Technische Universität München
Fakultät für Physik

Abschlussarbeit im Bachelorstudiengang Physik

TITEL

Luca Göcke

datum

Erstgutachter (Themensteller): Prof. Norbert Schuch
Zweitgutachter: Prof. Michael Knap
In Zusammenarbeit mit Prof. Robert König

Contents

Chapter 1

Introduction

Finding efficient algorithms to compute product state approximations for quantum many body systems plays a key role in investigating physical models. The general problem of product state approximation has been studied for nearly a century now and has applications in a remarkable variety of fields, including quantum many body physics, chemistry and condensed matter physics. Finding tractable methods to realize good approximations to groundstate configurations of quantum systems has attracted much attention, since it is the quantum analogue of constraint satisfaction where we are given a set of Boolean constraints on k variables each and try to satisfy as many constraints as possible. The topic of this thesis presents a beautiful intersection of several fields, since its theory combines physics, mathematics and computer science, and has promising benefits for different subfields of chemistry and physics.
Using classical algorithms to approximate the maximal energy approximation is a useful method which has recently been extensively studied, see for example [1–7]. The objective of this thesis is to provide an introduction to the paper by Bravyi *et al.* [6], which demonstrates an efficient algorithm for approximating the groundstate energy of traceless 2-local Hamiltonians, presenting an overview of the basic concepts and examining the mathematical reasoning behind them. I will also illustrate the challenge of generalizing such an algorithm to three-level systems, and propose a rounding scheme that parallelizes the one used for qubits.
The thesis is organized as follows: In the first chapter, an introduction to quantum information theory is given. I then proceed to provide an overview of methods from classical computing which are needed to understand the discussed paper. Chapter 4 breaks down the approach of proving the results of Bravyi *et al.* and presents the algorithm. In chapter 5, I present the results of implementing the algorithm and analyze the implementation of two models of Hamiltonians. I introduce the reader to the properties of qutrit systems and propose a way of generalizing the algorithm by Bravyi *et al.* This is also tested through implementation. All code that has been written for this thesis can be found in the appendix.

Chapter 2

Primer on quantum information theory

I will first give an introduction to quantum information theory, which uses quantum mechanical concepts to perform information processing and transmission of information. In quantum mechanics, we can associate a Hilbert space $\mathbb{H}$ with every quantum system. Quantum states are operators $\rho : \mathbb{H} \to \mathbb{H}$, which we can represent as density matrices. In general, a complex $M \times M$ matrix is a density matrix if it is:

1. Hermitian, $\rho = \rho^\dagger$,
2. positive, $\rho \geq 0$,
3. normalized, $Tr\rho = 1$.

The set of density matrices is a convex set and its pure states obey $\rho^2 = \rho$. In quantum computing, we mostly deal with N 2-level systems called qubits, the composite space of which is $H = H_1 \otimes H_2 \otimes \ldots \otimes H_N$. In this space, there are states ρ which can not be expressed through a tensor product of states in the subsystems $\rho = \rho_1 \otimes \rho_2 \ldots \otimes \rho_N$. We call these states entangled states. States which can be expressed as such are called separable or product states. We can represent a qubit state as

$$\rho = \frac{1}{2}\left(\mathbb{I} + \sum_{i=1}^{3} \sigma_i \tau_i\right).$$

This is called the Bloch representation of the state and is associated with the Bloch-vector $\boldsymbol{\tau}$. The σ_i are generators of $SU(2)$, in our case the Pauli matrices:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

For qubits, the positivity property is equivalent to $Tr\rho^2 \leq Tr\rho$, which implies $|\boldsymbol{\tau}| \leq 1$ and characterizes the Bloch-vector-space as a solid Ball with Radius 1, called the Bloch sphere.
Suppose $|0\rangle$ and $|1\rangle$ form an orthogonal basis for the 2-dimensional one qubit state space. An arbitrary vector in the space can then be written

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle$$

where a and b are complex numbers. The normalization condition of quantum states is equivalent to $\langle \psi | \psi \rangle = 1$ and $|a|^2 + |b|^2 = 1$. The orthogonal basis vectors of the state space are called computational basis. Per convention we let $|0\rangle$ and $|1\rangle$ be the ± 1 eigenvectors of Z.
The basic model of transmitting quantum information has three steps: We send a state ρ through a quantum channel $\mathcal{N}$ and the receiver has to measure the outcome in order to extract information. Quantum channels can be understood either as geometrical transformations associated with the Bloch representation, or as completely positive, trace preserving maps. A quantum channel has to be trace preserving i.e. $Tr(\mathcal{N}(\rho)) = Tr(\rho)$ in order for the outcome state to be normalized. It must be completely positive, i.e. the map $\mathbb{I} \otimes \mathcal{N}$ maps positive semidefinite hermitian matrices to positive semidefinite hermitian matrices for any identity matrix $\mathbb{I}$, in order for the outcome state to be positive. By the Kraus representation Theorem [8] a linear map Ψ is completely positive if and only if there exist operators $\{A_i\}$ such that

$$\Psi(\rho) = \sum_i A_i \rho A_i^\dagger.$$

A completely positive map is trace preserving if and only if $\sum_i A_i^\dagger A_i = \mathbb{I}$.
Maps that are both completely positive and trace preserving are called CPT maps. We discern between unital and non-unital maps. Unital maps map the identity to itself. Geometrically, we can interpret this as the image of the map having the same center as the Bloch sphere. Unital maps can be expressed as convex combinations of the Pauli operators and the identity. Their action in the Bloch sphere are different rotations with shrinking parameters, since the Pauli matrices are unitary [9].
The most commonly used model for quantum computation is the quantum circuit model, which generalizes its classical analogue. To classify quantum algorithms we use, in parallel to classical complexity theory, quantum complexity classes. The two prominent complexity classes in classical computation are P and NP. P is the set of problems which can be solved by a deterministic Turing machine in polynomial time, while NP is the set of problems which can be solved by a nondeterministic Turing machine in polynomial time. The class QMA is the quantum analogue of NP in a probabilistic setting, the class of all problems which can be solved by a quantum verifier probabilistically in polynomial time [3]. We call a problem complete if any other problem in its class can be reduced to it. Reduction means that for predicates L_1 and L_2 there is a polynomial f, such that $L_1(x) = L_2(f(x))$. We say that f reduces L_1 to L_2 polynomially.
The Hamiltonian of a system corresponds to its energy, the spectrum of the operator being the set of possible outcomes when measuring the total energy. A k-local-Hamiltonian is a hermitian matrix acting on N qubits, which can be written as a sum of Hamiltonians where each acts on at most k qubits. Physically, this corresponds

to system where the interaction energy between more than k qubits is negligible. Specifically, we look at 2-local-Hamiltonians on qubits of the form

$$H = H_1 + H_2.$$

where

$$H_1 = \sum_{j=1}^{3n} D_j P_j, \quad H_2 = \sum_{i,j=1}^{3n} C_{i,j} P_i P_j \tag{2.1}$$

with the Pauli-operators

$$P_{3a-2} = X_a,\ P_{3a-1} = Y_a,\ P_{3a} = Z_a.$$

The minimal eigenvalue of such a systems corresponds to its ground state. Since the quantum state achieving this optimal value might be an entangled state which might not be computable in polynomial time, we are interested in finding the product state that achieves the best approximation. It is equivalent to finding the best approximation to the maximal eigenvalue, because $\lambda_{max}(-H) = \lambda_{min}(H)$ [1].
In the local Hamiltonian problem, we are to determine, whether the ground state energy of a given k-local Hamiltonian is below one threshold or above another. It is equivalent to the maximum constraint satisfaction problem from classical computation. The 2-local Hamiltonian problem is QMA complete [3]. Specifically, we look at *traceless* 2-local Hamiltonians, as these are the quantum generalization of binary quadratic functions of the form

$$F(x) = x^T Bx + v^T x, \quad x \in \{\pm 1\}^n,$$

where $B \in \mathbb{R}^{n\times n}$ is a matrix with zero diagonal and $v \in \mathbb{R}^n$ a vector.

Chapter 3

Relevant classical methods

For product state approximation algorithms, many techniques from classical algorithms are used and generalized. Finding the maximal eigenvalue of a traceless 2-local hamiltonian is the quantum analogue to maximizing a binary quadratic program (MaxQP): Given a matrix A with $a_{ii} = 0$ maximize

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j \quad \text{s.t.} \quad x_i \in \{-1, 1\} \quad \forall\, i.$$

An important tool for solving such problems is the relaxation of a semidefinite program (SDP), which has been pioneered by Goemans and Williamson.
In semidefinite programing we try to maximize a linear function, such that an affine combination of symmetric matrices is positive semidefinite. An affine combination is a linear combination $\sum_{i=1}^{n} a_i x_i$ where x_i are elements of a vector space, such that $\sum_{i=1}^{n} a_i = 1$. Semidefinite programs are very useful, as they can be solved efficiently both in theory and in practice [10]. We can write a general SDP as:

$$\begin{aligned} \text{minimize} \quad & C \cdot X \\ \text{s.t.} \quad & A_i \cdot X = b_i, \quad i = 1, \ldots, m \\ & X \geq 0 \end{aligned}$$

where C and A_i are symmetric matrices and $b_i \in \mathbb{R}^m$ is a vector. This is called the primal problem. The dual of a SDP is its reformulated version, such that instead of minimizing (maximizing) an objective function, we maximize (minimize) another:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{m} y_i b_i \\ \text{s. t.} \quad & C - \sum_{i=1}^{m} y_i A_i \geq 0 \end{aligned}$$

If the optimal value of the primal and the dual problem are the same, we say that *strong dualitity* holds. While this is not the case in general, it usually holds for SDPs.

