Williams College Libraries
Thesis Release Form

All theses are available online to Williams users with a Williams log-in and password. Select **one** response for each question. Form to be completed jointly by student and faculty member.

ACCESS TO YOUR THESIS

Faculty claims co-authorship?
X_ No
___ Yes

When do you want your thesis made available to any user beyond Williams?
X_ Now
___ 5 years
___ 10 years
___ After lifetime of author(s)

OWNERSHIP/COPYRIGHT

Theses that contain copyrighted material cannot be made available beyond Williams users. Does your thesis contain copyrighted materials without copyright clearance?
___ No
X_ Yes (Copyrighted sections of the thesis will not be made available online. You have the option to submit a second version of the thesis omitting copyrighted material. Contact College Archives for details, archives@williams.edu)

You own copyright to your thesis. If you choose to transfer copyright to Williams, the College will make your thesis freely available online. When do you want to transfer copyright?
X_ Now
___ In 5 years
___ In 10 years
___ After lifetime of author(s)

Please provide a brief (1-5 sentences) description of your thesis.

We determined whether algorithms for the Quantum Fourier Transform (QFT) using qudits (quantum digits), rather than qubits (quantum bits), offer improved time performance. Numerical modeling of the QFT using gradient search methods in Matlab and C++ have suggested that qudit-based systems are less time-efficient than qubit-based systems for small systems.

Williams College Libraries
Thesis Release Form

Changes to the thesis release form require a new form to be completed, signed and returned to Special Collections in Sawyer Library.

Theses can be viewed in Special Collections; print copies of Division III and Psychology theses are available at Schow Science Library.

Direct questions about this form to the College Archivist (archives@williams.edu).

Title of thesis: *Quantum Optimal Control of Large-Dimensional Systems*

Author(s): Samuel T. Amdur

Signatures:

Student (Print): Samuel T Amdur Date: 11/17/2016

Student (Signature): Samuel T Amdur Date: 11/17/2016

Faculty (Print): _____ Date: _____

Faculty (Signature): _____ Date: _____

Faculty (Print): _____ Date: _____

Faculty (Signature): _____ Date: _____

Faculty (Print): _____ Date: _____

Faculty (Signature): _____ Date: _____

Faculty (Print): _____ Date: _____

Faculty (Signature): _____ Date: _____

Quantum Optimal Control of Large Dimensional Systems

Samuel TH Amdur IV

Professor Frederick W. Strauch, Advisor

A thesis submitted in partial fulfillment of the requirements
for the degree of Bachelor of Arts with Honors in Physics

Williams College
Williamstown, Massachusetts

May 22, 2015

Abstract

The quantum Fourier transform (QFT) has been shown to offer an exponential speedup over classical Fourier Transform algorithms. We attempt to use optimal control to determine whether QFT implementations using qudits, rather than qubits, offer improved time performance. Numerical modeling of the QFT using gradient search methods suggests that qudit-based systems may be more time-efficient than qubit-based systems for small systems. We also study the scaling of computational time for single-qudit systems implementing the QFT, finding preliminary evidence that the timesteps needed for a single-qudit system to implement the QFT is linear in the dimension of the system. We implement improvements to existing gradient search methods to allow for the scaling of numerical simulation to larger quantum systems.

Acknowledgements

It is hard for me to imagine completing my thesis without the support of my family, friends, and professors. I would first like to thank my loving parents, who have supported me throughout every turn in my life's journey through the world of science. I want to thank my brother and sister, who have been the best siblings I could ever ask for.

My friends in the physics department have been invaluable throughout my time at Williams. I want to give a huge thanks to Gabriel Samach and Cole Meisenhelder, who have been problem set, tutorial, and thesis partners through thick and thin. Our lively discussions about physics, and everything else, will always stand out in my memories of the Science Quad. Thanks to Julia Cline and Brandon Ling for their friendship, help, and support throughout uncountable nights in the PCR. I would like to thank my housemates, Jack Hoover, Liz Dietz, Phonkrit Tanavisarut, and Jeremy Boissevain, for their support, and for being great friends in general, throughout the past year.

I would like to thank my second reader, Professor Daniel Aalberts, for his thoughtful and thorough feedback that has greatly helped me create this final product. Most of all, I want to thank my adviser, Professor Frederick Strauch, for the many hours of patience and guidance he has devoted to my learning throughout this endeavor. He has been instrumental in making my thesis such a wonderful research experience.

Contents

Executive Summary **ix**

1 Introduction and Background **1**
1.1 Motivation . . . 1
1.2 Quantum Optimal Control . . . 2
1.3 Quantum Fourier Transform . . . 4
1.4 Research Goals . . . 5

2 Theoretical Background **7**
2.1 Qubits and Qudits . . . 7
2.2 Defining Unitary Operators . . . 8
2.3 Hamiltonians . . . 9
2.3.1 Control Hamiltonian . . . 10
2.3.2 Drift Hamiltonians . . . 10
2.4 Constructing Hamiltonians . . . 11
2.5 Qudit-Qudit Interactions . . . 11
2.6 The Discrete Fourier Transform . . . 13

3 GRAPE Implementation **15**
3.1 The Optimal Control Problem . . . 15
3.2 Algorithm Overview . . . 16
3.3 Matrix Exponentiation . . . 17
3.4 Gradient Calculation . . . 20
3.4.1 Preparing the Derivative Function . . . 21
3.4.2 Eigenvalue Decomposition (EVD) . . . 22
3.4.3 Commutator Approximation . . . 23
3.4.4 Simpson's Rule Approximation . . . 24
3.4.5 Comparison of Gradient Calculation Methods . . . 24
3.5 Adaptive Control Update Step . . . 26
3.5.1 Fixed Step Size . . . 26
3.5.2 Empirical Cauchy Method Adaptive Step . . . 26

3.5.3 Barzilai and Borwein Adaptive Step . . . 27
3.6 Implementing GRAPE in C++ . . . 27
3.7 Algorithm Behavior and Limitations . . . 29

4 Numerical Simulations **33**
4.1 Testing QFT Structure . . . 33
4.1.1 Analytic Bound . . . 34
4.1.2 Numerical Comparison . . . 35
4.2 Qubit-Qudit Comparison . . . 38
4.2.1 Considerations when Comparing Qubits and Qudits . . . 38
4.2.2 Qubit-Qudit Comparison Results . . . 39

5 Conclusion and Discussion **41**

A MPI Implementation **43**
A.1 MPI Overview . . . 43
A.2 Implementing MPI for use in GRAPE . . . 44
A.3 MPI Challenges and Future Additions . . . 45

B MATLAB Documentation **47**

C C++ Documentation **55**

Executive Summary

The discrete Fourier transform, used in a multitude of applications, is an enormously important algorithm. Because it is so extensively used, the development of a Fourier transform algorithm using quantum computing is of great interest. The quantum Fourier transform, or QFT, has the exact same mathematical properties as the discrete Fourier transform but acts upon an input vector composed of a quantum superposition of energy eigenstates. The QFT is known to be exponentially faster than the fast Fourier transform or any other discrete Fourier transform algorithm run on classical computers. While experimental implementations of the QFT have been performed on relatively small scale systems, it remains an important question to find efficient implementations for larger systems. One method of investigating large QFT operations is to conduct a numerical search for a series of operations that, when combined, will together compose the QFT.

In order to conduct our numerical search, we must first define the search space for our investigation. We attempt to re-express the QFT as a product of N_t time-independent Hamiltonian operators $\mathcal{H}$ acting over time on an input state $|X\rangle$:

$$U_{QFT}\,|X\rangle = e^{-i\Delta t\mathcal{H}_{N_t}/\hbar}e^{-i\Delta t\mathcal{H}_{N_t-1}/\hbar}\ldots e^{-i\Delta t\mathcal{H}_2/\hbar}e^{-i\Delta t\mathcal{H}_1/\hbar}\,|X\rangle\,, \tag{1}$$

Each Hamiltonian operator has freely varying parameters that can be controlled, as well as other parameters that are fixed and reflect either interactions between qubits or the physical characteristics of the qubits. Our expression models an implementation of a quantum computer where the system is transformed by discrete pulses of energy, such as from a laser acting upon trapped ions. With the use of a search algorithm, we can adjust the freely varying parameters, or controls, to create a combination of Hamiltonians that yields the QFT, or a close approximation of it.

There are two broad questions which we hope to address with the use of our numerical search strategy. The first: how can we perform numerical search as efficiently as possible? A faster search algorithm allows us to explore solutions for the QFT at larger scales which are computationally taxing to search. The second: what can our search algorithm tell us about the nature of the QFT and the types of quantum systems that are best suited to implement the QFT? Namely, we wish to investigate whether we can efficiently execute the QFT if we represent quantum information using quantum digits, or qudits, of size d, each representing a system with d energy eigenstates. How well would a quantum computer using qudits scale for larger QFT problems? And finally, is a computer that uses qudits

faster than one that uses qubits? These questions are addressed in the second half of the thesis.

We first worked to improve our search strategy, which is a form of gradient search known as GRadient Ascent Pulse Engineering (GRAPE). With GRAPE, we treat the calculation of the QFT as an optimal control problem. GRAPE is guided by a cost function, in this case the degree of overlap between our numerically generated operation and the QFT. At every iteration, we calculate the gradient of this overlap with respect to our freely varying parameters (controls), and use the gradient to adjust the controls and produce a numerical operation that comes closer to approximating the QFT. With repeated iterations, we thus use GRAPE to search for a set of controls that closely approximates the QFT. With this type of search, we saw three areas for improvement. Rather than calculating the exact gradient with every iteration, we used matrix exponential approximation techniques to allow for the calculation of an approximation of the gradient. This method gives us the flexibility of trading simulation accuracy for faster computational speed, if desired. We also raised the efficiency of our gradient search by improving how the gradient is applied to the controls with each iteration. By implementing an adaptive gradient "step," we improve the extent to which each iteration of our search makes progress towards replicating the QFT. Lastly, we implemented GRAPE in C++, where parallel processing can be used to scale the search algorithm onto larger computing clusters. With these improvements in hand, we now have the flexibility to implement a gradient search that can make time-accuracy tradeoffs when it is beneficial and have the ability to tackle large search problems solving for the implementation of large-scale QFT operations.

We next turned to numerical simulations of qudit systems running the QFT. The second half of the thesis studied what our search algorithm reveals about the implementation of the QFT using qudits. We first looked at a single-qudit system, where a single qudit stores information on the starting state and a Fourier transformation of the starting state will yield a qudit storing the Fourier transformed final state. If we operate upon this qudit with tridiagonal Hamiltonians, we predict that the number of elements in the matrix representation of the QFT should be roughly equivalent to the number of controllable parameters throughout our Hamiltonians. If we equate the number of QFT matrix elements to the the the number of controls, then we find a relation for the number of Hamiltonian timesteps N_t needed to build a QFT of size d: $N_t = (d^2 - 1)/(3d - 3)$. When we searched for solutions to the QFT using single-qudit systems from size 2 to 16, where size is the number of energy eigenstates represented by the qudit, we found that our successful searches required approximately as many timesteps as was predicted by our relation (See Fig. 1).

Next, we compared implementations of the QFT using qubits versus those using qudits. As a case study, we examined the situation in which the QFT is generated using either a 4 qubit system or a 2 qudit system where each qudit has a size of 4. Both systems can implement a QFT operating on a sample with 16 inputs. We defined a cross-Kerr coupling for the qudits that matches the coupling strength of spin-spin coupling of qubits

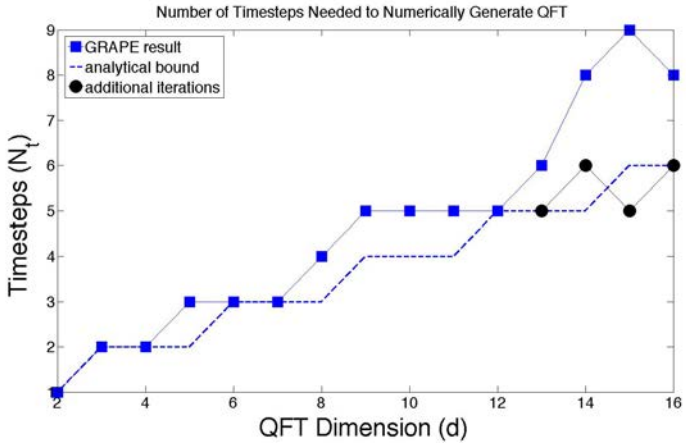
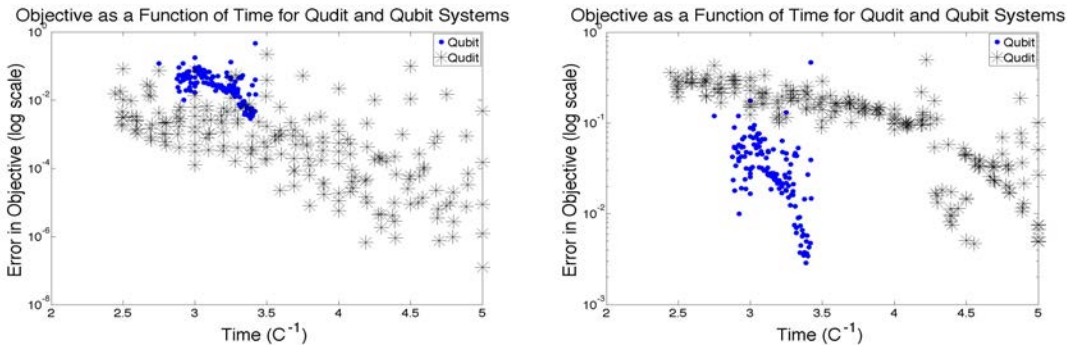


Figure 1: The number of Hamiltonian timesteps N_t required to numerically generate a QFT with objective error $\leq 10^{-5}$ for a given QFT size d. The dotted line denotes the timestep bound determined by our analytical prediction, while the markers indicate the results found by our GRAPE search. Circles mark results found after 2.5 times more iterations. For single-qudit systems of dimension 2 through 16, our empirical results adhere closely to the prediction, indicating that the number of timesteps required to generate the QFT has a $\sim d/3$ relationship to the qudit dimension.

in the Ising model. When we controlled for system size and coupling strength, our search found that the qudit system can implement the QFT in less time than the qubit system (See Fig. 2). Nevertheless, changes in coupling strength can have large effects on the amount of time necessary to construct the QFT, and if the coupling strength for the qudit system is weakened its runtime increases dramatically. Thus, we can only conclude that qudits are more effective than qubits for small implementations of the QFT if they can be physically implemented in a system with coupling strengths commensurate to those seen in qubit systems. In the future, we hope to use the tools we developed to examine larger systems. We look forward to using our search strategies to better understand what type of quantum system will lead to a faster quantum Fourier transform for large dimensional systems.



(a) Commensurate Coupling Force

(b) Weak Qudit Coupling

Figure 2: The objective as a function of scaled run time for various qudit and qubit implementations. (a) Comparison for a qudit system with a coupling coefficient equal to the coupling coefficient in the qubit system. (b) Comparison for a qudit system with a coupling coefficient 1/4 of the qubit system coupling coefficient.

Chapter 1

Introduction and Background

1.1 Motivation

Time matters in the world of quantum computing. It matters because a system that can execute a series of operations in a faster time is able to compute more quickly. Perhaps more importantly, it matters because of the fragile nature of quantum computers. Quantum computers, as currently conceived, trap systems of particles or packets of energy in a way that separates the system from the outside environment. By isolating these systems, operations and measurements can be performed on them, allowing us to "program" a system to perform a computation. The properties of a quantum computer degrade over time, however, due to the phenomenon of quantum decoherence: as time passes, a quantum system is more likely to interact with its environment and lose the information it has stored within itself. In short, decoherence introduces error in the computation. Because of this, executing a computation as quickly as possible is paramount in improving the performance of quantum computing devices.

There are numerous ways for any given algorithm to be implemented in a quantum system, and for this reason the set of instructions provided to the system matter greatly in determining time efficiency. In some ways, one can consider the search for a time optimal set of instructions to be analogous to the brachistochrone problem: how does one shape a wire starting at point A and ending at point B such that a frictionless bead sliding down the wire under the force of gravity traverses the wire in the least time? There are an infinite number of wire shapes that can accomplish this goal, but only one that minimizes the time required to get the bead to its target. Likewise, there are an infinite number of ways to operate upon a quantum system that will yield a desired computation, but only a certain subset of instructions that will yield a solution to the computation in minimal time.

