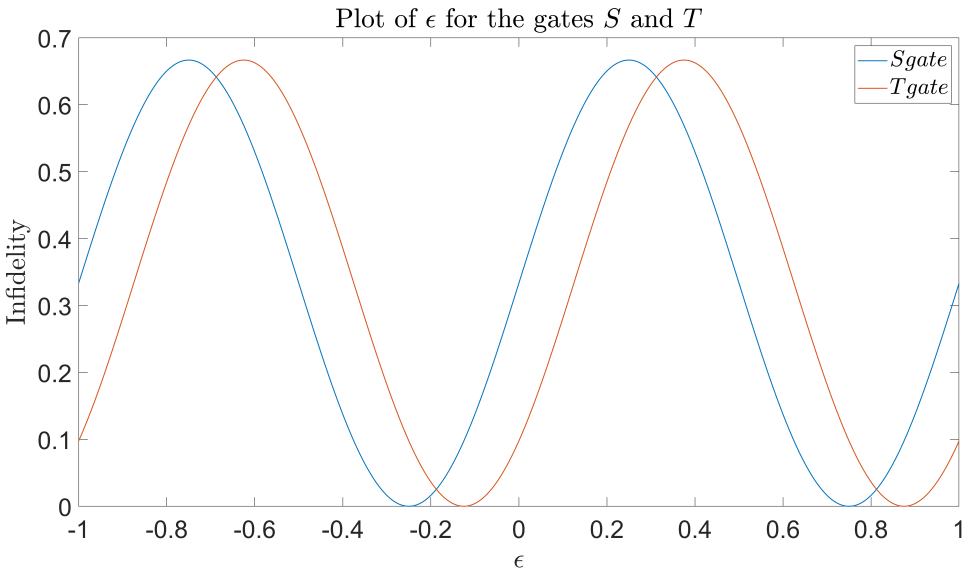

Figure 5.16: Infidelity for the S and T gates, varying ϵ with $\Omega_x = 0$ and $\Omega_y = 0$

$6.30 \cdot 10^{-5}$ with parameters shown in Tab. 5.5.

ϵ	6.96
Ω_x	6.07
Ω_y	0.87
Infidelity	$6.30 \cdot 10^{-5}$

Table 5.5: Lowest infidelity found in a plot with Adam optimizer for the T gate

Combined with the fact that they can be constructed using only ϵ as seen in Fig. 5.16, it may be one of the easier gates to construct. It only needs the static magnetic field from the ϵ which means that it even may be solved analytically.

5.3.2 Smooth and discrete pulses

As both magnetic pulses and laser beams have a little ramp on and ramp off effect, I did some testing to see what effect this had on the parameters required for optimal parameters. The Hamiltonian I used to simulate this ramp on and off was:

$$H = \frac{2\mathbf{B} \cdot \sigma}{\exp\left(s\left(|t - t_{mid}| - t_{pulse}\right)\right) + 1} \tag{5.6}$$

Where t is the time, t_{pulse} is the time a constant pulse would run, t_{mid} is the halfway time of all the time used when all the time used is greater than t_{pulse} to capture the tail and the head of the effect, and the s is the scale that states how sharply it ramps on and off. The time t will include a larger domain than t_{pulse} to capture the whole

smoothed out pulse, as when the scale s decreases it spreads out in time. When σ is expanded to its constitute forms it gives the Hamiltonian:

$$H = \frac{2}{\exp\left(s\left(|t - t_{mid}| - t_{pulse}\right)\right) + 1} \cdot \begin{bmatrix} \mathbf{B_z} & \mathbf{B_x} + i\mathbf{B_y} \\ \mathbf{B_x} - i\mathbf{B_y} & -\mathbf{B_z} \end{bmatrix} \tag{5.7}$$

When $s \to \infty$ the pulse becomes indistinguishable from a rectangular pulse.
As I did some tests I expanded the period used to measure the time in order to include the full effects of the pulse as seen in the ramp on and off in Fig 5.17, with a constant pulse for reference. I ended up using 2 times the time period and, as most tests were set to use 2π as the period used for the target pulse, I included the effects from acting in a time period π before and after, for a total time period of 4π.

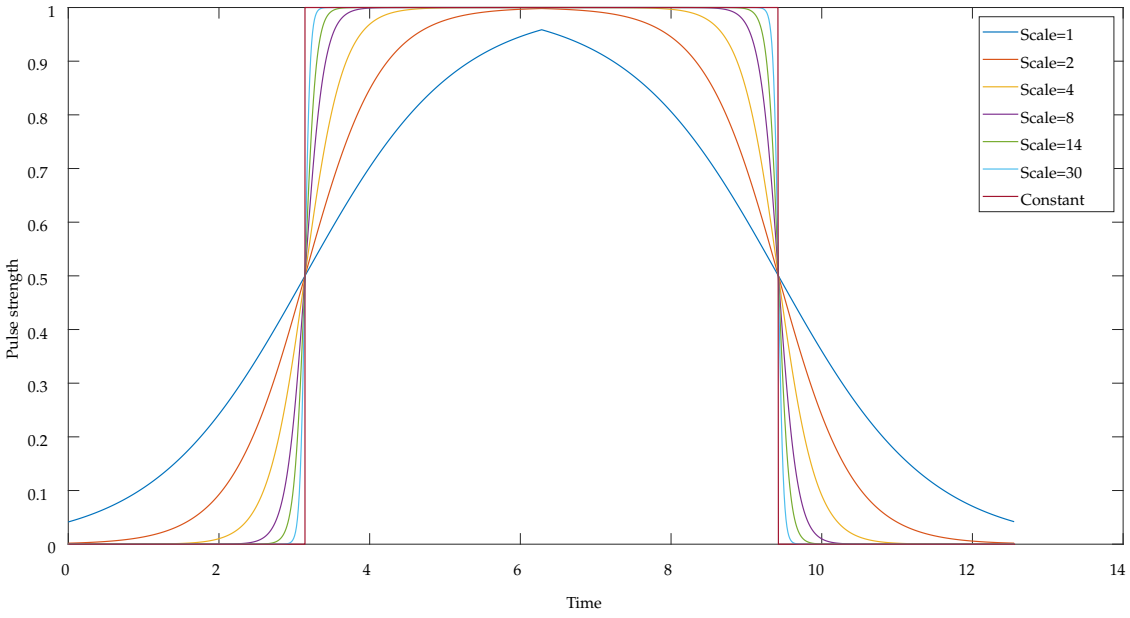

Figure 5.17: An example of several pulses with ramp on and off, for different values of scale

While having a ramp on and ramp off effect did change the result, compared to the rectangular pulses the difference was minor for any of the tests. Even with the scale variable s set to 1 a very rounded pulse, the result was only a $2.13 \cdot 10^{-8}$ difference from a pure discrete pulse, using the Average gate fidelity measure.
Moreover, for the X gate with $\Omega_x = \frac{1}{4}$, $\Omega_z = 0$ $\Omega_y = 0$ as described in Sec. 3.3.2, and the scale s set to 5 gives an infidelity of $9.06 \cdot 10^{-13}$. This is an excellent result, that may even have an infidelity of 0, but it is limited by the machine precision, and this result is so close to the effect of discrete pulses that it is not interesting for this project.
As these results were obtained without changing any parameters compared to the discrete pulses used to set up the test gates, I decided to focus on other possible solutions since the difference from discrete pulses was so small.

For comparison, I did some tests with an even larger time period, to capture the low effect when the pulse is the most spread out, for instance when s is set to 1, I did tests that had a time period of 6π. Increasing the time period increased the infidelity but adjusting the pulse length to aim for by only -0.0851 such that $t_{pulse} = 2\pi - 0.0851$, less than a 2% difference, gave an infidelity of $4 \cdot 10^{-8}$, this for the Xgate measured using the average gate fidelity. This shows that a minor adjustment in strength or length of the magnetic pulse would be enough to counteract the effects of a ramp on and off effect of the magnetic pulses.