For SDPs we can use Slaters condition for strong duality, which states that if there is an x^* that is strictly feasible (i.e. all constraints are satisfied and inequalities hold), then the problem is strictly dual [11].
The relaxation of a SDP was first proposed by Goemans and Williamson in [12] as part of an approximation algorithm for the max-cut problem, which is a special case of MaxQP. In max-cut, we are given a graph and are to find a partition of the vertices into two sets, such that the number of edges between the two sets is as large as possible. Goemans and Williamson start by reformulating the problem as an SDP. Given a vertex set $V = \{1, \dots n\}$ and non-negative weights $w_{i,j} = w_{j,i}$, maximize the objective function $\frac{1}{2}\sum_{i<j} w_{i,j}\,(1 - y_i y_j)$ such that $y_i \in S = \{-1, 1\} \quad \forall\ i \in V$. As this is in NP, we need to relax the constraints, which is accomplished by extending the objective function to a larger space, namely $S^n = \{-1, 1\}^n$. We then have to consider vectors v_i and look at the inner product $v_i \cdot v_j$. The algorithm proposed by Goemans and Williamson proceeds by partitioning the vertices of the graph based on randomized rounding. The rounding is based on a random hyperplane cut of the vectors. It has a approximation ratio of 0.878.
The first algorithm for approximating an optimal solution of MaxQP was proposed by Charikar and Wirth and has a $\Omega\left(\frac{1}{\log n}\right)$ approximation ratio [13]. Since the algorithm described in the next chapter uses a similar approach, I will now lay out the ideas of arriving at the classical result. It also uses relaxation of a SDP and randomized rounding, but instead of partitioning based on a random hyperplane cut through the origin, it takes into account the size of the projections of a random vector onto the solution vectors. For this, the relaxed semidefinite program is

$$\begin{aligned} \max \quad & \sum_{ij} a_{ij} v_i \cdot v_j \\ \text{s.t.} \quad & v_i \cdot v_i = 1 \quad \forall\ i \\ & v_i \in \mathbb{R}^n. \end{aligned}$$

Using this the algorithm which can be solved in polynomial time is:

1. Obtain an optimal solution $\{v_i\}$ to the SDP
2. Create vector r in which the r_i are independently distributed over the normal distribution
3. Let $z_i = v_i \cdot r/T$, where $T = \sqrt{4 \log n}$
4. If $|z_i| > 1$ then $y_i = \operatorname{sgn}(z_i)$, otherwise $y_i = z_i$
5. Round the y_i to ± 1

Step 4 truncates any values outside $[-1,+1]$ The last rounding step is based on the size of the $y_i \in [-1,1]^n$:

$$x_i = \begin{cases} -1, & \text{with probability} \frac{1-y_i}{2} \\ +1, & \text{with probability} \frac{1+y_i}{2} \end{cases}$$

The result then clearly lies in $[-1,+1]$. The proof that this algorithm indeed has an approximation ratio of $\Omega(\frac{1}{\log n}$ exploits the spherical symmetry of the distribution of the vector r to show that

$$T^2 \mathbb{E}\left[z_i z_j\right] = v_i \cdot v_j.$$

Therefore, if every $|z_i|$ would be at most one, the approximation ratio would be $\frac{1}{T^2}$. Since this is not the case, we have to proof that the expected value of $\Delta_{ij} = z_i z_j - y_i y_j$ is small in magnitude. In this case

$$|\mathbb{E}(\Delta_{ij})| < 8e^{-\frac{T^2}{2}}.$$

The latter can be used to finally show that the x_i are a good approximation to the y_i. This algorithm also efficiently solves max-cut and can therefore be seen as a generalization of [12]. In the next chapter, we will see the techniques used here to tackle the problem of approximating the maximal value of a traceless 2-local Hamiltonian. Others have studied the approximation algorithms for Hamiltonians which are positive semidefinite instead of traceless [1, 4, 7]. An interesting example of this case is the Heisenberg model, which has already been studied by Bethe in 1931 in [14], where the famous Bethe ansatz is presented. The Heisenberg model is a family of 2-local Hamiltonians of the form

$$H = \sum_{i,j} w_{ij} H_{ij},$$

where

$$H_{ij} = \alpha X_i X_i + \beta Y_i Y_j + \gamma Z_i Z_j.$$

In [1], constant approximation ratios are achieved for special cases of α, β, γ. For $\alpha + \beta + \gamma = 3$, the problem of finding the maximal eigenvalue of an equivalent version of H, namely with $H_{ij} = I - (\alpha X_i X_i + \beta Y_i Y_j + \gamma Z_i Z_j)$, is equivalent to the max-cut problem. Gharibian and Parekh demonstrated a classical approximation algorithm with an approximation ratio of at least 0.498. This year, Anshu *et al.*[7] have shown an efficient classical approximation algorithm with a ratio of at least 0.53 for the same problem, and that for specific instances of the problem there is a shallow quantum circuit that prepares a state with energy larger than the best product state and even its SDP relaxation. The circuit is efficiently computable and shallow means that its depth is independent of input size.

Chapter 4

Approximation algorithms for the groundstate energy of traceless 2-local-Hamiltonians

The algorithm discussed here is able to efficiently find a product state approximation for a general traceless 2-local Hamiltonian with an approximation ratio $\Omega(\frac{1}{\log n})$ [6]. I will explain the algorithm and the central results that enable and validate it.
Before looking at the algorithm itself, there is a preliminary lemma we have to look at. Hamiltonians of the kind $H = H_1 + H_2$ as defined in (2.1) have terms that are linear in Pauli operators. The following lemma will let us reduce this Hamiltonian to a purely quadratic one for the theorems presented in this paper. We form a new $n+1$-qubit Hamiltonian:

$$H' = H_2 + Z_{n+1}H_1.$$

Lemma 1. [6] $\lambda_{max}(H') = \lambda_{max}(H)$. *Moreover, given any* $(n+1)$*-qubit state* ω *we can efficiently compute an* n*-qubit state* ϕ *such that*

$$\langle\phi| H |\phi\rangle \geq \langle\omega| H' |\omega\rangle .$$

If ω *is a tensor product of single qubit stabilizer states then so is* ϕ*.*

The idea is, that for any n-qubit traceless 2-local Hamiltonian with linear terms, there is a purely quadratic $(n+1)$-qubit traceless 2-local Hamiltonian that has the same maximal eigenvalue and has an at best equally good product state approximation. Therefore, the bounds that we prove for quadratic Hamiltonians are valid also for Hamiltonians with linear terms. This enables us to set $H_1 = 0$. To prove this, we first show that all eigenvalues of H' are either eigenvalues of $H_2 - H_1$ or $H_1 + H_2$. This is the case because H' commutes with Z_{n+1} and therefore they share a common set of eigenvectors $|\psi\rangle$:

$$Z_{n+1}H_1 |\psi\rangle = \lambda(Z_{n+1})\lambda(H_1) |\psi\rangle = \pm\lambda(H_1) |\psi\rangle = \lambda(Z_{n+1}H_1) |\psi\rangle .$$

$H_2 - H_1$ can be obtained from $H_1 + H_2$ by the time reversal map:

$$\left(Y^{\otimes n} (H_2 + H_1) Y^{\otimes n}\right)^T = H_2 - H_1.$$

To see why this is true, we need the following properties of the Pauli matrices σ_i:

$$\sigma_a \sigma_b = \delta_{ab} \mathbb{I} + i\epsilon_{abc}\sigma_c$$

and

$$X^T = X, \quad Y^T = -Y, \quad Z^T = Z$$

We can see that the map takes any Pauli operator P_a to $-P_a$. This means it takes H_1 to $-H_1$ and leaves H_2 unchanged since it is quadratic in Pauli operators. Both the matrix transpose and the conjugation by a unitary operator conserve the spectrum, so we have proven that H and H' have the same spectrum and also the same maximal eigenvalue. Using similar arguments, we can then choose the product state $|\phi\rangle$ according to $|\omega\rangle$, such that its eigenvalues will always be at least equal.
The last statement in the lemma refers to an elegant concept that is very useful to quantum error correction. We say an operator A stabilizes a state $|\psi\rangle$ if $A|\psi\rangle = |\psi\rangle$. Conversely, a state is called a stabilizer state of an operator if it is in its $+1$-eigenspace. For practicality, we look at operators from the n-qubit Pauli group. This is favorable because they are unitary and their eigenvalues (± 1) differ significantly from another, such that we can easily perform phase estimations to find out the eigenvalue. If we are given a set of operators $S = \{A, B, C \dots\}$, we know that any errors (which are also from the Pauli group) either commute or anticommute with elements in S. One can correct any error E that anticommutes with S, and in case the error lies in S it is correctable if it commutes with S [15]. We say an operator commutes with a group or is in the normalizer of the group if for some $A, B \in S$: $EA = BE$ with possibly $A \neq B$. This criterion is very easy to check and gives us a useful mathematical toolbox. For choosing the product state $|\phi\rangle$, we use operations on $|\omega\rangle$ that take eigenvectors of matrices from the Pauli group to eigenvectors of matrices from the Pauli group. Therefore, if ω is a tensor product of single qubit stabilizer states, ϕ is, too.
The following Theorem is the main result of the part of the publication described here.