There have been many attempts to define an analytic approach to finding time-optimal instructions for quantum systems [6, 19, 16]. Most utilize elementary "quantum gates," logic operations that can be carried out by quantum systems and built up to construct

larger quantum computations in a manner that is analogous to the construction of classical computing operations from classical logic gates [19]. Quantum gates allow for operations that cannot be implemented in classical computing, however, thus weaving the benefits of quantum computing into analytically-solved control schemes. These approaches, while effective in generating instruction schemes for algorithms such as the Quantum Fourier Transform (QFT), have been shown to be less time-optimal than approaches that attempt to numerically solve for the fastest set of controls using algorithmic search strategies (See Fig. 1.1). The filled symbols in Fig. 1.1 show the gate complexity (left) and run time (right) of QFT algorithms determined analytically by two different research groups [6, 19], while the open symbols reflect the results found through numerical search approaches [20]. Analytic solutions are constrained by the structure of elementary quantum gates, whereas solutions found through numerical search allow for arbitrarily constructed gates. Fig. 1.1 indicates that for this reason, numerical approaches can offer much faster operations than analytical approaches.

Numerical approaches to quantum computing benefit from allowing for arbitrary operations upon the system. We can define freely-varying parameters (which we will call controls) in a quantum system and begin a search that iteratively improves the controls until a time optimal set of instructions is found. For our research, we used a gradient search process with the acronym of GRAPE (which stands for GRadient Ascent Pulse Engineering). GRAPE begins with a guess of the controls that together will generate a desired operation, and with each iteration it improves upon its guess until a solution is found with a desired degree of accuracy.

1.2 Quantum Optimal Control

Optimal control describes the numerical methods by which an "optimal" set of controls are found. In the case of quantum systems, it determines the set of controls that can be applied to a quantum system in order to carry out a given computation. We can better understand the nature of optimal control by extending our analogy of the brachistochrone problem as an example of optimization. The brachistochrone problem is a question of time optimization: in order to solve the problem of how to move a bead along a wire in the shortest time, one defines an *objective function* to be minimized. In this case, the objective function is the total time the bead spends along the wire. The objective function is used to help define a function, $y(x)$, that *controls* the amount of time spent along the wire by defining the shape of the wire's surface. In this case, with the use of the Euler-Lagrange equation, the controls (the shape of the wire) are optimized to minimize the objective function [22].

Quantum optimal control follows the same basic premises. The *objective function* in our case is a function quantifying the fidelity of a numerically approximated matrix transformation, U_{QFT}, to a desired transformation (here we look at the quantum Fourier transform, or QFT). In other words, it measures how close a computational operation built

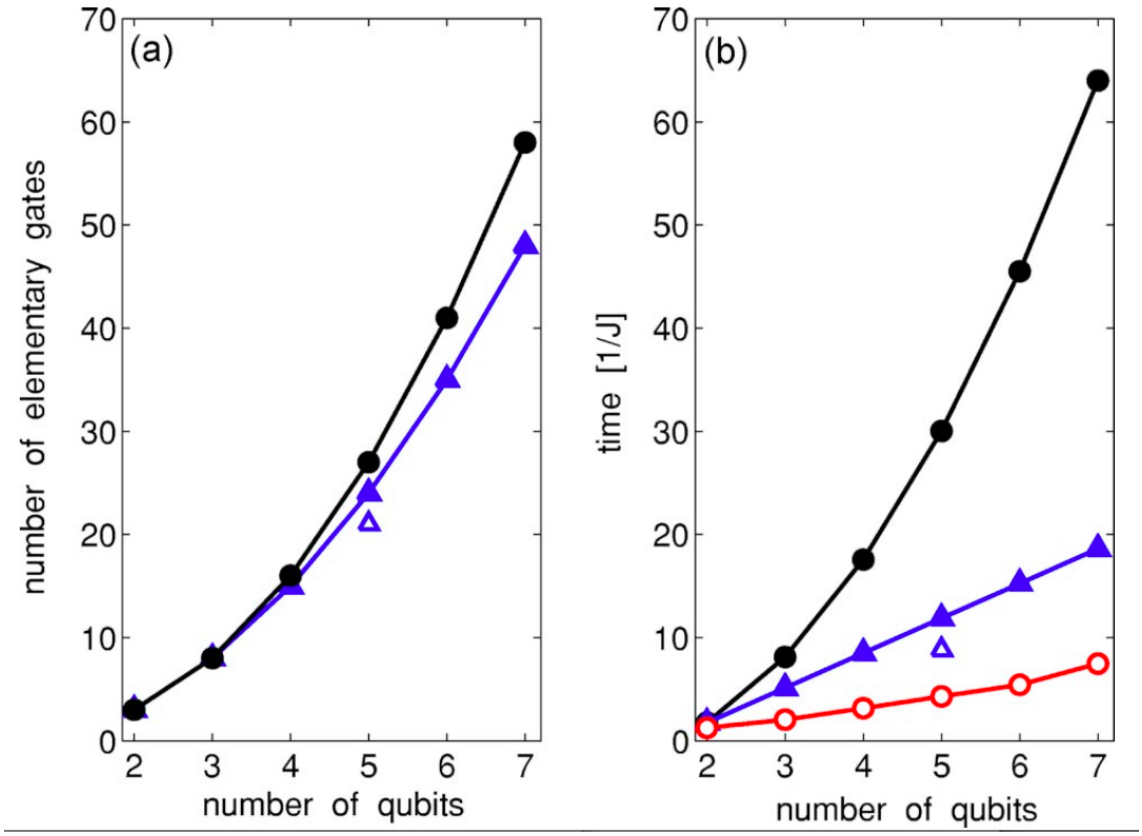


Figure 1.1: Reproduced from [20]: A comparison of gate complexity, left, and run time, right, for implementations of the QFT using analytic and numerical approaches.

by our set of controls comes to recreating the operation we want to replicate. We wish to maximize our objective function. Our *controls* are a set of complex numbers that define the changes in energy applied to our system in order to carry out quantum computations. These changes in energy are represented by time-independent Hamiltonian operators $\mathcal{H}_j$ that are each applied to the system over a time Δt:

$$U_{QFT}\left|X\right\rangle = e^{-i\Delta t\mathcal{H}_j/\hbar}e^{-i\Delta t\mathcal{H}_{j-1}/\hbar}\ldots e^{-i\Delta t\mathcal{H}_2/\hbar}e^{-i\Delta t\mathcal{H}_1/\hbar}\left|X\right\rangle, \tag{1.1}$$

where $|X\rangle$ is the state vector being acted upon. Rather than the Euler-Lagrange equation, we use the optimization technique of hill climbing, where the gradient of our objective function is calculated with respect to the controls. The gradient lays out the direction in which to change the controls such that we "climb" up to a higher objective function and U_{QFT} approaches the form of the actual quantum Fourier transform.

While quantum optimal control optimizes the fidelity of our controls, in each individual search the structure of the system is predefined and not subject to optimization. In our research, we can compare different system architectures by conducting multiple searches, changing for each search the dimension d_q of each quantum digit, the number of quantum digits in the system, N_q, and the total number of timesteps over which the system evolves to carry out a computation, N_t. By finding optimal controls for systems with different permutations of these variables, we can begin to get a picture of how quantum systems should be constructed to best carry out certain computations, such as the Fourier transform.

1.3 Quantum Fourier Transform

The quantum Fourier transform (QFT) promises to be an algorithm that can be run effectively on quantum computers. When expressed as an operator, it serves as a generalized form of the Hadamard gate, a quantum logic gate that is essential to many quantum operations. For example, QFT is a crucial component of Schor's algorithm, which can be used for finding prime factors and breaking RSA encryption [12].

We can understand the QFT better by first considering its classical analogue, the discrete Fourier transform. The discrete Fourier transform (DFT) maps the values of a discrete set of N points from a "time" domain to a "frequency" domain. It is a discrete version of the Fourier transform, which is expressed as follows:

$$g(k) = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(x)e^{-ikx}dx, \tag{1.2}$$

where $g(k)$ is the Fourier transform of $f(x)$. This mapping can be reversed, albeit without the coefficient of 2π in front.

The DFT mapping has traditionally been useful in the fields of signal processing and spectral analysis [1]. Instead of performing a transform on a continuous function, one can

also perform the Fourier transform on a discrete set of complex numbers $f_0, \ldots, f_k, \ldots, f_{N-1}$ The Fourier transformed coefficiencts $\tilde{f}_j$ are given by:

$$\tilde{f}_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega^{-jk} f_k, \tag{1.3}$$

where $\omega = \exp(\frac{2\pi i}{N})$ and jk is the product of two indices. This can also be expressed as a unitary transformation matrix of the form $\vec{Y} = \mathrm{DFT}_N \vec{X}$ where $\vec{X}$ represents the vector of complex numbers f_k. For DFT_4, the transformation operator upon a $\vec{X}$ of size 4 is given by:

$$DFT_4 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix}. \tag{1.4}$$

The fastest classical algorithm for the DFT, the fast Fourier transform, has a time complexity of $O(n2^n)$ [1], where n is the number of elementary bit operations.

The QFT is the DFT applied to a set of amplitudes representing quantum states and has the exact same matrix form. One can represent the DFT as a unitary matrix operation on a quantum system defined by an initial state represented by the vector $\vec{X}$. In the quantum case, $\vec{X}$ represents the superposition of energy eigenstates of a system of qubits or qudits. While the QFT performs the exact same operation as the DFT, we wish to use quantum computing to calculate Fourier transforms because algorithms for the QFT offer an exponential speedup over the DFT, with a time complexity of $O(n^2)$ elementary qubit operations [16]. Because of the benefits offered by the QFT over the DFT, it is an operation that merits further research and will be the focus of our thesis.

1.4 Research Goals

We have organized our research around two main questions that we wish to address:

1. **Can quantum optimal control be used to design the computational operations of large quantum systems?**

   Attempts to use optimal control in order to develop sets of instructions for quantum systems suffer from computational limitations. The amount of classical computation time required to construct a set of optimal controls increases as the dimension of the problem increases. As a result, large quantum systems are poorly examined, and searches for sets of optimal controls can prove futile due to computational limitations.

   We hope to improve a gradient search implementation of optimal control, known as GRAPE, in order to improve the ability of optimal control to address large scale

quantum systems. From a computational efficiency standpoint, GRAPE has the advantage of utilizing the unitary structure of control operators in order to conduct its search with less computational cost than other search methods. We developed approximation schemes that help to expedite the GRAPE search process, as well as a C++ implementation of GRAPE that utilizes parallel processing in order to harness the power of larger computing clusters.

2. **Are qudit-based quantum systems more efficient for certain applications than systems using qubits?**

   We are motivated to understand what types of quantum systems are best suited to carrying out certain types of algorithms. The QFT can be carried out on a quantum computer using different sized units of information. A quantum computer could be built upon the quantum implementation of the bit (known as a "qubit"), or a larger dimensional unit, or "qudit," that holds more information per unit. Additionally, there are numerous ways in which one can expect systems of qubits or qudits to interact with each other in physical systems. Ions, for example, can have electric and magnetic dipole moments that could influence nearby ions. Electromagnetic waves in media have been shown to control the index of refraction experienced by a second set of EM waves passing through the material, thus creating an effective photon-photon interaction [21]. Because both the choice of qubit vs. qudit and choice of interaction scheme affect the types of controls that can be manipulated in a system, we wish to understand which quantum structure provides a set of controls best suited to implementing the QFT.

Chapter 2

Theoretical Background

In this chapter, we introduce the mathematical tools that are used to model quantum computations. We introduce and define the properties of "qubits" and "qudits" and how they define the state of a quantum system. We then develop the quantum mechanical principles behind the evolution of a system over time, discussing how controls are used to construct Hamiltonian matrix operators that solve for the Schrödinger equation, thus building up the quantum Fourier transform (QFT).

2.1 Qubits and Qudits

A "qubit" is a portmanteau of "quantum" and "bit" and represents a state that can be expressed as a superposition of two basis states labeled 0 and 1 [12]. In Dirac notation, a qubit can be expressed as

$$|\psi_1\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}, \tag{2.1}$$

where α_0 and α_1 are complex numbers subject to the following normalization condition

$$|\alpha_0|^2 + |\alpha_1|^2 = 1. \tag{2.2}$$

This definition stands in stark contrast to that of a classical bit, whose value is restricted to either $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ or $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ states, with no possible superposition. For an example of a qubit in a physical system, one could imagine these two states corresponding to the wave functions of two energy levels for an electron in a trapped ion.

For a system of n qubits, each qubit can exist as a superposition of two states, thus allowing the entire system to exist as a superposition of 2^n possible states [12]. We can express the state of the system as such:

$$|\Psi\rangle = \sum_{0 \leq x < 2^n} \alpha_x |x\rangle, \tag{2.3}$$

subject to the normalization condition that the sum of the squared amplitudes must be equal to one

$$\sum_{0 \leq x < 2^n} |\alpha_x|^2 = 1. \tag{2.4}$$

The probability of finding the qubit system in a given state $|x\rangle$ is given by $|\alpha_x|^2$.

A quantum digit, or qudit, is a multidimensional generalization of a qubit whose state is a superposition of d_q basis states. It can be expressed as

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle + ... + \alpha_{d_q} |d_q - 1\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ ... \\ \alpha_{d_q - 1} \end{pmatrix}, \tag{2.5}$$

subject to the normalization condition

$$\sum_{i=0}^{d_q - 1} |\alpha_i|^2 = 1. \tag{2.6}$$

There are clear benefits to the use of qudits over qubits, when possible. There are 2^n possible states for a system of n qubits, while a qudit system of n qudits of size d_q can generate d_q^n possible states. In this framework, qubits are a special case of qudits where $d_q = 2$. The physical basis for a qudit system could stem from a particle or field with several possible modes. Examples of possible qudit systems include the ground states of Rubidium [7] or the energy states of superconductors [15].

2.2 Defining Unitary Operators

In the matrix form of quantum mechanics, a state vector $|\Psi\rangle$ composed of qubits or qudits can be manipulated by a series of unitary operators to perform a calculation. The situation is no different when $|\Psi\rangle$ represents a system of qubits or qudits. A quantum system that evolves over time can be described by the time-dependent Schrödinger equation

$$i\hbar \frac{\partial}{\partial t} |\Psi\rangle = \mathcal{H} |\Psi\rangle, \tag{2.7}$$

where $\mathcal{H}$ is the Hamiltonian operator, which acts as a matrix on our quantum system and is used to characterize the energy levels of the system.

The time-dependent Schrödinger equation is a separable differential equation that can be easily solved for the case of a time-independent Hamiltonian. When one does so and solves for the final state, one gets:

$$|\Psi(t)\rangle = e^{-i\mathcal{H}t/\hbar} |\Psi(0)\rangle. \tag{2.8}$$

The exponential of the time-independent Hamiltonian in equation 2.8 is itself a matrix operator, one that we define as the time-evolution unitary operator $U(t)$:

$$U(t) = e^{-i\mathcal{H}t/\hbar}. \tag{2.9}$$

We can now solve for the Schrödinger equation using matrix form:

$$|\Psi(t)\rangle = U(t)\,|\Psi(0)\rangle\,. \tag{2.10}$$

If the quantum system evolves with a time-dependent Hamiltonian, the picture is not quite as clear. Nevertheless, we can approximate the evolution of a time-varying Hamiltonian as a product of the exponential of several time-independent Hamiltonians, $\mathcal{H}_j$, each operating over a small time step $\Delta t = T/N_t$:

$$U(T) = \prod_{j=1}^{N_t} U_j = U_{N_t} U_{N_{t-1}} \dots U_1, \tag{2.11}$$

where we define

$$U_j = e^{-i\mathcal{H}_j \Delta t/\hbar}. \tag{2.12}$$

The product $\prod_j U_j$ of the unitary transformations yields a final evolved unitary $U(T)$. This unitary represents the total operation upon the system over time T. U is a unitary operator because it satisfies the following condition:

$$UU^\dagger = U^\dagger U = 1. \tag{2.13}$$

It is important to note that as a result of this property, the application of unitary operators to a state is reversible. If properly constructed, the evolution of unitary transformations can yield an operation corresponding to useful algorithms, such as the discrete Fourier transform.

