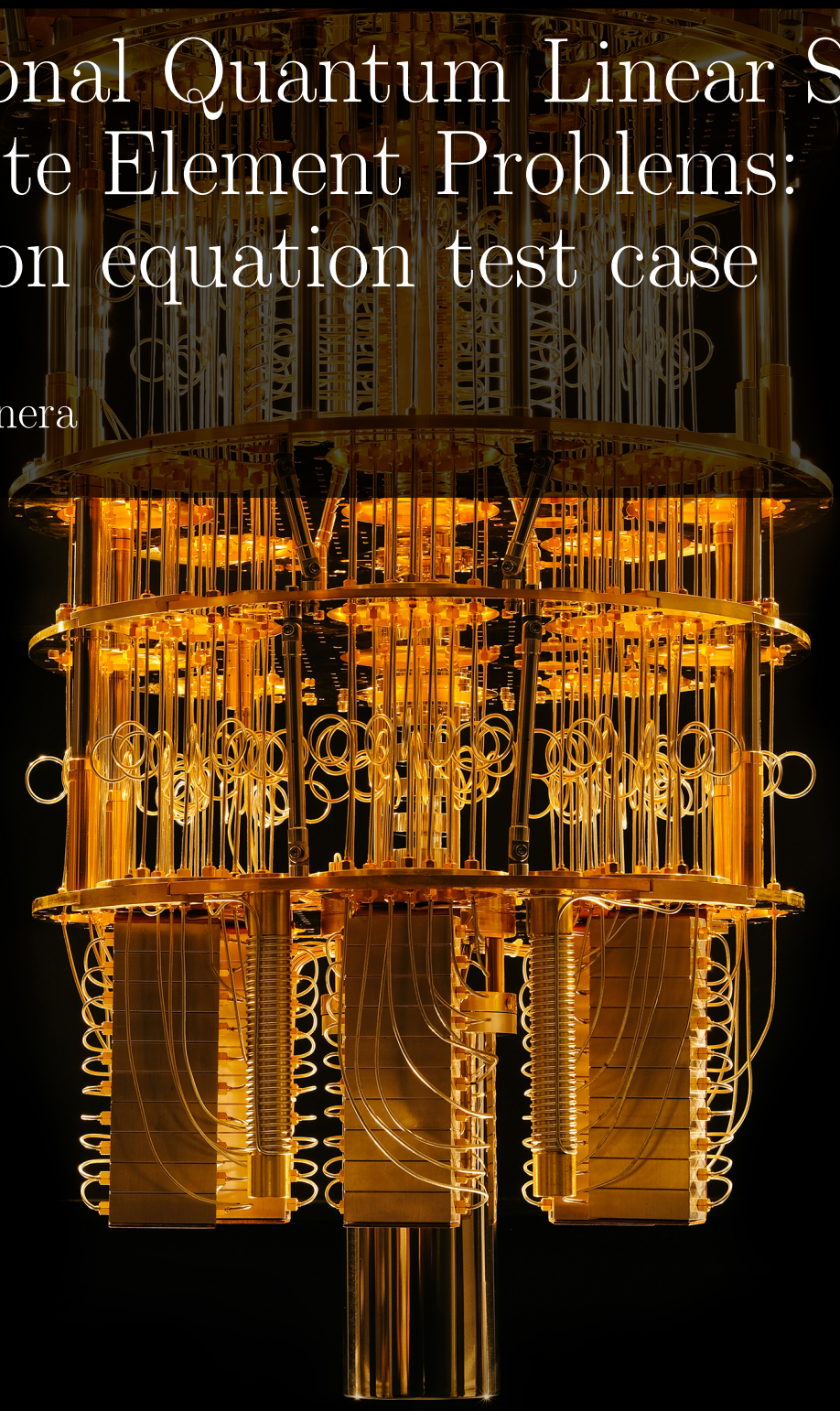
Variational Quantum Linear Solver for Finite Element Problems: a Poisson equation test case