5.4 Two qubits

When there are two qubits they can interfere with each other and when you look at the cost landscape for a two qubit Hamiltonian there are many local minima. This made tests take longer and the process of finding the minima became very dependent on the starting point, as the uneven landscape makes the gradient descent algorithm get stuck in the local minima since they are so large and often have areas of high infidelity between them and the global minima.

Square root of SWAP

After testing we found a research paper that found that the swap gate could be found analytically using only exchange interactions (Fan et al., 2005). It showed that they could find $(SWAP)^{\alpha}$ for any α by only adjusting the strength of interaction. This reduced the value of using the $SWAP$ gate to test with, but enabled us to confirm that my model showed the expected result of enabling the square root of SWAP gate trough only interactions between qubits in our system, this naturally extends to the normal SWAP gate.
Moreover, as expected the effect of the interaction is very periodic, as imposing an additional $\sqrt{SWAP}$ gets you the $SWAP$ state, and imposing $\sqrt{SWAP}$ twice more for four times total gets you back to the initial state, as you can see in Fig. 5.18. When compared to the SWAP gate seen in Fig. 5.3 the difference is minimal except for a minor difference in u the cost landscapes are identical. This can also be seen in Fig. 5.19, where only the parameter u is modified and all other parameters are set to 0. The only difference is in the phase of the resulting sine curve, these repeating patterns would also exist for all versions of $\sqrt{SWAP}^{\alpha}$.

Hadamard on two particles

The double Hadamard gate was one of the first gates that was used on two qubits simultaneously, it was chosen for the fact that I knew of one local minima. It could

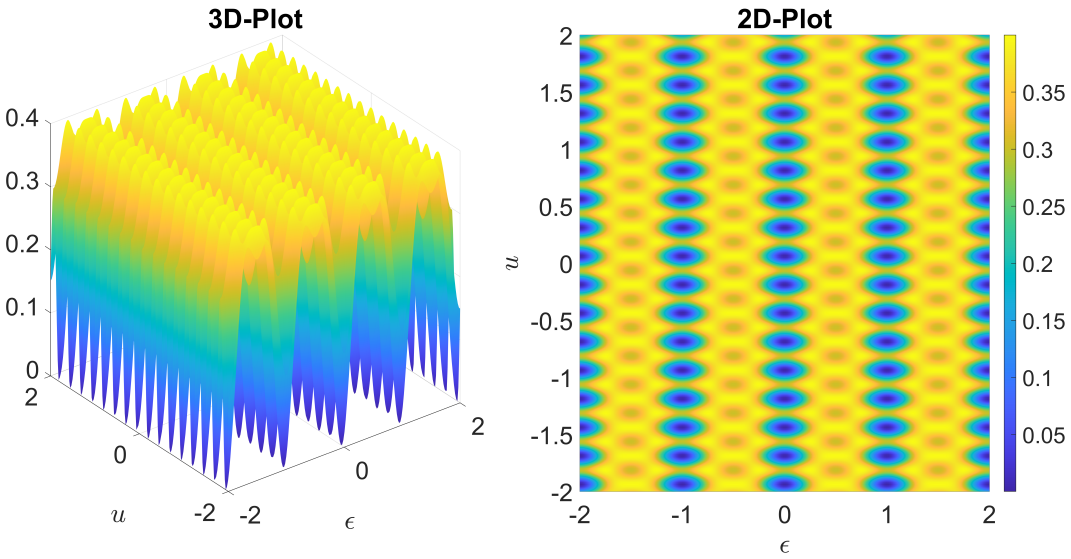

Figure 5.18: Cost landscape of SWAP gate varying ϵ and u with the other parameters set to 0

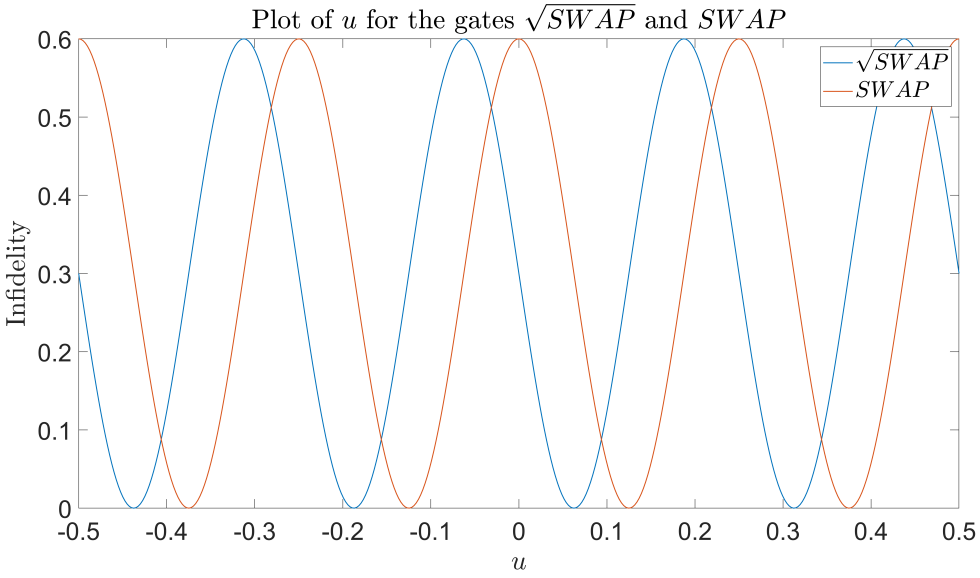

Figure 5.19: $SWAP$ and $\sqrt{SWAP}$ varying only u, the rest of the parameters set to 0

be used to see how good my optimization algorithm Adam was to find the global minima or a value close to it, when adding an additional parameter in the interaction parameter u. When running Hadamard on two qubits simultaneously with them interfering with each other, you can see that for some of the parameters, the optimal strength repeats periodically for some of the parameters, including for the interaction parameter u. I did some tests using interaction parameters that were different for each direction using the Hamiltonian from Eq. (3.15). But it did not seem to provide any advantage over using just one interaction parameter when using the Adam

optimizer with the Hamiltonian from Eq. (3.13), both Hamiltonians are described in Sec .3.3.1. While it is possible to set the interaction parameters u to 0 and use the same parameters as for the one qubit case, the optimization from random starting points was not able to find this point, instead, it found other minima. As the computational time needed to optimize over six parameters, with three interaction parameters, was significantly longer than for four parameters with only one interaction parameter, it was decided to only use four parameters to easier optimize the parameters with the Adam algorithm.

Variable	Value
ϵ	4.11
Ω_x	$5.60 \cdot 10^{-1}$
Ω_y	9.09
u	$5.00 \cdot 10^{-1}$
Infidelity	$6.74 \cdot 10^{-3}$

Table 5.6: Best result found with Adam optimizer from random start points for a double Hadamard gate for a four parameter Hamiltonian with interactions

For four parameters the best result where an infidelity of $6.74 \cdot 10^{-3}$ described in Tab 5.6. It showed poor results after 600 runs of the Adam optimizer with Average gate fidelity used as a measure, with random start parameters in the interval $(0, 10)$. Where only 14 of the attempts to optimize the parameter values got an infidelity below 0.1.
Fig. 5.20 shows how the parameters mirror around 0 for ϵ, but for these parameters there are none other valid values for the combination of ϵ and the interaction u. The result would have been even better if I had taken the parameters from the one qubit Hamiltonian in Tab. 5.2 and set the interaction parameter to zero. In this case, the interaction parameter u showed the expected cyclical behaviour as found for $SWAP$ and $\sqrt{SWAP}$ described in Sec. 5.4.