2.3 Hamiltonians

In general, the Hamiltonian operator probes the total energy in a system. We can better understand the operation of the Hamiltonian by considering an example, such as a spin-1/2 system in a magnetic field. Spin is a property of fermions that can be in a combination of a "spin up" and "spin down" state, thus representing a qubit. A magnetic field $\vec{B}$ will induce a torque on the spin that will align it parallel to the magnetic field, thus one can quantify the energy caused by this torque using the Hamiltonian operator [9]:

$$\mathcal{H} = -\gamma \frac{\hbar}{2} \begin{bmatrix} B_z & B_x - iB_y \\ B_x + iB_y & -B_z \end{bmatrix}, \tag{2.14}$$

where γ is the gyromagnetic ratio and the subscripts x, y, z denote the direction of the magnetic field. The directional dependence of the Hamiltonian is described by the Pauli spin matrices:

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{2.15}$$

These matrices compose the basic transformation operators on qubits. The energy in a quantum system, and thus the Hamiltonian operator, can be bifurcated into the energy inherent in a system (due to the nature and arrangement of the particles), and the energy levels that can be controlled as the system evolves. The first type of energy is described by the drift Hamiltonian while the second type is described by the control Hamiltonian.

2.3.1 Control Hamiltonian

In this research, we generalize the qubit control Hamiltonians introduced in Eq. (2.14) to the following form for a qudit of dimension d:

$$\mathcal{H}_j = \hbar \begin{bmatrix} b_0 & a_0 & 0 & \dots & 0 & 0 \\ a_0^* & b_1 & a_1 & \dots & 0 & 0 \\ 0 & a_1* & b_2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & b_{d-1} & a_{d-1} \\ 0 & 0 & 0 & \dots & a_{d-1}* & b_d \end{bmatrix}, \tag{2.16}$$

where b_i is a real number, a_i is a complex number and a_i^* is its complex conjugate. The structure of this operator assures that the control Hamiltonians will always be Hermitian, and thus yield real values when applied to a system, as expected. For our research, we assume that the elements within this matrix are independent of each other. We do not limit control amplitudes. The tridiagonal form of this Hamiltonian reflects the strong coupling to neighboring energy eigenstates seen in many systems, such as the spin system described in the prior section. Thus, the Hamiltonian represents an idealized model appropriate for certain systems, such as a Beryllium ion trapped in a harmonic potential [4].

2.3.2 Drift Hamiltonians

In addition to the control Hamiltonians, drift Hamiltonians appear in our modeled system as the underlying dynamics of the system that cannot be controlled. This includes the interactions between different qubits or qudits within the system. We assume that our system is subject to a constant drift Hamiltonian throughout its evolution so that it occurs in each timestep as a constant $\mathcal{H}_d$. Unlike the control Hamiltonians, the amplitude of the drift Hamiltonian is reflective of the intrinsic coupling strength between the elements of the system and cannot be controlled except indirectly through the length of time over

which the Hamiltonian is applied to the system. For this reason, one can measure the length of time over which a qubit or qudit system evolves by using a scaled time in units of the drift Hamiltonian's coupling strength.

2.4 Constructing Hamiltonians

For systems comprised of several qudits, a series of outer products is needed to generate the time-independent Hamiltonian for a given time step. Specifically, we make use of the Kronecker product of the Hamiltonians corresponding to each individual qudit. For example, consider a system of two qudits, where the state of the system is given by the Kronecker product

$$|\Psi_1\rangle \otimes |\Psi_2\rangle \,. \tag{2.17}$$

The total Hamiltonian $\mathcal{H}$ that operates on the system is a Kronecker product of the two individual qudit Hamiltonians with the identity operator:

$$\mathcal{H}_1 \otimes I_2 + I_1 \otimes \mathcal{H}_2. \tag{2.18}$$

The operation of the Kronecker product is as follows. Suppose A and B are both square matrices of dimension n by n. The matrix representation of the Kronecker product is of size n^2 by n^2 [16]:

$$A \otimes B = \begin{bmatrix} A_{11}B & A_{12}B & \dots & A_{1n}B \\ A_{21}B & A_{22}B & \dots & A_{2n}B \\ \vdots & \vdots & \vdots & \vdots \\ A_{n1}B & A_{n2}B & \dots & A_{nn}B \end{bmatrix}, \tag{2.19}$$

where the elements $A_{ij}B$ each denote a submatrix of dimension n by n that is proportional to B by a proportionality constant A_{ij}. For example, if A and B are 2 by 2 matrices, then their tensor product would be expressed as:

$$A \otimes B = \begin{bmatrix} a_{11}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} & a_{12}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \\ a_{21}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} & a_{22}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{bmatrix}. \tag{2.20}$$

2.5 Qudit-Qudit Interactions

In order to understand the type of qudit-qudit interactions we consider here, it is informative to first consider the two-qubit Ising model as it is applied to qubit systems. The Ising model considers the state of a spin-1/2 magnetic dipole moment in one direction, which we will call the z-direction, allowing for either an up state or a down state. This

can occur in the hyperfine coupling of nuclear and electron spins in an atom. We will call interactions between the dipole moments of each spin an Ising interaction. For a system of two qubits, each qubit representing the dipole moment of a spin in the Ising model, we can call the interaction between their magnetic moments an Ising interaction. The interaction can be modeled as the tensor product of two Pauli-Z rotations σ_z, weighted by the interaction strength J:

$$\mathcal{H}_{Ising} = -J\sigma_{z1} \otimes \sigma_{z2} = -J \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.21}$$

It is possible to generate the Ising interaction for qudit systems as well. To do so, we turn to the cross-Kerr interaction, which can be generated experimentally by one electromagnetic field changing the index of refraction experienced by another electromagnetic field. Importantly, the cross-Kerr interaction has been theoretically demonstrated for qudit systems [21]. The cross-Kerr interaction operator for two photon modes, each described by a qubit, can be described as the the following matrix product:

$$J_K \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = J_K \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.22}$$

where J_K is the cross-Kerr interaction strength. This operator can be re-expressed as a combination of Identity operators and Pauli-Z rotations:

$$J_K \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = J_K \left[\frac{1}{4} I \otimes I - \frac{1}{4}\sigma_z \otimes I - \frac{1}{4} I \otimes \sigma_z + \frac{1}{4}\sigma_z \otimes \sigma_z \right]. \tag{2.23}$$

When represented within the overall Hamiltonian of the system, the first three terms in Eq. (2.23) do not correspond to interactions between the qubits, but rather rotations undergone by each individual qubit. Thus, for the 2 qubit case, when $J_K = 4J$ the interaction strength for the cross-Kerr interaction is given by $J\sigma_z \otimes \sigma_z$ and is thus comparable to the Ising interaction.

For generalized systems of dimension d, the cross-Kerr interaction is generalized to a tensor product of diagonal operators weighted by the cross-Kerr interaction strength J'_K:

$$J'_K \begin{bmatrix} 0 & & & 0 \\ & 1 & & \\ & & \ddots & \\ 0 & & & d-1 \end{bmatrix} \otimes \begin{bmatrix} 0 & & & 0 \\ & 1 & & \\ & & \ddots & \\ 0 & & & d-1 \end{bmatrix} = J'_K \begin{bmatrix} 0 & & 0 \\ & \ddots & \\ 0 & & (d-1)^2 \end{bmatrix}. \tag{2.24}$$

If we consider, for argument's sake, the case where each qudit represents an atom whose electrons allow for an atomic spin $j = (d-1)/2$, then we can re-express our generalized cross-Kerr interaction as

$$J'_K \left(jI + \begin{bmatrix} -j & & \\ & \ddots & \\ & & j \end{bmatrix} \right) \otimes \left(jI + \begin{bmatrix} -j & & \\ & \ddots & \\ & & j \end{bmatrix} \right) \tag{2.25}$$

$$= J'_K \left[j^2 (I \otimes I) + j(I \otimes S_z) + j(S_z \otimes I) + J_z \otimes S_z \right], \tag{2.26}$$

where S_z is a generalized spin matrix for the z direction. Once again, the terms with an identity matrix do not correspond to interactions between the qudits and are thus ignored when considering qudit-qudit interactions. The last term, $S_z \otimes S_z$, is a multidimensional generalization of the term $J\sigma_z \otimes \sigma_z$ seen in the 2 qubit Ising interaction seen in Eq. (2.21). Thus, the cross-Kerr interactions described here are directly analogous to the Ising interactions between several qubits. With our numerical approach, we will use the cross-Kerr interactions to compare the efficacy of qudit systems to qubit systems in generating the QFT.

2.6 The Discrete Fourier Transform

The discrete Fourier transform (DFT) maps the values of a discrete set of N points from a "time" domain to a "frequency" domain. It is a discrete version of the Fourier transform, which, for example, is expressed in matrix form as follows for a system with 4 inputs:

$$DFT_4 = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix}, \tag{2.27}$$

where $\omega = \exp(\frac{2\pi i}{N})$. The QFT is the DFT applied to a set of amplitudes representing quantum states, given by an initial state $|\Psi(0)\rangle$. Because it has the same form as the DFT, the QFT is a unitary matrix operator that itself can be constructed by a set of unitary operations acting on a number of interacting qudits. In the next chapter, we will consider numerical search methods to generate the unitary operations necessary to build up the QFT.

Chapter 3

GRAPE Implementation

In this chapter, we discuss the GRAPE algorithm introduced in chapter 1 and examine key components of its implementation, including areas where we offered improvements to existing implementations.

3.1 The Optimal Control Problem

The goal of our quantum control is to find a set of controls $c_{j,k}$ that produce a total unitary operator, U_{QFT}, that approximates our goal unitary, the QFT. Optimal control accomplishes this task with approximation, replacing the continuous evolution of a quantum system over time with a set of time-independent Hamiltonian operators affecting the system over discretized time intervals. A physical example of such an example would be a system of trapped ions subject to a pulsed laser. Here, the control elements $c_{j,k}$ would represent the phase and amplitude of the laser. Within this approximation regime, each time-independent Hamiltonian $\mathcal{H}_j$ is an evolution of the system's wave function over a short timestep Δt:

$$\mathcal{H}_j = \mathcal{H}_d + \sum c_{j,k}\mathcal{H}^{(k)}, \tag{3.1}$$

where $c_{j,k}$ are the controls, $\mathcal{H}_d$ is the drift Hamiltonian and $\mathcal{H}^{(k)}$ is the Hamiltonian corresponding to each control. The exponential of $\mathcal{H}_j$ generates the timestep unitary operator U_j that operates upon the wave function:

$$U_j = e^{-i\mathcal{H}_j \Delta t/\hbar}. \tag{3.2}$$

The final unitary of the system is given by the product of every timestep unitary operator:

$$U_{QFT} = U(T) = \prod_{j=1}^{N_t} U_j, \tag{3.3}$$

where N_t is the total number of timesteps. With the help of an objective function, we can iteratively improve U_{QFT} until it approximates the QFT. Usually, we hope to find a set of controls that is time optimal, meaning that the evolution occurs in the least amount of time. Reducing computation time is always a desirable goal, and even more so in the case of quantum systems where shorter calculations help to maintain the coherence of the system.

Time optimization can be accomplished through a combination of two means. First, we can reduce the number of timesteps over which we evolve our system. By reducing the number of timesteps, however, we reduce the total number of Hamiltonian operators we define, thus reducing the number of controls that can be utilized to solve the problem. Second, we can reduce the amount of time, Δt, allotted to each timestep. Doing so, however, comes at the cost of reducing the amount by which each time-independent Hamiltonian can alter the system [14]. Using gradient search, we hope to numerically test qudit systems to empirically determine the parameters that allow for a time-optimal set of controls.

With gradient search, we also face the practical problem of how to search in the most computationally efficient way. There are two areas in the algorithm where a large number of computationally intensive operations occur. The first area is matrix exponentiation. Every timestep unitary is calculated by taking the exponential of a Hamiltonian, which is known to be a notoriously difficult operation [11]. The most direct method to compute the exponential of a square matrix is with a power series expansion of the form

$$e^A = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} \ldots, \tag{3.4}$$

where A is a square matrix. This method requires an intensive number of matrix multiplications and suffers from a relative lack of accuracy. Several alternative approximation schemes with better computational performance exist, and we outline our method of matrix exponentiation in following sections.

Another challenging computational operation is the derivative of matrix exponential functions. It is possible to find an exact solution using eigenvalue decomposition (EVD), but this method requires the diagonalization of matrices with every calculation. We describe implementations using EVD as well as approximation schemes that may offer computational speedups.

3.2 Algorithm Overview

GRAPE is an acronym for GRadient Ascent Pulse Engineering and is an instance of gradient search techniques. For a given set of controls, a series of unitary operators are defined that when multiplied together yield the evolved unitary U that approximates a goal unitary U_{ideal}. In our case, we want U_{QFT} to come as close as possible to the unitary matrix we hope to emulate, the QFT. From the outset, we define the number of timesteps

and total time over which the system will be allowed to evolve. We then begin with a "guess" of the controls that will yield an evolved unitary U that overlaps well with the target unitary. For our purposes, the initial guess need only be a randomly generated set of complex numbers. The overlap between U and the desired operator U_{ideal} is measured by the objective function:

$$\Phi = \frac{1}{d^2}|\operatorname{Tr}(U^\dagger U_{ideal})|^2, \tag{3.5}$$

where d is the dimension of the system and the objective Φ (which ranges from 0 to 1) approaches 1 as U becomes congruent with U_{ideal}. Thus, we say that our evolved unitary has a higher fidelity when the objective function is larger and there is greater overlap with the desired operator U_{ideal}. We describe the "error in objective" as the deviation of our objective from a value of 1. The partial derivatives of the objective function with respect to each control $c_{j,k}$ yield a gradient $\partial\Phi/\partial c_{j,k}$. The gradient can be used to update the controls, yielding a set of controls that comes closer to maximizing the objective function (see Fig. 3.1). Plotted on the left of Fig. 3.1 is the amplitude of a single control, u_j, over the timesteps from 1 to M. After the gradient of the objective function is calculated, the control is adjusted at every timestep to a new value (shown by the red arrows). In section 3.5, we discuss methods to determine how far our update should proceed along the gradient. At the end of the update, every control has a new amplitude at each timestep (as seen on the right), yielding an evolved unitary matrix that overlaps more closely with the QFT. These new controls will be used in the next iteration to find another local gradient that further improves the set of controls. This process, where we find the local gradient in order to improve the controls, can be done iteratively until U_{QFT} approaches the QFT with desired precision. If the quantum system is poorly configured to generate the QFT or there are an insufficient number of controls, however, U_{QFT} will cease to improve beyond a certain fidelity.

3.3 Matrix Exponentiation

Because unitary operators, and thus Hamiltonian exponentials, correspond to each discrete time step evolving the system, efficiently implementing matrix exponentiation within the GRAPE algorithm is of great importance. In order to approximate the matrix exponential to a desired precision, we use a "scaling and squaring" method to achieve third-order approximations [18, 13].

We start by using the results of the Baker-Campbell-Hausdorff (BCH) formula, which states that

$$\exp(X+Y) = \exp(X)\exp(Y)\exp\left(-\frac{1}{2}[X,Y]\right)\exp\left(\frac{1}{12}([Y,[Y,X]]+[X[X,Y]])\right)... \tag{3.6}$$

Figure 3.1 is redacted (copyrighted material removed)

Figure 3.1: A representation of the control update scheme used in GRAPE for a single control (Reproduced from [11]). The gradient of the objective updates every control to a new set of values over time that yield an evolved unitary closer to the ideal unitary.

