



**Università degli Studi di Parma**

---

DIPARTIMENTO DI SCIENZE MATEMATICHE, FISICHE E INFORMATICHE  
Corso di Laurea Magistrale in Scienze Informatiche

TESI DI LAUREA MAGISTRALE

**Introducing QDOS:  
software tools for generic QuDits OperationS**

Candidato:  
**Matteo Mezzadri**

Relatore:  
**Ch.mo Prof. Alessandro Chiesa**

Correlatore:  
**Ch.mo Prof. Stefano Carretta**



# Contents

<b>Introduction</b>	iii
<b>1 Theoretical Foundations</b>	<b>1</b>
1.1 Quantum Computing: A Brief Introduction . . . . .	1
1.2 Quantum Computing: Qubits Vs Qudits . . . . .	7
1.3 Dynamics of Closed Quantum Systems . . . . .	10
1.4 Open Quantum Systems: Lindblad Equation . . . . .	16
1.5 Decomposition of $SU(n)$ in Planar Rotations . . . . .	21
1.6 Generalized Measurements . . . . .	24
<b>2 QDOS Manual</b>	<b>29</b>
2.1 The QDOS Programming Language . . . . .	29
2.2 Built-in Gates . . . . .	33
2.3 How to Define a Custom Gate . . . . .	36
2.4 Measurements and Custom Logics . . . . .	39
2.5 How to Set Up and Run Simulations . . . . .	43
<b>3 QDOS Use Cases</b>	<b>47</b>
3.1 Quantum Fourier Transform . . . . .	47
3.1.1 QFT on Qubits . . . . .	48
3.1.2 QFT on Qudits . . . . .	50
3.1.3 QDOS Simulations . . . . .	53
3.2 Embedded QEC on Qudits . . . . .	59
<b>Conclusions</b>	<b>69</b>
<b>A: QDOS grammar</b>	<b>73</b>
<b>B: Built-in Gates List</b>	<b>75</b>
<b>C: Command Line Options</b>	<b>77</b>
<b>Bibliography</b>	<b>79</b>

<b>In-depth Readings</b>
--------------------------

<b>83</b>
-----------

# Introduction

Quantum computers are built upon a computing unit consisting of a two-level system, namely  $|0\rangle$  and  $|1\rangle$ . In recent years, the use of multi-level instead of two-level logical units (qudits) has attracted an increasing interest. Indeed, many new theoretical results have shown that the use of qudits could lead to remarkable advantages. For example, Quantum Error Correction (QEC) could be directly embedded on the qudit, cutting the overhead of *standard* QEC made by encoding a logical qubit into many physical qubits. Another field in which qudits show better performances than qubits is quantum simulations, where exponential speed-up on the simulation of quantum systems could be achieved compared to classical devices. Recently, Molecular Nanomagnets have been proposed as a particularly promising platform for building qudits. They present advantages also among other qudits, such as the possibility to precisely design and produce them. Such physical systems are the qudits at the base of the simulations reported in the Thesis.

Given the advantages that qudits have over qubits, programs able to simulate them become more and more necessary. The subject of the Thesis is a software for the simulation of general one- and two-qudit operations between generic qudits. We can divide the developed program into three main components: a simulator of generic qudits, a Domain Specific Language (DSL) for writing quantum programs then executed by the simulator and an optimizing compiler interfacing the simulator and the DSL.

The simulator is a solver of a system of ordinary differential equations. In particular, the ODE system that we want to solve is a model that governs the evolution of open quantum systems which in literature goes by the name of *Lindblad Equation*. Furthermore, this has been developed considering a generic Hamiltonian for the various qudits. By doing so, it is also possible

to investigate the quality of different physical platforms for the encoding of qudits.

The programming language was built starting from OpenQASM (Open Quantum Assembly), reference language in the scientific community for the description of quantum circuits (i.e., quantum programs). I decided to create a new language in order to give the user the possibility to execute arbitrary (C) code at any point in the circuit. This becomes necessary for the implementation of Quantum Error Correction schemes. In fact, for these schemes, after a measurement of the system, the instructions to be carried out depend directly on the outcome of the measurement itself, making them not known *a priori*. This flexibility is certainly one of the main strengths of the software developed.

The Thesis begins with the presentation of the theoretical foundations on which the software is based. In particular, we introduce the Lindblad Equation, generalized (ideal and non-ideal) measurements of quantum systems and the decomposition algorithm used by the compiler. Then, it continues with the description of the software produced. The program manual is also provided. Finally, we present a collection of examples showing the functionality of the software created for the execution of algorithms on generic qubits and qudits. For these examples we have focused on a specific kind of qudits, i.e. Molecular Nanomagnets. Such a platform has been shown to excel in customizability, simplicity of realization and robustness against pure-dephasing errors. Exploiting their strengths, schemes of Embedded Quantum Error Correction on Molecular Nanomagnets are presented, highlighting the excellent potential of this platform for the realization of quantum computers.

# Chapter 1

## Theoretical Foundations

In this Chapter we lay down the theoretical foundations on which our software is based. In Section [1.1](#) we introduce the newcomers to the quantum computing world, presenting the challenges that it is facing nowadays and the benefits that it could give us once such challenges have been overcome. In Section [1.2](#) we illustrate the benefits that one could achieve by using multi-level systems instead of standard two-level ones. Then, in Section [1.3](#) and [1.4](#), we derive the fundamental equation governing the evolution of an open quantum system. In Section [1.5](#) we describe an algorithm for the decomposition of a generic unitary operator into simpler operators directly executable on a quantum computer. As last, we end the Chapter with Section [1.6](#) that will include a treatment on generalized measurements, a key concept for quantum computing.

### 1.1 Quantum Computing: A Brief Introduction

A quantum computer is a machine that performs calculations according to the laws of quantum mechanics. Since the dawn of the idea itself, a quantum computer has been conceived as made of many quantum bits, also called *qubits*. This same comparison with the classical scheme of computation, combined with the instability and the difficulty of using the high energy levels of quantum systems, focused both the theoretical proposals and nearly all the quantum hardware available today to two-level systems.

As a classical bit, a qubit can encode one bit of information. But the analo-

gies end here. Indeed, if we call  $|0\rangle$  and  $|1\rangle$  the two levels of the quantum system that constitutes the qubit, then the generic state of the qubit is

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \text{ with } |\alpha|^2 + |\beta|^2 = 1.$$

As for a classical computer, the ultimate goal of a quantum computer is to compute functions. How it does so is crucial in understanding the possible advantages that it can bring. In particular, we have a very well known and strong result for a quantum system and the doable operations on its states.

**Theorem 1.1** (No Cloning Theorem). *Suppose we have two quantum systems  $A$  and  $B$  with a common Hilbert Space  $H_1 = H_2 = H$ . Then there is no unitary operator  $U$  on  $H \otimes H$  such that for all normalized states  $|\phi\rangle$  and  $|\sigma\rangle$  in  $H$*

$$U|\phi\rangle|\sigma\rangle = e^{i\alpha(\phi,\sigma)}|\phi\rangle|\phi\rangle$$

for some real number  $\alpha \in \mathbb{R}$ .

*Proof.* Suppose that such an  $U$  exists. Let  $|\phi\rangle_A$  and  $|\psi\rangle_A$  be two generic states in  $H$ . By the unitarity of  $U$  we would have

$$\begin{aligned} \langle\phi|\langle\sigma||\psi\rangle|\sigma\rangle &= \langle\phi|\langle\sigma|U^\dagger U|\psi\rangle|\sigma\rangle \\ &= e^{i\beta(\phi,\psi,\sigma)}\langle\phi|\langle\phi||\psi\rangle|\psi\rangle \\ &= e^{i\beta(\phi,\psi,\sigma)}\langle\phi|\psi\rangle\langle\phi|\psi\rangle. \end{aligned}$$

In other words, it is verified the following

$$|\langle\phi|\psi\rangle| = |\langle\phi|\psi\rangle|^2,$$

from which is trivial to see that we must have  $\langle\phi|\psi\rangle = 0$  or  $|\phi\rangle = e^{i\delta}|\psi\rangle$  for some  $\delta \in \mathbb{R}$ . The Thesis follows immediately from the arbitrariness of the states  $|\phi\rangle$  and  $|\psi\rangle$  in  $H$ .  $\square$

In less technical terms, Theorem [1.1](#) states that it is impossible to make a carbon copy of any quantum system. Therefore with a quantum computer we are unable to copy the state of a qubit onto another qubit. This implies that, for instance, *Error Correction* as we know it is not possible on

a quantum computer. Moreover, without going too deep into the details of how a measurement of a quantum system is made (see Section 1.6), since measurements destroy superposition and we can not make a perfect copy of a quantum system, classical control flow based on values of “variables” is impossible. Although, since its discovery, a range of partial cloning strategies for quantum states have been devised. For example, the optimal symmetric cloning of Bužek and Hillery or the perfect but probabilistic cloning of Duan and Guo. The interested reader can refer to [1] for a description of these and other partial cloning strategies.

Quantum computers do not only have disadvantages over their classical counterpart. Indeed, as we have mentioned before, a quantum system evolves under unitary transformations, and such are the operations that we are able to perform. Then, because unitary operators are always invertible (and have a unitary inverse), any operation on a quantum computer is invertible. So, given the output of a certain computation (and the operator  $U$  that has performed it), we can always retrieve its input. But just invertibility is not enough gain to compensate for Theorem 1.1

Before understanding the real advantage that a quantum computer can have over a classical one, we need to point out another subtle difference between the two machines. Let’s say that we want to compute a *non-injective* function  $f$  on the input state  $|x\rangle$ . Since we can perform only unitary operations on a quantum computer, and those preserve the norm of their input, there will be at least one input  $x$  for which we will have

$$U|x\rangle \neq |f(x)\rangle.$$

Indeed, suppose the contrary. Since  $f$  is not injective there will exist  $|x_1\rangle, |x_2\rangle$  such that  $|x_1\rangle \neq |x_2\rangle$  and  $|f(x_1)\rangle = |f(x_2)\rangle$ . Then we will have

$$1 \neq \langle x_1|x_2\rangle = \langle x_1|U^\dagger U|x_2\rangle = \langle f(x_1)|f(x_2)\rangle = 1$$

that cannot be.

Then, to be able to compute **any** function  $f$ , we must recur to a clever trick. We introduce a second bit string, initialized in an arbitrary state  $|y\rangle$ , so that

the quantum computer performs

$$U(|x\rangle|y\rangle) = |x\rangle|y \oplus f(x)\rangle.$$

Where  $y \oplus f(x)$  means the modulo 2 addition of  $y$  and  $f(x)$ . Moreover, initializing  $|y\rangle = |0\rangle$  one obtains

$$U(|x\rangle|0\rangle) = |x\rangle|f(x)\rangle.$$

Now, for the same non-injective function  $f$  we would get

$$\begin{aligned} \langle x_1|x_2\rangle &= \langle x_1|\langle 0|x_2\rangle|0\rangle \\ &= \langle x_1|\langle 0|U^\dagger U|x_2\rangle|0\rangle \\ &= \langle x_1|\langle f(x_1)|x_2\rangle|f(x_2)\rangle \\ &= \langle x_1|x_2\rangle. \end{aligned}$$

With this in mind we are now able to understand where the power of a quantum computer finds its root. Suppose to have a function  $f$  that takes as input any string  $s$  of  $n$  bits. And suppose also to have a quantum computer made of  $n + 1$  qubits ( $n + m$  if  $f$  has a vectorial output of dimension  $m$ ), and the Unitary operator  $U$  such that

$$\forall |x\rangle \in \{0, 1\}^n, \quad U|x\rangle|0\rangle = |x\rangle|f(x)\rangle.$$

Prepare the input  $|x\rangle$  as the direct product of the  $|+\rangle \doteq \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ , i.e.,  $|x\rangle \doteq 2^{-n/2}(|0\rangle + |1\rangle)^n$ . This input state is easily achievable by just setting  $|x\rangle = |0\rangle$  and then applying an Hadamard gate<sup>1</sup> to any of the input qubits, i.e.,  $|x\rangle \doteq H^n|0\rangle$ .

Then, by only one evaluation of  $U$ , we will compute  $f$  on all its input values. The input  $|x\rangle$ , prepared with the procedure described above, will contain any string of  $n$  bits (obviously scaled by a factor of  $2^{-n/2}$ ).

---

<sup>1</sup>As in classical computation theory a gate is an elementary operation executable on a quantum computer. Complex operations are achieved by applying in sequence elementary gates. In [15] is shown how a set of few gates on one and two qubits are enough to perform any given unitary operator involving an arbitrary big number of qubits. One can still refer to [15] for a deep discussion on the quantum circuit model for quantum computation.

Sadly, not all that shines is gold. As in classical computing, on a quantum computer we are able to perform only a restrict, but universal<sup>2</sup>, set of gates ([15]). We will show that (see Section 1.5), given an operator  $U$  we are always able to decompose it into  $o(N^2)$  elementary gates, where  $N$  is the number of qubits. In other words, the number of elementary gates needed to implement  $U$  will be exponential in the dimension of the Hilbert space spanned by the qubits. Therefore completely annihilating any advantage given by the parallelization of the evaluation of  $U$ . Then it should be clear that we must have “clever” ways of decomposing, or approximating,  $U$ . In more rigorous terms, we need decompositions/approximations into elementary gates where the number of gates grows as a polynomial function of the Hilbert space dimension, or, equivalently, as a logarithmic function of the number of qubits. In such cases we will effectively have a quantum exponential speed-up.

We have some cases where this happens brilliantly, such as the Quantum Fourier Transform for which we are able to achieve an exponential speed-up over the best classical algorithm known. One can find in [15] a quantum circuit implementing it.

One other example in which we are able to achieve an exponential speed-up over the classical counterpart is the factorization of numbers. Always in [15] is illustrated Shor’s algorithm, maybe the most famous quantum algorithm as of today, which enables us to factorize in its prime factors an  $N$  – bit long number in  $o(N)$  computations.

There are also algorithms that can give us some advantages over their classical counterpart but do not reach an exponential speed-up. One of these is Grover’s Search Algorithm, that enables us to search an unstructured database of  $N$  elements in  $o(\sqrt{N})$  computations. Again, in [15] is shown that Grover’s Search Algorithm is optimal, in the sense that an algorithm able to search an unstructured database of  $N$  elements faster than  $o(\sqrt{N})$  does not Exist.

As last, in terms of efficiency, there is at least one field in which a quantum computer, as of now, sits behind a classical one, Error Correction. With this,

---

<sup>2</sup>A set of gates is said to be universal whenever any unitary operation can be approximated with arbitrary precision by a sequence of the gate in the set itself. One can refer to [15] for a detailed discussion of universal gates sets on qubits or to [14] for an equivalently deep discussion over universal gates sets on generic qudits.

I am not stating that the quantum algorithms for error correction are less advanced than those developed for classical error correction. On the contrary, quantum error correction finds its root in classical error correction but then it develops in outstanding algorithms, such as topological codes ([9]) or Embedded codes ([5]). But errors in quantum computers are intrinsic to the quantum computer itself. The majority of them derives from the same power that enables quantum computers to offer an exponential speed-up, **entanglement**. In particular, they arise from the entanglement of the quantum computer and the environment. Hence, even if the theoretical research in QEC is well developed, the search for an easy-to-implement code is open, and high are the efforts of the scientific community put in the quest. This does not mean that we will not be able to perform error correction or that error correction is impossible. On the other hand, there is a really strong and reassuring theoretical result, the Threshold Theorem ([15]). This Theorem is beyond our scope and thus we will not report it. But put in simple words, the Threshold Theorem states that if we are able to perform physical operations on the hardware “well enough” (i.e., we are able to perform the universal gates with a small probability of errors), then we are able to perform effective Quantum Error Correction, i.e., we are able, by performing computations, to diminish the amount of errors in the quantum computer. The idea is that when we are below a certain threshold for the gates accuracy then we are able to enlarge the logical qudit -adding more physical qubit to it- lowering the probability of logical errors but keeping the “hardware errors”, given by imperfect gates, under control. The interested reader should definitively read [15] for a detailed treatment of repetition codes and the Threshold Theorem.

In Section 1.2 we will present a platform for a slightly different approach to quantum computing and QEC itself, Qudits. The change is small, just use more than two levels. Small changes but with clear improvements over qubits.

Around this same platform, and the need to have efficient ways to simulate operations on it, revolves the software subject of this thesis.

## 1.2 Quantum Computing: Qubits Vs Qudits

As already pointed out at the beginning of the previous Section, in current quantum computing hardware, each building block is typically used to encode a two-level quantum bit. This happens independently from the physical system that constitutes the computer. Indeed, as of today, superconducting resonators, trapped ions, quantum dots, etc., are all restricted to the Hilbert space formed by their two low energy levels. In this Section we will discuss possible advantages that using qudits (i.e., quantum bits with more than two levels available) instead of qubits could bring.

The crucial point is that the hardware beneath the computer is limited to two levels **not** by its nature (indeed in almost any case the physical object has more than two levels available), but because of the instability (they tend to be affected by strong decoherence) and the difficulty to address the other ones.

Making use of qudits presents clear challenges:

1. More controls will be needed to fully exploit the new degrees of freedom.
2. More controls would imply more opportunities for errors during a computation.
3. A larger number of levels could yield to an enhanced impact of decoherence.

More degrees of freedom will not imply only an overhead in controls on the qudit. As shown in [13], if we use these new degrees of freedom as ancillary levels we are able to decompose a multi-qubit gate into elementary gates in more efficient ways. In the paper a Toffoli gate between three qutrits (three level systems, in which the third level was used only as an ancillary level) has been decomposed into only three two-qutrit gates, while using standard qubits the best decomposition available is made of six two-qubit <sup>3</sup> gates. Indeed, fewer two-body gates will imply less errors to be corrected, since are these entangling gates that are more error prone.

---

<sup>3</sup>Here we are ignoring the number of one-qubit gates since those are much more easy to implement than their two-body counterparts. For completeness using the approach with qutrits other two one-qutrit gates are necessary. While with the standard approach with qubits another ten one-qubit gates are necessary.

But the gains are even bigger if we slightly increase the dimension of the problem. For instance, the procedure reported in [13] can be extended to the general case. For example, a 4-body (three-qubits-control one-qudits-target) can be decomposed in five two-body gates.

We have even more. Indeed this same procedure can be applied to the decomposition of generic controlled gates. The only request, to be able to perform such a compression, is the availability of enough levels on the target qudit. Again, this is quite a good advantage over qubits.

For the second and third challenges we could have a shared answer in exploiting the additional levels of a qudit to embed Quantum Error Correction into a single object. Indeed, there are two papers ([4], [5]) in which schemes for Quantum Error Correction embedded directly into the qudits are proposed and the structure of the qudit itself is investigated in order to suppress decoherence to a larger extent.