Theorem 1. [6] *There is an efficient classical algorithm which, given H of the form* (2.1)*, outputs a product state $|\phi\rangle = |\phi_1\rangle \otimes \ldots \otimes |\phi_n\rangle$ such that with probability at least $\frac{2}{3}$*

$$\langle\phi| H |\phi\rangle \geq \frac{\lambda_{max}(H)}{O(\log n)}.$$

Moreover, each single-qubit state ϕ_i in an eigenstate of one of the Pauli operators X, Y or Z.

The algorithm largely mirrors the MaxQP algorithm by Charikar and Wirth presented earlier. I will first present the semidefinite program, explain the algorithm and then demonstrate the ideas of proving Theorem 1. In our case, the semidefinite program is:

$$\begin{aligned} \max \quad & Tr\,(CM) \\ \text{s.t.} \quad & M_{i,i} = 1 \\ & M \geq 0 \end{aligned}$$

where M is a hermitian matrix. The ideal solution M is connected to our state in the following way [1]:

$$M_{i,j} = tr\,(\rho P_i P_j) \;\; i,j = 1 \dots 3n.$$

We can set M as a real symmetric matrix without loss of generality, because if the Pauli operators act on different matrices they commute, and therefore the matrix entry is real in this case. If they act on the same qubit, the matrix entry is purely imaginary because the operators anticommute. We can eliminate these terms using $M' = \frac{M+M*}{2}$ because they represent linear terms. This does not change the outcome of the objective function and is therefore fully without loss of generality.
From this perspective the constraints can be understood in the following way:

$$\begin{aligned} & M_{i,i} = tr\,(\rho P_i P_i) = tr\,(\rho) = 1 \quad \text{since} \quad P_i P_i = 1 \\ & M \geq 0 \Leftrightarrow x^T M x = tr(\rho(\sum_i^{3n} x_i P_i)(\sum_j^{3n} x_j P_j)) = tr(\rho X^2) \geq 0 \end{aligned}$$

where $X = \sum_i^{3n} x_i P_i$ and since $X^2, \rho \geq 0$. Since M is real and symmetric, we can express any matrix element as $M_{i,j} = \langle v^i | v^j \rangle$ for some unit vectors $v^1, v^2, \dots, v^{3n+1}$. The vectors have unit norm, since $M_{i,i} = 1$.
After solving the SDP, we choose a random vector $|r\rangle$, the components of which are drawn from the normal distribution and project our solution vectors on it. This is identical to the MaxQP algorithm discussed earlier, except for the constant T which has to be chosen differently. We can build a good geometrical intuition for the truncation step. Since we will project our optimal vectors obtained from the semidefinite program to the random vector $|r\rangle$, we have to make sure that the projections correspond to valid Bloch vector components. We ask the following question: What is the maximal value that a component of a valid Bloch vector can have if all three components have the same value? In other words, if we pick every value without knowing the others, what is the maximal value that we can assign to it such that the vector is part of the unit sphere?

$$\|z\| = \sqrt{z_1^2 + z_2^2 + z_3^2} = \sqrt{3z_1^2} = 1.$$

Therefore the maximal value, above which we should round down is $\frac{1}{\sqrt{3}}$. If we truncate such that no Bloch vector component can be above this value, we will always have a valid state. Visually, this is the same as fitting a cube inside the Bloch sphere and reducing our state space to inside the cube. The projections are $z_i = \left\langle r \middle| v^i \right\rangle / T$ with $T = c\sqrt{\log n}$ and $c = O(\log n)$. This constant will be discussed later. The algorithm looks like this:

1. Solve the relaxed semidefinite program, obtaining an optimal set of vectors v_i
2. Let $|r\rangle$ be a vector of $3n$ independently and identically distributed $N(0,1)$ random variables
3. If $|z_i| > \frac{1}{\sqrt{3}}$, we round down: $y_i = \frac{sgn(z_i)}{\sqrt{3}}$, otherwise $y_i = z_i$

As output we take $\rho = \rho_1 \otimes \ldots \otimes \rho_n$ where:

$$\rho_a = \frac{1}{2}\left(\mathbb{I} + y_{3a-2}P_{3a-2} + y_{3a-1}P_{3a-1} + y_{3a}P_{3a}\right)..$$

We want to show that this algorithm fulfills the approximation ratio $\Omega(\frac{1}{\log n})$ with probability $\frac{2}{3}$.
First we demonstrate that the approximation ratio is bounded below by a constant [16]:

Theorem 2. *Suppose H is traceless 2-local Hamiltonian. Then*

$$\lambda_{sep}(H) \geq \frac{1}{9}\lambda_{max}(H).$$

Where we call λ_{sep} the highest eigenvalue achievable by a product state:

$$\lambda_{sep}(H) = \max_{\phi_1,\ldots,\phi_n} \langle \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_n | H | \phi_1 \otimes \phi_2 \otimes \ldots \otimes \phi_n \rangle .$$

We can prove this using entanglement-breaking depolarizing channels. Depolarizing channels are a simple model for noise in quantum information theory [17]. They are implemented by a map Δ_λ, which maps a state ρ onto a linear combination of itself and the identity matrix:[18]

$$\Delta_\lambda(\rho) = \lambda\rho + \frac{1-\lambda}{d}\mathbb{I},$$

where d is the dimension of state. The parameter λ must satisfy

$$-\frac{1}{d^2-1} \leq \lambda \leq 1.$$

This channel maps pure states to mixed states. For $\lambda = 0$ we get the maximally noisy channel, for $\lambda = 1$ the identity. In our case we look at entanglement breaking channels. These are channels for which the output state is always separable, i.e. if any entangled density matrix is mapped to a separable one.[19] Here, the relevant map $\mathcal{E}_\delta$ is defined by its action on the Pauli group:

$$\mathcal{E}_\delta(I) = I \quad \mathcal{E}_\delta(P) = \delta P \quad P \in \{X, Y, Z\}.$$

Therefore, the action on a qubit state in Bloch representation is:

$$\mathcal{E}_\delta(\rho) = \frac{1}{2}\mathbb{I} + \delta \sum_{i=1}^{3} \tau_i P_i.$$

Geometrically, this reduces the length of any Bloch vector by a factor δ. Generally, a CPT map Φ can be written as $\Phi(\rho) = \frac{1}{2}(\mathbb{I} + (\boldsymbol{t} + T\tau)P_i)$ where $\boldsymbol{t}$ is a vector and T a matrix.[20] We can write this as $\boldsymbol{T} = \begin{pmatrix} 1 & 0 \\ \boldsymbol{t} & T \end{pmatrix}$, where we can assume without loss of generality that T is diagonal, which follows directly from the Kraus representation Theorem. $\boldsymbol{T}$ has then the canonical form

$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ t_1 & \lambda_1 & 0 & 0 \\ t_2 & 0 & \lambda_2 & 0 \\ t_3 & 0 & 0 & \lambda_3 \end{pmatrix}.$$

If $\boldsymbol{t} = 0$, this channel is unital. Unital qubit channels are entanglement breaking if and only if $\sum_j |\lambda_j| \leq 1$ (after T was diagonalized).[20] This implies that qubit channels of the form we look at in the paper are entanglement breaking for $\delta \leq \frac{1}{3}$ as we have $\lambda_1 = \lambda_2 = \lambda_3 = \delta$. Using our definition of $\mathcal{E}$, we see

$$Tr\left(\sigma P_{j_1} P_{j_2} \dots P_{j_L}\right) = \frac{1}{3^L} Tr\left(\rho P_{j_1} P_{j_2} \dots P_{j_L}\right)$$

Where $\sigma = \mathcal{E}_{\frac{1}{3}}^{\otimes n}(\rho)$. With this, we can prove the Theorem due to Lieb.
We consider the n-qubit state ψ satisfying $\langle\psi| H |\psi\rangle = \lambda_{max}(H)$. With the identity shown above, the depolarized state

$$\sigma = \mathcal{E}_{\frac{1}{3}}^{\otimes n}\left(|\psi\rangle\langle\psi|\right)$$

is separable and

$$\lambda_{sep}(H) \geq Tr(\sigma H) = \frac{1}{9}\langle\psi| H |\psi\rangle$$

which is the wanted statement.
It is then shown that the y_i are a good approximation to the z_i, i.e. that the expectation value of $\Delta_{ij} = z_i z_j - y_i y_j$ is sufficiently small. Specifically,

$$\mathbb{E}_r \, |\Delta_{i,j}| \leq e^{-\Omega(T^2)} \quad 0 \leq i < j \leq 3n.$$

Using this, one can show that the approximation ration holds. To find out more about the constant T, we can look at a recent publication by Harrow and Montanaro [5]. An algorithm is presented which gives an product state approximation ratio to traceless k-local Hamiltonians with respect to the 1-norm of the coefficients of the Hamiltonian, $C_{i,j}$ and D_j in our case. This result uses a different notion of approximation ratio but is concerned with the same kind of Hamiltonians. We use the following equation which is proved in the paper:

$$\sum_{i,j=1}^{3n} |C_{i,j}| \leq Kn\lambda_{max}(H).$$

$K > 0$ is an absolute constant, that we can use to show that the approximation ratio $\Omega(\frac{1}{\log n})$ holds if we choose T and c as described. For traceless 2-local Hamiltonians, Harrow and Montanaro show that

$$\lambda_{min}(H) \leq -\|\hat{H}\|_1/(24l)$$

where l is the maximal number of terms that each qubit participates in and $\|\hat{H}\|_1$ the 1-norm of the coefficients.