Thus, if we define two matrices composed of our Hamiltonian $\mathcal{H}$ to be composed of the diagonal and off-diagonal components of $\mathcal{H}$, then

$$\mathcal{H} = \mathcal{H}_{\text{off-diag}} + \mathcal{H}_{\text{diag}} \tag{3.7}$$

and

$$\exp\left(-i\mathcal{H}\Delta t\right) = \exp\left(-i(\mathcal{H}_{\text{off-diag}} + \mathcal{H}_{\text{diag}})\Delta t\right) \tag{3.8}$$

$$\approx \exp\left(-i\mathcal{H}_{\text{off-diag}}\Delta t\right)\exp\left(-i\mathcal{H}_{\text{diag}}\Delta t\right) + \mathcal{O}(\Delta t^2), \tag{3.9}$$

where Δt is the amount of time allotted to each timestep Hamiltonian. The diagonal Hamiltonian can be considered a sum of matrices each containing an element h_i along the diagonal of the original Hamiltonian. Thus, the combined diagonal Hamiltonian appears as:

$$\mathcal{H}_{\text{diag}} = \begin{bmatrix} h_0 & 0 & 0 & \dots & 0 \\ 0 & h_1 & 0 & \dots & 0 \\ 0 & 0 & h_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & h_{d-1} \end{bmatrix}. \tag{3.10}$$

The off-diagonal Hamiltonian can also be considered a sum of matrices, each containing a pair of off-diagonal elements $h_{j,k}$ and $h^*_{j,k}$ that are complex conjugates to each other:

$$\mathcal{H}_{\text{off-diag}} = \sum_{j=1}^{d}\sum_{k>j}^{d} \mathcal{H}_{\text{off-diag},j,k}, \tag{3.11}$$

$$\mathcal{H}_{\text{off-diag},j,k} = \begin{bmatrix} 0 & \dots & 0 & \dots & 0 \\ \vdots & 0 & h_{j,k} & \dots & 0 \\ 0 & h_{j,k}^* & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}, \tag{3.12}$$

where d is the dimension of the matrix. This definition of $\mathcal{H}_{\text{off-diag}}$ and $\mathcal{H}_{\text{diag}}$ thus ensures that every element in the combined Hamiltonian is represented. We can find the exponential of our diagonal component by taking the exponential of each element in the diagonal matrix $\mathcal{H}_{\text{diag}}$:

$$\exp\left(-i\mathcal{H}_{\text{diag}}\Delta t\right) = \begin{bmatrix} e^{-ih_0\Delta t} & 0 & 0 & \dots & 0 \\ 0 & e^{-ih_1\Delta t} & 0 & \dots & 0 \\ 0 & 0 & e^{-ih_2\Delta t} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & e^{-ih_{d-1}\Delta t} \end{bmatrix}. \tag{3.13}$$

The exponential of the matrices $\mathcal{H}_{\text{off-diag},j,k}$ can be found by finding the power series expansion of the matrix exponential. The power series expansion can be regrouped to express the matrix exponential of $\mathcal{H}_{\text{off-diag},j,k}$ as a combination of sine and cosine terms, in a process that is analogous to the derivation of Euler's Formula:

$$\exp\left(-i\mathcal{H}_{\text{off-diag},j,k}\Delta t\right) = \begin{bmatrix} 1 & 0 & \dots & \dots & 0 & 0 \\ 0 & \ddots & & & & 0 \\ \vdots & & \cos\left(|h_{j,k}\Delta t|\right) & \frac{-ih_{j,k}}{|h_{j,k}|}\sin\left(|h_{j,k}\Delta t|\right) & & \vdots \\ \vdots & & \frac{-ih_{j,k}^*}{|h_{j,k}|}\sin\left(|h_{j,k}\Delta t|\right) & \cos\left(|h_{j,k}\Delta t|\right) & & \vdots \\ 0 & & & & \ddots & 0 \\ 0 & 0 & \dots & \dots & 0 & 1 \end{bmatrix}, \tag{3.14}$$

With the BCH formula, we can now approximate the matrix exponential of $\mathcal{H}$ to the second-order as the product of the exponentials we calculated:

$$\exp\left(-i\mathcal{H}\Delta t\right) \approx \left(\prod_{j=1}^{\frac{1}{2}(d-1)(d)} \exp\left(-i\mathcal{H}_{\text{off-diag},j,k}\Delta t\right)\right) \exp\left(-i\mathcal{H}_{\text{diag}}\Delta t\right) + \Theta(\Delta t^2). \tag{3.15}$$

With symmetrization of the exponential product, this result can be further approximated

to the third-order:

$$\exp\left(-i\mathcal{H}\Delta t\right) \approx \left(\prod_{j=1}^{\frac{1}{2}(d-1)(d)} \exp\left[-i\mathcal{H}_{\text{off-diag},j,k}(\Delta t/2)\right]\right) \exp\left(-i\mathcal{H}_{\text{diag}}\Delta t\right)\ldots \left(\prod_{j=1}^{\frac{1}{2}(d-1)(d)} \exp\left[-i\mathcal{H}_{\text{off-diag},j,k}(\Delta t/2)\right]\right). \quad (3.16)$$

Note that the error in this method is subject to the size of the timestep Δt. If we wish to improve the accuracy of our results, we can introduce a small dt by using the identity

$$\left[e^{-i\mathcal{H}dt}\right]^N = e^{-i\mathcal{H}\Delta t}, \quad (3.17)$$

where $\Delta t = Ndt$. With $\log(N)$ squaring operations on $e^{-i\mathcal{H}dt}$, the original exponential $e^{-i\mathcal{H}\Delta t}$ can be recovered. This method, known as "scaling and squaring," allows us to approximate our matrix exponential to arbitrary accuracy [13]. We can approximate the exponential to an accuracy of $1/N^2$ through the use of $2M+1+\log(N)$ matrix multiplications, where M is the number of off-diagonal terms in the Hamiltonian. This is comparable to eigenvalue decomposition methods to calculate the matrix exponential, which have a cost of $O(d^3)$ where d is the dimension of the matrix [17]. For dense matrices, eigenvalue decomposition has a similar efficiency as scaling and squaring. For large dimensional matrices, however, our tridiagonal Hamiltonians are sparse matrices where matrix multiplication speedups can be found. Thus, scaling and squaring methods for large Hamiltonians are expected to require fewer multiplications than the eigendecomposition method.

3.4 Gradient Calculation

The gradient of the objective function is calculated by:

$$\frac{\partial\Phi}{\partial c_{k,j}} = \frac{1}{d^2}\left[\text{Tr}\left(U_{ideal}^{\dagger}\frac{\partial U}{\partial c_{k,j}}\right)\text{Tr}\left(U_{ideal}^{\dagger}U\right)^{*} + \text{Tr}\left(U_{ideal}^{\dagger}U\right)\text{Tr}\left(U_{ideal}^{\dagger}\frac{\partial U}{\partial c_{k,j}}\right)^{*}\right], \quad (3.18)$$

where the length of the resulting gradient vector is the total number or controls. Here, U is our approximation U_{QFT} and U_{ideal} is the QFT. For a system with tridiagonal qudit Hamiltonians, qudit dimension q_d, a total number of qudits N_q, and a total number of timesteps N_t, the control count is $N_tN_q(3q_d-2)$. Our controls are stored as real values, so we are interested in finding the real component of the gradient, which is a sum over all controls of:

$$\frac{\partial\Phi}{\partial c_{k,j}} = \frac{2}{d^2}\,\text{Re}\left[\text{Tr}\left(U_{ideal}^{\dagger}\frac{\partial U}{\partial c_{k,j}}\right)\text{Tr}(U_{ideal}^{\dagger}U)^{*}\right]. \quad (3.19)$$

In order to find the derivative of the evolved unitary U with respect to a control $c_{k,j}$, one only needs to consider the derivative of the unitary at time step j:

$$\frac{\partial U}{\partial c_{k,j}} = U_{N-1}U_{N-2}...U_{j+1}\frac{\partial U_j}{\partial c_{k,j}}U_{j-1}U_{j-2}...U_0 = \left(\prod_{n=N-1}^{j+1} U_n\right)\frac{\partial U_j}{\partial c_{k,j}}\left(\prod_{n=j-1}^{0} U_n\right), \quad (3.20)$$

where N is the total number of time steps and j is the time step belonging to the control $c_{k,j}$. By ordering the product of unitaries backwards, we maintain the proper time-ordering of the evolution.

Note that by finding the derivatives in this manner, we limit the total number of matrix multiplications and exponentiations that must be taken. When we take the derivatives of the unitary with respect to the controls in the first timestep, we require the final product of the unitaries, U_{N-1}, as well as the partial derivative of the local timestep unitary. After this first step, however, we only need to backwards-evolve our final unitary by one timestep to find the unitary products needed to take the derivative with respect to the subsequent timestep. This is an advantage of the GRAPE search strategy for finding evolved unitaries, and by structuring our algorithm in this manner our we see a run-time scaling of $O(N)$ with respect to the number of timesteps in the problem.

Recall that the unitary operator for each timestep is given by the exponential of the time step Hamiltonian. We can take the partial derivative of both sides with respect to the controls:

$$\frac{\partial U_j}{\partial c_{k,j}} = \frac{\partial}{\partial c_{k,j}}\exp\left(-i\Delta t\mathcal{H}_0 - i\Delta t\sum_{k'} c_{k',j}\mathcal{H}^{(k')}\right). \quad (3.21)$$

In order to solve for the right-hand side of Eq. (3.21), we can make use of either exact or approximate methods. We will first show how this expression can be re-expressed as an integral of noncommuting operators, then how it can be numerically evaluated in three different ways.

Because the derivative of the matrix exponential in Eq. (3.21) is used to update the set of controls for the next iteration, finding accurate values of the exponential derivatives is crucial in constructing a stable version of the GRAPE algorithm.

3.4.1 Preparing the Derivative Function

Let us begin by re-organizing the argument of the exponential in Eq. (3.21):

$$\frac{\partial U_j}{\partial c_{k,j}} = \frac{\partial}{\partial \delta c_{k,j}}\exp\left[-i\Delta t\left(\mathcal{H}_0 + \sum_{k'} c_{k',j}\mathcal{H}^{(k')} + \delta c_{k,j}\mathcal{H}^{(k)}\right)\right], \quad (3.22)$$

where $c_{k',j}$ is held constant (k' indexes our fixed controls) and $\delta c_{k,j}$ is our variable control. With this change of variables, we evaluate our derivative at zero:

Let $x = \delta c_{k,j}, A = -i\Delta t \left(\mathcal{H}_0 + \sum_{k'} c_{k',j}\mathcal{H}^{(k')}\right) = -i\Delta t\mathcal{H}_j$, and $B = -i\Delta t\mathcal{H}^{(k)}$:

$$\left.\frac{\partial U_j}{\partial c_{k,j}}\right|_{x=0} = \frac{\partial}{\partial x}\exp\left(A + xB\right). \tag{3.23}$$

Next, we work to rewrite Eq. (3.23) as an integral as follows. We first rewrite our expression as a limit of scaled and squared exponentials:

$$= \lim_{n\to\infty}\frac{\partial}{\partial x}\exp\left(A/n + xB/n\right)^n = \lim_{n\to\infty}\frac{\partial}{\partial x}\left[e^{(A/n)}e^{(xB/n)}\right]^n. \tag{3.24}$$

With manipulation, we can re-express this as an integral:

$$= \lim_{n\to\infty}\sum_{j=0}^{n-1}\left[e^{(A/n)}e^{(xB/n)}\right]^j e^{(A/n)}e^{(xB/n)}\frac{B}{n}\left[e^{(A/n)}e^{(xB/n)}\right]^{n-j-1}. \tag{3.25}$$

Evaluate at $x = 0$ and let $k = j + 1$:

$$= \lim_{n\to\infty}\frac{1}{n}\sum_{k=1}^{n} e^{(kA/n)}Be^{(1-k/n)A}. \tag{3.26}$$

Let $s = k/n$ and $ds = 1/n$:

$$\left.\frac{\partial U_j}{\partial c_{k,j}}\right|_{x=0} = \int_0^1 e^{sA}Be^{(1-s)A}ds = -ie^{-i\Delta t\mathcal{H}_j}\int_0^{\Delta t} e^{-i\tau\mathcal{H}_j}\mathcal{H}^{(k)}e^{i\tau\mathcal{H}_j}d\tau. \tag{3.27}$$

The right side of our expression is found when one plugs back in for A and B and lets $\tau = s\Delta t$. $\mathcal{H}^{(k)}$ is the control Hamiltonian. With our expression in Eq. (3.27), we now have a form that can be numerically solved by several means.

3.4.2 Eigenvalue Decomposition (EVD)

We begin by writing Eq. (3.27) in the form

$$\left.\frac{\partial U_j}{\partial c_{k,j}}\right|_{x=0} = -i\int_0^{\Delta t} e^{-i\tau\mathcal{H}_j}\mathcal{H}^{(k)}e^{i\tau\mathcal{H}_j}e^{-i\Delta t\mathcal{H}_j}d\tau. \tag{3.28}$$

By making use of the spectral theorem, we re-express the partial derivatives as

$$\left.\frac{\partial U_j}{\partial c_{k,j}}\right|_{x=0} = \sum_{m,n}\mathcal{D}^{(k)}_{m_j,n_j}\left|m_j\right\rangle\left\langle n_j\right|, \tag{3.29}$$

where we use the eigenvalues λ_{m_j} and λ_{n_j} corresponding to the eigenvectors $|m_j\rangle$ and $|n_j\rangle$ of the timestep Hamiltonian $\mathcal{H}_j$. We now re-express Eq. (3.28) as

$$\mathcal{D}^{(k)}_{m_j,n_j} = -i \int_0^{\Delta t} d\tau \, \langle m_j | \, e^{-i\tau\mathcal{H}_j} \mathcal{H}^{(k)} e^{i\tau\mathcal{H}_j} e^{-i\Delta t\mathcal{H}_j} \, | n_j \rangle \, . \tag{3.30}$$

In diagonalized form, the exponential functions evaluate at the eigenvalues λ_{m_j} and λ_{n_j} corresponding to the eigenvectors:

$$\mathcal{D}^{(k)}_{m_j,n_j} = -i \, \langle m_j | \, \mathcal{H}^{(k)} \, | n_j \rangle \int_0^{\Delta t} d\tau e^{-i\tau\lambda_{m_j}} e^{i\tau\lambda_{n_j}} e^{-i\Delta t\lambda_{n_j}} \, . \tag{3.31}$$

The evaluation of the integral depends upon whether m_j is equal to n_j for a given calculation. If $m_j = n_j$,

$$\mathcal{D}^{(k)}_{m_j,n_j} = -i\Delta t \, \langle m_j | \, \mathcal{H}^{(k)} \, | m_j \rangle \, e^{-i\Delta t\lambda_{m_j}} \, . \tag{3.32}$$

Otherwise, for $m_j \neq n_j$,

$$\mathcal{D}^{(k)}_{m_j,n_j} = \frac{\langle m_j | \, \mathcal{H}^{(k)} \, | n_j \rangle}{\lambda_{m_j} - \lambda_{n_j}} \left(e^{-i\Delta t\lambda_{m_j}} - e^{-i\Delta t\lambda_{n_j}} \right) . \tag{3.33}$$

3.4.3 Commutator Approximation

Refer back to Eq. (3.27):

$$\left. \frac{\partial U_j}{\partial c_{k,j}} \right|_{x=0} = \int_0^1 e^{sA} B e^{(1-s)A} ds = -ie^{-i\Delta t\mathcal{H}_j} \int_0^{\Delta t} e^{-i\tau\mathcal{H}_j} \mathcal{H}^{(k)} e^{i\tau\mathcal{H}_j} d\tau. \tag{3.34}$$

We use the Baker-Campbell-Hausdorff formula to express the integrand as an infinite series of commutators:

$$\left. \frac{\partial U_j}{\partial c_{k,j}} \right|_{\delta c_{k,j}=0} = -ie^{-i\Delta t\mathcal{H}_j} \int_0^{\Delta t} \mathcal{H}^{(k)} - i\tau[\mathcal{H}_j, \mathcal{H}^{(k)}] + \frac{1}{2!}(-i\tau)^2[\mathcal{H}_j, [\mathcal{H}_j, \mathcal{H}^{(k)}]] + \frac{1}{3!}(-i\tau)^3[\mathcal{H}_j, [\mathcal{H}_j, [\mathcal{H}_j, \mathcal{H}^{(k)}]]] + \ldots d\tau. \tag{3.35}$$

Taking the integral over τ is now a trivial process and we are left with

$$= -ie^{-i\Delta t\mathcal{H}_j} \left[\Delta t\mathcal{H}^{(k)} + (-i)\frac{\Delta t^2}{2!}[\mathcal{H}_j, \mathcal{H}^{(k)}] + (-i)^2\frac{\Delta t^3}{3!}[\mathcal{H}_j, [\mathcal{H}_j, \mathcal{H}^{(k)}]] + \ldots \right] . \tag{3.36}$$

We can now use only matrix multiplications to numerically solve for the derivative to whatever order of accuracy we would like.