Tautological gate

For testing purposes, I created a tautological gate, by using 10^6 time steps and setting all parameters to 1. I solved the Schrödinger equation with a Hamiltonian, which had all parameters set to 1, with the average gate fidelity to measure and the Crank-Nicolson solver to solve the differential equation. I used the resulting unitary matrix as a target gate. By doing this I could be certain that I knew what kind of parameters were guaranteed to give the right gate. It made it far easier to do many tests, as I could give gradient descent as good a start point as I wanted, so I could see how

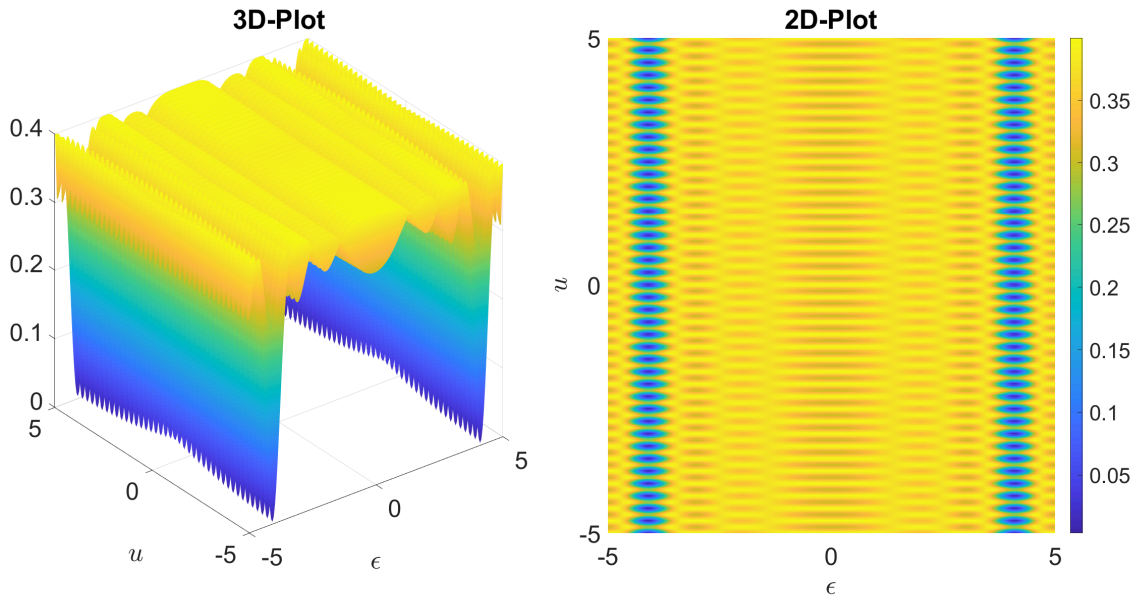

Figure 5.20: Cost landscape for a double Hadamard gate on two qubits, varying ϵ and u with $\Omega_x = 0.5598$ and $\Omega_y = 9.0934$, measured with average gate fidelity

effective it was to get back to the ideal point.

5.5 Lindblad optimization

Part of the goal of the project was to find if the use of dynamic magnetic fields had any effect when trying to combat noise. In the following subsections, I will look at the result of optimizing for three different dissipative jump operators. An Amplitude dampening jump operator, a randomly generated jump operator that is traceless and a jump operator found with tomography in Samach et al. (2022). To simplify my optimization I used a tautological gate generated with a Hamiltonian having all parameters be set to one as mentioned in Sec. 5.4.

5.5.1 Single qubit Lindblad with Amplitude damping

By using the Lindblad Eq. (3.27) from Sec. 3.6.2 I can simulate the effect of the qubit falling from its $|1\rangle$ position to it's $|0\rangle$ position.
In Fig. 5.21, when comparing to the target tautological gate, the difference in infidelity between the optimized and unoptimized parameters is too small to be visible. The figure shows the infidelity when optimizing from the ideal parameters for $\gamma = 0$ where $\epsilon, \Omega_x, \Omega_y$ are all set to 1, γ is increased until $\gamma = 1$. It may have been enough to stop at a lower value, as the infidelity reaches saturation at around 0.6, with a logarithmic growth of the infidelity, increasing rapidly for low γ values and flattening

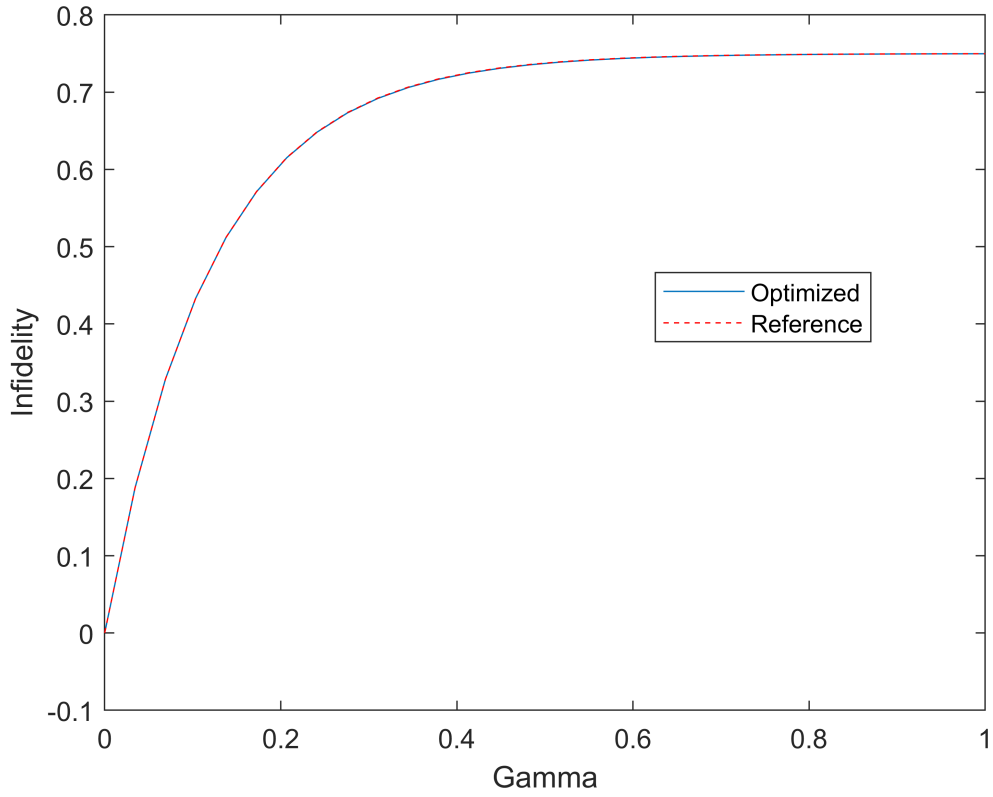

Figure 5.21: A Lindblad system called LambdaOne in my code, showing the difference from the ideal tautological gate as gamma increases

out as it approaches the maximal infidelity of 0.75 when $\gamma \to \infty$. It is only by looking at the difference between having optimized and unoptimized parameters in Fig. 5.22 that you see a difference. But this is only a difference on the scale of 10^{-4}, a small value that is close to the error in the simulation. All the optimizations, when using the tautological gate, start with the ideal parameters used to create the gate with a Hamiltonian based on Eq. (3.7), this shows an infidelity of $1.1159 \cdot 10^{-7}$. This is about the expected numerical error when using the Channel fidelity with Choi matrices, thus being an acceptable starting point.
But Fig. 5.23 shows that the optimal values of the parameters have a smooth moving trend until the infidelity reached saturation with a strength of the noise; γ around 0.8 where ϵ the strength of the field in the Z direction flattens.