Chapter 5

Testing specific models

I will now discuss the results of my implementation of the algorithm and the implemented Hamiltonians. The code is written in Python 3.8, while for the semidefinite program the interface PICOS is used. All of it can be found in appendix A. To produce informative plots, the implementation executes the algorithm with output y and computes the ratio $r = y^T Cy/\lambda_{max}$. It does this o times per number of qubits n. The average of the o ratios per n is plotted, where a logarithmic scale is used for the x-axis. The variance has been visualized as error bars. I have chosen to plot in the range of $n = 4$ to $n = 5200$ in 25 equidistant steps on the logarithmic scale, using $o = 20$. The computation time became infeasible for larger qubit numbers, as the complexity of the function *numpy.linalg.eig()*, used to to find the solution vectors, is $O\left(N^3\right)$.
Secondly, I have done the same computation but instead of taking the average of the lists, I have picked the maximal approximation ratio and plotted it. While this does not specifically test the code for the behaviour of the approximation ratio, it provides information on the practical uses of the algorithm. The Ising model with a transverse field has physical relevance because it can be used to study quantum phase transitions of ferroelectrics with a tunneling effect or systems of interacting magnetic spins with an outer field. The Hamiltonian

$$H = \alpha \sum_i Z_i + \beta \sum_i X_i X_{i+1}$$

has been exactly solved in one dimension in [21]. One way to do this by transforming the Hamiltonian into a quadratic form of Fermi operators via the Jordan-Wigner transformation [22]. To diagonalize this Hamiltonian, one can use the Bogoliubov transformation which is an isomorphism in the canonical commutation or anticommutation relation algebra [23].
The parameter α corresponds to the tunneling energy and β to the nearest neighbour interaction in the ferroelectric problem. We here look at a closed chain, meaning $1 \leq i \leq n, \quad X_{n+1} = X_1$. I have first implemented the model with $\alpha = 0, \quad \beta = 1$. This is a model of a one-dimensional chain with nearest neighbor interaction without an outer field. The state that is stabilized by all terms simultaneously is the state

achieving the maximal eigenvalue. In this case, it is the n-fold tensor product of the $+1$-eigenvector of X:

$$|\psi\rangle = |+\rangle_1 \otimes |+\rangle \otimes \ldots \otimes |+\rangle_n \,,$$

where

$$|+\rangle_i = \frac{|0\rangle_i + |1\rangle_i}{\sqrt{2}}.$$

The maximal eigenvalue of the Hamiltonian is therefore $\lambda_{max} = n$, since it has n terms. Figure 5.1 shows the result of computing numeric values of the approximation ratio over a range of n qubit Hamiltonians of this kind.

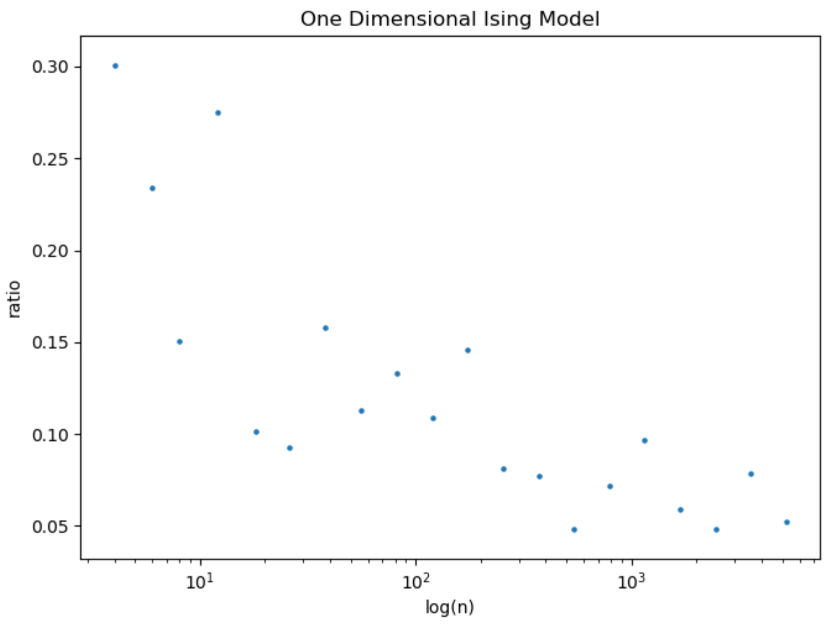

Figure 5.1: The average of 20 approximation ratios per n, plotted from 4 to 5200 qubits with logarithmic scale on the x-axis for the one dimensional Ising model with no transverse field.

While we can only be sure for significantly larger qubit numbers, it seems to exhibit the expected behaviour of $\Omega(\frac{1}{\log n})$.
I have also implemented the transverse field Ising model for non-zero α and arbitrary β. To compute the maximal eigenvalue, I have followed [21]. Since the Hamiltonian has terms that are linear in Pauli operators, we have to transform it as discussed in

chapter 4. The Hamiltonian that was implemented is

$$H = H_2 + Z_{n+1}H_1 = \alpha Z_{n+1}\sum_i Z_i + \beta \sum_i X_i X_{i+1}.$$

Fig. 5.2 shows a result of implementing this model for several ratios of α and β. The plot shows a similar behaviour to figure 5.1.

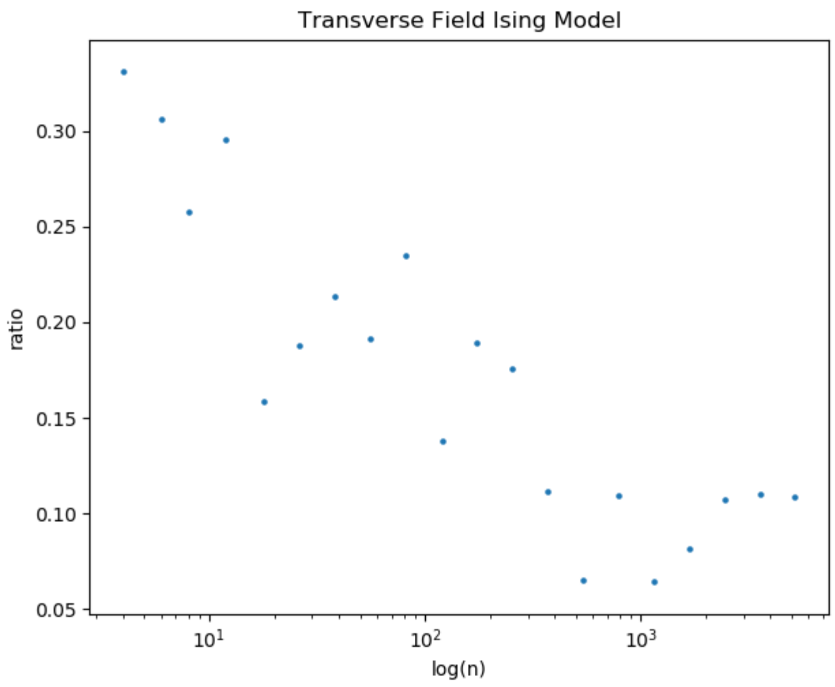

Figure 5.2: The average of 20 approximation ratios per n, plotted from 4 to 5200 qubits with logarithmic scale on the x-axis for the transverse field Ising model. For testing $\alpha = 2$ and $\beta = 3$ is used.

The second model implemented is

$$H = X_1X_2 + Z_1Z_2 + X_3 + X_4 + X_5X_6 + Z_5Z_6 + X_7 + X_8 \dots,$$

where we restrict n to be a multiple of 4. In contrast to the Ising model without a transverse field, the maximal eigenvalue can not be achieved by a product state in this case. To find out the maximal eigenvalue lets look at the first two quadratic terms:

$$H_2 = X_1X_2 + Z_1Z_2.$$

The state achieving the maximal eigenvalue $\lambda_{max}(H_2) = 2$ is the EPR-state $|\text{EPR}\rangle = \frac{\langle 00| + \langle 11|}{\sqrt{2}}$. This is a maximally entangled state. To find out the product state which approximates this best, maximize the overlap with a general product state:

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle = (a_1 |0\rangle + b_1 |1\rangle) \otimes (a_2 |0\rangle + b_2 |1\rangle)$$

with $a_1^2 + b_1^2 = a_2 + b_2^2 = 1$. The overlap is then:

$$\max_{\psi_1, \psi_2} \left(|\langle \text{EPR}| \psi\rangle|^2 \right) = \max \left(\left| \frac{1}{\sqrt{2}} (a_1 a_2 + b_1 b_2) \right|^2 \right) = \frac{1}{2},$$

with either $a_1 = a_2 = 1$ and $b_1 = b_2 = 0$ or $b_1 = b_2 = 1$ and $a_1 = a_2 = 0$. Therefore, the product states with the maximal overlap are $|00\rangle$ and $|11\rangle$ with maximal eigenvalue $\lambda_{sep}(H_2) = 1$, the approximation ratio being $\frac{\lambda_{sep} H(2)}{\lambda_{max}(H_2)} = 0.5$.
In general the maximal eigenvalue of the Hamiltonian is therefore $\lambda_{max}(H) = n$, and the best achievable eigenvalue by a product state is $\lambda_{sep}(H) = \frac{3}{4}n$. Fig. 5.3 seems to show the expected $\Omega\left(\frac{1}{\log n}\right)$ approximation ratio:

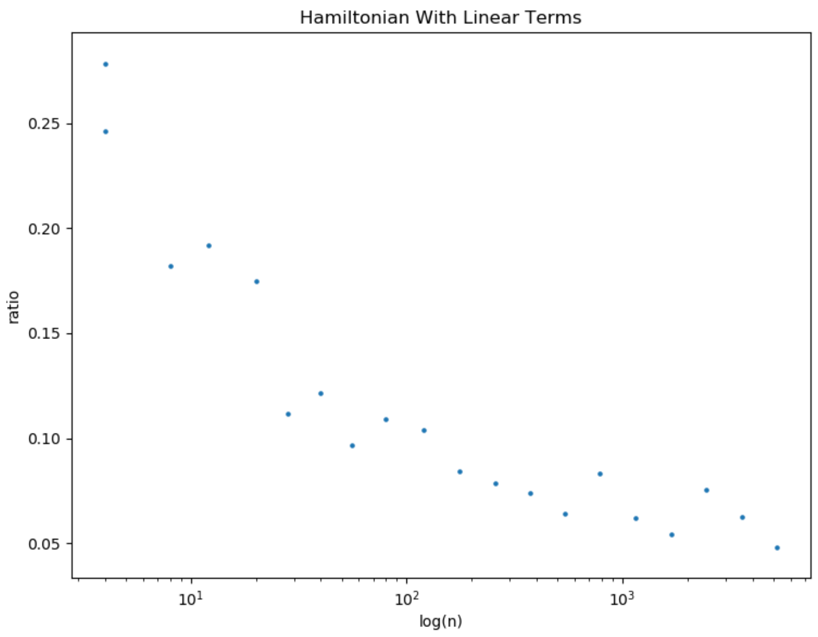

Figure 5.3: The average of 20 approximation ratios per n, plotted from 4 to 5200 qubits with logarithmic scale on the x-axis for a chain with linear terms.

Since the functionality and efficiency of this algorithm has been proved, the purpose of the implementation of the two models were of course mainly for testing the code itself. An open questions would be if it is possible to make the implementation more efficient, to achieve feasible computation for larger qubit numbers, to then test if the code achieves the proved results.

Chapter 6

Generalization to qutrits

Most quantum processors are currently based on qubits. Quantum computing based on 3-level systems is known to offer many advantages over qubit-based quantum computing. As an example, error correction can be done more efficiently [24] and quantum cryptography is more robust [25]. A qutrit processor has also recently been used to study thermalisation and loss of information in closed quantum systems [26]. In this chapter, I will introduce the reader to the basic concepts of 3-level systems and discuss the challenge of generalizing the algorithm discussed in chapter 4 to qutrit systems.
First, we will look at the generalization of the Bloch representation to d-level systems. We represent a state ρ with the help of a d^2-1-dimensional Bloch vector $\boldsymbol{\tau}$. We call the space of states that fulfill the conditions presented in chapter 1, the Bloch space. A state ρ can be represented in the following way:

$$\rho = \frac{1}{d}\mathbb{I} + \sum_{i=1}^{d^2-1} \tau_i \sigma_i$$

where σ_i are generators of $SU(d)$ obeying

$$\sigma_i \sigma_j = \frac{2}{d}\delta_{ij} + d_{ijk}\sigma_k + i f_{ijk}\sigma_k \tag{6.1}$$

The f_{ijk} and d_{ijk} are the structure constants of the Lie-Algebra. f_{ijk} is totally antisymmetric and equals the Levi-Civita-Symbol for $d=2$, d_{ijk} is totally symmetric and vanishing for $d=2$. We can construct the generators as follows:[27]

$$\{\sigma_i\}_{i=1}^{d^2-1} = \{u_{jk}, v_{jk}, w_l\}$$

where

$$u_{jk} = |k\rangle\langle k| + |k\rangle\langle j|\,,\ v_{jk} = -i(|j\rangle\langle k| - |k\rangle\langle j|),$$

$$w_l = \sqrt{\frac{2}{l(l+1}} \sum_{j=1}^{l} \left(|j\rangle\langle j| - l\,|l+1\rangle\langle l+1|\right),$$

$$1 \leq j \leq k \leq d, 1 \leq l \leq d-1.$$

The τ_i are the components of the Bloch vector and are the expectation values of the σ_i:

$$\tau_i = Tr(\rho\sigma_i)$$

The center of the Bloch space is the maximally mixed state $\rho_* = \frac{1}{d}\mathbb{I}$. For $d \geq 3$, there exist Bloch vectors with $|\tau| \leq 1$ which do not correspond to a positive semi-definite matrix. The space spanned by the Bloch-vectors is therefore not a solid ball with radius 1. We can deduce using (6.1) the following conditions that we have to put on the Bloch vector for the density matrix to describe a pure state [28]:

$$\rho = \rho^2 \Leftrightarrow \begin{cases} \tau^2 = \frac{d-1}{2d} \\ d_{ijk}\tau_j\tau_k = \frac{d-2}{d}\tau_i \end{cases} .$$

The fist condition implies the Bloch vector of a pure state being confined to a d^2-1 dimensional outsphere. The second condition says that the vectors of the pure states are a well defined subset of the surface of the outsphere. The generators of $SU(3)$ are the Gell-Mann-matrices λ_i:

$$\lambda_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \lambda_2 = \begin{pmatrix} 0 & -i & 0 \\ i & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \lambda_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

$$\lambda_4 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \lambda_5 = \begin{pmatrix} 0 & 0 & -i \\ 0 & 0 & 0 \\ i & 0 & 0 \end{pmatrix}, \lambda_6 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix},$$

$$\lambda_7 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -i \\ 0 & i & 0 \end{pmatrix}, \lambda_8 = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -2 \end{pmatrix}.$$

They are traceless, hermitian, and obey $Tr\left(\lambda_i\lambda_j\right) = \delta_{ij}$. The geometry of the Bloch space for qutrits has been studied by many authors [27, 29, 30] and many properties, such as sections in lower dimensions, have been found. The problem with generalizing the algorithm shown by Bravyi *et al.* to qutrits is that we can not directly map the solution states of the SDP to the Bloch space in the same way, since it is not a solid ball with radius one. We therefore have to find a rounding scheme that takes into account the geometric restrictions of the Bloch space for three level systems. An elgant and simple method to reduce the space to a solid ball in the Bloch space. As proved in [30], the boundary of the Bloch space of qutrits can never stray into the solid ball of radius $\frac{1}{2}$, meaning all bloch vectors with $|\tau| \leq \frac{1}{2}$ correspond to valid states. This is proved by using that boundary vectors $\boldsymbol{\tau} \in \mathbb{R}^8$ should necessarily satisfy

$$det\rho(\boldsymbol{\tau}) = 0.$$

We can therefore apply the same rounding scheme as in the qubit case, modified for the smaller sphere. The cut-off value therefore becomes $\frac{1}{2\sqrt{8}}$. Visually, we fit a hypercube into the 8-dimensional sphere of radius $\frac{1}{2}$ and round any states into the cube. As a field study I have implemented this algorithm and plotted the result, mirroring the procedure described in chapter 4. The model used is a one dimensional chain of the form:

$$H = \sum_{i=1}^{8n} \lambda_1^i \lambda_1^{i+1}.$$

The result can be seen in Figure 6.1

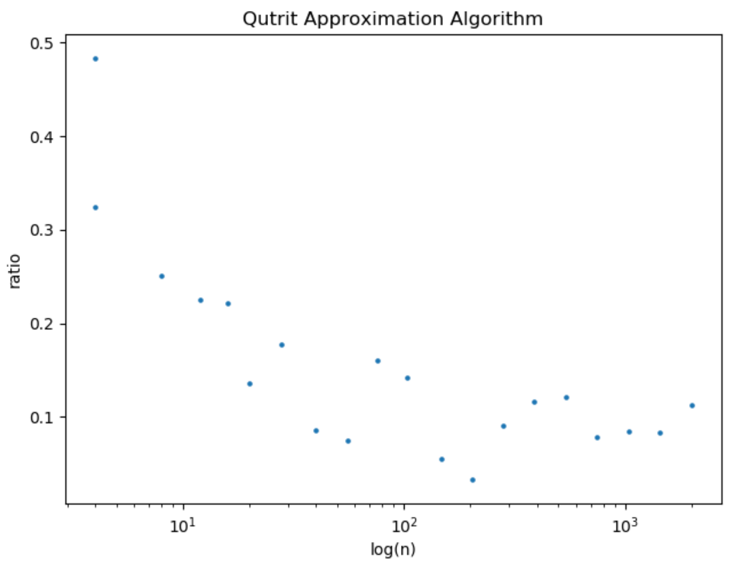

Figure 6.1: The result of the simplistically adapted algorithm on qutrits.

A better rounding scheme would include using the unique geometry of the Bloch space to make the cut-off smaller.