As pointed out in this two papers, magnetic molecules provide the ideal playground for such an architecture, thanks to their rich and chemically tunable spectrum, characterized by many available and individually addressable low-energy levels.

The idea in embedded QEC is to virtualize a logical qubit (or even qudit) on the qudit itself creating logical states using its multiple available levels. Doing so we cut drastically the typical hardware overhead of QEC algorithms, such as the surface code ([9]). Indeed, current estimates for the number of qubits required to perform practical calculations, such as simulating reaction mechanisms for nitrogen fixation, or factoring numbers of the sizes used in RSA encryption, are in the range of 500–1000 logical qubits. However, each logical units would be made of many physical qubits, thus requiring millions to billions of qubits to perform these calculations ([9]). So is easy to see that the ability to virtualize one logical high fidelity qubit on one physical qudit could lead to a great performance leap in quantum computing.

Molecular nanomagnets seem to be an optimal candidate for the implementation of qudits since it is possible -through accurate chemical processes- to synthesize them with the desired properties. In [4] the key ingredient ruling decoherence in multi-spin clusters have been shown and optimal molecule embedding Quantum Error Correction been designed. These are indeed an-

tiferromagnetically coupled systems with competing exchange interactions, characterized by many low-energy states in which decoherence is dramatically suppressed and does not increase with the system size.

Now we will talk about another application in which a qudits approach could bring clear advantages over a more “classical” qubits one, Quantum Simulations.

A quantum simulator is a system under high control of the experimenter, which is able to reproduce the dynamical behavior of a given physical model irrespective of the degree of internal correlations or entanglement. Analog quantum simulators have been proposed and developed, in which the physical properties of a targeted model are reproduced on a physical setup under externally controlled conditions. Then there is another kind of quantum simulators, digital quantum simulators, that are programmable and general purpose quantum devices [21].

To be able to understand why a quantum computer made of qudits could be better in quantum simulations than one based on qubits, we have to introduce how a quantum simulator works. The idea is to map the time evolution of the target physical model onto the quantum hardware degrees of freedom, in which the time evolution can be programmed in digital steps through a sequence of unitary operations defined by a quantum circuit. How this sequence of operators is determined is out of our scope, but the interested reader can refer to [20] for a detailed study of the problem.

After this brief explanation, it should seem more natural to use multi-level qudits as the base of the mapping of multi-level objects. For instance, this analogy between the simulator and the system simulated has reduced the hardware/software overhead in the quantum simulation of light–matter interaction processes as shown in [21]. For such problems, standard encodings -designed for multi-qubit architectures- either use a number of qubits equal to the number of simulated photons (i.e. an exponentially large Hilbert space) or reduce the number of qubits at the price of much more complex quantum circuits, involving interactions between distant qubits ([16]). Both the alternatives are far from being competitive compared to what is achievable with qudits. Indeed, in [21], using molecular spin qudits the authors have been able to reduce both the hardware overhead and the complexity

of manipulations by mapping each photon mode to a single spin  $S$  qudit. The qudit levels are then used to represent the number of photons in each mode.

It is clear that qudits can bring some serious advantages over the more “classical” concept of qubits. Still, this does not mean that qudits are the best path, nor that qubits are. But surely, there are plenty of reasons to investigate this promising alternative way of building quantum computers.

### 1.3 Dynamics of Closed Quantum Systems

To be able to control quantum systems we need a model that describes how it evolves and how to drive its evolution. The first step of such a model is the *Schrödinger Equation* [18].

Suppose we have a physical system whose state ket at time  $t_0$  is represented by  $|\alpha, t_0\rangle$ . Let us call  $|\alpha, t_0; t\rangle$  the state of the system at time  $t$ . For our need, suppose  $t > t_0$ . Then the two states are related by an operator which we will call the *time-evolution operator*  $\mathcal{U}(t, t_0)$

$$|\alpha, t_0; t\rangle = \mathcal{U}(t, t_0)|\alpha, t_0\rangle.$$

This special operator has some peculiar properties [18]. It is unitary, i.e.,

$$\mathcal{U}^\dagger(t, t_0)\mathcal{U}(t, t_0) = \mathbb{I}$$

from which follows that the norm is preserved through time-evolution. In other words, if we take a normalized ket as our initial state then also the final ket will be normalized,

$$1 = \langle\alpha, t_0|\alpha, t_0\rangle = \langle\alpha, t_0|\mathcal{U}^\dagger(t, t_0)\mathcal{U}(t, t_0)|\alpha, t_0\rangle = \langle\alpha, t_0; t|\alpha, t_0; t\rangle.$$

Is usual to refer to this property as probability conservation.

Another feature that the family of time-evolution operators has is the *composition property*

$$\mathcal{U}(t, t_0) = \mathcal{U}(t_2, t_1)\mathcal{U}(t_1, t_0), \quad \forall t_2 > t_1 > t_0.$$

Hence we have a semigroup with infinitesimal operator

$$\mathcal{U}(t_0 + dt, t_0) = \mathbb{I} - i\Omega dt$$

where we take  $\Omega$  to be a Hermitian operator ( $\Omega^\dagger = \Omega$ ) so that unitarity to the second order is granted at the infinitesimal generator [18].

In particular, borrowing from classical mechanics the idea that the Hamiltonian  $H$  of a system is the generator of time evolution [10], we have

$$\Omega = \frac{H}{\hbar}.$$

Thus the infinitesimal generator is written as

$$\mathcal{U}(t_0 + dt, t_0) = \mathbb{I} - \frac{i}{\hbar} H dt.$$

We are now able to derive the fundamental differential equation for the time-evolution operator  $\mathcal{U}(t, t_0)$ . By the composition property, with  $t_1 \rightarrow t$  and  $t_2 \rightarrow t + dt$ , we get

$$\mathcal{U}(t + dt, t_0) = \mathcal{U}(t + dt, t) \mathcal{U}(t, t_0) = \left( \mathbb{I} - \frac{i}{\hbar} H dt \right) \mathcal{U}(t, t_0),$$

from which we derive trivially that

$$\mathcal{U}(t + dt, t_0) - \mathcal{U}(t, t_0) = -\frac{i}{\hbar} H dt \mathcal{U}(t, t_0).$$

In differential form it becomes

$$i\hbar \frac{\partial}{\partial t} \mathcal{U}(t, t_0) = H \mathcal{U}(t, t_0). \quad (1.1)$$

Equation (1.1) is known as *The Schrödinger Equation* for the time-evolution operator. Multiplying both sides of equation (1.1) by the ket  $|\alpha, t_0\rangle$ , we trivially derive the Schrödinger Equation for the state ket

$$i\hbar \frac{\partial}{\partial t} \mathcal{U}(t, t_0) |\alpha, t_0\rangle = H \mathcal{U}(t, t_0) |\alpha, t_0\rangle$$

that becomes

$$i\hbar \frac{\partial}{\partial t} |\alpha, t_0; t\rangle = H |\alpha, t_0; t\rangle. \quad (1.2)$$

Both equations (1.1) and (1.2), once solved, give us a description of the temporal evolution of a quantum system with Hamiltonian  $H$ . As one would expect, the difficulties in finding the solutions to these equations are related to the properties of  $H$ . Indeed, if we have a time-independent Hamiltonian then the solution of equation (1.1) is trivially

$$\mathcal{U}(t, t_0) = e^{-\frac{i}{\hbar}H(t-t_0)}. \quad (1.3)$$

One way to retrieve it is to compound successively infinitesimal time-evolution operators, i.e.

$$\lim_{N \rightarrow +\infty} \left( \mathbb{I} - \frac{(iH/\hbar)(t-t_0)}{N} \right)^N = e^{-\frac{i}{\hbar}H(t-t_0)}.$$

Even easier one can just substitute equation (1.3) into (1.1) and verify that it is a solution.

Unfortunately equation (1.3) becomes a lot more complicated for Hamiltonians such that

$$\frac{\partial H}{\partial t} \neq 0.$$

With Perturbation Theory one can show [18] that the solution to equation (1.1) is written as

$$\mathcal{U}(t, t_0) = e^{-\frac{i}{\hbar} \int_{t_0}^t dt' H(t')}$$

if the Hamiltonian  $H$  is time-dependent and

$$[H(t_1), H(t_2)] = 0 \quad \forall t_1, t_2 \in (t_0, t).$$

Then, if this last hypothesis fails, the time-evolution operator assumes the following form

$$\mathcal{U}(t, t_0) = \mathbb{I} + \sum_{n=1}^{+\infty} \left( -\frac{i}{\hbar} \right)^n \int_{t_0}^t dt_1 \int_{t_0}^{t_1} dt_2 \dots \int_{t_0}^{t_{n-1}} dt_n H(t_1) H(t_2) \dots H(t_n).$$

Since we are interested in the simulation of the controlled evolution of a quantum system, we need actual ways to control its evolution. Such methods, seen from a physical point of view, can be as different as possible. From a more abstract and mathematical point of view they consist in a modification of the Hamiltonian of the system. In other words, if  $H_0$  is the Hamiltonian

of the free system then the Hamiltonian of the system will be  $H = H_0 + H_1$ . Where  $H_1$  is the perturbation of the Hamiltonian performed by us to drive the evolution of the system. We will focus on the case

$$\frac{\partial H_0}{\partial t} = 0, \quad \frac{\partial H_1}{\partial t} \neq 0. \quad (1.4)$$

Such a Hamiltonian, if conditions (1.4) hold, is usually written as

$$H(t) = H_0 + V(t). \quad (1.5)$$

Then, for the simulation of a quantum system evolving under our control, we need to simulate its evolution governed by such an Hamiltonian. For such simulations the Schrödinger Picture is not the most appropriate since we have shown that the evolution operator is all but trivial when the Hamiltonian is time-dependent. Moreover, if we would ever follow this approach, we will lose all the information known about  $H_0$ . A different approach is needed.

We will now present the method proposed by *P. Dirac*, also known as *The Interaction Picture*. Consider a Hamiltonian  $H$  in the form of equation (1.5) with a time-independent  $H_0$ . Call  $|n\rangle$  and  $E_n$  respectively the eigenvectors and the eigenvalues of  $H_0$ . We want to address the problem of finding the probability as a function of time for the system to be found in one of its eigenstates  $|n\rangle$ . Formally, suppose that at  $t = 0$  the state ket of the physical system is given by

$$|\alpha, t_0\rangle = \sum_n c_n(0) |n\rangle.$$

We want to find  $c_n(t)$  for  $t > 0$  such that

$$|\alpha, t_0; t\rangle = \sum_n c_n(t) e^{-iE_n t/\hbar} |n\rangle.$$

Then the probability of finding the system in the state  $|n\rangle$  at a time  $t$  is easily obtained with the evaluation of  $|c_n(t)|^2$ . This form clarifies that the potential  $V(t)$  drives the evolution of the system through the coefficient  $c_n(t)$ . Moreover, from this way of writing the problem is simple to grasp the idea of the Interaction Picture. Define

$$|\alpha, t_0; t\rangle_I = e^{\frac{i}{\hbar} H_0 t} |\alpha, t_0; t\rangle_S$$

where the subscript  $I$  and  $S$  refer to the Interaction Picture and to the Schrödinger Picture respectively. Operators are redefined in a similar way as

$$O_I = e^{\frac{i}{\hbar}H_0t}O_S e^{-\frac{i}{\hbar}H_0t}.$$

Hence the potential of our Hamiltonian becomes

$$V_I = e^{\frac{i}{\hbar}H_0t}V e^{-\frac{i}{\hbar}H_0t}.$$

From (1.2) follows the equation for the time-evolution of a state ket in the Interaction Picture. Indeed

$$\begin{aligned} i\hbar \frac{\partial}{\partial t} |\alpha, t_0; t\rangle_I &= i\hbar \frac{\partial}{\partial t} (e^{\frac{i}{\hbar}H_0t} |\alpha, t_0; t\rangle_S) \\ &= -H_0 e^{\frac{i}{\hbar}H_0t} |\alpha, t_0; t\rangle_S + e^{\frac{i}{\hbar}H_0t} \left( i\hbar \frac{\partial}{\partial t} |\alpha, t_0; t\rangle_S \right) \\ &= -H_0 e^{\frac{i}{\hbar}H_0t} |\alpha, t_0; t\rangle_S + e^{\frac{i}{\hbar}H_0t} (H_0 + V) |\alpha, t_0; t\rangle_S \\ &= e^{\frac{i}{\hbar}H_0t} V e^{-\frac{i}{\hbar}H_0t} e^{\frac{i}{\hbar}H_0t} |\alpha, t_0; t\rangle_S \\ &= V_I |\alpha, t_0; t\rangle_I. \end{aligned}$$

We thus see that

$$i\hbar \frac{\partial}{\partial t} |\alpha, t_0; t\rangle_I = V_I |\alpha, t_0; t\rangle_I. \quad (1.6)$$

Expanding the state ket of the Interaction Picture into the base of the eigenstates of  $H_0$  introduced earlier we obtain

$$|\alpha, t_0; t\rangle_I = \sum_n c_n(t) |n\rangle$$

where the  $c_n(t)$  are the same as before. Then, by the completeness property

of the eigenstates of  $H_0$  and by their time-independence, we can write

$$\begin{aligned}
 i\hbar \frac{d}{dt} c_n(t) &= i\hbar \frac{\partial}{\partial t} \langle n | \alpha, t_0; t \rangle_I \\
 &= i\hbar \langle n | \left( \frac{\partial}{\partial t} | \alpha, t_0; t \rangle_I \right) \\
 &= \sum_m \langle n | V_I | m \rangle \langle m | \alpha, t_0; t \rangle_I \\
 &= \sum_m \langle n | e^{\frac{i}{\hbar} E_n t} V e^{-\frac{i}{\hbar} E_m t} | m \rangle \langle m | \alpha, t_0; t \rangle_I \\
 &= \sum_m e^{\frac{i}{\hbar} (E_n - E_m) t} V_{nm} c_m(t).
 \end{aligned}$$

Hence obtaining the following system of ordinary differential equations for the coefficients  $c_n(t)$

$$i\hbar \frac{d}{dt} c_n(t) = \sum_m e^{i\omega_{nm}t} V_{nm} c_m(t), \quad (1.7)$$

where the coefficients  $\omega_{nm}$ , also called Rabi's frequencies, are defined as  $\omega_{nm} \doteq \frac{E_n - E_m}{\hbar} = -\omega_{mn}$ . Such frequencies are exactly what enables us to correctly address desired levels of our quantum system by potential  $\tilde{V}$  resonating with them [18]. For example, for a spin  $S = \frac{1}{2}$  system we can induce swaps of the states  $|\uparrow\rangle$  and  $|\downarrow\rangle$  with a potential as simple as

$$V(t) = K B_1 \cos(\omega_{12}t) \sigma_x$$

where with  $\sigma_x$  we refer to the respective Pauli operator. This, in practice, is obtained by a simple linearly oscillating magnetic field of the form [18]

$$\mathbf{B} = B_0 \hat{\mathbf{z}} + \tilde{K} B_1 \cos(\omega_{12}t) \hat{\mathbf{x}}.$$

System (1.7) enables us to perform accurate numerical simulations of the evolution of quantum systems under arbitrary potential once  $H_0$  is completely resolved. Moreover, it tells us how to exactly perform operations between any pair of states, that is by resonant pulses to the Rabi's frequencies  $\omega_{nm}$ . But unfortunately is still not enough to describe an actual quantum computer. Indeed (1.7) describes isolated quantum systems and fails completely in the description of the interaction that real systems have with the environment, from which arise the errors that heavily afflict quantum computers of today.

Then if we want to perform accurate numerical simulations of an operational quantum computer we can not follow system (1.7) as it is, generalizations are needed.

## 1.4 Open Quantum Systems: Lindblad Equation

From Section 1.3 it is clear that a quantum computer is not a closed quantum system, i.e. a system whose temporal evolution can be described through equation (1.1) alone. Before deriving a generalization of such equation for an open system, we need to introduce the *density operator formalism* pioneered by J. von Neumann in 1927.

A pure ensemble is, by definition, a collection of physical systems such that every member is characterized by the same state ket  $|\alpha\rangle$ . Instead, a mixed ensemble is a collection of physical systems where a fraction of the members  $w_1$  is characterized by  $|\alpha_1\rangle$ , another fraction  $w_2$  is characterized by  $|\alpha_2\rangle$  and so on. The only constraint on such populations is that

$$\sum_i w_i = 1.$$

Suppose we make a measurement on a mixed ensemble of some observable  $\mathbf{A}$ , the average measured value of  $\mathbf{A}$  is given by the so called ensemble average of  $\mathbf{A}$ , that is

$$[\mathbf{A}] \doteq \sum_i w_i \langle \alpha_i | \mathbf{A} | \alpha_i \rangle.$$

Then rewriting the ensemble average on a more general basis  $\{|b\rangle\}$ , which is also a basis for the ket space, we have

$$\begin{aligned} [\mathbf{A}] &= \sum_i w_i \sum_{b'} \sum_{b''} \langle \alpha_i | b' \rangle \langle b' | \mathbf{A} | b'' \rangle \langle b'' | \alpha_i \rangle \\ &= \sum_{b'} \sum_{b''} \left( \sum_i w_i \langle b'' | \alpha_i \rangle \langle \alpha_i | b' \rangle \right) \langle b' | \mathbf{A} | b'' \rangle. \end{aligned}$$

Defining the *density operator*  $\rho$  as

$$\rho \doteq \sum_i w_i |\alpha_i\rangle \langle \alpha_i|,$$

we have that in the basis  $\{|b\rangle\}$  it is

$$\langle b''|\rho|b'\rangle = \sum_i w_i \langle b''|\alpha_i\rangle \langle \alpha_i|b'\rangle.$$

Then, in terms of  $\rho$ , the ensemble average of  $\mathbf{A}$  is written as

$$[\mathbf{A}] = \sum_{b'} \sum_{b''} \langle b''|\rho|b'\rangle \langle b'|\mathbf{A}|b''\rangle = \text{Tr}(\rho\mathbf{A}).$$

Now that is clear from where it came the idea of the density operator  $\rho$ , we will derive evolution equations for  $\rho$  analogous to equations (1.1) and (1.7).