3.4.4 Simpson's Rule Approximation

Once again, refer to Eq. (3.27):

$$\left.\frac{\partial U_j}{\partial c_{k,j}}\right|_{x=0} = \int_0^1 e^{sA} B e^{(1-s)A} ds = -ie^{-i\Delta t \mathcal{H}_j} \int_0^{\Delta t} e^{-i\tau \mathcal{H}_j} \mathcal{H}^{(k)} e^{i\tau \mathcal{H}_j} d\tau. \tag{3.37}$$

We are also free to solve our integral using definite integral approximation methods. We found that the most effective integral approximation method is the composite Simpson's rule, which defines quadratic functions over subintervals of the integrand that together yield a close approximation of the integrand function. Its general form is given by

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\right], \tag{3.38}$$

where n is the number of subintervals and $h = (b-a)/n$ is the length of each interval. Because the error of Simpson's rule is proportional to $(1/n)^4$, it should be possible to achieve arbitrary accuracy in our approximation with the use of enough terms.

3.4.5 Comparison of Gradient Calculation Methods

For our MATLAB implementation of GRAPE, we developed scripts that allow for solutions of the control derivatives using EVD, the commutator approximation, and Simpson's rule. In our C++ implementation, we relied upon our two numerical approximation methods. Using a one qudit system of varying dimension, we then tested each method in MATLAB in order to determine their efficacy in GRAPE and their relative run times. The results are listed in Fig. 3.2. When controlling for the relative accuracy of each method, we find that EVD offers consistently faster results within GRAPE than the commutator method. We hypothesize that this is because the commutator approximation method requires a large order approximation to yield numerical solutions similar to the precision EVD offers.

Also note that the Simpson's rule method is only compared for $d = 8$. In general, the Simpson's method was too inaccurate to yield numerical solutions with a high degree of accuracy. For small-scale problems, it is not computationally taxing to find solutions with a high degree of accuracy (an objective error $< 10^{-12}$) using EVD or the commutator method, thus we do not use Simpson's rule. Nevertheless, because it is increasingly difficult to create high-fidelity solutions with optimal control for large systems, for larger-scale problems the Simpson's rule method offers an intriguing approach to generating approximate results with a faster run time than other methods. For larger problems, the best solution may be to utilize Simpson's rule to generate a rough approximation, then transition to EVD or the commutator method to generate a more accurate result.

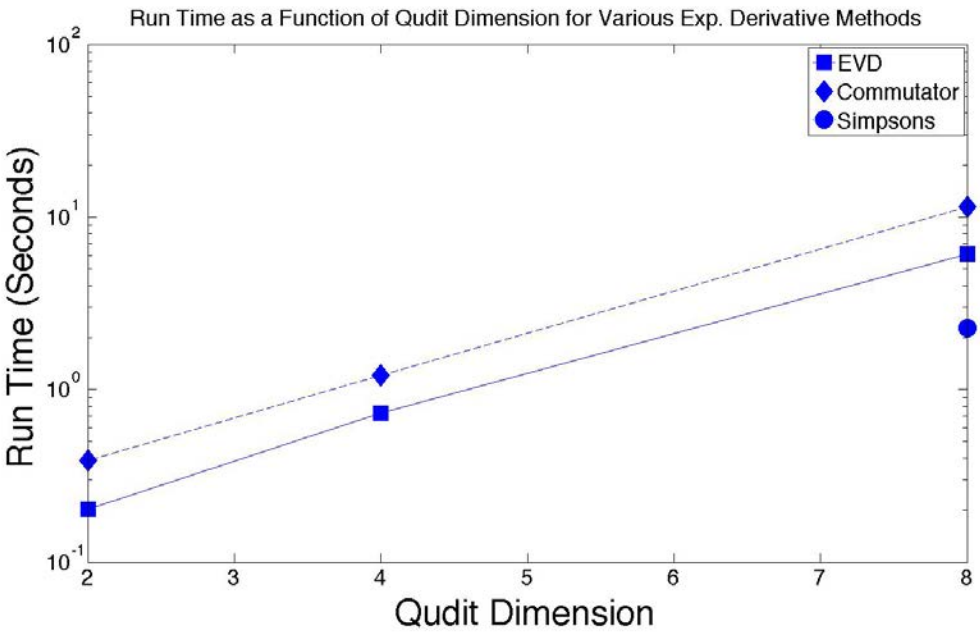


Figure 3.2: Run time as a function of qudit dimension for three different methods of finding the exponential derivative. For 2, 4, and 8 dimensional systems, the final objective function after 50 iterations of GRAPE using EVD was recorded. We then measured the run time required for implementations using the other two approximation methods to reach the same fidelity. Note that for the 2 and 4 dimension problems, the Simpson's rule method was unable to reach the level of fidelity achieved by EVD (objective error $< 10^{-12}$) over any run time.

3.5 Adaptive Control Update Step

After the gradient $\partial\Phi/\partial c_{j,k}$ of the objective function with respect to the controls is calculated, The controls are improved using the gradient before the next iteration begins:

$$(c_{j,k})_{new} = (c_{j,k})_{old} + \epsilon \frac{\partial \Phi}{\partial c_{k,j}}, \tag{3.39}$$

where ϵ is a small step size of our choosing. We have empirically found that the proper choice of step size can greatly influence the efficiency of the GRAPE algorithm. An update of the gradient that is too large may overshoot time optimal solutions, while an update that is too small will require an excessive number of search iterations. While standard implementations of GRAPE use a step size $\epsilon \ll 1$ for all iterations, we attempted to implement variations upon the step size that offered possible improvements upon the default GRAPE implementation.

3.5.1 Fixed Step Size

The basic implementation of GRAPE calls for a fixed step size for the gradient update every iteration. A choice of a small step size is recommended because the control "landscape" is not constant and the gradient can change greatly at different points. With the selection of a small step size, we make the assumption that small deviations from a given point in the control landscape will yield a similar gradient. Because the choice of a fixed step size must necessarily be conservative, the GRAPE algorithm proceeds along the control landscape more slowly than necessary given the other information at our disposal.

3.5.2 Empirical Cauchy Method Adaptive Step

Traditionally, the Cauchy method for gradient-descent searches calls for a minimization of the objective function in the direction of the gradient with each iteration [23]. For GRAPE searches involving large qudit dimensions or a large number of time steps, it is very costly to conduct a precise line search for the step size that will minimize the objective in the direction of the gradient. Instead of finding an exact solution to Cauchy's minimization conditions with each iteration, we implemented an empirical approximation of the Cauchy method.

For each iteration, we sample a spread of twenty different candidate step sizes, solving for the objective function that would result. We found that drawing candidates randomly from a uniform distribution $\pm.05$ around the previous step size was most effective in yielding improvements in the objective function. From these twenty different candidates, we select the step size most effective in improving the objective function and use that step size to modify our controls, with the spread of step sizes over the course of a search usually ranging between 0 and 1. While this method was clearly more effective than a fixed step size method at improving the objective function with each iteration (see Fig.

3.3), it can be ill-suited to some problems. Additionally, it suffers from the drawback of requiring computationally intensive samplings with each iteration. For an adaptive step that improves upon a fixed step size with less computational cost, we turn to the Barzilai and Borwein (BaB) method.

3.5.3 Barzilai and Borwein Adaptive Step

The Barzilai and Borwein (BaB) adaptive step method offers an improvement upon fixed step sizes by including information from the past iteration [5]. For each iteration, the step size is given by:

$$\epsilon = \left| \frac{\Delta \overrightarrow{g}^{\intercal} \cdot \Delta \overrightarrow{c}}{\Delta \overrightarrow{g}^{\intercal} \cdot \Delta \overrightarrow{g}} \right|, \tag{3.40}$$

where $\Delta \overrightarrow{c} = (c_{j,k})_{new} - (c_{j,k})_{old}$ is the change in the gradient vector from the prior iteration and $\Delta \overrightarrow{g} = \left(\frac{\partial \Phi}{\partial c_{k,j}}\right)_{new} - \left(\frac{\partial \Phi}{\partial c_{k,j}}\right)_{old}$ is the change in the control vector since the last iteration. Brazil and Borwein, as well as subsequent researchers [23], found that this method can offer an exponential improvement in efficiency over the Cauchy method despite being less computationally intensive. With the BaB method, the objective need only be calculated once per iteration. In our implementation of GRAPE, we found that the BaB method did indeed display a marked improvement over the Cauchy and fixed step methods for the single qudit case (see Fig. 3.3). As a result, all the results from our use of GRAPE make use of the BaB method implementation.

3.6 Implementing GRAPE in C++

The greatest potential for speedups from GRAPE lies in the parallelization of routines within the algorithm. Because the gradient search must compute the gradient of the control "landscape" with every iteration, there is an opportunity for parallel processing to simultaneously compute components of the gradient and reduce overall computational time. Recall that the gradient calculation requires the derivatives of the evolved unitary U with respect to the controls $c_{j,k}$. In order to find any given derivative, we only need to consider the derivative of the of the unitary at a time step j:

$$\frac{\partial U}{\partial c_{k,j}} = U_{N-1}U_{N-2}...U_{j+1}\frac{\partial U_j}{\partial c_{k,j}}U_{j-1}U_{j-2}...U_0 = \left(\prod_{n=N-1}^{j+1} U_n\right)\frac{\partial U_j}{\partial c_{k,j}}\left(\prod_{n=j-1}^{0} U_n\right) \tag{3.41}$$

By distributing the calculation of timestep unitary matrices to different processes, we can compute the derivative of the evolved unitary with respect to the controls in much less time.

Unlike MATLAB, C++ is compatible with MPI protocols that allow for parallel processing (See Appendix A), thus we implemented GRAPE in C++. Because C++ has a

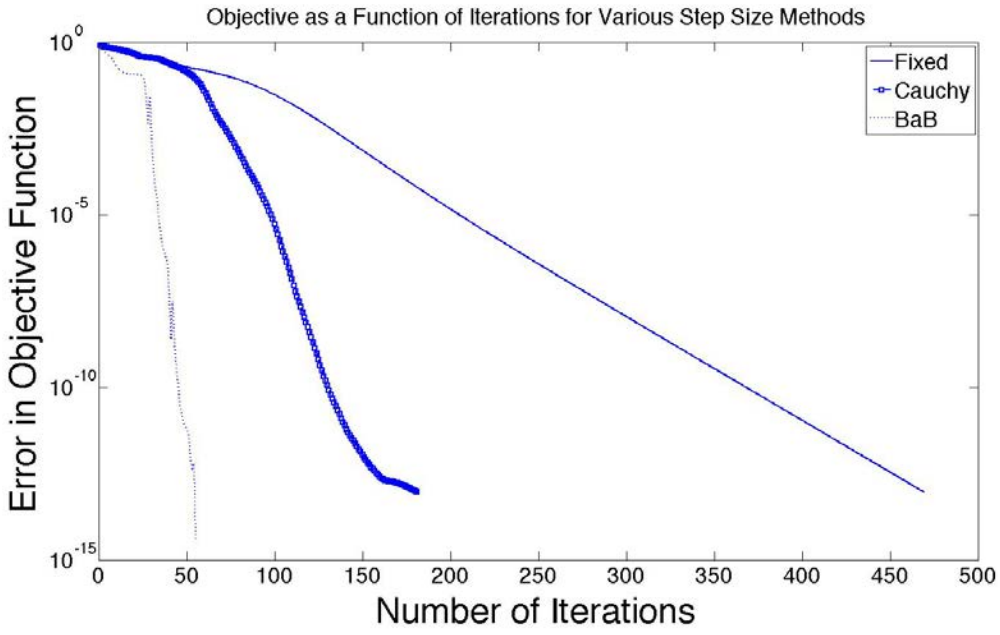


Figure 3.3: The objective as a function of number of iterations for various step size methods. This test was done on a 1 qudit system attempting the QFT with dimension $d = 4$ and $N_t = 5$ timesteps. Note that while the fixed and Cauchy methods yield relatively consistent improvements with each iteration, the BaB method is much faster, albeit inconsistent.

less extensive library of matrix operations than MATLAB, our GRAPE implementation in C++ lacks the EVD method of matrix derivatives and utilizes the scaling and squaring method for matrix exponentials. Otherwise, it offers an equivalent toolbox to our MATLAB implementation. In the future, we hope to use our C++ implementation on larger computing clusters where a large number of processes can be tasked with executing GRAPE.

3.7 Algorithm Behavior and Limitations

Because GRAPE is a numerical search strategy for the optimal control of qubit/qudit systems, one is never guaranteed that the solution found is truly time-optimal. We search for time optimal solutions by conducting searches with different numbers of timesteps N_t or timestep length Δt. While the objective can be minimized for some of these combinations to machine precision for a given system, there is always the possibility that one could further reduce the timesteps or timestep length and still find a solution by performing a given search for longer.

We can see a clear demonstration of this concept in Fig. 3.4. If we parameterize the system with a large number of timesteps, it is easy for the algorithm to find a solution to machine precision in a relatively small number of iterations. But as we constrain the problem by reducing the number of available timesteps with which to evolve the system, it takes a greater number of iterations to determine the solution. In this example, one may believe that the system cannot sufficiently perform the QFT with three timesteps after performing 100 iterations. With more iterations, however, it is clear that the system can yield a solution with as few as two timesteps. While testing with more iterations is practical for small systems, it is difficult to fully investigate large systems due to prohibitive classical computation time.

Fig. 3.5 illustrates the behavior of the gradient search algorithm as it improves the unitary with each iteration. Through 50 iterations, it appears that the system can be solved with an evolution of 5 timesteps, and that using 3 timesteps does not yield a solution with high fidelity to the QFT. Only with more iterations is it clear that we can achieve an implementation of the QFT with 3 timesteps. For larger systems, it can take orders of magnitude more iterations to determine whether a set of control parameters can yield a numerical solution of the QFT, and testing all possible combinations of timestep length and number of timesteps is prohibitively costly computationally. As a result, the time optimal controls found through GRAPE will almost always represent an overestimate of the time required to implement the QFT. This is especially true for large systems, where the run time for each iteration scales exponentially with the size of the system (See. Fig. 3.2).

There also exists the open question of the conditions under which GRAPE yields a local minimum for the objective function as opposed to a global minimum [14]. In other words, it is unclear whether gradient searches beginning from a different starting point

of control guesses will always converge upon a solution with an equal degree of overlap with the goal unitary. Because the topology of the control landscape is unclear, it can be beneficial to conduct a search with a variety of timestep counts, timestep lengths, and guesses for the initial controls rather than conducting a single search with a large number of iterations. Doing so provides more opportunities for the GRAPE algorithm to reach a global minimum. We employ this method in our following chapter, using a spread of initial control guesses, timestep counts, and timestep lengths to test implementations of the QFT in qubit and qudit systems.

The largest limitation for GRAPE is one shared by optimal control schemes in general: it is unlikely that quantum computers will exhibit ideal physical characteristics similar to those implied within GRAPE. Our implementation assumes that controls are sent to the system in a piecewise manner based upon the Heaviside unit step function, but in reality controls are likely to be fed in as a series of pulses, perhaps from lasers, that will little resemble a step function [14]. There will also inevitably be error in both the control pulses and state measurements that need to be accounted for. For our implementation of GRAPE, we also assume that our system is controllable by a tridiagonal Hamiltonian whose elements are all fully independent of each other and can exhibit unlimited amplitude; only the interaction Hamiltonians are constrained to a constant interaction strength. While this is inspired by the study of actual devices [4], in reality there are likely to be practical limits upon the amount of power that can be provided to alter the controls, as well as limitations to the number of controls that can be simultaneously modified. For this reason, we consider our simulations to provide a theoretical basis for understanding the optimal control of systems similar to the idealized one we used, and not an exact representation of any given physical system.