Appendix A

Implementation of the discussed methods

The following code is the implementation of the algorithm presented in [6]:

```
import numpy as np
import math
# for a 2-local Hamiltonian on qubits returns bloch vector
    upon input of C and c
def rund(v, c, C):
    N=C.size[0]
    n=N/3
    T = c*math.sqrt(math.log(n))
    rd = {}
    y = []
    r = np.random.normal(0,1,N)
    for i in range(0,N):
        z = np.inner(r, v[i])/T
        if np.linalg.norm(z) > 1/math.sqrt(3): y.append(np.
            sign(z)/math.sqrt(3))
        else: y.append(z)
    rd["blochvec"] = y
    return rd
```

This code uses the interface picos to compute the solution of the SDP:

```
import picos as pic
import cvxopt as cvx
import cvxopt.lapack
import numpy as np

def mc(C):
    N = C.size[0]
    maxcut = pic.Problem()

    # Add the hermitian matrix variable.
    X=maxcut.add_variable('X', (N,N), 'hermitian')

    # Constrain X to have ones on the diagonal.
    maxcut.add_constraint(pic.maindiag(X) == 1)

    # Constrain X to be positive semidefinite.
    maxcut.add_constraint(X >> 0)

    # Set the objective.
    maxcut.set_objective('max', C|X)


    # Solve the problem.
    maxcut.solve(solver='cvxopt')
    return X.value
```

The code for building the C and M for specific qubit numbers:

```
import numpy as np
from scipy.linalg import block_diag

# Assumes that n is a multiple of 4 it returns C for H' the
    form X1X2+Z1Z2#Z+X5X6+Z5Z6... upon input of the number of
    qubits
def buildC(n):
    o1 = np.zeros((3,3))
    a = np.array([[1,0,0], [0,0,0], [0,0,1]])
    b = np.block([[o1, a], [a, o1]])
    o2 = np.zeros((6,6))
    d = np.block([[b, o2], [o2, o2]])
    s = []
    for i in range(1,int(n/4)+1):
        s.append(d)
    s.append(o1)
    A = np.array(block_diag(*s))
    for i in range(0,int(n/4)):
        A[3*n+2][6+i*12] = 1
        A[6+i*12][3*n+2] = 1
```

```
            A[3*n+2][9+i*12] = 1
            A[9+i*12][3*n+2] = 1
    return A

# Returns the solution of the SDP for buildC(n)
def sdp1(n):
    A = np.add(buildC(n),np.eye(3*n+3))
    if int(n/4)>1:k=1
    else: k=0
    for i in range(0,int(n/4)):
        A[6+12*i][9+i*12] = 1
        A[9+i*12][6+12*i] = 1
        for j in range(0,int(n/4)-i-1):
            A[6+i*12][9+i*12+k*9+12*j] = 1
            A[9+i*12+k*9+12*j][6+i*12] = 1
            A[9+i*12][9+i*12+k*9+12*j] = 1
            A[9+i*12+k*9+12*j][9+i*12] = 1
            A[6+i*12][12+i*12+k*9+12*j] = 1
            A[12+i*12+k*9+12*j][6+i*12] = 1
            A[9+i*12][12+i*12+k*9+12*j] = 1
            A[12+i*12+k*9+12*j][9+i*12] = 1
    return A

# Returns the C for a chain of spins
def chainC(n):
    a = []
    a.extend([1,0,0] for i in range(n-1))
    C = np.diagflat(a,3) + np.diagflat(a,-3)
    C[0][3*n-3]=1
    C[3*n-3][0]=1
    return C

# Returns the sdp solution for chainC(n)
def sdp2(n):
    A = np.eye(3*n)
    for i in range(0,n):
        for j in range(0,n):
            A[i*3][j*3] = 1
    return A

# Returns C for the transverse field Ising Model
```

```
def tfiC(n,a,b):
    x = []
    x.extend([1,0,0] for i in range(n-1))
    x.append([0,0,0])
    C = -1*a*(np.diagflat(x,3) + np.diagflat(x,-3))
    C[3*n-3][0]=-1*a
    C[0][3*n-3]=-1*a
    for i in range(2,3*n,3):
        C[3*n+2][i]=-1*b
        C[i][3*n+2]=-1*b
    return C

# Returns the SDP solution for tfiC(n)
def tfisdp(n):
    a = np.eye(3*n+3)
    b,c=[],[]
    for i in range(int(n/2)):
        b.extend([1,0,0,-1,0,0])
        c.extend([-1,0,0,1,0,0])
    b.extend([0,0,0])
    c.extend([0,0,0])
    np.array(b).reshape(-1)
    np.array(c).reshape(-1)
    for i in range(0,3*n-5,6):
        a[i]=b
    for i in range(3,3*n-2,6):
        a[i]=c
    f=[]
    for i in range(n):
        f.extend([0,0,1])
    f.extend([0,0,0])
    np.array(f).reshape(-1)
    for i in range(2,3*n,3):
        a[i]=f
    for i in range(2,3*n,3):
        a[3*n+2][i]=-1
        a[i][3*n+2]=-1
    return a
```

This code is used to compute the optimal solution vectors $|v_i\rangle$ from the solution M of the SDP:

```
import numpy as np
import scipy.sparse.linalg


def findGenVecGram(gram):
    """Finds upon input of the Gram matrix gram a collection
         of vectors
    {v_1,...,v_n} such that gram_{ij} = <v_i, v_j>."""

    # Calculate the Decomposition such that gram = L*L^T
    L = choleskyPSDSym(gram)

    # Take the columns of L and write them into a list
    result = []
    for i in range(len(gram)):
        result += [L[:,i]]
    return result



def calcGram(v):
    """Returns upon input of the collection of vectors v = {
       v_1, ..., v_n}
    the associated Gram matrix gram, i.e., the matrix gram
       such that
    gram_{ij} = <v_i, v_j>."""
    n = len(v)
    result = np.zeros((n, n), dtype=float)
    for i in range(n):
        for j in range(n):
            result[i][j] = np.inner(v[i], v[j])
    return result



def choleskyPSDSym(A):
    """Returns a matrix L such that L*L^T = A, where A is a
       positive
    semidefinite symmetric matrix"""
    # Calculate spectral decomposition (eigenvalues +
        eigenvectors) of A
    w, v = np.linalg.eig(A)
    for i in range(len(w)):
            if w[i]<0:
                if abs(w[i])<10e-8: w[i]=0

    # Return the orthogonal matrix v multiplied with the
        square root of
    # the diagonal matrix whose diagonal consists of the
        eigenvalues
    return np.transpose(np.matmul(v, np.sqrt(np.diag(w))))
```