Let us assume that at some time  $t_0$  the state of the system is characterized by

$$\rho(t_0) = \sum_i w_i |\psi_i, t_0\rangle \langle \psi_i, t_0|,$$

where  $|\psi_i, t_0\rangle$  is a normalized statevector that evolves according to the Schrödinger equation (1.1). Hence

$$\rho(t) = \sum_i w_i \mathcal{U}(t, t_0) |\psi_i, t_0\rangle \langle \psi_i, t_0| \mathcal{U}^\dagger(t, t_0),$$

which can be written more concisely as

$$\rho(t) = \mathcal{U}(t, t_0) \rho(t_0) \mathcal{U}(t, t_0)^\dagger.$$

Differentiating it with respect to time, with equation (1.1) in mind, we get

$$\begin{aligned} \frac{d}{dt}\rho(t) &= \left(\frac{\partial}{\partial t}\mathcal{U}(t, t_0)\right)\rho(t_0)\mathcal{U}(t, t_0)^\dagger + \mathcal{U}(t, t_0)\rho(t_0)\left(\frac{\partial}{\partial t}\mathcal{U}(t, t_0)^\dagger\right) \\ &= -\frac{i}{\hbar}H\mathcal{U}(t, t_0)\rho(t_0)\mathcal{U}(t, t_0)^\dagger + \frac{i}{\hbar}[H(t), \rho(t)]\mathcal{U}(t, t_0)\rho(t_0)\mathcal{U}(t, t_0)^\dagger \\ &= -\frac{i}{\hbar}[H(t), \rho(t)]. \end{aligned}$$

Hence the Schrödinger equation for the density operator is written as

$$i\hbar \frac{d}{dt}\rho(t) = [H(t), \rho(t)]. \quad (1.8)$$

Now we will derive an analogous equation for the density operator in interaction picture. Let us recall that a state ket in interaction picture is defined as

$$|\alpha, t_0; t\rangle_I = e^{\frac{i}{\hbar} H_0 t} |\alpha, t_0; t\rangle_S.$$

Hence the density operator in the Interaction Picture is

$$\rho_I(t) = \sum_i w_i |\psi_i, t_0; t\rangle_I \langle \psi_i, t_0; t|.$$

Then, differentiating it with respect to time, by equation (1.6) we obtain the following

$$\frac{d}{dt} \rho_I(t) = -\frac{i}{\hbar} V_I(t) \rho_I(t) + \frac{i}{\hbar} \rho_I(t) V_I(t).$$

Thus we see that the sought equation is

$$i\hbar \frac{d}{dt} \rho_I(t) = [V_I(t), \rho_I(t)]. \quad (1.9)$$

Just to be explicit, all of this is still valid for *closed* systems only. The next step is to generalize equations (1.8) and (1.9) to *open* systems.

For clarity purposes, throughout the remaining of the section we will set the value of the Plank's constant  $\hbar$  to 1.

In general terms [2], an open system is a quantum system  $S$  which is coupled to another quantum system  $B$  called the environment. It thus represents a subsystem of the combined total system  $S + B$ . In most cases the combined system is assumed to be closed and thus it follows the Hamiltonian dynamics as described above. However, the state of system  $S$  will change as a consequence of its internal dynamics and of the interaction with system  $B$ . These interactions lead to system-environment correlations such that the resulting state changes of  $S$  can no longer be represented in terms of unitary, Hamiltonian dynamics. These dynamics are often referred to as *reduced system dynamics*, and  $S$  is usually called the *reduced* system. The observable referring to  $S$  are all of the form  $\mathbf{A}_S \otimes \mathbf{I}_B$  [2]. If the state of the total system is described by some density matrix  $\rho$ , then the expectation values of all the observables acting on the open system's Hilbert space are determined through the formula

$$\langle \mathbf{A}_S \rangle = \text{Tr}_S \{ \mathbf{A}_S \rho_S \},$$

where

$$\rho_S = \text{Tr}_B\{\rho\}$$

is the reduced density matrix of the open quantum system  $S$  obtained by tracing out the environment  $B$  [15]. It is clear that the reduced density operator  $\rho_S$  will be the quantity of central interest in the description of open quantum systems. Again, the reduced density operator  $\rho_S$  at time  $t$  is deduced by the density operator  $\rho(t)$  tracing over the degrees of freedom of the environment. Since the total density operator evolves unitarily, following equation (1.8), we have

$$\rho_S(t) = \text{Tr}_B\{\mathcal{U}(t, t_0)\rho(t_0)\mathcal{U}^\dagger(t, t_0)\}.$$

In an analogous way the equation describing the time evolution of  $\rho_S$  is

$$i\frac{d}{dt}\rho_S(t) = \text{Tr}_B\{[H(t), \rho(t)]\}.$$

where  $H$  is the Hamiltonian of the closed system  $S + B$  [2].

Unfortunately, in general, the dynamics of the reduced system defined by these exact equations will be quite complicated, if not intractable. However, under certain hypothesis on the interactions system-environment, meaningful approximations can be performed leading to more tractable models. In particular, in [2], is shown how to derive the Lindblad Equation both in the Schrödinger picture and in the Interaction Picture.

The first one reads as

$$\frac{d}{dt}\rho(t) = -i[H(t), \rho_S(t)] + \mathcal{D}(\rho_S(t))$$

and is derived by the general theory of quantum dynamical semigroups. Here  $H$  denotes the Hamiltonian of the closed system  $S + B$ ,  $\rho_S$  is the reduced density operator seen in the Schrödinger picture and  $\mathcal{D}(\rho_S(t))$  is called the *dissipator*. We will not go into details about the explicit form of the dissipator, this can be found in [2]. We will just say that, as is simple to understand, this term approximates the effects that the interactions system-environment have over the evolution of the system  $S$ .

Instead, the Lindblad Equation in the Interaction Picture is written as

$$\frac{d}{dt}\rho(t) = -i[H_{LS}(t), \rho_S(t)] + \mathcal{D}(\rho_S(t))$$

where  $\rho_S(t)$  is the reduced density operator of system  $S$  seen in the Interaction Picture, whilst  $H_{LS}$  is the so called *Lamb shift* Hamiltonian and  $\mathcal{D}(\rho_S(t))$  is the dissipator. Again, a detailed derivation of the equation through microscopic derivations and the explicit form of the Hamiltonian and the dissipator can be found in [2].

Following [15] the Lindblad Equation implemented in QDOS is given by

$$\frac{d}{dt}\rho_I(t) = -i[V_I(t), \rho_I(t)] + \sum_j \gamma_j \left( 2L_j \rho_I(t) L_j - \{L_j^\dagger L_j, \rho_I(t)\} \right). \quad (1.10)$$

where the  $L_j$  operators are called the *Lindblad Operators* and  $\{\cdot, \cdot\}$  is the anticommutator of its arguments. In all our simulations we have focused on generic spin systems affected by *pure-dephasing* errors. Such errors correspond to an exponential decaying of the off-diagonal elements of the reduced density operator  $\rho_S(t)$ . In this case, the Lindblad Equation governing the time evolution of our quantum computer based upon a spin  $S$  is written as [26]

$$\frac{d}{dt}\rho_I(t) = -i[V_I(t), \rho_I(t)] + \sum_{n=1}^N \frac{1}{T_{2n}} \left( 2S_{z_n} \rho_I(t) S_{z_n} - \{S_{z_n}^2, \rho_I(t)\} \right). \quad (1.11)$$

Here  $H_{LS}(t) = V_I(t)$  that is the potential in Interaction Picture driving the computations,  $\rho_I(t)$  is the density operator representing, in Interaction Picture, the state of our quantum computer. Lastly, the dissipator is given by the summation, where  $N$  is the number of qudits of the quantum computer and  $S_{z_n}$  spin operator on the  $n$ -th qudit. The  $T_2$  parameter is characteristic of each qudit, and gives info on the length of the coherence of the qudit itself.

With equation (1.10) we have now a model able to describe the evolution of the quantum computer both under an external potential and with environment interactions.

## 1.5 Decomposition of $SU(n)$ in Planar Rotations

We know how to simulate the temporal evolution of a quantum computer under external potential and interacting with the environment. What is left is:

- How, given a certain unitary operator  $U \in SU(n)$  on the qudit space we can actually implement it on the quantum computer. In simpler words, how we can translate  $U$  into a sequence of pulses  $V_i$  such that their application to the quantum system will perform the unitary  $U$ .
- How we can simulate a measurement of the quantum system described by a density operator  $\rho(t)$ . This is fundamental in at least two occasions. First, at the end of a quantum program, we would like to know what the results of the program itself are. Then, if we are interested in performing Quantum Error Correction, measurements of subsets of qudits are also necessary.

In this Section we will take care of the first point, illustrating an algorithm of decomposition [8] of  $SU(n)$ . This will enable us to decompose any  $U \in SU(n)$  in a product of operators easy to implement on a quantum computer. Then, in Section 1.6 we will illustrate generalized measurements of a quantum system [11].

The decomposition we are about to show is a known fact in linear algebra [8]. It provides a simple and explicit way to parameterize the group  $SU(n)$ .

A *planar rotation* in  $SU(n)$ ,  $U_{j,k}(\theta, \beta)$ ,  $j < k$ , i.e. an  $n \times n$  matrix which is equal to the identity except for the elements at the intersection between the  $j$ -th and  $k$ -th rows and columns which are occupied by

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta)e^{-i\beta} \\ \sin(\theta)e^{i\beta} & \cos(\theta) \end{pmatrix}.$$

An important property of planar rotations is

$$U_{j,k}^{-1}(\theta, \beta) = U_{j,k}^\dagger(\theta, \beta) = U_{j,k}(-\theta, \beta) = U_{j,k}(\theta, \beta + \pi).$$

We shall set

$$D(\alpha_1, \alpha_2, \dots, \alpha_{n-1}) \doteq \text{diag}(e^{i\alpha_1}, e^{i\alpha_2}, \dots, e^{i\alpha_{n-1}}, e^{-i\sum_{l=1}^{n-1} \alpha_l}).$$

Then the following holds,

**Theorem 1.2** (Decomposition of  $SU(n)$ ). *For any matrix  $X \in SU(n)$  there exist  $n - 1$  parameters,  $\alpha_1, \dots, \alpha_n$ ,  $\frac{n(n-1)}{2}$  parameters,  $\theta_1, \dots, \theta_{\frac{n(n-1)}{2}}$  and  $\frac{n(n-1)}{2}$  parameters,  $\beta_1, \dots, \beta_{\frac{n(n-1)}{2}}$ , such that*

$$\begin{aligned} X = & D(\alpha_1, \dots, \alpha_n) \times \\ & U_{1,2}(\theta_1, \beta_1) U_{1,3}(\theta_2, \beta_2) U_{2,3}(\theta_3, \beta_3) \cdots \times \\ & U_{1,n}(\theta_{\frac{(n-1)(n-2)}{2}+1}, \beta_{\frac{(n-1)(n-2)}{2}+1}) \cdots U_{n-1,n}(\theta_{\frac{n(n-1)}{2}}, \beta_{\frac{n(n-1)}{2}}) \end{aligned}$$

*Proof.* Consider a matrix  $X \in SU(n)$  and a planar rotation  $U_{n-1,n}(\bar{\theta}, \bar{\beta})$ . Multiplicating  $X$  on the right with  $U_{n-1,n}(\bar{\theta}, \bar{\beta})$  only affects the  $(n-1)$ -th and the  $n$ -th columns of  $X$ . In particular we select  $\bar{\theta}$  and  $\bar{\beta}$  so that element  $(n, n-1)$  of  $XU_{n-1,n}(\bar{\theta}, \bar{\beta})$  is zero. This is always possible. Indeed, if  $X_{n,n} = 0$ , then we can choose  $\bar{\theta} = \pi/2$  and  $\bar{\beta} = 0$ . Then suppose  $X_{n,n} \neq 0$  and set

$$\begin{cases} A = \frac{\text{Re}(X_{n,n-1})\text{Re}(X_{n,n}) + \text{Im}(X_{n,n-1})\text{Im}(X_{n,n})}{\|X_{n,n-1}\|^2} \\ B = \frac{\text{Re}(X_{n,n-1})\text{Im}(X_{n,n}) - \text{Im}(X_{n,n-1})\text{Re}(X_{n,n})}{\|X_{n,n-1}\|^2} \end{cases}.$$

Thus one get the desired values of  $\bar{\theta}$  and  $\bar{\beta}$  as follows

$$\begin{cases} (\bar{\theta}, \bar{\beta}) = (\arctan(\frac{1}{B}), \frac{\pi}{2}) & \text{if } A = 0 \text{ and } B \neq 0 \\ (\bar{\theta}, \bar{\beta}) = (\arctan(-\frac{1}{A}), 0) & \text{if } A \neq 0 \text{ and } B = 0 \\ (\bar{\theta}, \bar{\beta}) = (\arctan(-\frac{1}{A \cos(\bar{\beta})}), \arctan(-\frac{B}{A})) & \text{otherwise.} \end{cases}$$

Now, consider multiplicating,  $XU_{n-1,n}$  by a planar rotation  $U_{n-2,n}(\tilde{\theta}, \tilde{\beta})$  on the right. This only affects the  $(n-2)$ -th and  $n$ -th columns of  $XU_{n-1,n}$ . In particular, it does not affect the zero in the position  $n, n-1$  which was

previously introduced.

Again, we choose  $(\tilde{\theta}, \tilde{\beta})$  so that the matrix  $XU_{n-1,n}U_{n-2,n}(\tilde{\theta}, \tilde{\beta})$  has a zero in position  $(n-2, n)$ . Such values for  $(\tilde{\theta}, \tilde{\beta})$  always exist and can be computed as shown above for the first step. Iterating, we transform the  $n$ -th row of the matrix in all zeros except the  $n$ -th element. Since all the matrices involved in the process are unitary it follows that also the product must be unitary. Hence also the  $n$ -th column is made of all zeros except the last element. Moreover, the non-zero element, i.e., the element in position  $(n, n)$  must have magnitude equal to one. Then we are left with a matrix of the form

$$XU_{n-1,n} \cdots U_{1,n} = \begin{pmatrix} \tilde{X}_{(n-1) \times (n-1)} & 0 \\ 0 & e^{i\eta} \end{pmatrix}.$$

The thesis trivially follows by iteration of the argument on  $\tilde{X}$ .  $\square$

What is left to understand is how to implement planar rotations and diagonal operators such  $D(\alpha_1, \dots, \alpha_n)$  on a quantum computer. It turns out to be pretty simple, and to show that let us consider the example of a spin qubit, i.e. of a quantum system with two degrees of freedom. Suppose we want to apply the planar rotation  $U_{1,2}(\theta, \beta)$ . Then, to implement the rotation, one needs to generate a linearly oscillating magnetic field of the form

$$V(t) = K \cos(\omega_{12}t - \beta) \sigma_y$$

for a temporal span of  $2\theta$  where  $\omega_{1,2}$  is the Rabi's frequency between  $|0\rangle$  and  $|1\rangle$  for the specific qubit. Instead, if we would perform a dephasing between  $|0\rangle$  and  $|1\rangle$ , i.e. apply an operator of the form

$$\begin{pmatrix} e^{i\alpha} & 0 \\ 0 & e^{-i\alpha} \end{pmatrix},$$

we just need to apply the following two planar rotation,  $U_{12}(\pi, \pi)$  and  $U_{12}(\pi, \alpha)$ .

The procedure shown for a qubit remains basically unchanged for qudits. Indeed, the only difference is that, with a potential of  $S_y$  form we can directly perform operations between energetically near states of the qudit. In other

words, if, for example, we want to perform a planar rotation between the states  $|0\rangle$  and  $|2\rangle$  of the same qudit, we can do it only after the planar rotation  $U_{|0\rangle,|1\rangle}(\pi, \pi)$ . This rotation will exchange state  $|0\rangle$  with state  $|1\rangle$ . Then one would perform the original rotation between states  $|1\rangle$  and  $|2\rangle$ . Finally,  $U_{|0\rangle,|1\rangle}(\pi, 0)$  will restore the correct order of the levels of the qudit.

A remark on the decomposition algorithm presented above before proceeding to the discussion of generalized measurements is mandatory. As one can easily see from Theorem [1.2](#), in the general case, the number of rotations in which a unitary operator is decomposed goes as the square of the space dimension  $n$ . The problem comes to light when we recall that if our space is composed by qudits, each with  $d$  levels, then the dimension of the space is  $n = d^N$  where  $N$  is the number of qudits. This implies that, if we apply the above algorithm to operators involving a big number of qudits, the exponential speed-up that a quantum computer could give over a classical one would be completely annihilated by this exponential number of operations that we would be forced to perform. For this same reason we will apply the algorithm to only one- and two- qudit gates. This is not a serious constraint since it is a known fact that one- and two- qudit gates are universal, i.e. they are enough to perform, with arbitrary precision, any unitary operator on a quantum computer [\[15\]](#).