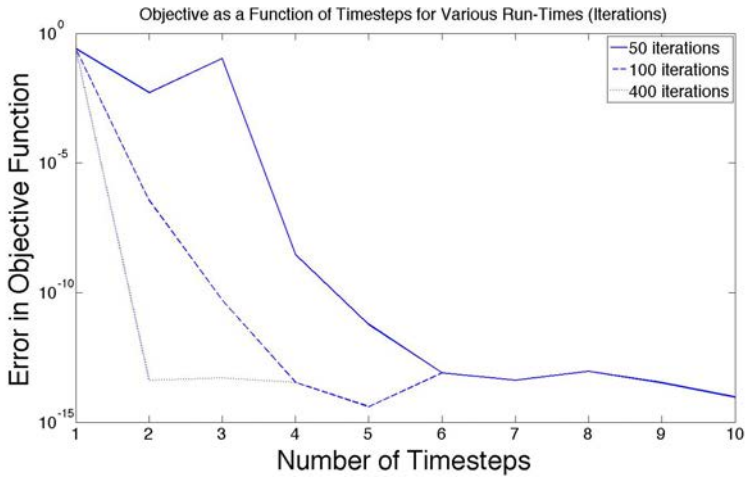


Figure 3.4: A plot of the final objective function as a function of number of timesteps for three different iteration run-times. Here, we use a single-qudit system of dimension $d = 4$ attempting the QFT. Run on a GRAPE implementation using EVD and BaB method gradient step size.

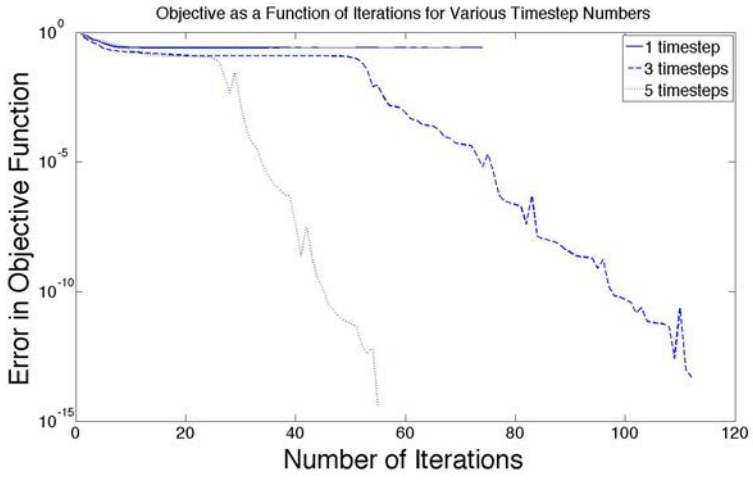


Figure 3.5: The objective as a function of Iterations for Various Timestep Numbers for a single qudit system with dimension $d = 4$. EVD was used to calculate matrix derivatives and we used the BaB method gradient step size. This plot illustrates the challenges of determining the time optimal configuration of time steps even for a small system: only with a sufficient number of iterations is it clear that a 4 dimensional qudit can generate a QFT with 3 timesteps.

Chapter 4

Numerical Simulations

Here, we describe numerical simulations run on GRAPE that help to answer two main questions:

1. How do optimal control algorithms of the QFT on a single-qudit system scale with increasing dimensional size?

2. Are qudit-based systems more efficient for certain applications than systems using qubits?

4.1 Testing QFT Structure

Recall that the QFT operator can be expressed as a square unitary matrix. For the case of a 4-dimensional system, the QFT has the following form:

$$QFT_4 = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix}, \tag{4.1}$$

where $\omega = \exp(\frac{2\pi i}{d})$ and $d = 4$ is the dimension of the operator. The QFT operator features d^2 non-zero elements. Nevertheless, because the matrix operator is a representation of a sum of d exponential functions with a phase rotation for each element of the state vector, it is clear that there is structure to the matrix and that the d^2 elements in the QFT are not truly independent of each other. Because of this structure, we suppose that it should be possible to find time optimal controls that require fewer than d^2 controls. We also wish to examine whether the number of timesteps needed to generate the QFT using optimal control scales with the number of elements in the QFT. In order to determine whether these hypotheses are true, we used GRAPE to test whether fewer controls were needed to implement the QFT than implied by the worst case scenario, in which more controls are required than there are elements in the QFT matrix.

4.1.1 Analytic Bound

In order to generate an upper analytic bound on the number of controls that should be required to generate the QFT, we assume that the QFT has no structure beyond its unitary nature. With this assumption, one finds that for a QFT transform of dimension d, there are d real values along the diagonal, $d(d-1)/2$ real elements along the off-diagonal, and $d(d-1)/2$ imaginary elements along the off-diagonal. We do not specify the global phase of our QFT, so subtracting one degree of freedom from our assessment this yields

$$d + d(d-1) - 1 = d^2 - 1 \tag{4.2}$$

possible degrees of freedom (in a mathematical sense). This is the worst case scenario where all values are independent of one another. Next, we compare this to the number of controls available to a single qudit system with the same dimension as our goal QFT. We counted the number of controls in each tridiagonal Hamiltonian timestep in order to determine the minimum number of timesteps needed to generate the worst-case scenario QFT. We assume that if each timestep Hamiltonian is given enough time to fully rotate a qudit state, than each control within the timestep Hamiltonian introduces a degree of freedom into the evolved unitary operator. Once again, we do not specify the global phase of our Hamiltonian. With these assumptions, each tridiagonal Hamiltonian has d diagonal values and $2(d-1)$ off-diagonal real and imaginary values for a total of

$$d + 2(d-1) - 1 = 3d - 3 \tag{4.3}$$

degrees of freedom. In our worst-case scenario of a unitary matrix with no structure, in order to implement the QFT operator the degrees of freedom of our evolved unitary must exceed the maximum possible degrees of freedom of the QFT. Recall that the final evolved unitary is the product of N_t timesteps:

$$U = e^{-i\mathcal{H}_{N_t}\Delta t} e^{-i\mathcal{H}_{N_t-1}\Delta t} \dots e^{-i\mathcal{H}_2\Delta t} e^{-i\mathcal{H}_1\Delta t}, \tag{4.4}$$

where each timestep Hamiltonian $\mathcal{H}_j$ has $3d-2$ degrees of freedom. For an evolution with N_t timesteps, this relation can be characterized as

$$N_t(3d-3) \geq d^2 - 1. \tag{4.5}$$

Solving for N_t we find a relation for the expected number of timesteps required:

$$N_t \geq \frac{d^2-1}{3d-3}, \tag{4.6}$$

rounding to the next largest integer value. In order to test whether the timestep scaling of the QFT for single qudit systems obeys the analytic bound we derived, we turn to GRAPE for numerical simulations.

4.1.2 Numerical Comparison

We developed a test using GRAPE to determine the number of timesteps required to numerically generate with high fidelity the QFT for a single qudit system of a given dimension.

For each qudit size between 2 and 16, we ran GRAPE for a variety of timestep numbers N_t, allowing each timestep Hamiltonian to fully rotate the system ($\Delta t = 1$). For selected qudit dimensions d_q, the resulting objective error for each simulation is shown in Fig. 4.1. If the resulting objective error is less than 10^{-5} (shown by the dotted line), then we classify the simulation as a successful generation of the QFT. As expected for lower qudit dimensions, fewer time steps are needed to implement the QFT compared to higher dimension systems, as predicted by our analytical bound. We observe that in these smaller systems, systems with a sufficient number of timesteps to implement the QFT are able to reach objective errors on the order of machine precision. Note that values $\leq 10^{-12}$ likely indicate issues of machine precision in the simulation, and the relative values below this error are not reflective of any structural differences between the systems.

For larger dimension systems, the picture is less clear. For the $d = 16$ case, there are a range of timesteps where it appears that the system can approximate a QFT to near the desired threshold. Because larger systems have more controls, it takes more iterations to explore the control "landscape" and find an optimal control result. In Fig. 4.2 we see evidence of this. When the largest systems were given 2.5 times more iterations, the required number of timesteps drops to around the analytical bound.

Cases in Fig. 4.2 where the number of timesteps exceeds our theoretical bound suggest that the number of controls do not directly correspond to the number of degrees of freedom in the problem, as we assumed. Because the evolved unitary matrix is the result of several unitary matrix multiplications, there is no one-to-one relationship between the specified controls and the resulting elements of the QFT approximation. Overall, however, there is a surprisingly close relationship between our empirical results for the required number of timesteps and the timestep number predicted by our degrees of freedom analysis (See Fig. 4.2). For every numerical result beyond $d = 2$, our simulations represent an upper bound on the number of timesteps necessary to replicate the QFT - it is possible that with a longer search and more iterations we could find more numerical solutions to the QFT that require less timesteps and beat our bound. The close adherence to our analytical curve suggests that for single-qudit systems, the number of timesteps required to generate the QFT has a $\sim d/3$ relationship to the qudit dimension, as Eq. (4.6) predicts.

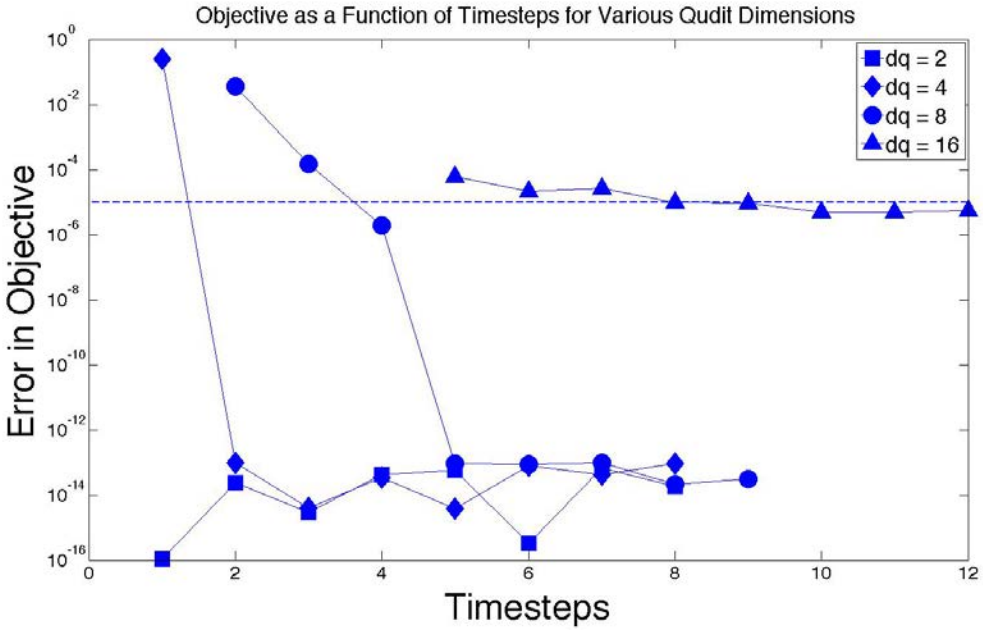


Figure 4.1: Objective error for the QFT as a function of number of timesteps for single qudit systems of varying dimension. Each plotted point represents the final objective error for a system with a given qudit dimension and number of timesteps. Each objective was reached after 2,000 iterations of GRAPE. The dotted line is our error tolerance threshold of 10^{-5} where we believe U begins to accurately represent the QFT.

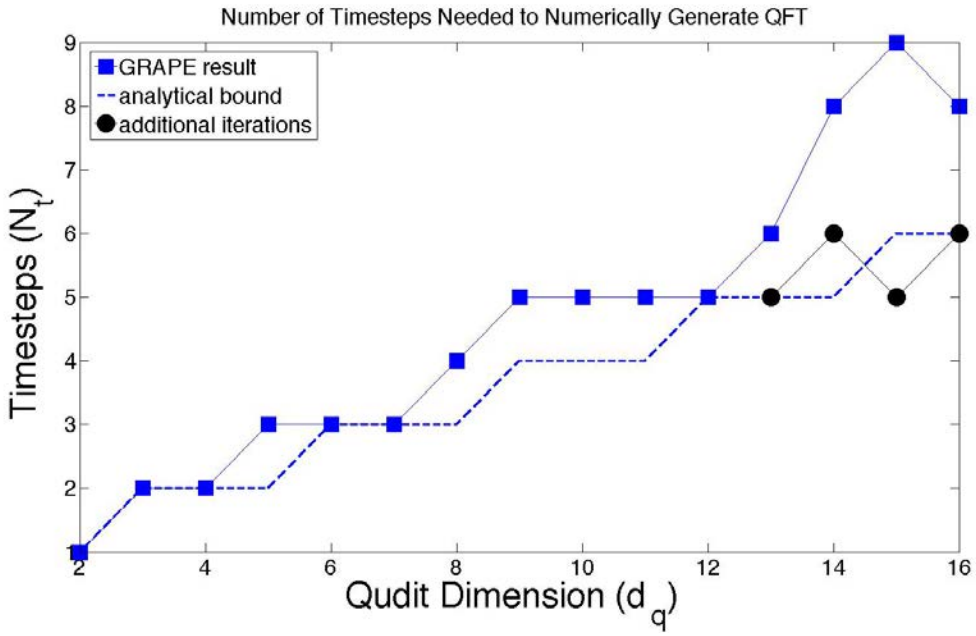


Figure 4.2: The number of timesteps required to numerically generate a QFT with objective error $\leq 10^{-5}$ for a given qudit dimension. Each objective was reached after 2,000 iterations of GRAPE. The dotted line denotes the timestep bound determined by our analytical prediction, while the markers indicate the results found by our GRAPE simulation. Circles mark results found after 5,000 iterations. For single-qudit systems of dimension 2 through 16, our empirical results adhere closely to the prediction, indicating that the number of timesteps required to generate the QFT has a $\sim d/3$ relationship to the qudit dimension.

4.2 Qubit-Qudit Comparison

In this section, we discuss our comparisons of qubit and qudit systems in implementing the QFT16 transformation.

4.2.1 Considerations when Comparing Qubits and Qudits

We wish to know whether, for a QFT implementation for a given dimension, a qubit-based system or qudit-based system will yield more time optimal results. At first, there appear to be reasons why this seems like an invalid endeavor. Physical implementations of qubit and qudit systems will likely be drastically different, with different means for control and different timescales over which controls can be implemented. It is also unlikely that qudit and qubit systems would have analogous coupling interactions.

For our research, however, a comparison is worthwhile because we wish to know whether the structure of interactions within qudit systems is better suited to implementations of the QFT than the structure of qubit systems. In order to test this, we controlled for differences between the two types of systems in multiple ways. Both our qudit and qubit systems solve for the QFT16 transformation: this requires a system of 4 qubits or two 4-dimensional qudits. Both systems make use of the tridiagonal Hamiltonian as the control structure for each qudit/qubit. We also attempt to ensure that both systems use analogous coupling interactions: ZZ Ising interactions for the qubit system and Cross-Kerr interactions for the qudit system (see section 2.5). The structure of a timestep Hamiltonian $\mathcal{H}_j$ for the 4 qubit system is

$$\mathcal{H}_j = \mathcal{H}_1 \otimes I \otimes I \otimes I + I \otimes \mathcal{H}_2 \otimes I \otimes I + I \otimes I \otimes \mathcal{H}_3 \otimes I + I \otimes I \otimes I \otimes \mathcal{H}_4 + \sum_{n=1}^{6} J\chi_n, \quad (4.7)$$

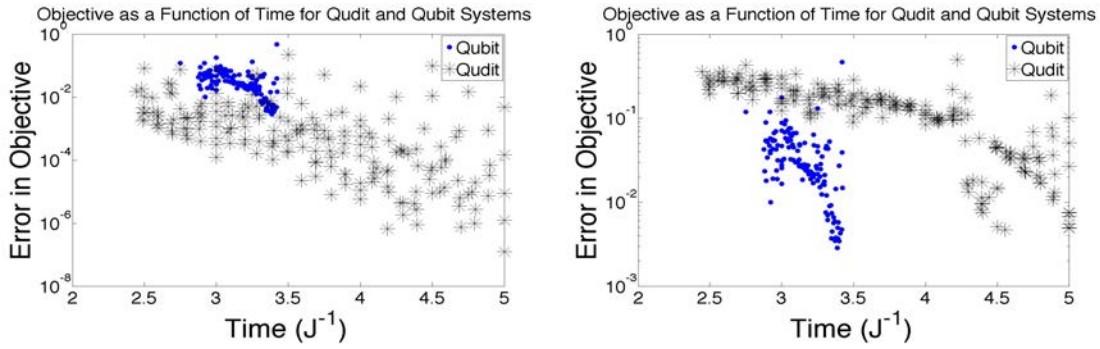
where $\mathcal{H}_i$ is a qubit Hamiltonian, C is the coupling strength, and χ_n is the ZZ interaction matrix for two qubits. All qubits are coupled with each other for a total of 6 coupling interactions. The structure of a timestep Hamiltonian $\mathcal{H}_j$ for the 2 qudit system is

$$\mathcal{H}_j = \mathcal{H}_1 \otimes I + I \otimes \mathcal{H}_2 + J'_K\chi, \quad (4.8)$$

where $\mathcal{H}_i$ is a tridiagonal qudit Hamiltonian, J'_k is the cross-Kerr coupling strength, and χ is the Kronecker product of two cross-Kerr matrices. The last important step is ensuring that both systems use the same timescale so that the time optimal results can be compared. Recall that the unitary operator for each timestep is given by

$$U_j = e^{-i\mathcal{H}\Delta t/\hbar}, \quad (4.9)$$

where $\mathcal{H}$ is a sum of the drift and control Hamiltonian. The control Hamiltonian can have arbitrary amplitude, but the drift Hamiltonian reflects coupling interactions and has an amplitude given by J. Thus, for our systems an increase in the timestep length Δt is



(a) Commensurate Coupling Force (b) Weak Qudit Coupling

Figure 4.3: The objective as a function of scaled run time for various qudit and qubit implementations. (a) Comparison for a qudit system with a coupling coefficient equal to the coupling coefficient in the qubit system. (b) Comparison for a qudit system with a coupling coefficient 1/4 of the qubit system coupling coefficient.

analogous to an increase in the coupling strength, and we can measure time in terms of the inverse coupling strength, J^{-1}. The total time it takes for to execute the QFT is the number of timesteps N_t times the time for each timestep, Δt. We parameterize the qudit system to have the same inter-qudit coupling strength as the qubit system, thus the same units of time can be used for both systems.