This code is used to compute ratios and make subsequent plots:

```
import numpy as np
import picos as pic
import cvxopt as cvx
import cvxopt.lapack
import matplotlib.pyplot as plt
import scipy
from scipy import special
import math
import matplotlib.scale as scale
import statistics

from timeit import default_timer as timer
from hamiltonians import *
from sdp import *
from gram import *
from algo import *
from operations import *

# returns 1. the solution to the SDP 2. the gram vectors 3.
    dictionary with the state and bloch vector upon input of
    C and c(O(1)).
def solve(C, c):
    V = np.array(mc(C))
    K = (np.conj(V)+V)/2
    v = findGenVecGram(K)
    sol = rund(v,c,C)
    print(sol)
    return sol

# returns upon input of C, y and the maximal eigenvalue
    corresponding to C the ratio.
def ratio(C, y, maxeig):
    return np.inner(np.dot(y,C),np.transpose(y))/maxeig

# returns list of ratios upon input of number of qubits, c
    and number of iterations o, for buildC(n).
def sample(n, c, o, C):
    print('finding␣gram␣vectors...')
    start = timer()
    v = findGenVecGram(sdp1(n))
    end = timer()
    print('elapsed␣time:',end-start)
    ratlist=[]
    i=0
    while i < o:
        ratlist.append(ratio(C, rund(v,c,cvx.matrix(C))["
            blochvec"], n))
        print(100*i/25,'%')
        i=i+1
    return ratlist

def chainsample(n, c, o, C):
    v = findGenVecGram(sdp2(n))
    ratlist=[]
    i=0
    while i < o:
        ratlist.append(ratio(C, rund(v,c,cvx.matrix(C))["
            blochvec"], n))
        i=i+1
    return ratlist

# returns the highest eigenvalue of the n-qubit productstate
     after performing the rounding o times
def maxsample(n, c, o, C):
    v = findGenVecGram(sdp1(n))
    ratlist=[]
    i=0
    while i < o:
        ratlist.append(ratio(C, rund(v,c,cvx.matrix(C))["
           blochvec"], n))
        i=i+1
    return max(ratlist)

def maxchainsample(n,c,o,C):
    v = findGenVecGram(sdp2(n))
    ratlist=[]
    i=0
    while i < o:
        ratlist.append(ratio(C, rund(v,c,cvx.matrix(C))["
           blochvec"], n))
        i=i+1
    return max(ratlist)

# Returns the average of the list.
def avg(l):
    return sum(l)/len(l)

def forward(x):
    return x**2
def inverse(x):
    return x**(1/2)

# For the buildC(n) returns a plot of samples in a range of
   qubitnumbers upon input of start and end of range, c and
   the number of rounding attemps per qubitnumber
def plotn(ni, nf, c, o, s):
    x=[]
    y=[]
    for i in range(ni,nf+1,s):
        x.extend([i for j in range(o)])
        y=y+sample(i,c,o,buildC(i))
    plt.scatter(x,y,marker=".",s=3)
    plt.xlabel('n')
    plt.ylabel('ratio')
    plt.title('nplot')
    plt.show()
```

```
# Does the same as plotn but returns the averages over the
    lists per n, and the steps are choosable for efficiency
def avgplot(ni,nf,c,o,s):
    y=[]
    x=[]
    var=[]
    a = np.exp(np.linspace(np.log(ni), np.log(nf), s))
    for i in a:
        i = int(4*round(i/4.))
        x.append(i)
    for i in range(len(x)):
        print(x[i])
        list=np.real(sample(x[i],c,o,buildC(x[i])))
        y.append(avg(list))
        var.append(statistics.variance(list))
    plt.errorbar(x,y,yerr=var,fmt='o',elinewidth=1,capsize
        =3,lw=0)
    plt.xlabel('n')
    plt.ylabel('ratio')
    plt.xscale('log')
    plt.title('Hamiltonian␣With␣Linear␣Terms')
    plt.show()

def maxplot(ni,nf,c,o,s):
    y=[]
    x=[]
    a = np.exp(np.linspace(np.log(ni), np.log(nf), s))
    for i in a:
        i = int(4*round(i/4.))
        x.append(i)
    for i in range(len(x)):
        print(x[i])
        y.append(maxsample(x[i],c,o,buildC(x[i])))
    plt.scatter(x,y,marker="o",s=5)
    plt.xlabel('log(n)')
    plt.ylabel('ratio')
    plt.xscale('log')
    plt.title('Hamiltonian␣With␣Linear␣Terms')
    plt.show()
```

```
# For the buildC(n) returns a plot of samples for a range of
     c's to hopefully find out something about what the
    optimal c is. Not used.
def plotc(ci, cf, n, o):
    C = buildC(n)
    x=[]
    y=[]
    for i in range(ci,cf+1):
        x.extend([i for j in range(o)])
        y=y+sample(n,i,o,C)
    plt.scatter(x,y,marker=".",s=3)
    plt.xlabel('c')
    plt.ylabel('ratio')
    plt.title('cplot')
    plt.show()

# For the chainC(n) returns a plot of samples in a range of
    qubitnumbers upon input of start and end of range, c and
    the number of rounding attemps per qubitnumber.
def chainplot(ni, nf, c, o,s):
    x=[]
    y=[]
    var=[]
    for i in range(ni,nf+1):
        x.append(i)
        y=y+avg(chainsample(i,c,o,chainC(i)))
    plt.scatter(x,y,marker=".",s=3)
    plt.xlabel('n')
    plt.ylabel('log(ratio)')
    plt.yscale('log')
    plt.title('nplot')
    plt.show()


# For the chainbuildC(n) returns a plot of samples for a
    range of c's to hopefully find out something about what
    the optimal c is.
def chainplotc(ci, cf, n, o):
    C = chainC(n)
    x=[]
    y=[]
    for i in range(ci,cf+1):
        x.extend([i for j in range(o)])
    for i in range(ci,cf+1):
        y=y+chainsample(n,i,o,C)
    plt.scatter(x,y,marker=".",s=3)
    plt.xlabel('c')
    plt.ylabel('ratio')
    plt.title('cplot')
    plt.show()

# Plots the averages of the lists of ratios for chainC(n).
def avgchainplot(ni,nf,c,o,s):
    x=[]
    y=[]
    var=[]
    a = np.exp(np.linspace(np.log(ni), np.log(nf), s))
    for i in a:
        i = int(round(i))
        x.append(i)
    for i in range(len(x)):
        print(x[i])
        list=np.real(chainsample(x[i],c,o,chainC(x[i])))
        y.append(avg(list))
        var.append(statistics.variance(list))
    plt.errorbar(x,y,yerr=var,fmt='o',elinewidth=1,capsize
        =3,lw=0)
    plt.xlabel('n')
    plt.ylabel('ratio')
    plt.xscale('log')
    plt.title('One␣Dimensional␣Ising␣Model')
    plt.show()

def chainmaxplot(ni,nf,c,o,s):
    y=[]
    x=[]
    a = np.exp(np.linspace(np.log(ni), np.log(nf), s))
    for i in a:
        i = int(4*round(i/4.))
        x.append(i)
    for i in range(len(x)):
        y.append(avg(maxsample(x[i],c,o,buildC(x[i]))))
        print(x[i])
    plt.scatter(x,y,marker="o",s=5)
    plt.xlabel('log(n)')
    plt.ylabel('ratio')
    plt.xscale('log')
    plt.title('s')
    plt.show()
# Returns the maximal eigenvalue of the transverse field
   Ising model upon input of a, b, n and a constant d.
def tfimaxeig(n,a,b,d):
    x = a+b/2+(2*a*n/math.pi)*(1+b/(2*a))*d
    return x

# Returns list of ratios for the transverse field Ising
   model.
def tfisample(n, c, o, C,a,b,d):
    print('finding␣gram␣vector...')
    start = timer()
    v = findGenVecGram(tfisdp(n))
    end = timer()
    print('elapsed␣time:', end-start)
    ratlist=[]
    i=0
    while i < o:
        print(100*i/25,'%')
        ratlist.append(ratio(C, rund(v,c,cvx.matrix(C))["
           blochvec"], tfimaxeig(n,a,b,d)))
        i=i+1
    return ratlist

def tfimaxsample(n,a,b,d):
    v = findGenVecGram(tfisdp(n))
    ratlist=[]
    i=0
    while i < o:
        ratlist.append(ratio(C, rund(v,c,cvx.matrix(C))["
           blochvec"], tfimaxeig(n,a,b,d)))
        i=i+1
    return np.max(ratlist)
```

```
# Plots the average of the list of ratios for the transverse
    field Ising model.
#def tfiplot(ni,nf,c,iterations,steps,a,b):
#     l = math.sqrt((2*b/a)/((1+b/2*a)**2))
#     d=scipy.special.ellipeinc(math.pi/2,l)
#     x=[]
#     y=[]
#     var=[]
#     a = np.exp(np.linspace(np.log(ni), np.log(nf), steps))
#     for i in a:
#         i = int(2*round(i/2.))
#         x.append(i)
#     for i in range(len(x)):
#         print(x[i])
#         list=tfisample(x[i],c,iterations,tfiC(x[i],a,b),a,b
    ,d)
#         y.append(avg(list))
#         var.append(statistics.variance(list))
#     plt.errorbar(x,y,yerr=var,fmt='o',elinewidth=1,capsize
    =3,lw=0)
#     plt.xlabel('n')
#     plt.ylabel('ratio')
#     plt.xscale('log')
#     plt.title('Transverse Field Ising Model')
#     plt.show()

def tfiplot(ni,nf,c,iterations,steps,a,b):
    l = math.sqrt((2*b/a)/((1+b/2*a)**2))
    d=scipy.special.ellipeinc(math.pi/2,l)
    x=[]
    y=[]
    var=[]
    k = np.exp(np.linspace(np.log(ni), np.log(nf), steps))
    for i in k:
        i = int(2*round(i/2.))
        x.append(i)
    for i in range(len(x)):
        print(x[i])
```

```
            alist=np.real(tfisample(x[i],c,iterations,tfiC(x[i],
                a,b),a,b,d))
            y.append(avg(alist))
            var.append(statistics.variance(alist))
        plt.errorbar(x,y,yerr=var,fmt='o',elinewidth=1,capsize
            =3,lw=0)
        plt.xlabel('n')
        plt.ylabel('ratio')
        plt.xscale('log')
        plt.title('Transverse␣Field␣Isisng␣Model')
        plt.show()

def tfimaxplot(ni,nf,c,iterations,steps,a,b):
        l = math.sqrt((2*b/a)/((1+b/2*a)**2))
        d=scipy.special.ellipeinc(math.pi/2,l)
        x=[]
        y=[]
        k = np.exp(np.linspace(np.log(ni), np.log(nf), steps))
        for i in k:
            i = int(2*round(i/2.))
            x.append(i)
        for i in range(len(x)):
            print(x[i])
            alist=np.real(tfisample(x[i],c,iterations,tfiC(x[i],
                a,b),a,b,d))
            y.append(avg(alist))
        plt.scatter(x,y,marker="o",s=5)
        plt.xlabel('n')
        plt.ylabel('ratio')
        plt.xscale('log')
        plt.title('Transverse␣Field␣Isisng␣Model')
        plt.show()
```