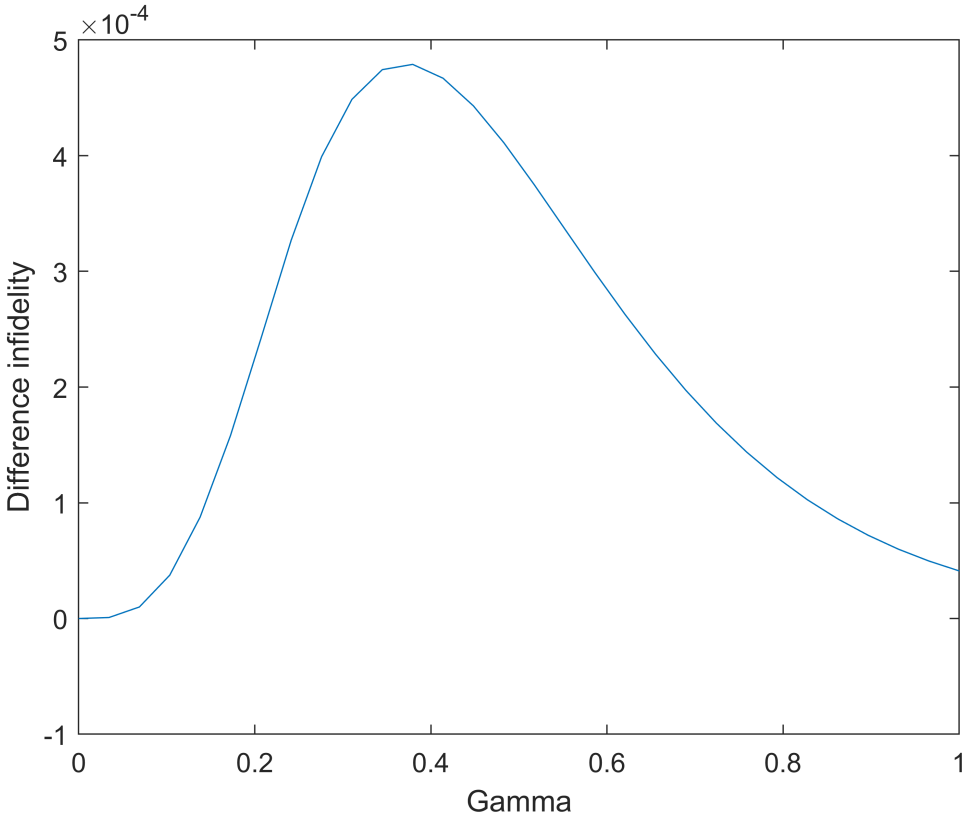

Figure 5.22: LambdaOne difference between optimized and reference result as gamma increases

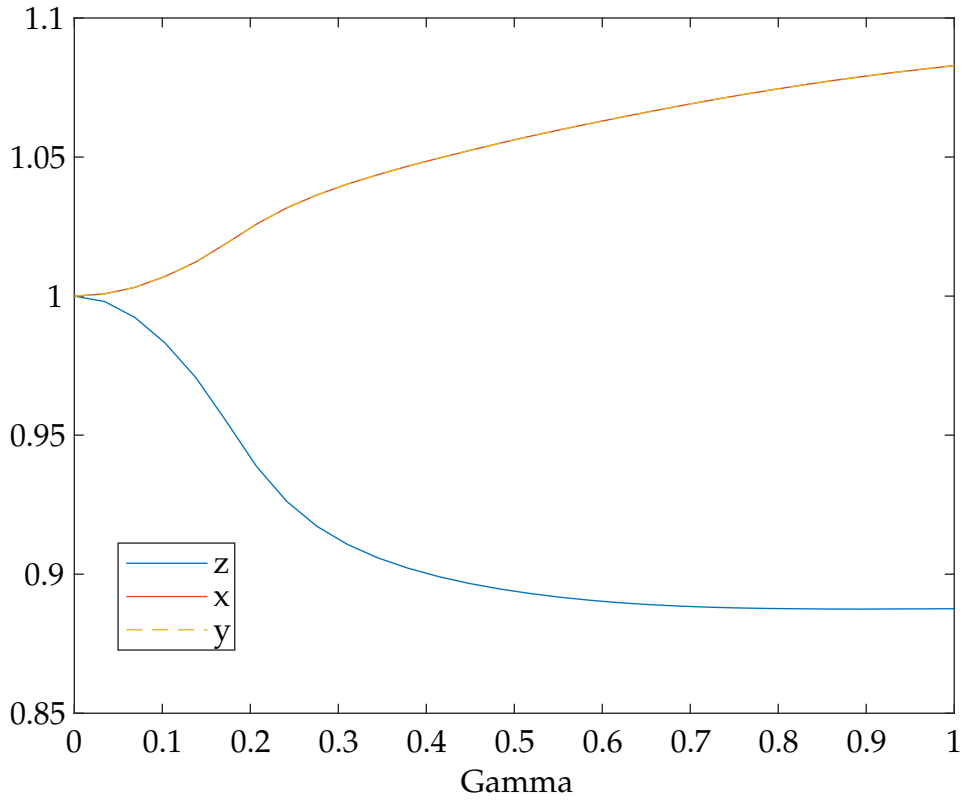

Figure 5.23: LambdaOne the value of the optimized parameters as gamma increases, the changes to x and y parameters overlap

5.5.2 Optimization for Lindblad with tomography jump operator

When trying to mitigate noise for a two qubit Lindbladian, the first jump operators I used were found using tomography in Samach et al., 2022. But mitigating this was even harder than mitigating the amplitude dampening for a single qubit. The jump operators are smaller, so each with $\gamma_j \leq 0.1$, requires ten times the γ value to reach the same effect as the amplitude dampening and the randomly generated jump operators I use later.
In all figures I use a tautological gate created with a Hamiltonian so that all four parameters $\epsilon, \Omega_x, \Omega_y, u$ are set to one. This is also the start point for when I optimize when γ increases.
For Fig. 5.24 you can see how the infidelity increases logarithmic, fast at the start and then more slowly approaching the maximum of the measurement as gamma increases. Starting at $\gamma = 0$ for no noise, with the infidelity rapidly increasing, before flattening out as the effect of the noise is set to$\gamma = 7$ and slowly approaching the maximum infidelity of the measure $1 - \frac{1}{2^4} = 0.9375$ as $\gamma \to \infty$. The small dip in infidelity in the span $0 - 1$ is on the scale of 10^{-6} and may simply be errors from using the channel fidelity and Choi matrices, as both the optimized and the starting parameter values give smaller infidelity there, the value is simply too small to be certain.