Enrico Cappanera



Delft University of Technology

TU Delft

Variational Quantum Linear Solver for Finite Element Problems: a Poisson equation test case

by

Enrico Cappanera

to obtain the degree of Master of Science at
the Delft University of Technology, to be
publicly defended on 17th September 2021

Student Number: 4943872
Date Completed: 17th September 2021
Thesis Supervisors: Dr.ir. M.I. Gerritsma, TU Delft
Dr.ir. M. Möller, TU Delft

TU Delft

Abstract

Computational fluid dynamic is governed by Navier Stokes equation, a set of partial differential equations which are notoriously difficult to solve and computationally expensive when tackled numerically. Over the years, simplified models of these equations have been developed to solve these issues, achieving faster convergence at the expense of accuracy. On the other hand, this research investigates the possibility of implementing these equations on quantum computers, an emerging computational technology that promises computational speed-ups in specific settings.

In particular, this work focuses on the Variational Quantum Linear Solver (VQLS) and evaluates the feasibility of solving a one dimensional Poisson's equation as a simplified proxy for Navier Stokes. While many quantum algorithms exist, VQLS is one of the few capable of solving linear equations that was proven feasible on real hardware for simple test cases and was therefore chosen as most suitable for this proof of concept. This research is divided into two main blocks: the first is dedicated to the pre-processing necessary to solve Poisson's equation with VQLS; the second contains numerical runs of VQLS, the evaluation of its performance and scaling challenges.

In this work, Poisson's equation is firstly discretized so that the resulting linear system can be cast in VQLS. To this extent, the linear system matrix A has to be decomposed as a linear combination of unitary matrices that are then represented by quantum logic gates. This step is crucial because the number of unitaries in the decomposition is tied to the overall algorithm efficiency. Therefore, the number of gates in the decomposition and their complexity (e.g. interconnectivity requirements) were chosen as a compatibility metric between discretized Poisson's equation and VQLS.

When using Pauli Gates as a basis, both hybrid finite element discretization and first-order finite element are inefficiently decomposed, with respectively $\mathcal{O}(N^{m>1})$ and exactly N terms, with $N = 2^n$ and n being the number of qubits. On the other hand, using gates with higher entanglement allowed decomposing A in $n + 3$ unitaries, which is polynomial in n, hence efficient. Moreover, the identified gate decompositions for first-order finite element discretization can be easily generalized to a higher number of dimensions and become progressively more efficient while doing so. In summary, it was not possible to find a gate decomposition both efficient and with limited hardware requirements because a simple de-

composition requires exponentially many unitaries, whereas an efficient one requires complex multi controlled gates. Therefore, it is deemed unlikely that Poisson's equation can be efficiently implemented on a quantum computer in the near term.

The high-entanglement decomposition of first-order finite element discretization was chosen as a test case for numerical runs because it was the only efficient decomposition found. These were carried out on a quantum simulator (Qiskit QasmSimulator) using a hardware efficient ansatz. For each numerical run, iterations to success and minimum cost achieved were measured as an efficiency metric. In general, iterations to success were larger than the best classical counterpart and scaled exponentially for increasing qubits numbers. Across all the experiments performed, scalability was an issue: while 2-qubits runs converged to satisfactory results for most runs, 3-qubits runs seldom converged, and 4-qubits runs never converged.

The major hurdle preventing convergence was the plateauing of the cost function for increasing qubit numbers. This gradient vanishing was measured by extensive random sampling of the cost function gradient through the domain. Moreover, for this test case, because the gradient tends to vanish, maintaining the same level of precision on its evaluation would require an exponential increase in shot number for an increasing number of qubits, which would hamper the scalability of VQLS.

Throughout the analysis, several techniques were attempted to improve convergence. Acting on the ansatz by various types of gradual optimization did not improve convergence, whereas using a shallow ansatz improved convergence but compromised the solution's accuracy. On the other hand, using a non-normalized cost function allowed easier convergence compared to a normalized one.