The following code is the algorithm and code for plots for the algorithm on qutrits:

```
import numpy as np
import math
from whole import *

# For a traceless 2-local hamiltonian on qutrits returns
```

```
    bloch vector  upon input of the solution vectors , C and c
    .
def trund(v, c, C):
    N=C.size[0]
    n=N/8
    T = c*math.sqrt(math.log(n))
    rd = {}
    y = []
    r = np.random.normal(0,1,N)
    for i in range(0,N):
        z = np.inner(r, v[i])/T
        if np.linalg.norm(z) > 1/2*math.sqrt(8): y.append(np
            .sign(z)/2*math.sqrt(8))
        else: y.append(z)
    rd["blochvec"] = y
    return rd

# Builds the C for a simple closed chain of the form H_i =
    gm[1]_igm[1]_(i+1) where gm[i] is the ith Gell-Mann
    matrix.
def tC(n):
    a = []
    a.extend([1,0,0,0,0,0,0,0] for i in range(n-1))
    C = np.diagflat(a,8) + np.diagflat(a,-8)
    C[0][8*n-8]=1
    C[8*n-8][0]=1
    return C

# Returns the SDP solution for aforementioned C.
def tsdp(n):
    A = np.eye(8*n)
    for i in range(0,n):
        for j in range(0,n):
            A[i*8][j*8] = 1
    return A

# Computes the the ratio upon C, y (output of 'trund'), and
    the maximal eigenvalue of the model.
def tratio(C, y, maxeig):
    return np.inner(np.dot(y,C),np.transpose(y))/maxeig
```

```
# Outputs a list of o ratios for a given qutrit number, c, o
    and C.
def tsample(n, c, o, C):
    v = findGenVecGram(tsdp(n))
    ratlist=[]
    i=0
    while i < o:
        ratlist.append(tratio(C, trund(v,c,cvx.matrix(C))["
            blochvec"], n))
        i=i+1
    return ratlist

# Plots the average of aforementioned samplelists within a
    range of qutrit numbers, upon input of c, sample size o
    per step, and total steps s. The steps are made such that
    they are equidistant on a logarithmic scale.
def tavgplot(ni,nf,c,o,s):
    y=[]
    x=[]
    a = np.exp(np.linspace(np.log(ni), np.log(nf), s))
    for i in a:
        x.append(i)
    for i in range(len(x)):
        y.append(avg(tsample(x[i],c,o,tC(x[i]))))
        print(x[i])
    plt.scatter(x,y,marker="o",s=5)
    plt.xlabel('log(n)')
    plt.ylabel('ratio')
    plt.xscale('log')
    plt.title('Qutrit␣Approximation␣Algorithm')
    plt.show()
```

Bibliography

[1] Sevag Gharibian and Ojas Parekh. »Almost optimal classical approximation algorithms for a quantum generalization of Max-Cut«. In: *arXiv preprint arXiv:1909.08846* (2019).

[2] Sevag Gharibian and Julia Kempe. »Approximation Algorithms for QMA-Complete Problems«. In: *SIAM Journal on Computing* 41.4 (2012), pp. 1028–1050. DOI: 10.1137/110842272. eprint: https://doi.org/10.1137/110842272. URL: https://doi.org/10.1137/110842272.

[3] Julia Kempe, Alexei Kitaev and Oded Regev. »The Complexity of the Local Hamiltonian Problem«. In: *SIAM Journal on Computing* 35.5 (2006), pp. 1070–1097. DOI: 10.1137/S0097539704445226. eprint: https://doi.org/10.1137/S0097539704445226. URL: https://doi.org/10.1137/S0097539704445226.

[4] Fernando G. S. L. Brandão and Aram W. Harrow. *Product-state Approximations to Quantum Ground States.* 2014. eprint: 1310.0017.

[5] Aram W. Harrow and Ashley Montanaro. »Extremal eigenvalues of local Hamiltonians«. In: *Quantum* 1 (Apr. 2017), p. 6. ISSN: 2521-327X. DOI: 10.22331/q-2017-04-25-6. URL: http://dx.doi.org/10.22331/q-2017-04-25-6.

[6] Sergey Bravyi et al. »Approximation algorithms for quantum many-body problems«. In: *Journal of Mathematical Physics* 60.3 (Mar. 2019), p. 032203. ISSN: 1089-7658. DOI: 10.1063/1.5085428. URL: http://dx.doi.org/10.1063/1.5085428.

[7] Anurag Anshu, David Gosset and Karen Morenz. »Beyond product state approximations for a quantum analogue of Max Cut«. In: *arXiv preprint arXiv:2003.14394* (2020).

[8] Man-Duen Choi. »Completely positive linear maps on complex matrices«. In: *Linear Algebra and its Applications* 10.3 (1975), pp. 285–290. ISSN: 0024-3795. DOI: https://doi.org/10.1016/0024-3795(75)90075-0.

[9] Sandor Imre and Laszlo Gyongyosi. *Advanced Quantum Communications: An Engineering Approach.* Dec. 2012. DOI: 10.1002/9781118337462.

[10] Lieven Vandenberghe and Stephen Boyd. »Semidefinite Programming«. In: *SIAM Review* 38.1 (1996), pp. 49–95. DOI: `10.1137/1038003`. eprint: `https://doi.org/10.1137/1038003`. URL: `https://doi.org/10.1137/1038003`.

[11] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* USA: Cambridge University Press, 2004. ISBN: 0521833787.

[12] M. X. Goemans and D.P. Williamson. »Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming«. In: *Journal of the ACM* 42 (1995), pp. 1115–1145.

[13] M. Charikar and A. Wirth. »Maximizing quadratic programs: extending Grothendieck's inequality«. In: *45th Annual IEEE Symposium on Foundations of Computer Science* (2004), pp. 54–60.

[14] H. Bethe. »Zur Theorie der Metalle«. In: *Zeitschrift fur Physik* 71.3-4 (Mar. 1931), pp. 205–226. DOI: `10.1007/BF01341708`.

[15] Daniel Gottesman. *Stabilizer Codes and Quantum Error Correction.* 1997. eprint: `quant-ph/9705052`.

[16] Elliott H. Lieb. »The classical limit of quantum spin systems«. In: *Communications in Mathematical Physics* 31.4 (Dec. 1973), pp. 327–340. DOI: `10.1007/BF01646493`.

[17] Isaac L. Chuang Michael A. Nielsen. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* 10 Anv. Cambridge University Press, 2011. ISBN: 1107002176,9781107002173.

[18] C. King. *The capacity of the quantum depolarizing channel.* 2002. eprint: `quant-ph/0204172`.

[19] Michael Horodecki, Peter W. Shor and Mary Beth Ruskai. »Entanglement Breaking Channels«. In: *Reviews in Mathematical Physics* 15.06 (Aug. 2003), pp. 629–641. ISSN: 1793-6659. DOI: `10.1142/s0129055x03001709`. URL: `http://dx.doi.org/10.1142/S0129055X03001709`.

[20] Mary Beth Ruskai. »Qubit Entanglement Breaking Channels«. In: *Reviews in Mathematical Physics* 15.06 (2003), pp. 643–662. DOI: `10.1142/S0129055X03001710`. eprint: `https://doi.org/10.1142/S0129055X03001710`. URL: `https://doi.org/10.1142/S0129055X03001710`.

[21] Pierre Pfeuty. »The one-dimensional Ising model with a transverse field«. In: *Annals of Physics* 57.1 (1970), pp. 79–90. ISSN: 0003-4916. DOI: `https://doi.org/10.1016/0003-4916(70)90270-8`.

[22] Michael A. Nielsen. *The Fermionic canonical commutation relations and the Jordan-Wigner transform.* 2005. URL: `http://michaelnielsen.org/blog/archive/notes/fermions_and_jordan_wigner.pdf`.

[23] N. N. Bogoljubov, V. V. Tolmachov and D. V. Širkov. »A New Method in the Theory of Superconductivity«. In: *Fortschritte der Physik* 6.11-12 (1958), pp. 605–682. DOI: 10.1002/prop.19580061102.

[24] Earl T. Campbell. »Enhanced Fault-Tolerant Quantum Computing in d-Level Systems«. In: *Physical Review Letters* 113.23 (Dec. 2014). ISSN: 1079-7114. DOI: 10.1103/physrevlett.113.230501. URL: http://dx.doi.org/10.1103/PhysRevLett.113.230501.

[25] Helle Bechmann-Pasquinucci and Asher Peres. »Quantum Cryptography with 3-State Systems«. In: *Physical Review Letters* 85.15 (Oct. 2000), pp. 3313–3316. ISSN: 1079-7114. DOI: 10.1103/physrevlett.85.3313. URL: http://dx.doi.org/10.1103/PhysRevLett.85.3313.

[26] M. S. Blok et al. *Quantum Information Scrambling in a Superconducting Qutrit Processor*. 2020. eprint: 2003.03307.

[27] Gen Kimura. »The Bloch vector for N-level systems«. In: *Physics Letters A* 314.5-6 (Aug. 2003), pp. 339–349. ISSN: 0375-9601. DOI: 10.1016/s0375-9601(03)00941-1. URL: http://dx.doi.org/10.1016/S0375-9601(03)00941-1.

[28] Karol Życzkowski Ingemar Bengtsson. *Geometry of Quantum States an Introduction to Quantum Entanglement*. 2nd. Cambridge University Press, 2017. ISBN: 9781107026254,1107026253,9781107656147,1107656141.

[29] Istok P. Mendas. »The classification of three-parameter density matrices for a qutrit«. In: *Journal of Physics A: Mathematical and General* 39.36 (Aug. 2006), pp. 11313–11324. DOI: 10.1088/0305-4470/39/36/012. URL: https://doi.org/10.1088%2F0305-4470%2F39%2F36%2F012.

[30] Sandeep K Goyal et al. »Geometry of the generalized Bloch sphere for qutrits«. In: *Journal of Physics A: Mathematical and Theoretical* 49.16 (Mar. 2016), p. 165203. DOI: 10.1088/1751-8113/49/16/165203. URL: https://doi.org/10.1088%2F1751-8113%2F49%2F16%2F165203.