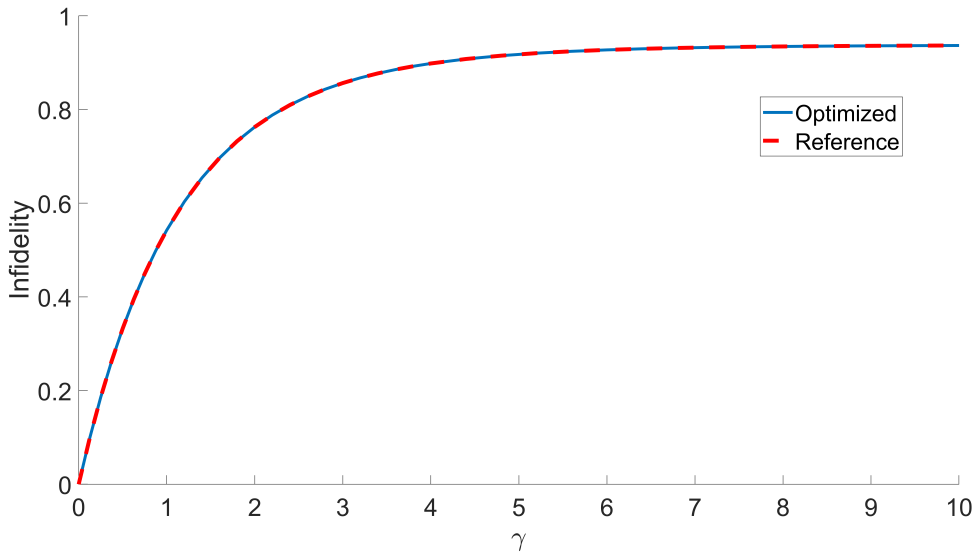

Figure 5.24: The measure for the difference from an ideal tautological gate created with 1 as parameters and using the jump operators from the tomography, showing the plot for optimized parameters and for unoptimized reference parameters

As seen in Fig. 5.25 the difference between the optimized parameters and unoptimized parameters increases as gamma increases, but it never exceeds $3 \cdot 10^{-5}$. This

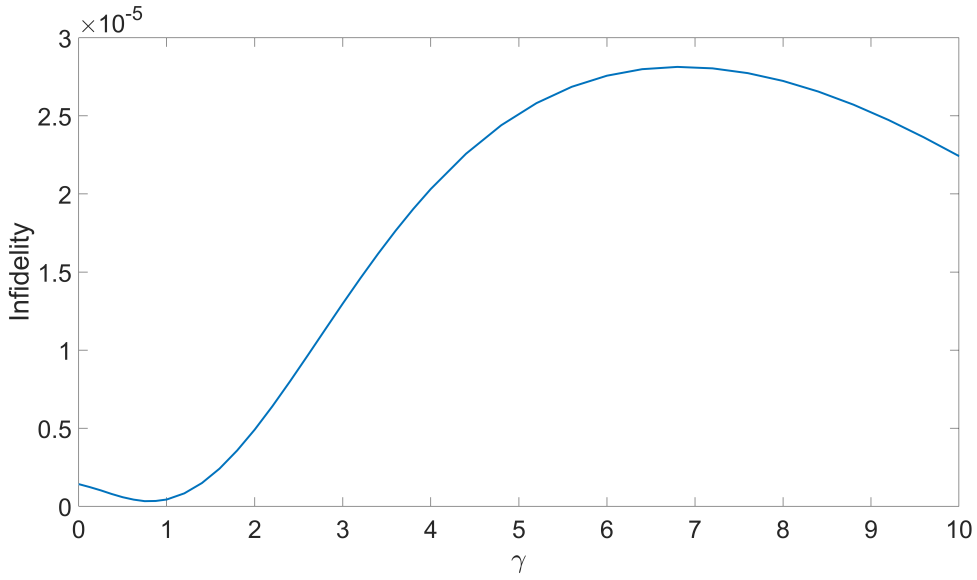

Figure 5.25: The difference between using optimized parameters and non optimized parameters on the infidelity compared to a tautological target gate, for a given gamma. Using jump operators from the tomography

makes the effect of the optimization rather uncertain. There is a correlation with Fig. 5.24 as the effect increases until the infidelity reaches saturation at a γ value of around 7. But these measurement values are so small that they cannot say anything definite. I used different solvers for the Lindblad equation and the Hamiltonian used to construct the target gate. I cannot dismiss the possibility that a part of the difference in infidelity when optimizing, can come from using different solvers. Using different solvers can change what parameters are optimal, but reducing the size of the time steps used in the solvers mentioned in Sec. 3.8 should theoretically reduce this error, as it becomes infinitely small.
As these results are so small it is hard to be certain it is not caused by random noise, but they are still an indication that it can be interesting to look closer at these jump operators using a method with fewer numerical errors. As the channel fidelity on the Choi matrix has shown in tests a measurement error of up to 10^{-7} when measuring two identical matrices. Some of this error comes from taking the matrix square root, it may need a higher precision than the 64-bit double precision used by default in MATLAB.
The interesting part is that the change in optimal parameter values has a stable change with increasing gamma γ as seen in Fig. 5.26. But the effect is so small it may be only numerical errors.

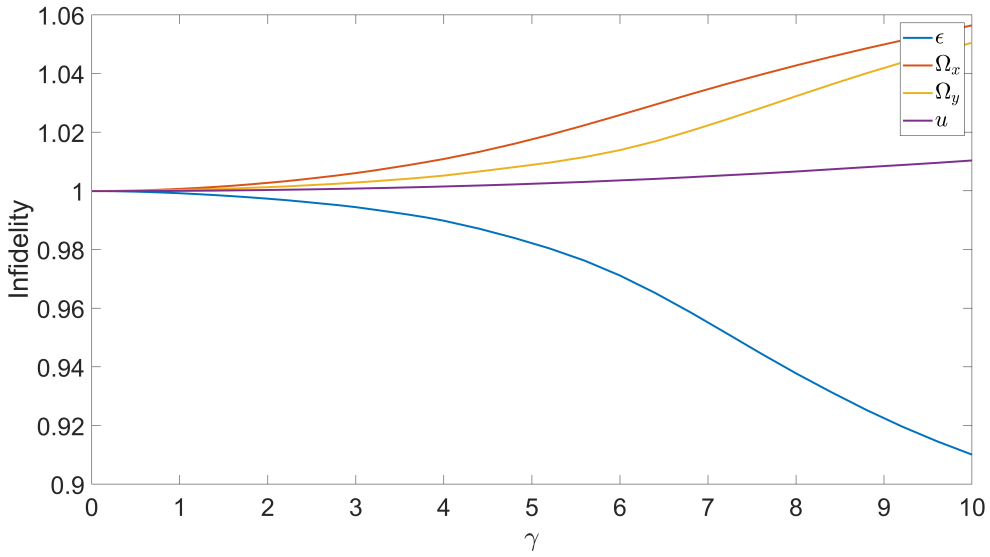

Figure 5.26: The evolution of the ideal parameters with increasing gamma. Using the jump operators from the tomography and the tautological gate created using 1 as parameters as target gate

5.5.3 Optimization for Lindblad with random jump operator

When I tried to use a randomly generated jump operator, created by the code in Lst. 4.16, the implementation of the method described in Sec. 3.6.5, gave me the jump operators in Eq. (4.10). I optimized the parameters for different γ values. This may be one of the easier jump operators to optimize since it may be less dominated by single qubit noise. In all figures I use a tautological gate created with a Hamiltonian so that all four parameters $\epsilon, \Omega_x, \Omega_y, u$ are set to one. This is also the starting point for when I optimize when γ increases.
In Fig. 5.27 you can see how the infidelity increases logarithmic, fast at the start and then more slowly approaching the maximum of the measurement as gamma increases. Starting at $\gamma = 0$ for no noise, with the infidelity rapidly increasing, before flattening out as $\gamma = 0.02$ and slowly approaching the maximum infidelity of the measure $1 - \frac{1}{2^4} = 0.9375$ as $\gamma \to \infty$. Compared to Fig. 5.24 in Sec. 5.5.2 the gamma values are much smaller, this is because the jump operators for the tomography have a prefactor that reduces the effect of these jump operators.