## 1.6 Generalized Measurements

A mathematical formulation of the simplest form of measurement was given by J. von Neumann, and we shall refer to measurements of this type as *von Neumann measurements* or projective measurements. Let  $\mathcal{A}$  be an observable quantity and  $\mathbf{A}$  its associated Hermitian operator. Call  $|a_n\rangle$  and  $a_n$  the eigenvectors and the eigenvalues of  $\mathbf{A}$  respectively, i.e.

$$\mathbf{A} = \sum_n a_n |a_n\rangle \langle a_n|.$$

Then, the probability that a measurement of  $\mathbf{A}$  will give result  $a_n$  is [\[1\]](#)

$$P(a_n) = \langle a_n | \rho | a_n \rangle = \text{Tr}\{\rho |a_n\rangle \langle a_n|\},$$

where  $\rho$  is the density operator describing the state of the quantum system immediately prior to the measurement. Introducing the projector operator  $P_n \doteq |a_n\rangle\langle a_n|$  we can rewrite the formula for the probability of a certain observation as

$$P(a_n) = \text{Tr}\{\rho P_n\}. \quad (1.12)$$

This allows us to deal in a straightforward manner with the possibility that the eigenvectors of  $\mathbf{A}$  may be degenerate. Suppose that each eigenvalue of  $\mathbf{A}$  has degeneracy  $j_n$ , i.e. there exists  $|a_n^{(j)}\rangle$  such that  $\forall j = 1, \dots, j_n$

$$\mathbf{A}|a_n^{(j)}\rangle = a_n|a_n^{(j)}\rangle.$$

For all  $a_n$  we define

$$P_n \doteq \sum_{j=1}^{j_n} |a_n^{(j)}\rangle\langle a_n^{(j)}|$$

from which follows that the probability of observing the value  $a_n$  is again

$$P(a_n) = \text{Tr}\{\rho P_n\}.$$

What is left to understand is how a measurement affects the state of the system, i.e. how the state of the system will be after a projective measurement. Again, the mathematics depicting such state change is quite easy. Indeed, a measurement of  $\mathbf{A}$  will be accompanied by a change in the density operator of the form [1]

$$\rho \longrightarrow \rho' = \frac{P_n \rho P_n}{\text{Tr}\{P_n \rho P_n\}}. \quad (1.13)$$

Then, by the cyclic property of the Trace operator [15]

$$\text{Tr}(AB) = \text{Tr}(BA)$$

and by the idempotence of the projector operator

$$P_n^2 = \sum_{j=1}^{j_n} |a_n^{(j)}\rangle\langle a_n^{(j)}| \sum_{k=1}^{j_n} |a_n^{(k)}\rangle\langle a_n^{(k)}| = \sum_{j,k=1}^{j_n} |a_n^{(j)}\rangle \delta_j^k \langle a_n^{(k)}| = P_n,$$

we can rewrite (1.13) as

$$\rho \longrightarrow \rho' = \frac{P_n \rho P_n}{P(a_n)}. \quad (1.14)$$

The von Neumann description of a measurement is insufficiently general for the simple reason that most observations that we can perform are not of this type. The real world is noisy, as are quantum computers of today, and this ensures that our observation will include errors. Now we will consider the effects of noise-induced errors on ideal von Neumann measurements. Let  $i$  denote the outcome of a (hypothetical) von Neumann measurement and  $r$  denote the outcome of the real measurement. An ideal von Neumann measurement of the observable  $\mathbf{A}$  will give one of the results  $\{a_n\}$  with probabilities calculated using equation (1.12), that is

$$P(i = a_n) = \text{Tr}(\rho P_n).$$

The statistical errors associated with the operation of the measuring device are described by the set of conditional probabilities  $P(r = a_m | i = a_n)$ . This is the probability that the measurement gives the result  $a_m$  given that an ideal measurement would have given  $a_n$ . Bayes rule then gives the probability that the measured result is  $a_m$

$$\begin{aligned} P(r = a_m) &= \sum_n P(r = a_m | i = a_n) P(i = a_n) \\ &= \sum_n P(r = a_m | i = a_n) \text{Tr}(\rho P_n). \end{aligned}$$

Then, by introducing the operators

$$\pi_m \doteq \sum_n P(r = a_m | i = a_n) P_n,$$

we can rewrite the probability of observing  $a_m$  as

$$P(r = a_m) = \text{Tr}(\rho \pi_m). \quad (1.15)$$

Let us show an example of how all of this will translate for a non ideal von Neumann measurement of a qubit in our simulator. The operators  $\pi_j$  would simply become

$$\begin{aligned} \pi_0 &= (1 - p)P_0 + pP_1 = (1 - p)|0\rangle\langle 0| + p|1\rangle\langle 1|, \\ \pi_1 &= (1 - p)P_1 + pP_0 = (1 - p)|1\rangle\langle 1| + p|0\rangle\langle 0|. \end{aligned}$$

Hence, the probabilities for the two outcomes of a measurement are

$$\begin{aligned} P(0) &= \text{Tr}(\rho\pi_0), \\ P(1) &= \text{Tr}(\rho\pi_1). \end{aligned}$$

So, the first step in the simulation process would be to compute the operators  $\pi_j$ . Then we compute the correct probabilities for any possible observable value (in the case of a qubit, 0 and 1). We pseudo-randomly choose one of the observable values according with the distribution found. Once we have the value of the observation, following the distribution that builds the respective  $\pi_j$  operator, we choose on which eigenspace to project the density operator, i.e. we choose which operator  $P_k$  to apply. At last, with the chosen operator  $P_k$ , we calculate the new density operator following equation (1.14).

Having described the measurement process we have covered all the theoretical foundations of the simulator. In the next Chapter, we shift the focus from the simulator to the actual program and to the programming language realized for it.



## Chapter 2

# QDOS Manual

In this Chapter we discuss QDOS, the software object of this Thesis. We can divide it into three main components: a simulator of generics qudits, a Domain Specific Language (DSL) for writing quantum programs then executed by the simulator and an optimizing compiler interfacing the simulator and the DSL. The simulator is a solver of a system of ordinary differential equations. In particular, the ODE system that we want to solve is given by equation (1.10). The compiler is a straightforward application of Theorem 1.2. Having widely discussed the theoretical background of both the simulator and the compiler in Chapter 1, here we concentrate on the programming language developed and on the general use of QDOS.

### 2.1 The QDOS Programming Language

In this section we illustrate the QDOS programming language. It has been developed by analogy to OpenQASM (Open Quantum Assembly)<sup>1</sup> but with some clear distinctions. As should be already clear from Chapter 1, quantum computers are intrinsically susceptible to errors due to interaction with the environment. Then, to make quantum computing realizable, at least in the near terms, schemes of Quantum Error Correction have to be implemented.

---

<sup>1</sup>Open Quantum Assembly Language (OpenQASM; pronounced open kazm) is an intermediate representation for quantum instructions. The language was first described in a paper published in July 2017, and a reference source code implementation was released as part of IBM's Quantum Information Software Kit (Qiskit) for use with their IBM Quantum Experience cloud quantum computing platform. The language has similar qualities to traditional hardware description languages such as Verilog.

One of the reasons to develop a new programming language and not keep OpenQASM was to make the implementation of such schemes easier.

The generic QEC code has the following structure

1. Encoding of the local qubits
2. Perform some of the circuit operations
3. Error detection
4. Error correction
5. Iterate step 2. to 5. until all the operations of the quantum circuit have been executed.

The steps of error detection and error correction are repeated many times through the execution of the circuit, and while the error detection step is always the same, the error correction one, in general, depends heavily on the outcome of the detection step. Moreover, the detection step is based on the measurement of the part of the quantum system performing the computations and then by the analysis of these measurements. The analysis step usually requires a lot of classic informatics to be carried out.

OpenQASM, as the name suggests, is a low level programming language that, even in its most spread version (2.0), does not permit either the definition of sub-circuits to be repeated in a quantum circuit or the call to classical algorithms, inside the circuit itself, to be performed at execution time.

Moreover, the goal of the developed software was to simulate arbitrary operations among generic qudits. OpenQASM unfortunately is limited to qubit. Hence we needed a new and more abstract programming language. Such language should be focused on qudits. It should give the possibility to perform measurements during the execution of a circuit. But most importantly it should be able to schedule operations based on such measurements results.

If we add these two shortcomings of OpenQASM to the fact that it is a language for qubits only, the need for a new programming language, more flexible and more powerful, based on generic building blocks (the qudits) and able to interface standard quantum circuit operations with classical algorithms, should be more than clear.

We now illustrate QDOS grammar. A plain version of it can be found in Appendix A. For sake of clarity we will present the grammar free from any semantics, which can be found in the source code of the parser of the language. For the same reason we will present the lexical syntax of all the `TOKEN` of the grammar at the end of the Chapter.

A standard QDOS program consists of definitions, as shown in the following snippet of its grammar

```
list:      /* Nothing */
          | list ';'
          | list def ';'
          | list error ';'
          ;
```

A definition in QDOS is written as

```
def:       QUDIT '[' INTEGER ']' '=' '{' integer_sequence '}'
          | GATE '[' INTEGER ']' '=' INDEX
          | NAME '=' '{' inst_sequence '}'
          | NAME '=' INTEGER
          | NAME '=' FLOAT
          ;
```

The first production describes the definition of an array of qudits used through the QDOS program. The `integer_sequence`, as the name suggests, is a sequence of integer numbers that defines the dimensions of the qudits used. The dimension of each qudit is arbitrary but must be greater than 1.

The second production describes the definition of a *custom gate*. Gates are the basic operations used to define a quantum circuit (i.e., a quantum program). The `INDEX` is the path to the file containing the operator describing the custom gate. We will explain in detail how to define a custom gate in Section [2.3](#).

The last two productions are for the definition of variables that can hold integer or floating point values. The third one is the definition of a block of instructions where each instruction is obtained as

```
inst:      GATE '[' integer_sequence ']' '(' float_sequence ')' ';'
          | GATE '[' integer_sequence ']' ';' ;
```

```

| MEASUREMENT '[' integer_sequence ']' ';'
| CUSTOM_LOGIC NAME ';'
| NAME ';'
;

```

Here the first two productions are for gates: the first one for parametric gates the second for non-parametric ones (see Section 2.2). The integer numbers inside the square brackets are the indexes of the qudits on which the gate operates, while the floating point numbers inside the round brackets are the parameters of the gate. The third production is for the measurement of the qudits listed as the argument of the instruction. The forth one is what really makes QDOS different from OpenQASM: the call (at runtime) of an arbitrary C function of name `NAME`. These functions are called `custom_logic` since they have been introduced with the clear idea to enable the final user to perform custom operations on measurement outcomes and to be able to correctly detect and correct errors. We will describe in details both measurement and custom logic functions in Section 2.4. The latter allows us to perform code nesting by inserting the execution of instruction blocks inside other instruction blocks.

Here I anticipate a fundamental feature of QDOS, that will be also stressed in Section 2.5. The entry point of any QDOS program is a block called **main**, and all the definitions used inside a block of instructions must be defined before their use (this applies also to blocks themselves).

We stop here the discussion of the grammar of QDOS since we have already covered all the principal productions. This clearly shows that the QDOS programming language is still incredibly simple but also much more flexible than a standard assembly language, such as OpenQASM. A full version of the grammar of the QDOS programming language can be found in Appendix A.

At last we report the regular expressions describing the syntax of grammar tokens.

DIGIT	[0-9]	
INT_NUMBER	-?{DIGIT}+	--> INTEGER
FLOAT_NUMBER	-?{INT_NUMBER}*"."{INT_NUMBER}	--> FLOAT
GATE_NAME	\${[A-Z][A-Z_]*[0-9]*}	--> GATE
FUNC_VAR_BLOCK_NAME	[a-z_]+[0-9_]*	--> NAME

STRING                                      \ "[a-zA-Z \_/"."] \* \ "                                      --> INDEX

Then we have the keywords **q** for the QUDIT token, **custom\_logic::** for the CUSTOM\_LOGIC token and **measure** for the MEASUREMENT token.

Now that we are conscious of the general structure of a QDOS program, we describe in detail all of its components, starting from gates.

## 2.2 Built-in Gates

In this Section we illustrate which gates are built-in in QDOS. These have been carefully chosen in order to provide a universal set. With universal we mean that such gates are enough to decompose any unitary operator on the quantum computer.

The first two gates that we will present are the rotations of Theorem [1.2](#). Specifically we call these two gates \$RP and \$RZ. They are both parametric gates and are declared in the following way:

\$RP[int  $idx_{q_0}$ ](int  $lv_0$ , int  $lv_1$ , float  $\theta$ , float  $\beta$ );  
 \$RZ[int  $idx_{q_0}$ ](int  $lv_0$ , int  $lv_1$ , float  $\phi$ );

An \$RP is a simple planar rotation. Hence, it will apply on qudit  $idx_{q_0}$  an identity operator except in the intersection of the  $lv_0$ -th and  $lv_1$ -th rows and columns that are occupied by the following

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta)e^{-i\beta} \\ \sin(\theta)e^{i\beta} & \cos(\theta) \end{pmatrix}.$$

Similarly, an \$RZ will apply on qudit  $idx_{q_0}$  an identity operator except in the intersection of the  $lv_0$ -th and  $lv_1$ -th rows and columns that are occupied by the following

$$\begin{pmatrix} e^{i\phi} & 0 \\ 0 & e^{-i\phi} \end{pmatrix}.$$

With these two gates we are able to perform any single-qudit unitary gate, independent of its dimension (cfr. Theorem [1.2](#)). All the other single qudit gates built-in in QDOS have been included to increase the level of abstraction. These simplify the writing of qudit circuits and are the straightforward

generalization of the most common gates for qubit based quantum computing. Such single qudit gates are

$$\begin{aligned} &X[\text{int } idx_{q_0}]; \\ &Z[\text{int } idx_{q_0}]; \\ &H[\text{int } idx_{q_0}]; \\ &S[\text{int } idx_{q_0}]; \end{aligned}$$

For a qubit, such gates are represented by the matrices

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

in the standard computational basis given by the eigenstates of  $Z$  (i.e.,  $|0\rangle$  and  $|1\rangle$ ). For the remainder of the section, all the matrix representation of gates will be written in this basis. The qudit generalizations of these gates are [11, 17]

$$\begin{aligned} X &= |j \oplus_d\rangle \langle j| & H &= \frac{1}{\sqrt{d}} \sum_{j,k=0}^{d-1} \omega^{jk} |k\rangle \langle j| \\ Z &= \omega^j |j\rangle \langle j| & S &= \sum_{j=1}^{d-1} \omega^{j(j+1)/2} |j\rangle \langle j| \end{aligned}$$

where  $d$  is the dimension of the qudit ( $d > 2$ ) and  $\omega \doteq e^{\frac{2\pi i}{d}}$ . For example, if  $d = 4$ , the above gates individuate the following operators

$$\begin{aligned} X &= \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, & H &= \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}, \\ Z &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{\frac{\pi i}{2}} & 0 & 0 \\ 0 & 0 & e^{\pi i} & 0 \\ 0 & 0 & 0 & e^{\frac{3\pi i}{2}} \end{pmatrix}, & S &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & e^{\frac{\pi i}{2}} & 0 & 0 \\ 0 & 0 & e^{\frac{3\pi i}{2}} & 0 \\ 0 & 0 & 0 & e^{3\pi i} \end{pmatrix} \end{aligned}$$

Ended the description of single qudit gates, we now focus on two-qudit gates. To achieve general universality [15] on qudits, usually to universal single

qubit gates is associated the CNOT gate, also called CX. Such a gate has the following syntax in QDOS

$$\text{CX}[\text{idx}_{q_0}, \text{idx}_{q_1}];$$

and, when applied to qubits  $q_0$  and  $q_1$ , gives the matrix

$$\text{CX} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

that simply flips the states of the target qubit ( $q_1$ ) if the control ( $q_0$ ) is in state  $|1\rangle$ . In fact, this is a conditional X applied to  $q_1$  when  $q_0$  is in  $|1\rangle$ . This makes its generalization intuitive, i.e. we apply a generalized X to  $q_1$  if  $q_0$  is in  $|d-1\rangle$ . This is indeed correct. Thus we define the generalized CX in QDOS to be

$$\begin{cases} |j, k\rangle \rightarrow |j, k\rangle & \text{if } j < d-1 \\ |j, k\rangle \rightarrow |j, k \oplus_d 1\rangle & \text{if } j = d-1 \end{cases}$$

An example of the operator associated with a generalized CX could be the following

$$\text{CX} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix},$$

if the control qudit has dimension  $d_0 = 2$  and the target qudit has dimension  $d_1 = 4$ . Moreover, it turns out [17] that this generalization of the CX gate associated with the other built-in gates available in QDOS form a universal set of gates. Hence, any possible quantum circuit can be written with the gates discussed so far.

Along with this set of universal gates we have decided to implement other two two-qudit gates. The CZ gate, that works with the same logic of a CX

but replaces the conditional X with a conditional Z, and the CEX gate. This last gate has the following syntax

$$\text{CEX}[idx_{q_0}, idx_{q_1}](\text{int } lv_c, \text{int } lv_0, \text{int } lv_1);$$

and has the effect of exchanging the  $lv_0$  and  $lv_1$  levels of the target qudit  $idx_{q_1}$  when the control qudit  $idx_{q_0}$  is in the state  $lv_c$ . Hence, it is formally defined as

$$\begin{cases} |lv_c, lv_0\rangle \leftrightarrow |lv_c, lv_1\rangle \\ |j, k\rangle \rightarrow |j, k\rangle & \text{if } j \neq lv_c, k \neq lv_0, lv_1 \end{cases}.$$

With CEX we have nearly finished the list of built-in gates of QDOS. Indeed there are still two built-in gate to be discussed. These gates are fundamental, respectively, for an easy implementation of the Quantum Fourier Transform and an meaningful verification of Embedded Quantum Error Correction codes for quantum memories. Hence we leave their discussion to the next Chapter, in which both of these subject will are discussed. In Appendix B is reported the complete list of the QDOS built-in gate, one should refer to it as a reference index.

In the next Section we will show how one can define a custom gate on the fly without needing to modify the source code of QDOS to add a new built-in.

## 2.3 How to Define a Custom Gate

To figure out how a custom gate can be defined we first need to understand how operations are represented in QDOS. The classical flow of information in QDOS can be viewed as a three steps process.

1. Take a unitary operator as input.
2. Decompose it, following Theorem [1.2](#), in a sequence of rotations, both planar and Z.
3. Translate such a sequence of rotations into a sequence of parallel resonant pulses able to drive the evolution of the physical system simulated in the desired way (this step strongly depends on the physical system being simulated, more on this in Section [2.5](#)).

As one can understand, step one, i.e. the input step, can be shifted to any other point of the translating routine. In other words, our algorithm can take in input a sequence of rotations or a sequence of pulses, instead of a generic unitary. Hence, when I have interfaced the simulator with the programming language I have decided to give a particular semantics to gate operators. Specifically, a gate operator inside QDOS can be a unitary matrix, a sequence of rotations or a sequence of pulses.

While I discourage to define gate operators as sequences of pulses, since those are heavily dependent on the hardware simulated (more on this in Section 2.5), the other two options are safely usable, as they are completely independent from the physical system being simulated. Moreover, in this manual I will not go into details on how to define proper sequence of pulses. Nevertheless, I leave the possibility to do it since, for the most experienced, could be useful to directly try pulses on the hardware.

At last, as a useful (not forced) convention, I suggest to differentiate the possible kind of gates with the '\$' symbol. In particular:

- No '\$' before the gate name implies that the gate will be represented by a unitary operator.
- A single '\$' before the gate name indicates that the gate operator associated to the gate is a sequence of rotations.
- Two '\$' before the gate name imply a sequence of pulses for the gate operator.

This convention applies also to the builtin gates of QDOS described in Section 2.2. This explain the reason for the '\$' in the name of the \$RP and \$RZ gates. Both of them will directly generate a sequence of rotations. In this specific case, they will generate a sequence consisting of only one rotation, a planar rotation and a Z rotation, respectively.

After this small premise, let's now see how to define a custom gate. The process is quite simple. It is rooted into the creation of a binary file containing the appropriate information. The first information that has to be stored is the code representing the kind of gate operator. Specifically, you need to write `sizeof(size_t)` (which is platform dependent) bit and, inside these bits you have to write the correct code: 0 for a unitary operator, 1 for a

sequence of rotations or 2 for a sequence of pulses.

If you want to define a custom gate with an arbitrary unitary operator you just need to write the unitary matrix representing it on the file. Be careful to write 64-bit long floating point numbers both for the real and imaginary parts of the matrix. Such a matrix is read with the gsl interface `gsl_matrix_complex_fread`. Particular attention must be given to checking that the matrix written on file is a special unitary matrix ( $\in SU(n)$ ), if not QDOS will warn about it.