4.2.2 Qubit-Qudit Comparison Results

For each value plotted in Fig. 4.3, a different combination of time step number and time step length was parameterized for the search. Because some combinations of time step number and length are more effective than others, the lower bound of the qudit and qubit curves should be considered most pertinent to whether a time-optimal solution can be found for a given total time $T = N_t \Delta t$. For both qudits and qubits, below a certain time threshold a set of optimal controls cannot be found. For qubits, at around 3.5 J^{-1} a set of optimal controls can be found that reproduce the QFT to within an objective error $< 10^{-2}$. When the coupling strength is the same for both systems, a QFT approximation scheme with an error $\leq 10^{-2}$ is found in 30 percent less time for the qudit system than the qubit system. Notice, however, that if the coupling strength of the qudit system is reduced by a factor of four, the qudit system performs poorly in comparison to the qubit system, requiring a longer run time to achieve the same degree of overlap with the QFT. Thus, we see that when coupling strength disparities are large enough, these disparities appear to outweigh differences in system structure.

Chapter 5

Conclusion and Discussion

The goal of this thesis is to create a toolbox that allows us to numerically investigate the behavior of qubit and qudit systems using gradient search techniques. To that end, we developed implementations of the GRAPE algorithm in MATLAB and C++ that allow for modular customizability to fit the form and scale of a given problem. Our approaches to matrix exponentiation, matrix exponential derivatives, and gradient step sizes offer improvements to the computational efficiency of GRAPE while the use of parallelization on C++ allows for scaling to accommodate problems involving larger systems.

We then used our implementation of gradient search in MATLAB to study the generation of the quantum Fourier transform (QFT) in two systems. With single qudit systems, we examined whether the number of required timesteps scales based upon a relation of tridiagonal Hamiltonian controls to total QFT elements. We also examined two-qudit systems to see how their performance in the generation of the QFT compared to equivalent qubit systems. In both systems, there are indications that qudits offer advantages over qubits in the generation of the QFT. Below, we outline the finer details of our conclusions.

Gradient Search Implementation Discussion

We first examined the GRAPE algorithm and considered improvements to its computational efficiency. Because matrix exponentials are a computationally costly facet of GRAPE, we implemented the scaling and squaring method to yield efficient approximations of the exponential. Every iteration of GRAPE also requires an exponential derivative with respect to each control, thus we sought ways of efficiently approximating the exponential derivative. By implementing derivative tools using eigenvalue decomposition (EVD), commutator approximation, and the Simpson's rule approximation, we have the ability to choose derivative methods suited to a variety of problems. While EVD offers accuracy for small-scale problems, Simpson's rule approximations have been shown to be more computationally efficient for gradient searches on large systems when less accuracy is required, such as when we wish to understand algorithm scaling rather than generate an accurate

approximation. The last step of each GRAPE iteration is the update of the controls using the gradient. Here, the use of the Barzilai and Borwein gradient step method offers an order of magnitude improvement upon fixed-step updates. Lastly, we implemented GRAPE on C++, allowing us to take advantage of parallel processing protocols that can scale the GRAPE algorithm for searches on larger systems.

Numerical Simulation Discussion

With a tool box of gradient search routines in hand, we next turned to numerical simulations of qudit systems. We first examined the computation of the QFT using a single-qudit system. We have found that the number of timesteps needed to compute the QFT closely matches a $\sim d/3$ relation where d is the dimension of the QFT. Next, we considered whether qudit or qubit based systems are better suited to the computation of the QFT. By matching the coupling strengths of a qubit and qudit system, we were able to compare the run time efficiency of QFT algorithms run on qubit and qudit based networks. Here, we find evidence that when coupling strengths are equal, qudit systems offer faster computations of the QFT than equivalent qubit systems.

Future Work

There remain several lines of inquiry that merit further investigation:

- Whenever one carries out numerical approximations in the interest of computational efficiency, performance-accuracy trade-offs should be taken into account. In the future we would like to more carefully quantify the introduced error of our approximation schemes, especially for simulations of larger systems that we would like to examine in the future.

- We have yet to take full advantage of our GRAPE toolbox and study large-dimensional systems. Because GRAPE now accommodates parallel processing, simulations on large computing clusters could offer insights into qudit systems orders of magnitude larger than the ones examined here. For these larger systems, there may be emergent behavior that is unseen in our smaller simulations.

- There remain many opportunities for us to consider other configurations of quantum systems that reflect possible physical implementations of quantum computing. For example, it may be practically difficult to implement piecewise Hamiltonian controls, thus other researchers have examined how to simulate optimal control for slowly-modulating control pulses [14]. We have yet to test such non-idealized configurations.

Appendix A

MPI Implementation

In order to implement our gradient algorithm in a way that would allow for scalability to accommodate large problems, we turned to MPI (Message Passing Interface). MPI is a specification outlining the manner in which processors should communicate with one another [2]. Processors running the same implementation of MPI are able to send messages to one another in a consistent manner, allowing for parallel processes to be run. In our research, we made use of MPI to distribute computational tasks from our GRAPE search algorithm.

A.1 MPI Overview

MPI allows for messages to be passed between different processes, enabling parallel computing. When a routine is executed, such as our GRAPE algorithm, each individual process runs the routine locally but is assigned a different "rank" by MPI. Any parts of the routine that do not discriminate based upon rank are run by all processes. Using MPI, however, we can break up certain parts of our routine to be executed only by processes of a certain rank. When these processes complete their task, they can then use MPI calls to broadcast the results of their task to other processes. In this way, a large task can be broken up into several smaller tasks that are executed by individual processes.

Because our gradient search algorithm must compute the gradient of the control landscape with each iteration, there is an opportunity for MPI to parallelize this time-intensive process and reduce overall computational time. Recall that in order to find the derivative of our evolved unitary with respect to a control $c_{k,j}$, we only need to consider the derivative of the unitary at time step j:

$$\frac{\partial U}{\partial c_{k,j}} = U_{N-1}U_{N-2}...U_{j+1}\frac{\partial U_j}{\partial c_{k,j}}U_{j-1}U_{j-2}...U_0 = \left(\prod_{n=N-1}^{j+1} U_n\right)\frac{\partial U_j}{\partial c_{k,j}}\left(\prod_{n=j-1}^{0} U_n\right). \quad \text{(A.1)}$$

By distributing the calculation of timestep unitary matrices to different processes, we can

compute the derivative of the evolved unitary with respect to the controls in much less time.

A.2 Implementing MPI for use in GRAPE

Because our gradient search algorithm must compute the gradient of the control landscape with each iteration, there is an opportunity for MPI to parallelize this time-intensive process and reduce overall computational time.

For our C++ implementation of GRAPE, we utilized the open source compiler "mpi C++". The program `optimalp.cc` calls the MPI routines and must begin with a header file `#include "mpi.h"` . Once compiled and executed, `optimalp.cc` will run simultaneously on every process. Each process then uses the following calls to initialize MPI and receive a ranking:

```
char processor_name[MPI_MAX_PROCESSOR_NAME];
int numprocs, rank, namelen, subcount;
MPI_Init(NULL, NULL);

MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Get_processor_name(processor_name, &namelen);
```

These calls fetch information on the number of total processes available, the rank of the local process, and the assigned name of the local process. Next, we define two arrays, `recvcounts` and `displs`, that store the number of controls and position of the controls, respectively, that each process is responsible for. Each process will solve for the derivative of the unitary with respect to the controls it is responsible for.

At the beginning of each iterative step in GRAPE, `MPI_Bcast` is called to send the controls and gradient step size to every sub-process. `MPI_Bcast` broadcasts a message to every process within the routine, and is called with the format:

```
MPI_Bcast (&buffer, count, datatype, root, comm)
```

Where `buffer` is the array to be broadcast, `count` is the size of the passed array, `datatype` characterizes the data type of the array being passed, `root` is the rank of the process that begins the broadcast, and `comm` is the environment within which the messages are being passed (`MPI_COMM_WORLD` in this case).

Because MPI only allows for a set of objects to be passed as an array, each process must then unpack the broadcasted array into the vector format that is used by our C++ GRAPE implementation. Each process then computes the portion of the gradient calculation assigned to it. Meanwhile, the root process is left to complete other tasks within the

algorithm, such as finding the proper gradient step and calculating the objective based upon the current iteration of controls.

Once every sub-process has completed its calculation of a portion of the gradient, the derivatives are sent to the root process using the call `MPI_Gatherv`. This call has the following form [3]:

```
MPI_Gatherv (sbuf, scount, stype, rbuf, rcounts, displs, rtype, root, comm)
```

`MPI_Gatherv` requires a large number of parameters because it instructs the root process how to assemble the information coming from several different processes into one cohesive object. `sbuff` is the object being sent to the root process, `scount` is the size of the object being sent, `stype` characterized the data type being passed to the root process, `rebuff` is the object that holds the received objects, `rcounts` is the size of the object storing the received objects, `displs` describes the location at which each received object should be stored within `rbuf`, `rtype` describes the data type of the final object, `root` is the rank of the root process that will assemble the objects, and `comm` is the environment within which the messages are being passed (`MPI_COMM_WORLD` in this case).

With the `MPI_Gatherv` call, the complete gradient vector is assembled by the root process and used to update the controls. The next iteration can then begin and this process is repeated.

A.3 MPI Challenges and Future Additions

While MPI allows us to parallelize the calculation of the derivative, there are other bottlenecks in GRAPE where parallel processing cannot be as easily implemented. While the derivative of the evolved unitary with respect to each control can be broken up into pieces, there remains the calculation of of the product of timestep unitaries (see eq. A.1). The product of j timestep unitaries is a time-intensive process that must be taken up by each process. We considered implementing a system by which processes worked together to multiply unitary matrices, but we believe that such a solution would in fact slow down computational speed by drastically increasing the amount of communication between processes.

Appendix B

MATLAB Documentation

The following is a code glossary for functions and scripts that are used to implement GRAPE in MATLAB.

adaptive_step

Purpose Generates the step size used to calculate how much to alter the controls with each iteration. Possible types of gradient steps are a fixed size, simple inverse method, empirical Cauchy method, and Barzilai and Borwein method.

Parameters

phi The objective error from the previous iteration.

type the type of adaptive step to be used ("none", "simple_inverse", "cauchy", or "BaB").

ukx The control vector for off-diagonal reals.

uky The control vector for off-diagonal imaginaries.

ukz The control vector for on-diagonal elements.

derx Vector of derivatives with respect to ukx controls.

dery Vector of derivatives with respect to uky controls.

derz Vector of derivatives with respect to ukz controls.

old_ukx ukx controls from the last iteration.

old_uky uky controls from the last iteration.

old_ukz ukz controls from the last iteration.

old_derx ukx derivatives from the last iteration.

old_dery uky derivatives from the last iteration.

old_derz ukz derivatives from the last iteration.

epsilon_old The step size from the last iteration.

u_ideal The goal unitary matrix.

step The current iteration number.

Called By `optimal`.

calc_der_U

Purpose Calculates the derivative of the timestep unitary U_j with respect to a control c_k, j using the EVD method.

Parameters

e_vec The set of eigenvectors for the current timestep Hamiltonian.

e_val The set of eigenvalues for the current timestep Hamiltonian.

d The dimension of the goal unitary.

dt The time allotted to each timestep evolution.

Hk The control Hamiltonian $\mathcal{H}^{(k)}$.

Called By `optimal`.

calc_der_U_com

Purpose Calculates the derivative of the timestep unitary U_j with respect to a control c_k, j using the commutator approximation method.

Parameters

op1 The timestep Hamiltonian $\mathcal{H}_j$.

op2 The control Hamiltonian $\mathcal{H}^{(k)}$.

dt The time allotted to each timestep evolution.

n The number of terms to be used in the approximation.

Called By `optimal`.

calc_der_U_sim

Purpose Calculates the derivative of the timestep unitary U_j with respect to a control c_k, j using Simpson's rule of approximation.

Parameters

op1 The timestep Hamiltonian $\mathcal{H}_j$.

op2 The control Hamiltonian $\mathcal{H}^{(k)}$.

dt The time allotted to each timestep evolution.

n The number of terms to be used in the approximation. n defines the number of slices Simpson's rule uses to approximate the integral.

Called By `optimal`.

delta

Purpose Creates a matrix that is used to build a control Hamiltonian for on-diagonal controls.

Parameters

k The index marking the position of the on-diagonal control within a qudit.

d The dimension of qudit Hamiltonians in the system.
Called By `kron_prod`.

evolvedU

Purpose Generates the evolved unitary U by taking the product of timestep unitaries U_j.
Parameters
ukx The control vector for off-diagonal reals.
uky The control vector for off-diagonal imaginaries.
ukz The control vector for on-diagonal elements.
qd The dimension of qudits in the system.
dt The time allotted to each timestep evolution.
N The number of timesteps N_t.
qn The number of qudits in the system.
Refers To `generateU`.
Called By `optimal`.

generateH_qudit

Purpose Constructs a timestep Hamiltonian $\mathcal{H}_j$ by summing qudit and interaction Hamiltonians.
Parameters
ukx The control vector for off-diagonal reals.
uky The control vector for off-diagonal imaginaries.
ukz The control vector for on-diagonal elements.
qd The dimension of qudits in the system.
j The index of the timestep Hamiltonian.
qn The number of qudits in the system.
Refers To `kerr`, `kron`, `kronH`.
Called By `generateU`, `optimal`.

generateHc

Purpose Creates a qudit Hamiltonian at timestep j for qudit c.
Parameters
ukx The control vector for off-diagonal reals.
uky The control vector for off-diagonal imaginaries.
ukz The control vector for on-diagonal elements.
qd The dimension of qudits in the system.
j The index of the timestep when the Hamiltonian is generated.

c The index of the qudit.
Called By `kronH`.

generateU

Purpose Generates the timestep unitary matrix U_j.
Parameters
ukx The control vector for off-diagonal reals.
uky The control vector for off-diagonal imaginaries.
ukz The control vector for on-diagonal elements.
qd The dimension of qudits in the system.
dt The time allotted to each timestep evolution.
j The index of the timestep for the unitary matrix.
qn The number of qudits in the system.
Refers To `generateH_qudit`.
Called By `evolvedU`, `leftUj`.

kerr

Purpose Constructs a cross-Kerr interaction matrix dependent upon the qudit size qd.
Parameters
d The dimension of each qudit.
Called By

kron_prod

Purpose Takes the Kronecker product of control Hamiltonians with identity matrices to help create the timestep Hamiltonian H_j by constructing a qudit control Hamiltonian.
Parameters
type A string describing the type of control Hamiltonian that is being taken with identity matrices (options are 'sigmaX' (real off-diagonal control), 'sigmaY' (imaginary off-diagonal control), and 'delta' (on-diagonal control).
c The index of the qudit the Hamiltonian is being constructed for.
k The index of the control in the qudit Hamiltonian.
qd The dimension of qudits in the system.
qn The number of qudits in the system.
Refers To `delta`, `kron`, `sigmaX`, `sigmaY`.
Called By `generateH_qudit`.

kronH

Purpose Generate a Kronecker product of identity and qudit Hamiltonians to help create a timestep Hamiltonian H_j.