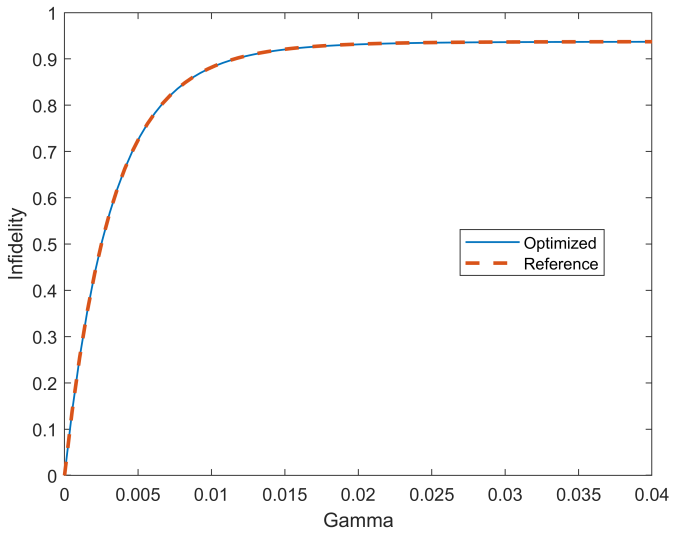

Figure 5.27: The measure for difference from an ideal tautological gate created with 1 as parameters and using random jump operators, showing both optimized and the original reference parameters

When optimizing parameters there is a significant difference when using a randomly generated jump operator, as seen in Fig. 5.28. Reaching a scale of 10^{-4} it is noticeable, but this may not be significant in a real quantum computer.
As you can see in Fig. 5.27 and keeping in mind Fig. 5.28 the difference is largest when the gamma value increases and the effect of the noise makes the infidelity of the Lindblad equation, compared to the target gate, larger. In this case, a gate

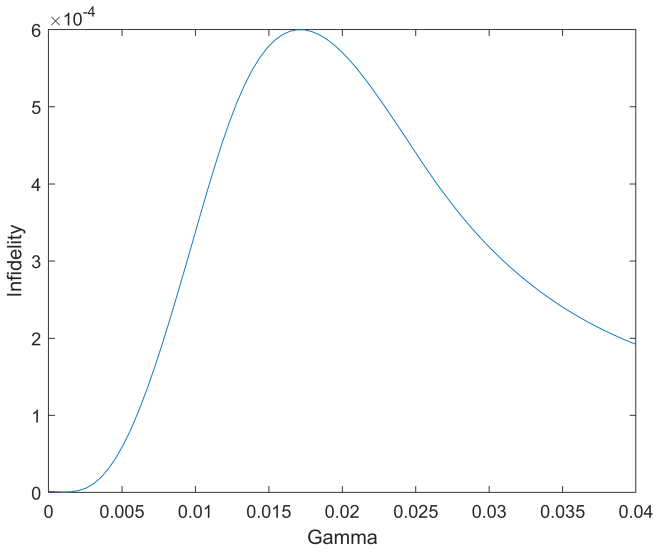

Figure 5.28: The difference between using optimized parameters and non optimized parameters on the infidelity comparing to a tautological target gate, for a given gamma. Using randomly generated jump operators

is constructed by finding the results of a Hamiltonian with a value of 1 for all parameters. The effect of the optimization decreases as it reaches a saturation point where the noise is so large that no changing of the parameters can compensate at larger γ values.
The fact that the parameters in Fig. 5.29 do not have a monotonic change while gamma increases, gives rise to some concern over how stable these optimized points are. But the change in parameters is so large as to be on the scale of 10%, this makes it more certain that the effects are real.
The fact that the parameters only begin to become unstable after γ reaches a value around 0.15 and the effect of the optimization is reduced, gives hope that the instability is only because the infidelity starts to become saturated around that point.

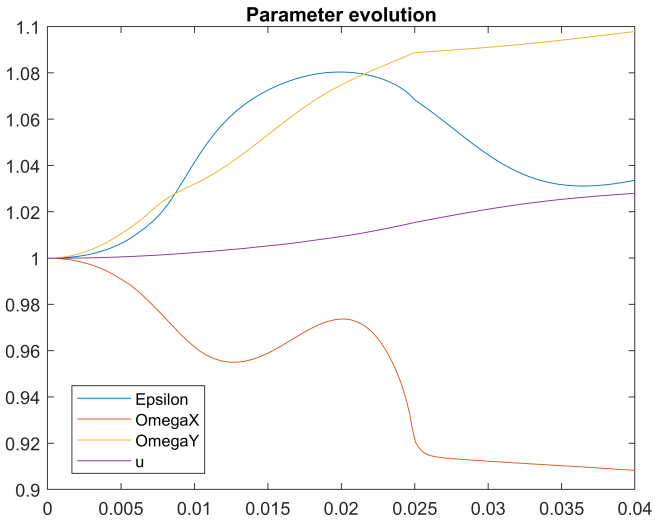

Figure 5.29: The evolution of the ideal parameters with increasing gamma. Using the random traceless jump operators and the tautological gate created using 1 as parameters as target gate

Chapter 6

Discussion

In this chapter I will discuss the results found in the previous chapter 5, noting some limitations. Though the results found may have limited effect, the trend of the effect does hint that some effect is existing.

6.1 Unitary maps

The cost landscapes were non-convex and hard to search using a gradient descent based algorithm such as the Adam optimizer. This made it harder to find good starting points, compared to using discrete magnetic pulses where for simple systems, such as a single qubit gate, it is possible to solve the Schrödinger equation or even the Lindblad equation analytically. The non-convex cost landscape made it harder to find start points to explore from, when not using the tautological gates. The tautological gates gave a good point to start when working with dynamic magnetic fields, they are only used to see the behaviour of a quantum system when manipulated with dynamic magnetic fields while optimising the parameters. While working with such uneven cost landscapes with unknown minima, a tautological gate guarantees that a solution does exist, and gives you a good starting point for optimization when exposing the qubit to disturbances given by the jump operators. You still need a good starting point for the gates you want to use, if you want to apply what you found when optimizing for the tautological gates in a quantum computer, as what is found with the tautological gates only gives information about what is possible and the effects may vary for different gates.

6.2 Constructing gates

As I can construct the Hadamard gate as seen in Sec. 5.3 and both the T gate and S gate as seen in Sec. 5.3.1, I can construct a universal set of gates for one qubit (M.

Nielsen & Chuang, 2010). The need for the CNOT gate for a completely universal set of gates limits me. The CNOT can be constructed by the $\sqrt{SWAP}$ gate, in combination with single qubit gates. The Hamiltonian I use, cannot discriminate between the qubits, therefore it cannot use a single qubit gate on only one qubit when working with two qubits. This means that it cannot be used to construct a universal set of gates for any number of qubits. Creating a Hamiltonian that can discriminate between qubits would expand the gates I can create to any arbitrary qubit gate. This means that using dynamic magnetic fields for the Ω_x and Ω_y fields and a static ϵ, can be used to create any arbitrary qubit gate, as long as you have sets of three individual magnetic fields for each qubit.

6.3 Ramp On and Ramp Off

The effect of using ramp on and ramp off, was so minor that it may not be worth focusing on. A real world laser may have different rates of turning on and off or the ramp on and off effect may not be symmetrical, it may have some gain in fidelity if you optimize for that effect. But it may only be applicable if you interact with a qubit with more than one laser at a time, as the fluctuation of a single laser may be counteracted by adjusting how long or how strong the pulses are.
It may be more interesting to see if differently shaped magnetic pulses would have an effect if their overlap were not complete.

6.4 Lindblad optimization

When using the Lindblad equation, it seems like there is a small but consistent effect of optimizing over the parameters when the simulated system is exposed to noise. This was only significant when using the generated random noise. But the effect was significant enough that it can be worth looking into this with a simulator that can have lower numerical errors. While the effect on a real quantum computer is limited this can be a part of the work to reduce the impact of noise. Even if the impact of noise reduction from dynamic magnetic fields is too small on its own.

6.4.1 Limitations

While there are simulations for four different sources of noise, the effect of the optimization for the noise from a tomography of a quantum computer and the amplitude dampening on a single qubit very limited effect. At some points the effect is so small it is hard to tell if it is caused by numerical errors or if it is an actual

effect. The largest improvement found was on the scale of 10^{-4}, a value that may be significant but has a negligible effect when used alone, so at most these results can be used to supplement other solutions.