If we want to define the gate operator as a sequence of rotations the procedure is the following. After the code of the gate operator one must write the number of rotations that make up the sequence in exactly `sizeof(size_t)` bit. Then, for any rotation it must be written again in `sizeof(size_t)` bit the number 0 for a planar rotation and the number 1 for a Z rotation. After this code has been written, one needs to write on file the parameters of the respective rotations, that are simply the parameters requested by the \$RP and \$RZ gates. For both kind of rotations one will write the indexes of the two levels on which to apply the rotation (indexes start always from 0). Again such indexes must be written in `sizeof(size_t)` bit. After that, for a Z rotation one will write a 64-bit long floating point number for the angle of rotation whilst for a planar rotation with the same format are requested the  $\theta$  and  $\beta$  parameters.

Once we have written correctly the gate operator in binary format on a file, one just needs to provide a name for the gate, let's say for example \$CG, and define it in the QDOS circuit before any use. By doing so, the second production of definitions presented in Section 2.1 must be followed. This will basically consist in stating the gate name, the number of qudit on which it operates and the path to the file, that can be either absolute or relative.

To give an example, suppose our custom gate operates among 2 qudit. Then, inside a QDOS circuit, it would be defined in the following way

```
$CG[2] = "./relative/path/to/file";
```

or also

```
$CG[2] = "/absolute/path/to/file";
```

We have now fully discussed one of the three instructions that a QDOS program is able to perform, gates. In the next section we will look into the remaining two: measurements and custom logics.

## 2.4 Measurements and Custom Logics

Measurements in QDOS are carried out as described in Section 1.6. Two are the important parameters of QDOS that are modifiable through command line options: the probability of measurement errors (see Section 1.6) and the number of consecutive measurements to keep in memory. Such options are shown in Appendix C.

Measurements outcomes will be saved in an abstract data structure called `lista_misure`. Such data structure will be provided as the only parameter when invoking any custom logic function. One should iterate through the measurement outcomes with the following interface.

```
/*
 * Interfaccia per recuperare la misura i-esima data una lista di misure.
 * L'ordinamento è decrescente in senso temporale, i.e., la misura 0
 * corrisponde all'ultima misurazione effettuata
 */
extern
int *get_misura(lista_misure *lst_mis, size_t idx);
```

Calling such a function on a list of measurement outcomes will return a pointer to the first elements of an integer array. This array contains a number of elements equal to the number of qudits in the simulated physical system. Any element of the array is equal to  $-1$  if the corresponding qudit has not been measured. Otherwise it contains the measurement outcome for that qudit (i.e., a number between  $0$  and  $d - 1$ ). The second argument of the interface is the index of the measurements to be retrieved. The ordering of measurements decreases in  $t$ , i.e., the latest measurement is available through index  $0$ . The effective number of measurements available in the list is stored in `lst_mis->dim_eff`.

The QDOS program provides then a usual feature to simulate quantum hardware: the possibility, at the end of the circuit, to perform a final measurement of the physical system an arbitrary number of times. To do this, one should modify the QDOS program behavior by the command line option

--n\_shots(more on this in Section 2.5). Lastly, measurements in QDOS are generalized measurements (see Section 1.6).

We have covered all the nuances of the measurement instructions in QDOS. Now we talk about custom logic instructions. When a custom logic is invoked in a QDOS circuit, the QDOS source code will look into the table of custom logic functions to find the requested function. If the search is positive then QDOS invokes the appropriate custom logic through its pointer, where a custom logic function pointer in QDOS is defined as follows:

```
typedef blocco_gate* (*custom_logic_ptr) (lista_misure *tabella_misure);
```

Hence, a custom logic function, given its input, will return a pointer to a data structure called **blocco\_gate**. This data structure is the abstraction with which a block of instructions is represented in the QDOS source code.

We leave out how a custom logic function determines the operations to perform depending on the measurements outcomes given as input. Now, we illustrate how one can define from scratch a new block of instructions directly from C code using the interface of the QDOS source code.

The first step in the process is to allocate the memory for a new **blocco\_gate** through the interface

```
/*
 * Interfaccia per l'allocazione di un blocco di gate
 */
extern
blocco_gate *alloc_blocco_gate(void);
```

Then one should give a name to the newly created block. Afterwards, one needs to fill the block of instructions with actual instructions. As should already be clear, in QDOS there are three different types of instructions: gates, measurements and custom logics. Inside a **blocco\_gate** all of those three can be inserted, but custom logics are not designed to schedule any new measurements or calls to other custom logics. Hence, we will limit to the description of how to insert gates inside a **blocco\_gate**.

The process is pretty standard and easy:

1. Search for the desired definition of gate through the following interface

```
/*
```

```

    *  Interfaccia per la ricerca di un gate all'interno della lista
    *  delle definizioni
    */
extern
gate *cerca_definizione(lista_definizioni_gate *lst, char *nome_gate);

```

where, as first parameters, one should give the global variable `lst_def` which contains all the defined gates (both custom and built-in) in the QDOS circuit. The second and last parameter is simply the name of the desired gate.

2. Initialize the arguments and parameters (if any) of the gate requested. For this purpose the following interface is made available

```

/*
 *  Funzione per l'allocazione dei parametri del gate
 */
extern
parametri_gate *alloc_parametri_gate(uint8_t n_qudit, size_t n_par);

```

where the first argument is the number of qudits on which the gate works and the second one is the number of parameters of the gate. Once verified the allocation of `parametri_gate` we initialize them. The field `size_t *idx_qudit` must contain the indexes of the qudits on which the gate will be applied while `double *arg` contains the parameters of the gate.

3. Allocate an appropriate instruction to be inserted into the block. To do this there is the following interface

```

/*
 *  Funzione per l'allocazione di un gate da eseguire. Prende
 *  come argomenti la definizione del gate da implementare ed i
 *  parametri del gate da eseguire
 */
extern
istruzione *alloc_istruzione_gate(gate *def_g, parametri_gate *par);

```

where the first argument is the gate definition retrieved and the second one the list of parameters properly initialized.

4. Insert the newly created instruction inside the list of instructions of the gate. For this purpose the following interface is made available

```

/*
 * Funzione per l'inserimento di un'istruzione all'interno di una
 * 'lista_gate'
 */
extern
void inserisci_istruzione(lista_gate *lst, istruzione *ist);

```

where the first argument will be the attribute `lista_gate *lst` of our `blocco_gate` while the second one is the allocated instruction.

Here I show an example of a possible code for the creation of a block containing only an Hadamard gate (H) on the first qudit.

```

blocco_gate *blk = alloc_blocco_gate();
/* Error checking omitted */
blk->nome_blocco = "prova_csl";

gate *g = cerca_definizione(lst_def, "H");
/* Error checking omitted */
parametri_gate *p = alloc_parametri_gate(1, 0);
/* Error checking omitted */
p->idx_qudit[0] = 0;

istruzione *ist = alloc_istruzione_gate(g, p);
/* Error checking omitted */
inserisci_istruzione(blk->lst, ist);

```

Files `custom_logic.c` and `custom_logic.h` are the places in which one should write all codes relative to custom logics.

Once we have defined a custom logic other three steps are required in order to be able to use it inside a QDOS circuit. The first one is to change the value of the macro definition `#define N_CSL` to reflect the number of custom logics to be seen by QDOS. Then it should be updated the global definition of the variable `custom_logic_definitions[N_CSL]` in the file `custom_logic.c`. This is the global table in which QDOS looks for the pointer of the custom logic to be executed in a circuit. It is pretty simple to initialize it in the correct way. Each entry requires a string with the name of the custom logic that we want to use in our circuits and the pointer to the custom logic. Thus, a possible initialization has the form

```

custom_logic custom_logic_definitions[N_CSL] =
{

```

```

...
{"custom_logic_name", &custom_logic_function},
...
};

```

Finally, one should recompile the code to make the new custom logic available. The procedure to do this is described in details inside the README and INSTALL files packed with the source code.

We have now fully discussed the QDOS programming language. In the next Sections we turn our attention at the description of physical systems to be simulated.

## 2.5 How to Set Up and Run Simulations

In this section we illustrate how to describe the qudits that make up our quantum computer. Then we will conclude showing how to set up all the parameters needed by QDOS in order to run a simulation.

QDOS requires, for any qudit defined inside a circuit, four main objects:

1. An Hermitian matrix, of size  $d \times d$ , where  $d$  is the dimension of the qudit, representing its Hamiltonian.
2. An Hermitian matrix, of size  $d \times d$ , representing the operator of the potential  $V$  with which we want to drive the evolution of the qudit. We refer to this matrix also as the *connection matrix* since it defines also how the levels of the qudit are connected to each other.
3. [OPTIONAL] A matrix, of size  $d \times d$ , representing the error operator affecting the qudit. Errors are disabled by default and can be enabled through the command line option `--errors_enabled`.
4. [OPTIONAL] A number representing the error parameter. Errors will be then added to the evolution of the system through equation (1.10), with the error operator  $L_j$  and the error parameter  $\gamma_j$ .

When dealing with spin systems as base for qudits, we need a way to be able to distinguish multi-qudits operations from single-qudit ones. For this reason, we have introduced in QDOS a mechanism to interconnect qudits which relies on a tensor  $J$ , also called the *connection tensor*. Such object

will affect the Hamiltonian of the physical system when gates among multiple qudits have to be executed. In particular, if a gate between qudit  $j$  and qudit  $k$  has to be performed, then a term of the form

$$J_{j,k} [S_{z_j} \otimes S_{z_k}].$$

is added to the system Hamiltonian.

$J$ , since we enable gates between no more than 2 qudits, can be represented by a matrix of dimension  $n \times n$  where  $n$  is the number of qudits defined in the circuit. Hence only one matrix representing  $J$  is needed for the physical system simulated. This interconnection mechanism can be disabled (it is enabled by default) through the command line option `--connection_enabled`.

The procedure to write on file a complex matrix is the same used for the definition of custom gates. The only difference is where one needs to write the data describing qudits. File names are fixed, the directory must be the same for all the files but the directory is arbitrary and can be customized with the command line option `--in_dir`. The names of the files that contain such information are the following:

- `in.H.bin` must contain, one after the other, the Hamiltonian of the qudits defined inside the QDOS circuit. These Hamiltonian ( $H_j$ ) will be used by QDOS to build the total Hamiltonian of the system

$$H_{\text{tot}} = \sum_j H_j + \sum_{k,l} \Theta_{kl}(t) J_{kl} \sigma_{z_k} \otimes \sigma_{z_l}$$

where  $\Theta_{kl}(t)$  is defined as

$$\begin{cases} \Theta_{kl}(t) = 1 & \text{if we are executing an operation between qudits } k \text{ and } l \\ \Theta_{kl}(t) = 0 & \text{otherwise.} \end{cases}$$

Hence, in the absence of multi-qudit gates being executed, the Hamiltonian simulated by QDOS is

$$\bar{H} = \sum_j H_j.$$

Order is important since the first matrix to be read is for the first qudit and so on.

- `in.M.bin` must contain, one after the other, the connection matrix ( $M_j$ ) for each qudit defined inside the QDOS circuit. The total connection matrix is obtained by:

$$M_{\text{tot}} = \sum_j M_j.$$

As for the Hamiltonians order is important.

- `in.E.bin` must contain, one after the other, all the error operators only if errors are enabled. Again order is important.
- `in.R.bin` must contain, one after the other, all the error parameters only if errors are enabled. Order is important.
- `in.J.bin` must contain the matrix representing the connection tensor  $J$  only if the interconnection mechanism is enabled.

To be able to feed-in correct data to the connections matrices, one should be aware on how those will be used by QDOS. This is pretty much straightforward. Let us call  $\Sigma$  the operator that the total connection matrix  $M_{\text{tot}}$  represents. Then to drive the evolution of qudit  $i$ , QDOS use a potential  $V$  defined as follows

$$V = \mathcal{G}(t) \cos(\omega_{nm}t - \beta) \Sigma$$

where  $\mathcal{G}(t)$  is a Gaussian modulation of the pulse and  $\omega_{nm}$  is the Rabi's frequency between level  $n$  and  $m$ . Then it consider  $V$  in interaction picture as shown in Section 1.3 and, as stated in Section 1.4, it use equation (1.10) to approximate the evolution of the physical system, either with or without errors.

Since we are ultimately solving a system of differential equations we need initial conditions, i.e. density matrix of the system before any operation. This density matrix has to be written inside the file `in.P.bin`. Moreover, in the file `in.P.bin` have to be written also another parameter, that I call the fire-up parameter. This parameter must be written as first in the file `in.P.bin`. It is a time in nanoseconds and it model the time of free evolution of the system between two consecutive pulses. It is a floating point number

of 64-bit and can be set to 0.

Having prepared all the input information necessary for QDOS we can launch a simulation by simply typing in a terminal the following prompt

```
$: qdos [options] circ.q
```

QDOS will then run and eventually warn if anything is wrong.

Once finished the simulation of the instructions listed in the circuit, as already mentioned in Section 2.4, one can ask QDOS to perform multiple measurements of the final density matrix obtained, thus simulating multiple executions of the circuit. In order to perform N of such measurements, one just needs to call QDOS with the command line option `--n_shots=N`. The outcome of these measurements will be printed by QDOS on standard output.

Lastly, QDOS can produce different kind of intermediate output on file. Specifically QDOS can print:

- the sequence of pulses effectively simulated,
- the resulting final density matrix,
- the temporal evolution of the density matrix.

All of this output information can be controlled through appropriate command line options. Such options are shown in Appendix C. As for inputs, outputs file are located in the same directory and this can be set through the command line option `--out_dir`.

With this we have concluded our QDOS treatment. Use cases are left for the next Chapter in which we use QDOS to simulate actual quantum circuits.

## Chapter 3

# QDOS Use Cases

In this chapter we use QDOS to simulate two prototypical algorithms, namely the QFT and a QEC protocol on qudits. Indeed, QFT is at the heart of many quantum algorithms, such as the Shor’s factoring algorithms. We derive the QFT for qubits and then we generalize it for qudits. Then we implement it in QDOS. In the second part of the Chapter we consider, as a different application, an example of Embedded Quantum Error Correction on qudits. We report logical states, encoding and correction procedures as proposed in [3], then we simulate them with QDOS. This highlights the strengths of QDOS: being able to perform partial measurements throughout the execution of a circuit, and the possibility to schedule conditional code based on the outcome of these measurements.

### 3.1 Quantum Fourier Transform

In this Section we present the *Quantum Fourier Transform* for qubits and its generalization to qudits. The QFT is the key ingredient for quantum factoring and many other interesting quantum algorithms. It is an efficient quantum algorithm to perform a Fourier transform of quantum mechanical amplitudes. For instance, QFT is the fundamental subroutine of the quantum *phase estimation* algorithm, i.e. the approximation of the eigenvalues of a unitary operator [15]. This allows us to solve several other interesting problems, including the *order-finding problem* and the *factoring problem* (through Shor’s algorithm) [15]. Moreover, the Quantum Fourier Transform

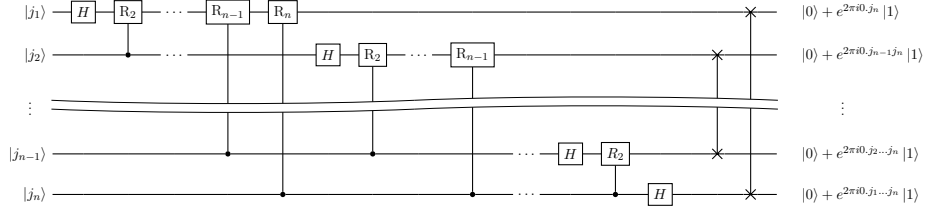


Figure 3.1: Efficient circuit for the execution of the Quantum Fourier Transform on  $n$  qubits [15].

may be used to solve the *hidden subgroup problem* [15], a generalization of phase estimation and order-finding problems, that has, among its special cases, an efficient quantum algorithm for the *discrete logarithm problem*, another problem thought to be intractable on a classical computer.

### 3.1.1 QFT on Qubits

The Discrete Fourier Transform takes as input a vector of complex numbers,  $x_0, \dots, x_{N-1}$  where the length  $N$  of the vector is a fixed parameter. Its output is a vector of complex numbers  $y_0, \dots, y_{N-1}$  defined by

$$y_j \doteq \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} x_k e^{2\pi i k j / N}.$$

The Quantum Fourier Transform on an orthonormal basis  $|0\rangle, \dots, |N-1\rangle$  is defined to be a linear operator with the following action on the basis states,

$$|j\rangle \longrightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i k j / N} |k\rangle. \quad (3.1)$$

Hence, its action on an arbitrary state may be written as

$$\sum_{j=0}^{N-1} x_j |j\rangle \longrightarrow \sum_{k=0}^{N-1} y_k |k\rangle.$$

Now we derive a circuit performing the QFT on  $n$  qubits (see, e.g., [15]). Let  $N = 2^n$ , then  $|0\rangle, \dots, |2^n - 1\rangle$  is the computational basis for our system of  $n$  qubits. It is helpful to write the state  $|j\rangle$  using the binary representation  $j = j_1 j_2 \dots j_n$ . More formally,  $j = j_1 2^{n-1} + \dots + j_n 2^0$ . It is also convenient

to adopt the notation  $0.j_l j_{l+1} \dots j_m$  to represent the binary fraction

$$j_l/2 + j_{l+1}/4 + \dots + j_m/2^{m-l+1}.$$

With simple algebra it follows an equivalent and useful formulation of the QFT

$$\begin{aligned} |j\rangle &= \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e^{2\pi i k j / N} |k\rangle \\ &= \frac{1}{2^{n/2}} \sum_{k_1=0}^1 \dots \sum_{k_n=0}^1 e^{2\pi i j (\sum_{l=1}^n k_l 2^{-l})} |k_1 \dots k_n\rangle \\ &= \frac{1}{2^{n/2}} \sum_{k_1=0}^1 \dots \sum_{k_n=0}^1 \bigotimes_{l=1}^n e^{2\pi i j k_l 2^{-l}} |k_l\rangle \\ &= \frac{1}{2^{n/2}} \bigotimes_{l=1}^n \left[ \sum_{k_l=0}^1 e^{2\pi i j k_l 2^{-l}} |k_l\rangle \right] \\ &= \frac{1}{2^{n/2}} \bigotimes_{l=1}^n \left[ |0\rangle + e^{2\pi i j 2^{-l}} |1\rangle \right] \\ &= \frac{(|0\rangle + e^{2\pi i 0.j_n} |1\rangle)(|0\rangle + e^{2\pi i 0.j_{n-1}j_n} |1\rangle) \dots (|0\rangle + e^{2\pi i 0.j_1 j_2 \dots j_n} |1\rangle)}{2^{n/2}}. \end{aligned}$$

With this equivalent formulation one can easily show [15] that the circuit reported in Figure 3.1 implements the Quantum Fourier Transform on the input  $|j_1 \dots j_n\rangle$ , where the gate  $R_k$  denotes, in the computational basis, the following unitary transformation

$$\begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i / 2^k} \end{pmatrix}.$$

Since the controlled  $R_k$  is fundamental for the implementation of an important routine as the QFT (see Figure 3.1) we have opted to implement such gate as a built-in in QDOS. Obviously it has been implemented not only for qubits. Indeed, its generalization to qudits is necessary for the implementation of the QFT on qudits as its qubit version for the QFT on qubits. The QDOS syntax for such gate is

$$\text{CR}[idx_{q_0}, idx_{q_1}](\text{int } k);$$

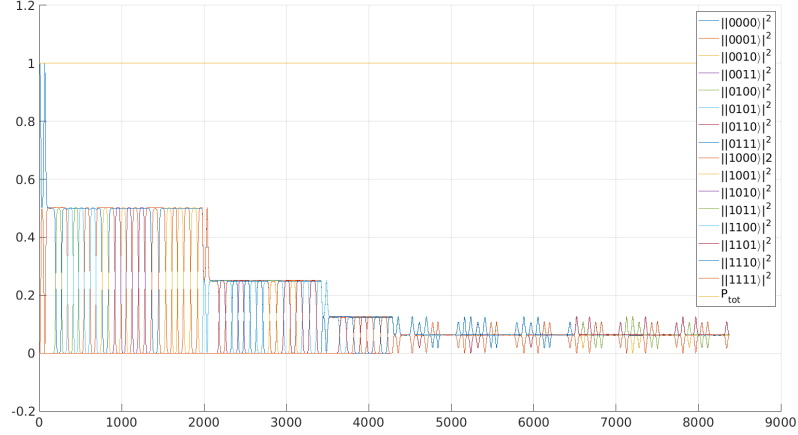


Figure 3.2: Here is shown the temporal evolution of the diagonal elements of the density matrix for a system of 4 qubits performing a Quantum Fourier Transform. These diagonal elements represent the probability of observing a state after a complete measurement of the system. Starting with only the state  $|0000\rangle$  the QFT creates an equal superposition of all the states of the system.