Parameters

ukx The control vector for off-diagonal reals.

uky The control vector for off-diagonal imaginaries.

ukz The control vector for on-diagonal elements.

qd The dimension of qudits in the system.

j The index of the timestep for the control.

c The index of the qudit that is involved in the Kronecker product.

qn The number of qudits in the system.

Refers To `generateHc`, `kron`.

Called By `generateH_qudit`

leftUj

Purpose Backwards evolves the evolved unitary U by one timestep unitary matrix. This operation helps generate the objective derivative with respect to controls for a given timestep.

Parameters

ukx The control vector for off-diagonal reals.

uky The control vector for off-diagonal imaginaries.

ukz The control vector for on-diagonal elements.

dt The amount of time allotted to each timestep unitary.

j The local timestep of the unitary matrix.

Uf The evolved unitary matrix that is to be de-evolved by one step.

qn The number of qudits in the system.

qd The dimension of each qudit in the system.

Refers To `generateU`.

Called By `optimal`.

optimal

Purpose The function that executes the GRAPE algorithm. It is called by a main function and can be modified to return varying information on the iterative process, final approximation, and objective error.

Parameters

ukx The control vector for off-diagonal reals.

uky The control vector for off-diagonal imaginaries.

ukz The control vector for on-diagonal elements.

u_ideal The matrix operation that GRAPE attempts to approximate (the QFT in our case).
tol A double representing the threshold of objective error below which the approximation has sufficient fidelity and the algorithm ceases.
iter The maximum number of iterations to execute.
adaptive_type A string specifying the type of gradient step to be used.
epsilon The gradient step size to be used in the first iteration.
qd The dimension of each qudit.
dt The amount of time allotted to each timestep unitary.

Refers To `adaptive_step`, `calc_der_U`, (alternative implementations: `calc_der_U_com`, `calc_der_U_sim`) `evolvedU`, `generateH_qudit`, `kron_prod`, `leftUj`, `phi_final`, `rightUj`.

phi_final

Purpose Calculates and returns the objective error.
Parameters
op1 The evolved unitary approximation.
op2 The ideal unitary matrix.
d The dimension of the ideal unitary.
Called By `optimal`.

rightUj

Purpose Evolves a unitary by another timestep unitary matrix in order to generate a unitary matrix used to calculate the objective derivative for a given timestep.
Parameters
ukx The control vector for off-diagonal reals.
uky The control vector for off-diagonal imaginaries.
ukz The control vector for on-diagonal elements.
dt The amount of time allotted to each timestep unitary.
j The local timestep of the unitary matrix.
Uinit The evolved unitary matrix that is to be evolved by an additional step.
qn The number of qudits in the system.
qd The dimension of each qudit in the system.
Refers To
Called By `optimal`.

sigmaX

Purpose Creates a matrix that is used to build a control Hamiltonian for real off-diagonal controls.

Parameters

k The index of the control within the qudit.

d The dimension of the goal unitary.

Called By `kronH`.

sigmaY

Purpose Creates a matrix that is used to build a control Hamiltonian for imaginary off-diagonal controls.

Parameters

k The index of the control within the qudit.

d The dimension of the goal unitary.

Called By `kronH`.

Appendix C

C++ Documentation

The following is a code glossary for functions that are used to implement GRAPE in C++. Note that with the exceptions of `optimal.cc` and `optimalp.cc`, all functions are found within the file `qobjects.cc`. The header file is found at `qobjects.h`.

adaptive_step

Purpose Generates the step size used to calculate how much to alter the controls with each iteration. Possible types of gradient steps are a fixed size, simple inverse method, and Barzilai and Borwein method.

Parameters

phi_Old The objective error from the previous iteration.

adname the type of adaptive step to be used ("none", "simple_inverse", or "BaB").

ukx The control vector for off-diagonal reals.

uky The control vector for off-diagonal imaginaries.

ukz The control vector for on-diagonal elements.

derivative_wrt_x Vector of derivatives with respect to ukx controls.

derivative_wrt_y Vector of derivatives with respect to uky controls.

derivative_wrt_z Vector of derivatives with respect to ukz controls.

ukx_old ukx controls from the last iteration.

uky_old uky controls from the last iteration.

ukz_old ukz controls from the last iteration.

old_derivative_wrt_x ukx derivatives from the last iteration.

old_derivative_wrt_y uky derivatives from the last iteration.

old_derivative_wrt_z ukz derivatives from the last iteration.

old_epsilon The step size from the last iteration.

uideal The goal unitary matrix.

step The current iteration number.

Called By `optimal`.

der_exp

Purpose Computes the derivative of exp(A + B) using the commutator approximation method.
Parameters
op1 Matrix A to be exponentiated.
op2 Matrix B to be exponentiated.
dt The amount of time allotted to operators A and B to rotate the state.
n The number of commutated terms to be used in approximation. More terms yield a higher degree of accuracy at the cost of computational time.
Called By `optimal`.

der_exp_sim

Purpose Computes the derivative of exp(A + B) using Simpson's approximation method.
Parameters
op1 Matrix A to be exponentiated.
op2 Matrix B to be exponentiated.
dt The amount of time allotted to operators A and B to rotate the state.
n The number of subintervals to be used in the Simpson's approximation. More terms yield a higher degree of accuracy at the cost of computational time.
Called By `optimal` (if implemented instead of `der_exp`).

evolvedU

Purpose Calculates the evolved unitary product of all the timestep unitary matrices.
Parameters
ukx The control vector for off-diagonal reals.
uky The control vector for off-diagonal imaginaries.
ukz The control vector for on-diagonal elements.
qd The dimension of each qudit.
dt The amount of time allotted to each timestep unitary.
N The total number of timesteps.
qn The number of qudits.
Refers To `generateU`.
Called By `optimal`.

exp_op

Purpose Calculates the matrix exponential of the argument $-i * A$.
Parameters

A The matrix that forms the argument of the exponential.
dt The time allotted to the matrix operator to act upon the state.
n The number of squaring operations applied to A.
Refers To `Givens`.
Called By `GenerateU`.

generateH_qudit

Purpose Generates the combined timestep Hamiltonian that includes all qudit-qudit interactions.
Parameters
ukx The control vector for off-diagonal reals.
uky The control vector for off-diagonal imaginaries.
ukz The control vector for on-diagonal elements.
qd The dimension of each qudit.
j The local timestep of the Hamiltonian.
qn The number of qudits in the system.
Refers To `kronH`, `tensor`.
Called By `generateU`, `optimal`.

generateHc

Purpose Calculates the tridiagonal Hamiltonian for a qudit in a given timestep by pulling the control values from control vectors.
Parameters
ukx The control vector for off-diagonal reals.
uky The control vector for off-diagonal imaginaries.
ukz The control vector for on-diagonal elements.
qd The dimension of each qudit.
j The local timestep of the Hamiltonian.
i Denotes the qudit in the system for which the tridiagonal Hamiltonian is generated.
Called By `kronH`.

generateU

Purpose Constructs the unitary matrix operator for the system for a single timestep.
Parameters
ukx The control vector for off-diagonal reals.
uky The control vector for off-diagonal imaginaries.
ukz The control vector for on-diagonal elements.
qd The dimension of each qudit.

dt The amount of time allotted to each timestep unitary.
j The local timestep of the Hamiltonian.
qn The number of qudits in the system.
Refers To `generateH_qudit`, `exp_op`.
Called By `evolvedUj`, `leftUj`, `right_Uj`.

Givens

Purpose An object construction function that builds a Givens rotation matrix using the object type `qoperator`.
Parameters
opname The type of matrix to be constructed ("givens" in this case).
rotations The vector holding the 4 rotations that are the non-null elements of the Givens matrix.
size The size of the rotations vector (must be four).
row_index The row index marking where the Givens block begins.
column_index The column index marking where the Givens block begins.
Refers To `swap_elem`.
Called By `exp_op`.

kron_prod

Purpose Generates the control Hamiltonian matrices that are used in generateHc.
Parameters
opname A string containing the type of control Hamiltonian to be generated.
c Denotes the qudit in the system for which the control Hamiltonian is generated.
k Specifies the location of the control for which the control Hamiltonian is generated.
qd The dimension of each qudit.
qn The number of qudits in the system.
Refers To `tensor`.
Called By `optimal`.

kronH

Purpose Generates a matrix tensor product of identity operators and tridiagonal Hamiltonians.
Parameters
ukx The control vector for off-diagonal reals.
uky The control vector for off-diagonal imaginaries.
ukz The control vector for on-diagonal elements.
qd The dimension of each qudit.

j The local timestep of the Hamiltonian.
qn The number of qudits in the system.
Refers To `generateHc`, `tensor`.
Called By `generateH_qudit`.

leftUj

Purpose Backwards evolves the evolved unitary by one timestep unitary matrix. This operation helps generate the objective derivative with respect to controls for a given timestep.
Parameters
ukx The control vector for off-diagonal reals.
uky The control vector for off-diagonal imaginaries.
ukz The control vector for on-diagonal elements.
dt The amount of time allotted to each timestep unitary.
j The local timestep of the unitary matrix.
left The evolved unitary matrix that is to be de-evolved by one step.
qd The dimension of each qudit.
qn The number of qudits in the system.
Refers To `generateU`
Called By `optimal`.

optimal

Purpose The function that executes the GRAPE algorithm. It is called by a main function and can be modified to return varying information on the iterative process, final approximation, and objective error.
Parameters
ukx The control vector for off-diagonal reals.
uky The control vector for off-diagonal imaginaries.
ukz The control vector for on-diagonal elements.
uideal The matrix operation that GRAPE attempts to approximate (the QFT in our case).
tol A double representing the threshold of objective error below which the approximation has sufficient fidelity and the algorithm ceases.
iter The maximum number of iterations to execute.
adaptive_type A string specifying the type of gradient step to be used.
epsilon The gradient step size to be used in the first iteration.
qd The dimension of each qudit.
dt The amount of time allotted to each timestep unitary.
qn The number of qudits in the system.

Refers To `adaptive_step`, `der_exp`, `evolvedU`, `generateH_qudit`, `kron_prod`, `leftUj`, `phi_final`, `rightUj`.

phi_final

Purpose Calculates and returns the objective error.
Parameters
op1 The evolved unitary approximation.
op2 The ideal unitary matrix.
d The dimension of the ideal unitary.
Called By `optimal`.

right_Uj

Purpose Evolves a unitary by another timestep unitary matrix in order to generate a unitary matrix used to calculate the objective derivative for a given timestep.
Parameters
ukx The control vector for off-diagonal reals.
uky The control vector for off-diagonal imaginaries.
ukz The control vector for on-diagonal elements.
dt The amount of time allotted to each timestep unitary.
j The local timestep of the Hamiltonian.
Uinit The evolved unitary matrix that is to be evolved by one step.
qd The dimension of each qudit.
qn The number of qudits in the system.
Refers To `generateU`
Called By `optimal`.

swap_elem

Purpose Swaps out the value at index x, y of a matrix with a new value z.
Parameters
x The row index.
y The column index.
z The new value to be inserted.
Called By `Givens`.

tensor

Purpose Generates the tensor product between multiple matrices (multiple functions).

Parameters

op1 The first matrix in the tensor product.

op2 The second matrix in the tensor product.

op3 The third matrix in the tensor product.

Called By `generateH_qudit`, `kronH`, `kron_prod`.

References

[1] Dave Bacon, *The Quantum Fourier Transform and Jordan's Algorithm*, University of Washington Lecture Notes (Unpublished).

[2] Blaise Barney, *Message Passing Interface (MPI)*, Lawrence Livermore National Laboratory Tutorial, https://computing.llnl.gov/tutorials/mpi/.

[3] Brandon Barker, *MPI Collective Communications*, Cornell Center for Advanced Computing Virtual Workshop Tutorial, https://www.cac.cornell.edu/vw/MPIcc/default.aspx.

[4] A. Ben-Kish, B. DeMarco, V. Meyer, M. Rowe, J. Britton, W. M. Itano, B. M. Jelenković, C. Langer, D. Leibfried, T. Rosenband, and D.J. Wineland. Experimental Demonstration of a Technique to Generate Arbitrary Quantum Superposition States of a Harmonically Bound Spin-1/2 Particle. *Physical Review Letters*, 90(3): 037902, 2003.

[5] Jonathan Barzilai and Jonathan M. Borwein. Two-Point Step Size Gradient Methods. *IMA Journal of Numerical Analysis*, 8: 141-148, 1988.

[6] Alexandre Blais. Quantum network optimization. *Physical Review A*, 64: 022313, 2001.

[7] Gavin K. Brennan, Dianne P. O'Leary, and Stephen S. Bullock. Criteria for exact qudit universality. *Physical Review A*, 71(5): 052318, 2005.

[8] David P. DiVincenzo. Two-bit gates are universal for quantum computation. *Physical Review A*, 51: 1015-1022, 1995.

[9] David J. Griffiths. *Introduction to Quantum Mechanics*. Pearson Prentice Hall, Upper Saddle River, 2005.

[10] Navin Khaneja, Timo Reiss, Cindie Kehlet, Thomas Schulte-Herbrüggen, Steffen J. Glaser. Optimal control of coupled spin dynamics: design of NMR pulse sequences by gradient ascent algorithms. *Journal of Magnetic Resonance*, 172(2): 296-305, 2005.

[11] S. Machnes, U. Sander, S.J. Glaser, P. de Fouquieres, A. Gruslys, S. Schirmer, T. Schulte-Herbruggen. Comparing, optimizing, and benchmarking quantum-control algorithms in a unifying programming framework. *Physical Review A*, 84(2): 022305, 2011.

[12] N. David Mermin. *Quantum Computer Science: An Introduction*. Cambridge University Press, Cambridge, 2007.

[13] Cleve Moler and Charles Van Loan. Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later. *SIAM Review*, 45(1): 1-46, 2003.

[14] F. Motzoi, J. M. Gambetta, S. T. Merkel, F.K. Wilhelm. Optimal control methods for fast time-varying Hamiltonians. *Physical Review A*, 84(2): 022305, 2011.

[15] Matthew Neeley, Markus Ansmann, Radoslaw C. Bialczak, Max Hofheinz, Erik Lucero, Aaron D. O'Connell, Daniel Sank, Haohua Wang, James Wenner, Andrew N. Cleland, Michael R. Geller, and John M. Martinis. Emulation of a Quantum Spin with a Superconducting Phase Qudit. *Science*, 325(5941): 722-725, 2011.

[16] M. A. Nielson, I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, 2000.

[17] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*, volume 2nd. Cambridge University Press, 1988.

[18] John L. Richardson. Visualizing quantum scattering on the CM-2 supercomputer. *Computer Physics Communications*, 63: 84-94, 1991.

[19] A. Saito, K. Kioi, Y. Akagi, N. Hashizume, and K. Ohta. Actual computational time-cost of the Quantum Fourier Transform in a quantum computer using nuclear spins. eprint. arXiv: quant-ph/0001113, 2000.

[20] T. Schulte-Herbrüeggen, A. Spörl, N. Khaneja, and S. J. Glaser. Optimal control-based efficient synthesis of building blocks of quantum algorithms: A perspective from network complexity towards time complexity. *Physical Review A*, 72: 042331, 2005.

[21] Gary F. Sinclair and Natalia Korolkova. Cross-Kerr interaction in a four-level atomic system. *Physical Review A*, 76, 033803, 2007.

[22] John R. Taylor. *Classical Mechanics*. University Science Books, Sausalito, 2005.

[23] Ya-xiang Yuan. Step-Sizes for the Gradient Method. *AMS/IP Studies in Advanced Mathematics*, 785-796, 2008.

[24] Haruo Yoshida. Construction of higher order symplectic integrators. *Physics Letters A*, 150(5,6,7) 262-268, 1990.