Amplitude damping

The optimization on a single qubit was on the edge between being insignificant and significant, making the result rather uncertain. The evolution of the parameters gives, as with the other types of noise, an indication that the effect may exist. But it is too small to be certain of the result.

Tomography noise

For the tomography noise, it may be the case that it is dominated by the noise on a single qubit. This makes it quite uncorrelated and harder to mitigate, therefore it is reducing the possibility that it is possible to mitigate it with dynamic magnetic fields. But as this is a noise that was measured with tomography, it is known to have existed in a quantum computer at some point. As this was the case where the mitigation was the smallest, being at the scale of 10^{-5} its limited effect is disappointing.

Generated noise

The randomly generated noise was less dominated by single qubit noise, compared to the noise captured with tomography. Maybe this means that it is possible to get a better optimization for this noise source. I only use one jump operator for the randomly generated noise, it may therefore have been easier to correct for the effect of this noise, but it was still such a small improvement that it may be of limited use to reduce the noise experienced by the qubit.
There is a hint of an effect for the randomly generated traceless noise, but the result of optimizing for a randomly generated noise seems to have a limited effect when optimizing while using dynamic magnetic fields to counteract noise.

Chapter 7

Summary and Conclusions

This chapter starts with a small summary to refresh how the study was done and what was found. After this, I will come to a conclusion on what has been achieved and how useful it is, before ending with future work that can be done.

7.1 Summary

This project started with optimizing and finding parameters for single qubit gates using a dynamic magnetic field in the x and y direction and a static magnetic field in the z direction. To measure how good my simulation was to approximate the different gates, I used several measurement methods. I started with a norm measure, but because it gave problems to my optimization algorithm I swapped to average gate fidelity based on the quantum fidelity measure. Meanwhile, when measuring the simulations using the Lindblad equation I used Choi matrices to simplify the measurement process. The Lindblad equation was added to be able to simulate open quantum systems, and thereby be able to simulate the effect of noise.
The use of dynamic Hamiltonian and Lindbladian made it necessary to solve a time dependent differential equation. The method used is a Crank-Nicolson based solver that I used for Hamiltonians, and one version of the Lindblad equation, while the other versions of the Lindblad equation use a solver based on Runge-Kutta.
The code I created is with the use of the Lindblad master equation able to simulate three different types of noise in an open quantum system, by using jump operators to describe the different types of noise: amplitude damping, noise found with tomography, and a randomly generated noise. The program is created to be easy to expand, simplifying the method to add additional types of jump operators.
As I found a good approximation for the Hadamard gate, on one qubit with dynamic magnetic fields, I started to expand to two qubit gates, starting with a double Hadamard gate and the SWAP gate. This gave larger problems when I optimized the

parameters for the Hadamard gate, as the additional interaction parameter u made the optimization take longer with poorer results. In an attempt to get better infidelity, I tried dividing the interaction parameter into three, with a different interaction strength in all three directions. These additional parameters gave a slightly lower infidelity, but optimizing for six parameters greatly increased the time needed to perform the optimization, since it still had trouble with local minima, I decided to use another target when working with noise.
The choice was made to use tautological gates, gates constructed with known parameters, in an attempt to counteract the effect of noise. This was done both to have a good starting point to optimize when I tried to counteract the noise given by the Lindblad equation, and to make sure that there actually existed parameters that were guaranteed to give the best possible infidelity. While finding a good minima for the double Hadamard gate would be good, and may be possible, the primary focus for my study was on the possibility of mitigating the impact of noise.

7.2 Conclusion

The first takeaway of this study is that dynamic magnetic fields can be used to construct a set of universal quantum gates. This can then be used to create any arbitrary quantum gate. But the use of dynamic magnetic fields is not enough to meaningfully reduce noise as introduced by jump operators inducing non unitary evolution. The results have shown minimal improvements when counteracting the effects of noise. The noise reduction is on the border of being significant while the effect is too small for practical use, especially when the effect of optimizing parameters had the largest effect when the infidelity was too high already for any practical purpose.
During this project, I explored several different ways to measure the fidelity between two quantum maps. I found that the norm measure could be numerically unstable, creating difficult cost landscapes. To reduce the possibility that the measurement method introduced additional difficulty I tried average gate fidelity. It gave a more forgiving cost landscape to optimize in, similar to the channel fidelity I use for the Lindblad equation, since the average gate fidelity is based on the channel fidelity.
Even though the result of the optimization was minimal, I feel that this project has greatly increased my knowledge of quantum mechanics, outside of the single quantum information technology course in my studies. This project has given me knowledge with a focus on spin interactions, open quantum systems and the use of the Lindblad master equation. The need for density matrices and the use of Choi matrices when used to measure the difference between a quantum map and a target

gate, took a while to understand, but gave insight into different ways of representing a quantum state and quantum maps. Kraus operators were never directly used, but instead simplified away for one of the measurement methods or simply not chosen as the implementation of Choi matrices was seen as easier to implement.
During this project, I created a general code framework, that can be used to rapidly create new Hamiltonians or Lindbladians and test how easily the Adam algorithm can optimize the parameters. When using new Lindbladians I can rapidly test how different forms of magnetic fields behave, when exposed to many types of noise. The code is able to quickly solve dynamic equations, which are in the form of differential equations. It is useful, because when you want to see the effect of a Hamiltonian, you have to solve a differential equation which may need many timesteps to be accurate, as the equation is dynamic and therefore different for each timestep.
The spin interactions this project models are able to be generalized to any two level quantum system. The method used to construct a qubit does not matter, but the implementation may change what tools are available to manipulate the qubit.

7.3 Future work

The smooth evolution of the optimal parameters suggests that a more detailed future study, of what effects have the largest impact on reducing noise, might be interesting. Because the effect found in this study is so small when used on its own when mitigating noise, it will need confirmation with a setup less sensitive to machine errors than the one used here. As it is unclear whether it is possible to counteract non unitary noise using unitary interactions.
A topic that can be studied in more detail is how the dynamic magnetic field affects the construction of a double Hadamard gate on two qubits, while setting a non-zero constant interaction parameter. It can be trivially constructed without interaction between the parameters, so the interesting part is if you can counteract the effect of the interaction only using the dynamic magnetic fields.
Another possibility is to try another type of noise, having magnetic fields acting on two qubits, that interact with each other, and tracing out the second one when you measure it. This gives rise to a different type of noise than the jump operators in the Lindblad equation. This new noise will be non-Markovian, containing the memory of earlier interactions.
To reduce the time needed to find the gradient, the code can be modified to use other methods than the finite difference method to find the gradient, for example finding it analytically or with auto differentiation. The time taken to find the gradient limited how much I could use the Adam optimizer to find minima.

There was never a goal that the Hamiltonian should be an exchange symmetrical model, the Hamiltonian is more restrictive than needed. The code I produced can in a future project be easily extended with different Hamiltonians and Lindbladians that can discriminate between the qubits being manipulated, choosing which one to manipulate.
A project with broader prospects may instead be to study the effect of pulses closer to the magnetic or laser pulses in use in the systems of today, and from there change them to experiment with the shape and overlap of the pulses and see if that has a greater effect. This way, you may see the effect of manipulating the pulses in use in the quantum computers of today.

Bibliography

Aaronson, S., & Gottesman, D. (2004). Improved simulation of stabilizer circuits. *Phys. Rev. A*, *70*, 052328. https://doi.org/10.1103/PhysRevA.70.052328

Benioff, P. (1980). The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of Statistical Physics*, *22*, 563–591. https://doi.org/10.1007/BF01011339

Braunstein, S. L., & Pati, A. K. (2007). Quantum information cannot be completely hidden in correlations: Implications for the black-hole information paradox. *Phys. Rev. Lett.*, *98*, 080502. https://doi.org/10.1103/PhysRevLett.98.080502

Breuer, H.-P., Petruccione, F., et al. (2002). *The theory of open quantum systems*. Oxford University Press on Demand.