It should be noted that with the circuit reported in Figure 3.1 we are able to implement the QFT with a number of gates that is polynomial in the number  $n$  of qubits used! This implies an exponential speed-up over the best classical algorithm known for the computation of the Discrete Fourier Transform, the so called Fast Fourier Transform that is polynomial in  $N = 2^n - 1$ .

Before discussing how to implement the QFT on qubits with QDOS we derive a possible generalization on qudits. After that, we discuss and compare the execution of the QFT both on qubits and on qudits either with or without errors.

### 3.1.2 QFT on Qudits

Recalling the definition of the generalized Hadamard transform given in Section 2.2 and the definition of the QFT given by equation (3.1), it follows immediately that the Hadamard gate applied on a single qudit is exactly the Quantum Fourier Transform on its  $d = N - 1$  levels ( $|0\rangle, \dots, |d\rangle$ ). We must remark that this could seem to be a big advantage for qudits over qubits, but a few comments are in order. As already stated in Section 1.5, the decom-

position of a unitary operator through Theorem 1.2 results in a number of operations that is polynomial on the dimension of the operator itself. Thus, in this case, we would get a number of operations that is polynomial in  $N$ , de facto loosing the quantum speed-up over classical algorithms. Nevertheless, all these operations are *local*, while the corresponding multi-qubit version involves several (or many) controlled two-qubit operations. In addition, this scaling holds if we perform the QFT on only one qudit using Theorem 1.2 for the decomposition of the operator  $H$ . Instead if we derive a circuit (similar to the one shown for qubits) that performs the QFT on an arbitrary number of qudits, then it would be polynomial in the number of qudits achieving again an exponential speed-up over the FFT. With the difference that for qudits of dimension  $d$  the base of the exponential is  $d$  and not 2. Hence, for  $d \gg 2$ , in practical terms a fixed number of objects, we can achieve a discrete speed-up even on the qubits version of the QFT. We stress however, that the crucial advantage is in the reduction of the number of multi-qubit gates, typically much more error-prone compared to local operations. Moreover, if we do not assume a complete connection between the qudits (as for superconducting transmon) multi-object controlled gates would require several (or many) **SWAP** operations between qudits before being able to perform the required gate. Furthermore **SWAP** operations are composed by multi-object controlled operations, typically **CNOT**, and require a lot of time to be performed as Figure 3.2 highlights. Here we can see that a simple sequence of 3 **SWAP** take nearly half of quantum computation time for the execution of the QFT on 4 qubit, moreover it takes so long even assuming a complete connection between qudits. It is clear that, if we have to contrast errors that are exponential with the flow of time, the less the **SWAP** operations the better the result. The simplest way to reduce **SWAP** operations is to use as few many-body gates as possible. Hence, theoretically the qudit version of the QFT could bring good advantages over its qubits implementation.

To retrieve a circuit implementing the Quantum Fourier Transform on qudits in an efficient way we can proceed in a similar way to what done for qubits (see, e.g., [22]). Suppose we are interested in QFT over  $|0\rangle, \dots, |N-1\rangle$ , with  $N = d^n$  where  $d$  is the dimension of the qudits and  $n$  the number of the qudits. Again, it is helpful to write the state  $|j\rangle$  using the  $d$ -ary representation  $j = j_1 j_2 \dots j_n$ . Formally,  $j = j_1 d^{n-1} + \dots + j_n d^0$ . It is also convenient to adopt the notation  $0.j_l j_{l+1} \dots j_m$  to represent the binary

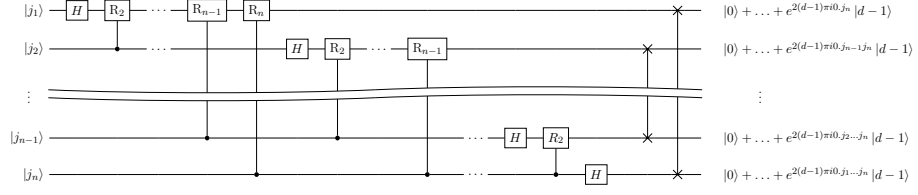


Figure 3.3: Efficient circuit for the execution of the Quantum Fourier Transform on  $n$  qudits [22].

fraction

$$j_l/d + j_{l+1}/d^2 + \dots + j_m/d^{m-l+1}.$$

As done for qubits, starting from the QFT definition given by equation (3.1), with some simple algebra we get an equivalent and useful formulation

$$\begin{aligned}
 |j\rangle &= \frac{1}{d^{n/2}} \sum_{k=0}^{2^n-1} e^{2\pi i k j / N} |k\rangle \\
 &= \frac{1}{d^{n/2}} \sum_{k_1=0}^{d-1} \dots \sum_{k_n=0}^1 e^{2\pi i j (\sum_{l=1}^n k_l d^{-l})} |k_1 \dots k_n\rangle \\
 &= \frac{1}{d^{n/2}} \sum_{k_1=0}^{d-1} \dots \sum_{k_n=0}^{d-1} \bigotimes_{l=1}^n e^{2\pi i j k_l d^{-l}} |k_l\rangle \\
 &= \frac{1}{d^{n/2}} \bigotimes_{l=1}^n \left[ \sum_{k_l=0}^{d-1} e^{2\pi i j k_l d^{-l}} |k_l\rangle \right] \\
 &= \frac{1}{d^{n/2}} \bigotimes_{l=1}^n \left[ |0\rangle + e^{2\pi i j d^{-l}} |1\rangle + \dots + e^{2\pi i j (d-1) d^{-l}} |d-1\rangle \right].
 \end{aligned}$$

As one can easily see (we have just changed 2 with  $d$ ), this is a straightforward generalization of the QFT for qubits. In Figure 3.3 an efficient circuit for the implementation of the QFT on qudits is reported [22]. At this level of abstraction one can see that there is no difference at all between the circuit implementing the QFT on qubits and the circuit implementing the QFT on generic qudits. The operations that will be performed are different but these differences are not visible at this level of abstraction. In Figure 3.3 the

QDOS simulations of the QFT							
$n$	$d$	Levels	Pulses	2-body G	$T_2$	Fidelity	Execution Time
4	2	16	110	12	-	$\sim 99.90\%$	$\sim 8700$ ns
1	16	16	764	0	-	$\sim 97.80\%$	$\sim 7300$ ns
2	4	16	152	2	-	$\sim 99.00\%$	$\sim 7800$ ns
4	2	16	110	12	$100 \mu s$	$\sim 87.50\%$	$\sim 8700$ ns
1	16	16	764	0	$100 \mu s$	$\sim 37.00\%$	$\sim 7300$ ns
2	4	16	152	2	$100 \mu s$	$\sim 73.75\%$	$\sim 7800$ ns
4	2	16	438	48	$100 \mu s$	$\sim 66.90\%$	$\sim 33600$ ns

Table 3.1: Table summarizing the execution of the QFT on different systems. Here  $n$  is the number of qudits considered and  $d$  is their dimension. The "pulses" column reports the total number of pulses executed to perform the QFT for each specific hardware configuration while 2-body G is the number of gates between more than one qudit.  $T_2$  represents the characteristic time of dephasing errors simulated on each qudit. The first three runs have no errors simulated. The last line is the execution of the 4 qubits QFT considering a linear connectivity topology between qubits. The fidelity of the output density matrix  $\rho$  is computed as  $\mathcal{F} = \langle \psi_{\text{fin}} | \rho | \psi_{\text{fin}} \rangle$  where  $|\psi_{\text{fin}}\rangle$  is the correct final solution that the circuit should have produced. The error is then computed as  $\mathcal{E} = 1 - \mathcal{F}$ . The execution time is the quantum execution time, i.e. the time needed for a quantum processor to execute the circuit implementing the Quantum Fourier Transform.

H is the generalized Hadamard already discussed, while the gate  $R_k$  is the generalization to qudits of its analogous for qubits. This is defined as

$$\begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & e^{2\pi i/d^k} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & e^{2\pi i(d-1)/d^k} \end{pmatrix}$$

and its controlled version is applied  $j$  times to the target qudit for a control qudit in state  $|j\rangle$ .

### 3.1.3 QDOS Simulations

Now we turn our attention to the implementation of the QFT in QDOS. In particular, we show a QDOS program for the execution of the QFT on 4 qubits. Along this, we report QDOS programs for the QFT on 1 and 2 qudits with, respectively, 16 and 4 levels each, thus keeping the dimension of the Hilbert space fixed in the different implementations. We thus obtain

the QFT for the states  $|0\rangle, \dots, |15\rangle$  executed both on qubits and on qudits. This choice enables us to verify how errors can affect different kind of physical realizations of the same algorithm on different quantum computing hardware. We execute such codes either with or without errors that, in this case, are pure-dephasing errors. Pure dephasing errors, as already described at the end of Section 1.4, are simulated with the Lindblad Equation (1.11). To keep the analysis as general as possible, we only give a brief description of the examined physical system for the implementation of the quantum computer. Briefly, we have modeled all the terms of the Hamiltonian as spin systems ( $S = 1/2$  for qubits,  $S = 15/2$  and  $3/2$  for qudits), hence the connection matrices are proportional to  $S_y$ . The general form of the Hamiltonian simulated is the following

$$H = \sum_l \mu_B g_l B S_{z_l} + D S_{z_l}^2 + \sum_{j,k} \Theta_{j,k}(t) J_{j,k} S_{z_j} \otimes S_{z_k}$$

where  $\Theta_{jk}(t)$  is one if we are performing operations between qudit  $j$  and qudit  $k$  and zero otherwise. The  $T_2$  parameter chosen is always  $100 \mu\text{s}$  for all qubits/qudits. This value is optimistic but feasible in specific molecular system (see, for example [25]). We recall that, in the case of qudits, it represents the coherence time for  $|\Delta m| = 1$  transitions.

Table 3.1 shows the results of the executed simulations. Focusing on the first three lines of the table, i.e. the simulations without pure dephasing errors, we can clearly see that to perform the QFT on a qudit of 16 levels directly through an Hadamard gate we need a much bigger number of pulses compared to the QFT executed on qubits or smaller qudits. This was expected, as we have already remarked that Theorem 1.2 gives us decompositions that are polynomial in the space dimension. This is the main reason for which we get a lower fidelity for the QFT on a single  $S = 15/2$  qudit. Indeed, as we can see from Table 3.1, without pure dephasing errors the fidelity decreases as the number of pulses increases. This is due to small leakage (coherent) errors accumulating during each pulse, and hence the total error is roughly proportional to the number of pulses.

When we turn on pure dephasing errors we can see their dramatic effect on the fidelity obtained. Indeed, even with optimistic  $T_2$  we get a  $-12.5\%$

on the fidelity for the QFT on 4 qubits, while we reach a  $-60.70\%$  for the QFT on a single  $S = 15/2$  qudit. These dramatic outcomes could have been predicted since pure dephasing errors make the off-diagonal elements of the density matrix of a quantum spin  $S$  system decaying exponentially with the square of the difference between the respective eigenvalues of  $S_z$  on the pair of examined states. In other words, if we are considering the element  $\rho_{j,k}$ , under the action of pure dephasing errors it will decay exponentially as  $\sim e^{\Delta m^2 t/T_2}$ , where  $\Delta m$  is the difference between the  $j$ -th and the  $k$ -th  $S_z$  eigenvalues. This means that, for example, the element  $\rho_{15,0}$  decays as  $\sim e^{15^2 t/T_2}$  and then for  $t = 7.3 \mu\text{s}$  we have that  $\rho_{15,0}$  is multiplied by a factor of  $\sim e^{-16.5!}$

But also qubits are significantly affected by pure dephasing errors, and a fidelity of  $87.50\%$  for a QFT is not near enough to what is needed to perform successfully all the algorithms in which the QFT is a key component. Moreover, the QFT for qubits requires more two-qubit gates than the QFT implemented on qudits. This, in practice can translate in big disadvantages for qubits. Indeed, while for molecular spin qudits multi-body gates are not more complex than single-body ones for other architectures such as superconducting transmon qubits (one of the most diffused realization of qubits at the time of writing) multi-qubits gates are much more prone to errors [12, 23].

To quantify this effect, if we call  $p$  the probability of error in a single qubit gate on a superconducting transmon quantum computer (STQC), then on the same hardware the probability of error for a two-qubit gate is about  $10p$  [19, 24]. Moreover, for STQCs the connections between qubits is far from easy and usually qubits are arranged in a planar or even linear grid with only nearest-neighbor connectivity. Hence, as already pointed out in Section 3.1.2, on such a mesh, the implementation of the QFT on 4 qubits requires many SWAP operations, each made of 3 CNOT. In practical purposes, if we would have performed the QFT on a system like `ibmq_manila` with a linear mesh of qubits, an average error of CNOT operations of about  $0.72\%$  and considering for the CR gate the same error of CNOT, we would run a circuit containing 48 two-qubit gates, resulting in an error probability (leaving out single-qubit gates) of about  $34.56\%$ . Then we should account for the reduction of the fidelity produced by pure dephasing errors that, as reported in the

last line of Table 3.1, is about 33.10%, hence giving us a final mean fidelity of about 50.45%. This clearly show that, at least if multi-qubit gates are difficult to perform and connections among qubits is poor (as for STQCs), a more viable solutions would be to use molecular spin qudits. Indeed for this platform multi-qudit gates does not represent a greater challenge than single-qudit ones ???. Moreover, for molecular spin qudits we have ways to counter pure dephasing errors, both via hardware and software. In [4] is shown how is possible to increase the coherence of a qudit by the synthesis of different multi-spin molecules with proper pattern of spin-spin interaction. Then, from a software point of view, in Section 3.2 we illustrate an Embedded Quantum Error Correction code [3] on molecular nanomagnet, originally proposed in [3], which has been developed to correct pure dephasing errors.

To end this Section on the Quantum Fourier Transform we report the QDOS circuits used for the simulations. The following is the QDOS circuit implementing the QFT on 4 qubits with complete connections among qubits.

```
q[4] = {2,2,2,2};
```

```
qft =
{
    H[0];
    CR[1,0] (2);
    CR[2,0] (3);
    CR[3,0] (4);

    H[1];
    CR[2,1] (2);
    CR[3,1] (3);

    H[2];
    CR[3,2] (2);

    H[3];

    CX[0,3];
    CX[3,0];
    CX[0,3];

    CX[1,2];
    CX[2,1];
```

```

    CX[1,2];
};

main = { qft; };

```

In this QDOS circuit the SWAP gates that should be performed at the end of the QFT have been implemented with the usual decomposition

$$\text{SWAP}[j, k] = \text{CX}[j, k] \text{CX}[k, j] \text{CX}[j, k].$$

While the following implements the QFT on 4 qubit with restricted connectivity among them (specifically a linear topology).

```

q[4] = {2,2,2,2};

qft =
{
    H[0];
    CR[1,0](2);

    CX[2,1];
    CX[1,2];
    CX[2,1];
    CR[1,0](3);
    CX[2,1];
    CX[1,2];
    CX[2,1];

    CX[0,1];
    CX[1,0];
    CX[0,1];
    CX[3,2];
    CX[2,3];
    CX[3,2];
    CR[2,1](4);
    CX[0,1];
    CX[1,0];
    CX[0,1];
    CX[3,2];
    CX[2,3];
    CX[3,2];

    H[1];
    CR[2,1](2);

```

```

    CX[3,2];
    CX[2,3];
    CX[3,2];
    CR[2,1](3);
    CX[3,2];
    CX[2,3];
    CX[3,2];

    H[2];
    CR[3,2](2);

    H[3];

    CX[0,1];
    CX[1,0];
    CX[0,1];
    CX[3,2];
    CX[2,3];
    CX[3,2];
    CX[1,2];
    CX[2,1];
    CX[1,2];
    CX[0,1];
    CX[1,0];
    CX[0,1];
    CX[3,2];
    CX[2,3];
    CX[3,2];

    CX[1,2];
    CX[2,1];
    CX[1,2];
};

main = { qft; };

```

Moving on to qudits, the QFT can be implemented in QDOS as follows.

```

q[1] = {16};

qft = { H[0]; };

main = { qft; };

```

for the QFT on 1 qudit and

```
q[2]    = {4,4};

SWAP[2] = "./custom_gate/SWAP.bin";

qft =
{
    H[0];
    CR[2,0] (2);

    H[1];

    SWAP[0,1];
};

main = { qft; };
```

for the QFT on 2 qudits. In this last circuit, implementing the QFT on 2 qudits, we show the use of a custom gate. Specifically, the SWAP gate between two qudits has been defined through the unitary matrix representing it in the computational basis. This highlights the ease of use of custom gates inside a QDOS circuit.

### 3.2 Embedded QEC on Qudits

In this Section we introduce and simulate with QDOS a possible scheme of Embedded Quantum Error Correction on qudits [3]. In the last Section we have been focused on a specific quantum algorithm (QFT) and on how to implement it on a quantum computer. Little importance was given to the specific hardware used for the realization of the quantum computer itself. This has been done to show that QDOS can also be used as a generic simulator for quantum algorithms, even without knowing anything on the hardware used to build a quantum computer. But this is not the reason why QDOS has been developed. Indeed, since the introduction of the Thesis we have stressed out that one of our goals was to be able to simulate in detail the physical realization of the quantum computer itself. This is necessary as we want to evaluate the qualities of a possible hardware platform, especially against errors. In this Section we give some attention to the physical implementation of the qudits that we use, molecular nanomagnets.

The idea behind QEC is to encode the quantum information into “logical qubits”, objects with more than two possible energy levels. Logical qubits are designed such that errors bring the system in a state outside the computational subspace, making errors in logical qubits detectable and correctable. In standard approaches these extra-states are obtained by encoding a logical qubit into many physical units. However, this makes the practical implementation of QEC and the corresponding quantum computation extremely difficult because nonlocal quantum gates on a large set of physically distinct objects are needed. A possible way to overcome this hurdle is to employ a single multilevel quantum object to encode a logical qubit. Molecular nanomagnets offer many accessible spin states which could be used to encode a protected qubit.

In [3] it is shown how to encode a single logical qubit into  $d$  levels of a molecular nanomagnet (qudit encoding) endowed with a QEC scheme to protect it against the most harmful errors occurring in molecular qubits, namely, pure dephasing. To do this, a code has been derived [3] which is specific for the class of systems under consideration and introduces error-protected states in such a molecular qudit. Here we summarize the procedure of [3], we give a circuit in QDOS for the implementation of such scheme, and, at last, we simulate the execution of the proposed embedded QEC code with QDOS.

We consider simple molecules described by the Hamiltonian

$$H = g_z \mu_b B S_z + g_z^A \mu_b B \sigma_z^A + D S_z^2 + S \cdot \underline{\Gamma} \cdot \sigma^A \quad (3.2)$$

where  $S$  is the qudit spin used to embed the error-protected qubit,  $\sigma^A = 1/2$  is an electronic spin, exploited as ancilla for error detection, and  $\mu_B$  is the Bohr magneton. This Hamiltonian is realized, for example, in a dimer consisting of a (molecular) spin  $S$  weakly coupled to a spin  $1/2$ . The first two terms in equation (3.2) represent the Zeeman interaction with an external magnetic field  $B$  along  $z$ ,  $D$  is an axial zero-field splitting term. The last term represents a weak exchange or hyperfine ancilla-qudit coupling tensor  $\underline{\Gamma}$ . Because  $\Gamma_{x,y}$  is much smaller than the difference of qudit and ancilla excitation energies, the eigenstates are simple tensor products of the eigenstates of  $S_z$  ( $|m\rangle$ ) and  $\sigma_z^A$  ( $|l\rangle$ ), with  $m = -S, \dots, S$  and  $l = \downarrow, \uparrow$ .

The most important error in molecular qubits is given by pure dephasing (see equation (1.11)). Conversely, spin relaxation is usually very slow at low temperature in these systems, with electronic relaxation times reaching  $\sim 10^2$  ms. For small  $t/T_2$  is possible [3] to perform a perturbative expansion  $\rho(t) = \sum_{k=0}^{\infty} E_k \rho(0) E_k^\dagger$ , with the *error operators*

$$E_k \doteq \sqrt{\frac{(2t/T_2)^k}{k!}} e^{-S_z^2 t/T_2} S_z^k.$$

This shows that at short  $t/T_2$  only low powers of  $S_z$  affect the dynamics. Hence, by considering the matrix elements of  $S_z^n$  they define protected qubit states (code words) and a QEC procedure to recover the effects of pure dephasing up to a given order in  $(t/T_2)^n$ . These code words have been defined as

$$|0_L\rangle = \frac{1}{\sqrt{2^{2S-1}}} \sum_{k \text{ odd}}^{2S} \sqrt{\binom{2S}{k}} |k - S\rangle, \quad (3.3)$$

$$|1_L\rangle = \frac{1}{\sqrt{2^{2S-1}}} \sum_{k \text{ even}}^{2S} \sqrt{\binom{2S}{k}} |k - S\rangle. \quad (3.4)$$

Thus, to correct dephasing up to the order  $(t/T_2)^n$  one needs at least  $2n$  levels. In [3] is shown how these code words ensure that (i)  $E_k$  errors bring  $|0_L\rangle$  and  $|1_L\rangle$  to orthogonal states and (ii) the coefficients  $\alpha$  and  $\beta$  of a generic superposition of the logical states  $\alpha|0_L\rangle + \beta|1_L\rangle$  are preserved. Therefore,  $E_k$  errors are detectable, for (i), and correctable, for (ii). In [3] is reported the general procedure for arbitrary  $S$ . Here we limit to the discussion of the embedded QEC code for the case  $S = 3/2$ , hence correcting errors up to order  $(t/T_2)^n$ , i.e. we limit to the recovery of errors of type  $S_z$ .

Before presenting the QEC code, we briefly discuss the physical realization of our system. As reported in [6], a suitable system to test the code is the CrYb complex, consisting of a  $S = 3/2$   $\text{Cr}^{3+}$  electronic spin qudit coupled to the electronic  $\sigma^A = 1/2$  spin given by the  $\text{Yb}^{3+}$  ion. We assume realistic parameters for a  $\text{Cr}^{3+}$  ion in an octahedral crystal field, i.e.  $D = -0.24 \text{ cm}^{-1}$  and an isotropic  $g = 1.98$ . Conversely for  $\text{Yb}^{3+}$  we assume  $g_z^A = 4.2$  and  $g_{x,y}^A = 2.9$ , that are typical for a Yb(trensal) complex [6]. Then we assume  $\Gamma_z = -3.3 \times 10^{-2} \text{ cm}^{-1}$  as one could obtain with a dipole-dipole coupling,

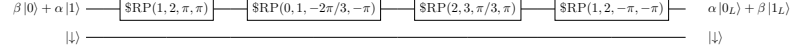
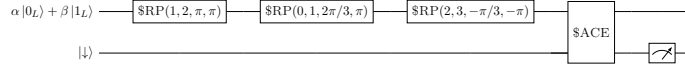
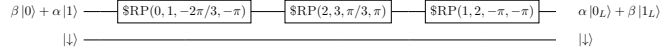
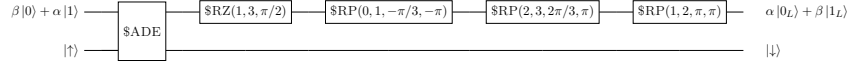
(a) Circuit implementing the **encoding** step of the QEC code.(b) Circuit implementing the **detection** step of the QEC code. The \$ACE gate is a sequence of rotations that will perform an ancilla conditional excitation. It is described in details in the main text.(c) Circuit implementing the appropriate **recovery** step for the ancilla measurement outcome  $|\downarrow\rangle$ .(d) Circuit implementing the appropriate **recovery** step for the ancilla measurement outcome  $|\uparrow\rangle$ . The \$ADE gate is a sequence of rotations that will perform an ancilla de-excitation. It is described in details in the main text.

Figure 3.4: Schematic representation of the Embedde Quantum Error Correction procedure described in [3].

while we set  $\Gamma_{x,y} = 0$ , a slight difference from parameters reported in [6] which however does not significantly change the result of QEC. According to [6] we have chosen  $B = 1$  T,  $B_1 = 100$  G and  $B_1^A = 45$  G. Where  $B_1$  is the amplitude of the oscillating field used to resolve all transitions. As coherence times we assume  $T_2 = 50 \mu s$  and  $T_2^A = 3 \mu s$ .

Now we report the procedure proposed in [3] for the implementation of the error correction scheme for  $S = 3/2$ , as shown in Figure 3.4. The code, as usual in QEC, is composed of three main steps: state encoding, error detection and error correction. For each of these steps we describe the sequence of QDOS gates performing it along with a possible QDOS circuit implementing them. With this we highlight two of the main strengths of QDOS:

1. being able to perform partial measurements throughout the execution of a circuit, and

2. the possibility to schedule conditional code based on the outcome of these measurements.

But also, being forced to define the entire circuit to the rotation level, we will run into an annoying shortcoming of QDOS: the lack of simple arithmetic between variables, thus forcing us to define one distinct variable for each unique angle of rotation, instead computing values starting from the same constant  $\pi$ . This is not a fundamental problem, but still is one of the features which I believe should be implemented. In the Conclusions of the Thesis we talk about this and other useful features not present at this stage in the QDOS programming language.

Returning to Quantum Error Correction, the procedure for the state encoding proposed in [3] is as follows. Starting from a generic superposition of  $|0\rangle$  and  $|1\rangle$ , let's say  $|\psi\rangle = \beta|0\rangle + \alpha|1\rangle$ , the objective is to transform the state  $|\psi\rangle$  into the logical state  $|\psi\rangle = \alpha|0_L\rangle + \beta|1_L\rangle$ .

With this aim, one starts with a  $\pi$ -pulse between  $|1\rangle$  and  $|2\rangle$ . Then a  $(-2\pi/3)$ -pulse and a  $(\pi/3)$ -pulse will follow between, respectively,  $|0\rangle$ - $|1\rangle$  and  $|2\rangle$ - $|3\rangle$ . To complete the encoding procedure a  $(-\pi)$ -pulse is needed between  $|1\rangle$  and  $|2\rangle$ . A possible implementation of these rotations in QDOS can be achieved by the following block of instruction and a more schematic representation can be seen in Figure 3.4a.

```
q[2] = {2,4};

pi      = 3.141592653589793238;
mpi     = -3.141592653589793238;
pi_t    = 1.047197551196597746;
mt_pi_t = -2.094395102393195492;

encoding =
{
    $RP[1](1, 2,      pi,  pi);
    $RP[1](0, 1, mt_pi_t, mpi);
    $RP[1](2, 3,      pi_t, pi);
    $RP[1](1, 2,      mpi, mpi);
};
```

To perform the step of detection we need to reconstruct the initial superposition of coefficients  $\alpha$  and  $\beta$  but being careful to produce distinguishable (orthogonal) results in case of an  $S_z$  error. To achieve this we first need to

apply the inverse of the last three pulses of the encoding step. After that we perform a conditional ancilla excitation for the two states of the spin  $S$  qudit which are populated if and only if a single  $S_z$  error has occurred. Then, the detection step ends with a measurement of the ancilla. Resulting in a detection of an error only if the outcome of such measure is  $|\uparrow\rangle$ . This step is shown in Figure 3.4b and the following is a possible QDOS circuit implementing it.

```
q[2] = {2,4};

pi      = 3.141592653589793238;
mpi_t   = -1.047197551196597746;
t_pi_t  = 2.094395102393195492;

$ACE[2] = "./custom_gate/ACE.bin";

detection =
{
    $RP[1](1, 2, pi, pi);
    $RP[1](0, 1, t_pi_t, pi);
    $RP[1](2, 3, mpi_t, mpi);

    $ACE[0,1];

    measure[0];
};
```

Here the custom gate **\$ACE** performs the step of the ancilla conditional excitation. It performs an **\$RP** of an angle of  $\pi$  between the states  $|1, \downarrow\rangle$  and  $|1, \uparrow\rangle$ , and an identical **\$RP** on the states  $|3, \downarrow\rangle$  and  $|3, \uparrow\rangle$ . It had to be defined as a custom gate since the **\$RP** gate acts on a single qudit, while for the ancilla conditional excitation we need to turn on the coupling between the spin  $S = 3/2$  and  $\sigma^A = 1/2$ , i.e. the term  $S \cdot \underline{\Gamma} \cdot \sigma^A$  of the Hamiltonian reported in equation (3.2).

To complete the QEC code we need to perform the error correction (or recovery) step. This depends on the outcome of the measurement performed at the end of the detection step. Then, if an error has not been detected (i.e. if the ancillary spin has been projected into state  $|\downarrow\rangle$ ), we perform the sequence of rotations of Figure 3.4c. While, if an error has been detected, we first perform an ancilla de-excitation, bringing back the states  $|1, \uparrow\rangle$  and

$|3, \uparrow\rangle$  into, respectively, the states  $|1, \downarrow\rangle$  and  $|3, \downarrow\rangle$ . After that, we perform the corresponding sequence of rotations reported in Figure 3.4d. A QDOS circuit implementing this recovery sequence could be the following:

```

q[2] = {2,4};

pi      = 3.141592653589793238;
mpi     = -3.141592653589793238;
pi_2    = 1.570796326794896619;
pi_t    = 1.047197551196597746;
mpi_t   = -1.047197551196597746;
t_pi_t  = 2.094395102393195492;
mt_pi_t = -2.094395102393195492;

recovery_no_error =
{
    $RP[1](0, 1, mt_pi_t, mpi);
    $RP[1](2, 3, pi_t, pi);
    $RP[1](1, 2, mpi, mpi);
};

recovery_error =
{
    $ADE[0,1];

    $RZ[1](1,3, pi_2);

    $RP[1](0, 1, mpi_t, mpi);
    $RP[1](2, 3, t_pi_t, pi);
    $RP[1](1, 2, pi, pi);
};

recovery =
{
    custom_logic::check_ancilla;
};

```

The `$ADE` custom gate is the inverse of the `$ACE`. The custom logic used in the block `recovery` checks the outcome of the last measurement, and based on the outcome it chooses to perform the appropriate block of instructions between `recovery_error` and `recovery_no_error`. A (C) implementation for this custom logic could be the following.

```

/*
 * Custom logic function for the Embedded Quantum Error Correction
 * scheme proposed in Section 3.2 of the Thesis.
 *
 * It simply check the last measurement outcome of the ancilla.
 * Then if we get a down (0) it will schedule the 'blocco_gate'
 * recovery_error. While if we get an up (1) it will schedule
 * the recovery_no_error block.
 */
blocco_gate *EQEC_check_ancilla(lista_misure *tabella_misure)
{
/*
 * Check input
 */
    if(tabella_misure == NULL || tabella_misure->dim_eff == 0)
    {
        fprintf(stderr, "ERROR:: Custom logic invoked before "
            "any measurement.\n"
            "ABORTING...\n\n");

        return NULL;
    }

    int *measurement_outcome = get_misura(tabella_misure, 0);
/*
 * Error Check
 */
    if (measurement_outcome == NULL)
    {
        fprintf(stderr, "ERROR:: impossible to retrieve the last measurement "
            "outcome.\n"
            "ABORTING...\n\n");

        return NULL;
    }

    blocco_gate *blk = NULL;

/*
 * Check the measurement outcome
 */
    if (measurement_outcome[0] == 1)
    {
/*
 * We have detected an error, hence the recovery_error block has to be
 * executed
 */

```

```

*/
    blk = cerca_blocco(tabella_blk, "recovery_error");
}
else
{
/*
*   We haven't detected any error, hence the recovery_no_error block has
*   to be executed
*/
    blk = cerca_blocco(tabella_blk, "recovery_no_error");
}

    return blk;
}

```

A complete QDOS circuit implementing a full step of the described QEC code could be given by the following main block of instructions.

```

main =
{
    encoding;

    $$IDLE[0](5000);

    detection;
    recovery;
};

```

Here we come in touch with the last QDOS built-in gate, the gate `$$IDLE`. This generate a zero pulse for the duration given as a parameter (in nanoseconds). It is used to simulate a free evolution of the system. This is used to evaluate the performance of the Embedded QEC code proposed in [3] and reported in this Section for quantum memory applications, where data are stored in a quantum state and corrected by the code after a while.

In Figure 3.5 it is reported the error  $\mathcal{E} \doteq 1 - \langle \psi | \rho | \psi \rangle$  against the ratio  $t/T_2$ , showing, after a given amount of time  $t$ , how well the proposed procedure can correct pure dephasing errors. For comparison we plot also, for the same  $t/T_2$ , the error for a  $S = 1/2$  spin qubit. As the Figure shows we can see a clear advantage over free evolution, in particular, the ratio between uncorrected and corrected error is maximum for  $t/T_2 \sim 5 \times 10^{-2}$ , i.e. for  $t \sim 2.5 \mu s$ . This is in agreement with results reported in [6] for a CrYb compound with similar parameters, thus confirming again the correctness of

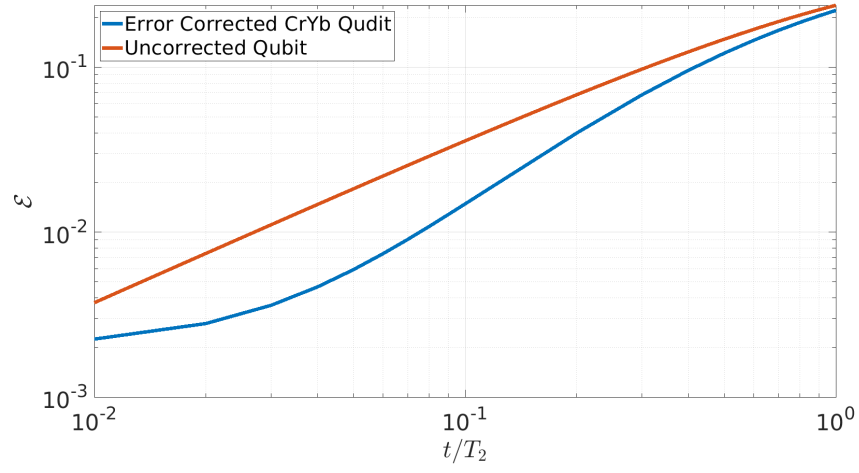


Figure 3.5: Comparison between an error corrected  $S = 3/2$  spin Qudit (blue line) and an uncorrected  $S = 1/2$  spin qubit subject to decoherence. The error  $\mathcal{E}$  is plotted as a function of the ratio  $t/T_2$  where  $t$  is the time of free evolution to which the system has been exposed.

the simulations performed with QDOS.

# Conclusions

With this Thesis I create software tools that can automate and speed-up the evaluation of hardware platforms for the realization of quantum computers. After having introduced in the first Chapter the theoretical foundations necessary for the correct simulation of open quantum systems (see Sections 1.3 and 1.4) and arbitrary operations on them (see Sections 1.5 and 1.6), in Chapter 2 I included a broad discussion of the tools made.

To simplify the user interaction with the simulator, I have developed a simple programming language, called QDOS (which stands for QuDit OperationS). This programming language speeds-up the definition of quantum algorithms on generic qudits and abstracts the simulated hardware from the algorithm. Indeed, the approach followed to carry out the simulation of a quantum computer involves the numerical approximation of a system of differential equations that describes its time evolution, the so called Lindblad Equation (see Section 1.4). This approach has both strengths and weaknesses. Compared to the multiplication between matrices, which most of the available simulators implement [7], it has a significantly higher computational cost. However, it is also one of the few methods that allows one to faithfully simulate the time evolution of the quantum system in the presence of errors (such as, for example, pure dephasing). In fact, through linear algebra we are able to obtain the final state of a quantum system but we have no insights on the evolution itself. Given therefore the purpose that I had set for myself, i.e. a tool for the accurate simulation of the hardware that constitutes a quantum system, the choice of simulation through Lindblad Equation seemed obvious if not forced.