Chruściński, D., & Pascazio, S. (2017). A brief history of the gkls equation. *Open Systems & Information Dynamics*, *24*(03), 1740001. https://doi.org/10.1142/S1230161217400017

Cook, S. (2001). The p versus np problem. *The millennium prize problems*. http://www.claymath.org/millennium-problems/p-vs-np-problem

Crank, J., & Nicolson, P. (1947). A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Mathematical Proceedings of the Cambridge Philosophical Society*, *43*(1), 50–67. https://doi.org/10.1017/S0305004100023197

Deutsch, I. H., Brennen, G. K., & Jessen, P. S. (2000). Quantum computing with neutral atoms in an optical lattice. *Fortschritte der Physik*, *48*(9-11), 925–943. https://doi.org/10.1002/1521-3978(200009)48:9/11<925::aid-prop925>3.0.co;2-a

Dial, O. (2022). *Eagle's quantum performance progress*. Retrieved May 18, 2022, from https://research.ibm.com/blog/eagle-quantum-processor-performance

Fan, H., Roychowdhury, V., & Szkopek, T. (2005). Optimal two-qubit quantum circuits using exchange interactions. *Phys. Rev. A*, *72*, 052323. https://doi.org/10.1103/PhysRevA.72.052323

Feynman, R. P. (1982). Simulating physics with computers. *International Journal of Theoretical Physics*, *21*(6), 467–488. https://doi.org/10.1007/BF02650179

Finnila, A., Gomez, M., Sebenik, C., Stenson, C., & Doll, J. (1994). Quantum annealing: A new method for minimizing multidimensional functions. *Chemical Physics Letters*, *219*(5), 343–348. https://doi.org/https://doi.org/10.1016/0009-2614(94)00117-0

Gerald, T. (2014). Mathematical methods in quantum mechanics with applications to schrödinger operators, mathematical methods in quantum mechanics with applications to schrödinger operators. *American Mathematical Society, Providence, RI*. https://doi.org/https://doi.org/10.1090/gsm/157

Gorini, V., Kossakowski, A., & Sudarshan, E. C. G. (1976). Completely positive dynamical semigroups of N-level systems. *Journal of Mathematical Physics*, *17*(5), 821–825. https://doi.org/10.1063/1.522979

Griffiths, D. J. (1995). *Introduction to quantum mechanics* (1st).

Hellwig, K. E., & Kraus, K. (1969). Pure operations and measurements. *Communications in Mathematical Physics*, *11*(3), 214–220. https://doi.org/10.1007/BF01645807

Johansson, M. (2022, June 9). Three different ways of connection hpc and qc. Retrieved June 2022, from https://enccs.github.io/NordIQuEst-workshop/hybrid_hpc_qc/

Karlsson, E. B. (2022). Internal dynamics in condensed matter, as studied by spin relaxation: Some examples from 75 years. *The European Physical Journal H*, *47*(1), 4. https://doi.org/10.1140/epjh/s13129-021-00030-9

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. https://doi.org/10.48550/ARXIV.1412.6980

Lindblad, G. (1976). On the generators of quantum dynamical semigroups. *Communications in Mathematical Physics*, *48*(2), 119–130. https://doi.org/10.1007/BF01608499

Magesan, E., Blume-Kohout, R., & Emerson, J. (2011). Gate fidelity fluctuations and quantum process invariants. *Physical Review A*, *84*(1). https://doi.org/10.1103/physreva.84.012309

Manzano, D. (2020). A short introduction to the lindblad master equation. *AIP Advances*, *10*(2), 025106. https://doi.org/10.1063/1.5115323

Mills, A. R., Guinn, C. R., Gullans, M. J., Sigillito, A. J., Feldman, M. M., Nielsen, E., & Petta, J. R. (2022). Two-qubit silicon quantum processor with operation fidelity exceeding 99%. *Science Advances*, *8*(14), eabn5130. https://doi.org/10.1126/sciadv.abn5130

Nielsen, E. K. (2011). *Quantum computation, theory of.* Retrieved May 22, 2022, from http://encyclopediaofmath.org/index.php?title=Quantum_computation,_theory_of&oldid=50733 Updated 1 July 2020

Nielsen, M., & Chuang, I. (2010). *Quantum computation and quantum information* (10th Anniversary Edition). Cambridge University Press.

Nielsen, M. A. (2002). A simple formula for the average gate fidelity of a quantum dynamical operation. *Physics Letters A*, *303*(4), 249–252. https://doi.org/10.1016/s0375-9601(02)01272-0

Papanikolaou, C. (2021). *Construct single qubit gates with spin in magnetic fields* (Unpublished Work).

Pechen, A., & Il'in, N. (2012). Trap-free manipulation in the landau-zener system. *Phys. Rev. A*, *86*, 052117. https://doi.org/10.1103/PhysRevA.86.052117

Pechen, A. N., & Tannor, D. J. (2011). Are there traps in quantum control landscapes? *Phys. Rev. Lett.*, *106*, 120402. https://doi.org/10.1103/PhysRevLett.106.120402

Preskill, J. (2018). Quantum Computing in the NISQ era and beyond. *Quantum*, *2*, 79. https://doi.org/10.22331/q-2018-08-06-79

Rabitz, H. A., Hsieh, M. M., & Rosenthal, C. M. (2004). Quantum optimally controlled transition landscapes. *Science*, *303*(5666), 1998–2001. https://doi.org/10.1126/science.1093649

Riebesell, J. (2022, April). Bloch-sphere.tex. https://github.com/janosh/tikz/blob/main/assets/bloch-sphere/bloch-sphere.tex

Samach, G. O., Greene, A., Borregaard, J., Christandl, M., Barreto, J., Kim, D. K., McNally, C. M., Melville, A., Niedzielski, B. M., Sung, Y., Rosenberg, D., Schwartz, M. E., Yoder, J. L., Orlando, T. P., Wang, J. I.-J., Gustavsson, S., Kjaergaard, M., & Oliver, W. D. (2022). Lindblad tomography of a superconducting quantum processor. *Physical Review Applied*, *18*(6). https://doi.org/10.1103/physrevapplied.18.064056

Samal, J. R., Pati, A. K., & Kumar, A. (2011). Experimental test of the quantum no-hiding theorem. *Phys. Rev. Lett.*, *106*, 080401. https://doi.org/10.1103/PhysRevLett.106.080401

Schrödinger equation. (2011). Encyclopedia of Mathematics. Retrieved April 29, 2023, from http://encyclopediaofmath.org/index.php?title=Schr%C3%B6dinger_equation&oldid=40003. Updated 14 December 2016

Sen, R. (2013). *A first course in functional analysis: Theory and applications*. Anthem Press. https://doi.org/10.7135/9780857282224

Shor, P. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 124–134. https://doi.org/10.1109/SFCS.1994.365700

Süli, E., & Mayers, D. F. (2003). *An introduction to numerical analysis*. Cambridge University Press. https://doi.org/10.1017/CBO9780511801181

Vandersypen, L. M. K., & Eriksson, M. A. (2019). Quantum computing with semiconductor spins. *Physics Today*, *72*(8), 38–45. https://doi.org/10.1063/PT.3.4270

Xue, C., Chen, Z.-Y., Wu, Y.-C., & Guo, G.-P. (2019). Effects of quantum noise on quantum approximate optimization algorithm. https://doi.org/10.48550/ARXIV.1909.02196

Appendix A

Code

A.1 Repository

The code repository used is located at: https://github.com/DiedrikL/Master-Project-Semester-1