By simulating the temporal evolution of the quantum system through Lindblad Equation, the operator associated with the gate is not enough to know

the evolution and final state of the system after the gate itself. Indeed, in order to control the temporal evolution of the quantum system we need to use correctly modulated pulses, but these are totally hardware dependent. In the first chapter we showed how to decompose, with the Theorem [1.2](#), the unitary representation of a gate into a sequence of rotations that, once the physical properties of the hardware are known, is easily translatable into an appropriate sequence of pulses.

This is one of the main reasons for the introduction of the QDOS programming language: to be able to abstract the simulated hardware from the defined circuit. Indeed, by performing this abstraction, we recover the programming simplicity that simulators based on linear algebra have, and at the same time, we greatly facilitate the porting of already written quantum circuits. Furthermore, the user does not have to worry about the handwriting of all the pulses to be performed that can be a highly tedious and absolutely error-prone task, as even for short and simple QDOS circuits the number of pulses that compose them is in the order of hundreds.

Finally, having the opportunity to develop a new programming language, I was able to achieve a much more demanding goal, i.e. to simplify the simulation of Quantum Error Correction codes. To do this, both efforts at the simulator level and efforts at the QDOS language level were required. But in the end I am very satisfied with the solution reached, which in Section [3.2](#) allowed the effortless implementation of an Embedded Quantum Error Correction algorithm on qudits. This is a remarkable success, as evidenced also by the Quantum Flagship project sponsored by the European Union, which reads, verbatim:

"Fault tolerant quantum computing represents a collective research, engineering and quantum software challenge. **Achieving fault-tolerance relies on choosing a quantum error correction code which is suitable for the dominant noise model in the system**, and a level of control such that error accumulation can be effectively suppressed using quantum-compatible feedback routines. Finally, the routines need to be implemented sufficiently fast such that errors do not increase during the computation. The biggest likely challenge in this field will be to show a universal gate-set, the hallmark for arbitrary quantum computing, employing two logical qubits. **While highly demanding, this achievement is nothing short of the**

**equivalent to being the first nation to land on the moon."**

I certainly don't think of the QDOS programming language as a complete language. In the near future some other features will need to be implemented to make it actually effective. A short, personal list of these features follows.

- Implement simple arithmetic between variables.
- Making the definition of instruction blocks parametric, by doing this we actually make it possible to define real subroutines.
- Implement control flow mechanisms, first of all the `for` loop that could be extremely useful for a language of such low level, and therefore repetitive, as QDOS is.

To conclude, in the last chapter of the thesis I presented two situations in which the use of the software developed leads to considerable benefits. On the one hand, with the simulation of the Quantum Fourier Transform we were able to highlight how the choice of hardware and the limits that this has have heavy repercussions on the accuracy of the results obtained. With the simulation of an Embedded Quantum Error Correction technique we have shown the simplicity of implementation and testing of arbitrary QEC procedures.

For all these reasons I believe that the software developed, subject of this Thesis, can be of help for the design of the quantum computers of tomorrow. But above all, I am convinced that its continuous development will help me considerably in my future academic career.



## A: QDOS grammar

```
list:      /* Nothing */
| list ';'
| list def ';'
| list error ';'
;

def:       QUDIT '[' INTEGER ']' '=' '{' integer_sequence '}'
| GATE '[' INTEGER ']' '=' INDEX
| NAME '=' '{' inst_sequence '}'
| NAME '=' INTEGER
| NAME '=' FLOAT
;

inst:      GATE '[' integer_sequence ']' '(' float_sequence ')' ';'
| GATE '[' integer_sequence ']' ';'
| MEASUREMENT '[' integer_sequence ']' ';'
| CUSTOM_LOGIC NAME ';'
| NAME ';'
;

inst_sequence:  inst
| inst_sequence inst
;

integer_sequence:  /* Nothing */
| INTEGER
| integer_sequence ',' INTEGER
;

float_sequence:   /* Nothing (N c Q)*/
| VAR
| FLOAT
| INTEGER
| float_sequence ',' VAR
| float_sequence ',' FLOAT
| float_sequence ',' INTEGER
;
```



## B: Built-in Gates List

Below is a list of all built-in gates recognized by QDOS.

**\$\$IDLE**[int  $idx_{q_0}$ ]( float  $t$ );

It generates a null pulse thus simulating a free evolution of the quantum system.

**\$RP**[int  $idx_{q_0}$ ](int  $lv_0$ , int  $lv_1$ , float  $\theta$ , float  $\beta$ );

It generates a planar rotation on qudit  $idx_{q_0}$ , between levels  $lv_0$  and  $lv_1$  of an angle  $\theta$  and a phase  $\beta$

**\$RZ**[int  $idx_{q_0}$ ](int  $lv_0$ , int  $lv_1$ , float  $\phi$ );

It generates a Z rotation on qudit  $idx_{q_0}$ , between levels  $lv_0$  and  $lv_1$  of an angle  $\phi$ .

**\$X**[int  $idx_{q_0}$ ];

It generates the generalized X gate on qudit  $idx_{q_0}$ .

**\$Z**[int  $idx_{q_0}$ ];

It generates the generalized Z gate on qudit  $idx_{q_0}$ .

**\$H**[int  $idx_{q_0}$ ];

It generates the generalized H gate on qudit  $idx_{q_0}$ .

**\$\$S**[int  $idx_{q_0}$ ];

It generates the generalized S gate on qudit  $idx_{q_0}$ .

$\$CX[\text{int } idx_{q_0}, \text{int } idx_{q_1}];$

It generates the generalized CX gate between qudit  $idx_{q_0}$  and qudit  $idx_{q_1}$ .

$\$CEX[\text{int } idx_{q_0}, \text{int } idx_{q_1}](\text{int } lv_c, \text{int } lv_0, \text{int } lv_1);$

It generates the generalized CEX gate that will exchange levels  $lv_0$  and  $lv_1$  of qudit  $idx_{q_1}$  when the control qudit  $idx_{q_0}$  is in level  $lv_c$ .

$\$CR[\text{int } idx_{q_0}, \text{int } idx_{q_1}](\text{int } k);$

It generates a generalized CR gate between qudit  $idx_{q_0}$  and qudit  $idx_{q_1}$  of parameter  $k$ .

## C: Command Line Options

Here follows a list of the command line options accepted by QDOS. This same list can be seen in the manual page of QDOS accessible on LINUX through

```
$: man qdos
```

The list is:

```
--connection_enabled={true|false}
    Enable connections among qudit to resolve levels.
--errors_enabled={true|false}
    Enable the simulation of pure dephasing errors.
--frequency_output={integer}
    Sets the number of times the density matrix will be printed when
    simulating a pulse.
--help
    Display this Information.
--in_dir={string}
    Set the directory for input files.
--measurement_error={double}
    Set the probability of errors in a measurement of a qudit.
--measure_memory={integer}
    Set the number of measurement to save.
--n_shots={integer}
    Set the number of complete measurements to be performed at the
    end of the simulation.
--out_dir={string}
    Set the directory for the output files.
--out_rho_final={false|bin|dec}
    Set the format of the final (and before any measurement) density
    matrix to be printed.
--out_rho_simulation={false|bin|dec}
```

```

    Set the format of the intermediate density matrix to be printed.
--out_schedule={false|bin|dec}
    Set the format of the sequence of pulses to be printed.
--random_seed={false|true|integer}
    Set a custom seed for the RNG of QDOS or auto-generate a random
    seed for it.
--simulation_enabled={true|false}
    Enable the simulation of the circuit provided as input.
--simulation_progres={0|1|2}
    Set the level of verbosity of the simulator.
--version
    Get the version of the installed software.

```

Any of these options have a default value. A plain call to QDOS, i.e.

```
$: qdos circ.q
```

would be, indeed, equivalent to

```

$: qdos --connection_enabled=true --dephasing_enabled=false
  --frequency_output=32 --in_dir="./input/" --measurement_error=0.0
  --measure_memory=10 --n_shots=0 --out_dir="./output/"
  --out_rho_final=dec --out_rho_simulation=dec
  --out_schedule=dec --random_seed=false
  --simulation_enabled=true --simulation_progres=2
  circ.q

```

# Bibliography

- [1] S. BARNETT, *Quantum Information*, Oxford University Press, Inc., USA, 2009.
- [2] H.-P. BREUER AND F. PETRUCCIONE, *The Theory of Open Quantum Systems*, Oxford University Press, 01 2007.
- [3] A. CHIESA, E. MACALUSO, F. PETIZIOL, S. WIMBERGER, P. SANTINI, AND S. CARRETTA, *Molecular nanomagnets as qubits with embedded quantum-error correction*, The Journal of Physical Chemistry Letters, 11 (2020), pp. 8610–8615. PMID: 32936660.
- [4] A. CHIESA, F. PETIZIOL, M. CHIZZINI, P. SANTINI, AND S. CARRETTA, *Theoretical design of optimal molecular qudits for quantum error correction*, J. Phys. Chem. Lett., 13 (2022), pp. 6468–6474.
- [5] A. CHIESA, F. PETIZIOL, E. MACALUSO, S. WIMBERGER, P. SANTINI, AND S. CARRETTA, *Embedded quantum-error correction and controlled-phase gate for molecular spin qubits*, AIP Advances, 11 (2021), p. 025134.
- [6] M. CHIZZINI, L. CRIPPA, L. ZACCARDI, E. MACALUSO, S. CARRETTA, A. CHIESA, AND P. SANTINI, *Quantum error correction with molecular spin qudits*, Phys. Chem. Chem. Phys., 24 (2022), pp. 20030–20039.
- [7] K. DE RAEDT, K. MICHELSEN, H. DE RAEDT, B. TRIEU, G. ARNOLD, M. RICHTER, T. LIPPERT, H. WATANABE, AND N. ITO, *Massively parallel quantum computer simulator*, Computer Physics Communications, 176 (2007), pp. 121–136.

- [8] D. D’ALESSANDRO, *Introduction to quantum control and dynamics*, Chapman and Hall, 2021.
- [9] A. FOWLER, M. MARIANTONI, J. MARTINIS, AND A. CLELAND, *Surface codes: Towards practical large-scale quantum computation*, Physical Review A, 86 (2012).
- [10] H. GOLDSTEIN, C. POOLE, J. SAFKO, AND N. COCCA, *Meccanica classica*, Zanichelli, 2005.
- [11] M. HOWARD AND J. VALA, *Qudit versions of the qubit  $\pi/8$  gate*, Physical Review A, 86 (2012), p. 022316.
- [12] M. KJAERGAARD, M. E. SCHWARTZ, J. BRAUMÜLLER, P. KRANTZ, J. I.-J. WANG, S. GUSTAVSSON, AND W. D. OLIVER, *Superconducting qubits: Current state of play*, Annual Review of Condensed Matter Physics, 11 (2020), pp. 369–395.
- [13] B. P. LANYON, M. BARBIERI, M. P. ALMEIDA, T. JENNEWEIN, T. C. RALPH, K. J. RESCH, G. J. PRYDE, J. L. O’BRIEN, A. GILCHRIST, AND A. G. WHITE, *Simplifying quantum logic using higher-dimensional hilbert spaces*, Nature Physics, 5 (2009), pp. 134–140.
- [14] A. MUTHUKRISHNAN AND C. R. STROUD, *Multivalued logic gates for quantum computation*, Phys. Rev. A, 62 (2000), p. 052309.
- [15] M. A. NIELSEN AND I. L. CHUANG, *Quantum Computation and Quantum Information: 10th Anniversary Edition*, Cambridge University Press, USA, 10th ed., 2011.
- [16] P. J. OLLITRAULT, A. KANDALA, C.-F. CHEN, P. K. BARKOUTSOS, A. MEZZACAPO, M. PISTOIA, S. SHELDON, S. WOERNER, J. M. GAMBETTA, AND I. TAVERNELLI, *Quantum equation of motion for computing molecular excitation energies on a noisy quantum processor*, Phys. Rev. Research, 2 (2020), p. 043140.
- [17] M. RINGBAUER, M. METH, L. POSTLER, R. STRICKER, R. BLATT, P. SCHINDLER, AND T. MONZ, *A universal qudit quantum processor with trapped ions*, Nature Physics, 18 (2022), pp. 1–5.
- [18] J. J. SAKURAI AND J. NAPOLITANO, *Modern Quantum Mechanics*, Cambridge University Press, 3 ed., 2020.

- [19] R. SCHUTJENS, F. A. DAGGA, D. J. EGGER, AND F. K. WILHELM, *Single-qubit gates in frequency-crowded transmon systems*, Phys. Rev. A, 88 (2013), p. 052330.
- [20] F. TACCHINO, A. CHIESA, S. CARRETTA, AND D. GERACE, *Quantum computers as universal quantum simulators: state-of-art and perspectives*, (2019).
- [21] F. TACCHINO, A. CHIESA, R. SESSOLI, I. TAVERNELLI, AND S. CARRETTA, *A proposal for using molecular spin qubits as quantum simulators of light-matter interactions*, Journal of Materials Chemistry C, 9 (2021).
- [22] Y. WANG, Z. HU, B. C. SANDERS, AND S. KAIS, *Qudits and high-dimensional quantum computing*, Frontiers in Physics, 8 (2020).
- [23] D. WILLSCH, M. NOCON, F. JIN, H. DE RAEDT, AND K. MICHIELSEN, *Gate-error analysis in simulations of quantum computers with transmon qubits*, Phys. Rev. A, 96 (2017), p. 062302.
- [24] Y. XU, J. CHU, J. YUAN, J. QIU, Y. ZHOU, L. ZHANG, X. TAN, Y. YU, S. LIU, J. LI, F. YAN, AND D. YU, *High-fidelity, high-scalability two-qubit gate scheme for superconducting qubits*, Phys. Rev. Lett., 125 (2020), p. 240503.
- [25] J. M. ZADROZNY, J. NIKLAS, O. G. POLUEKTOV, AND D. E. FREEDMAN, *Millisecond coherence time in a tunable molecular electronic spin qubit*, ACS Central Science, 1 (2015), pp. 488–492. PMID: 27163013.
- [26] W. ZHONG, Z. SUN, J. MA, X. WANG, AND F. NORI, *Fisher information under decoherence in bloch representation*, Physical Review A, 87 (2013), p. 022337.



# In-depth Readings

- [27] C. CAFARO, F. MAIOLINI, AND S. MANCINI, *Quantum stabilizer codes embedding qubits into qudits*, Phys. Rev. A, 86 (2012), p. 022308.
- [28] S. CARRETTA, D. ZUECO, A. CHIESA, A. GOMEZ-LEON, AND F. LUIS, *A perspective on scaling up quantum computation with molecular spins*, Applied Physics Letters, 118 (2021), p. 240501.
- [29] S. CHICCO, A. CHIESA, G. ALLODI, E. GARLATTI, M. ATZORI, L. SORACE, R. DE RENZI, R. SESSOLI, AND S. CARRETTA, *Controlled coherent dynamics of  $[vo(tpp)]$ , a prototype molecular nuclear qudit with an electronic ancilla*, Chem. Sci., 12 (2021), pp. 12046–12055.
- [30] A. CHIESA, F. TACCHINO, M. GROSSI, P. SANTINI, I. TAVERNELLI, D. GERACE, AND S. CARRETTA, *Quantum hardware simulating four-dimensional inelastic neutron scattering*, Nature Physics, 15 (2019), pp. 455–459.
- [31] S. CLARK, *Valence bond solid formalism for d-level one-way quantum computation*, Journal of Physics A, 39 (2006), pp. 2701–2721.
- [32] D. COZZOLINO, B. DA LIO, D. BACCO, AND L. OXENLOWE, *High-dimensional quantum communication: Benefits, progress, and future challenges*, Advanced Quantum Technologies, 2 (2019), p. 1900038.
- [33] C. HATHHORN, *Engineering a compiler, second edition by keith d. cooper and linda torczon*, SIGSOFT Softw. Eng. Notes, 37 (2012), p. 36–37.
- [34] M. JACKSON, *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices*, ACM Press/Addison-Wesley Publishing Co., USA, 1995.

- [35] B. W. KERNIGHAN AND R. PIKE, *The UNIX Programming Environment*, Prentice Hall Professional Technical Reference, 1984.
- [36] C. P. KOCH, M. LEMESHKO, AND D. SUGNY, *Quantum control of molecular rotation*, Rev. Mod. Phys., 91 (2019), p. 035005.
- [37] T. KRAFT, C. RITZ, N. BRUNNER, M. HUBER, AND O. GUHNE, *Characterizing genuine multilevel entanglement*, Phys. Rev. Lett., 120 (2018), p. 060502.
- [38] P. J. LOW, B. M. WHITE, A. A. COX, M. L. DAY, AND C. SENKO, *Practical trapped-ion protocols for universal qudit-based quantum computing*, Phys. Rev. Research, 2 (2020), p. 033128.
- [39] E. A. MARTINEZ, C. A. MUSCHIK, P. SCHINDLER, D. NIGG, A. ERHARD, M. HEYL, P. HAUKE, M. DALMONTE, T. MONZ, P. ZOLLER, AND R. BLATT, *Real-time dynamics of lattice gauge theories with a few-qubit quantum computer*, Nature, 534 (2016), pp. 516–519.
- [40] S. PIRANDOLA, S. MANCINI, S. L. BRAUNSTEIN, AND D. VITALI, *Minimal qudit code for a qubit in the phase-damping channel*, Phys. Rev. A, 77 (2008), p. 032309.
- [41] M. RINGBAUER, T. R. BROMLEY, M. CIANCARUSO, L. LAMI, W. Y. S. LAU, G. ADESSO, A. G. WHITE, A. FEDRIZZI, AND M. PIANI, *Certification and quantification of multilevel quantum coherence*, Phys. Rev. X, 8 (2018), p. 041007.
- [42] D. RITCHIE AND B. KERNIGHAN, *The C programming language*, Bell Laboratories, 1988.
- [43] P. SCHINDLER, D. NIGG, T. MONZ, J. BARREIRO, E. MARTINEZ, S. WANG, S. QUINT, M. BRANDL, V. NEBENDAHL, C. ROOS, M. CHWALLA, M. HENNRICH, AND R. BLATT, *A quantum information processor with trapped ions*, New Journal of Physics, 15 (2013), p. 123012.
- [44] D. G. TEMPEL AND A. ASPURU-GUZI, *Relaxation and dephasing in open quantum systems time-dependent density functional theory: Properties of exact functionals from an exactly-solvable model system*, Chem-

ical Physics, 391 (2011), pp. 130–142. Open problems and new solutions in time dependent density functional theory.