Overall, in the pre-processing analysis an efficient yet simple implementation of one dimensional Poisson's equation was not found, and numerical experiments showed serious convergence difficulties. Therefore, this research hints at an incompatibility between variational quantum solver and Poisson's equation. However, future works should attempt other convergence improvement strategies such as local cost function and higher-order discretization methods before a conclusive assessment can be drafted.

Preface

What a journey! This thesis marks the end of my stay at TU Delft, a great learning experience from the very first day until the last. Certainly, a journey marked with sacrifices and hard work but also with fun and personal growth.

Meeting talented people of my study course was an inspiring and eye-opening experience. In particular, I would like to thank Giulio, Jacopo and Andrea for all the times we helped each other and the fun we had at the library. You managed to make 12-hours long study sessions fun, quite an achievement! Also, special thanks to my former flatmates Klemens and Jasper for the fun times together. Our dinners were literally the only fun thing I did in Delft during the pandemic.

A thank you to the people that helped me complete this work: my supervisors Marc and Matthias, for the courage of embarking on a journey outside our core competences and comfort zone; Giorgio for sharing our quantum struggles and your advice; Gawel for the useful emails and tips; Mario for helping me find the Pauli decomposition pattern; and the Stack Exchange community for the brilliant answers.

Thank you to my long-time friends from Ancona, which remained close despite the passing years.

Thank you to my girlfriend Polina for being patient enough to cope with me for all these years, the great times we had together, and the faith we could make it happen no matter the difficulties.

Above all, I would like to thank my parents and my sister for the everlasting support and for believing in me notwithstanding my delays in delivering this work. Undoubtedly, this thesis would not have been possible without your help.

Enrico Cappanera
Ancona, September 2021

Contents

List of Figures

Chapter 1

Introduction

Fluid flow phenomena can be described by a set of partial differential equations, namely Navier-Stokes equations. For most cases, these equations do not have an analytical solution. Therefore, it is necessary to solve these equations numerically, which is generally achieved by discretization of the domain. For example, some methods compute the equations only at specific points in space and time. As a result, the initial set of partial differential equations is approximated as a set of algebraic equations, which can be solved by a computer [5] (p.25). This procedure is at the essence of computational fluid dynamics (CFD), which generally consists of using numerical analysis to solve fluid flow problems.

Often these equations have proven to be computationally expensive, if not impossible to solve in some cases. For example, for some applications, a direct numerical simulation (DNS) of Navier Stokes equations for high Reynolds number is beyond the capabilities of the world's most powerful supercomputers [6]. At the moment, simplified models of these equations are generally employed, at the expense of accuracy. Some examples are Reynolds averaged Navier–Stokes (RANS) where only the time-averaged flow is computed [5] (page 397), or Large-eddy simulation (LES) where only the largest and more relevant length-scales of the flow are computed directly, whereas smaller scales are modelled [7]. While in some cases simplifying the problem can lead to satisfactory results, increasing computation power would be beneficial to solve more complex problems or increase accuracy. Therefore, it is interesting to investigate if a novel computational technology such as quantum computers can be employed to solve CFD problems. In literature, only a few instances are addressing this problem [8, 9, 10][1], so it is an open and interesting question whether this is feasible or not.

The idea of a quantum computer dates back to the 80s, when scientists started to investigate whether quantum mechanics could be used to perform computations. Noticeable is the early work from Benioff, who introduced a quantum mechanical model of a computer [11]. Feynman himself speculated that since the nature of the world is governed by quantum mechanics, then a computational machine based on those laws would be appropriate to

[1] To the best of the author's knowledge, not necessarily exhaustive

simulate it [12].

Since then, the development of quantum computers progressed noticeably: in autumn 2019 a team from Google was able to claim quantum supremacy for a specific computational task [13][2]. In simple words, quantum supremacy means that the quantum computer can perform a computational task that its classical counterpart would not be able to perform. However, quantum computers are not yet developed enough for most practical applications. One could say that they are in their early stages, in a similar way to classical computers in the middle of the last century.

Despite quantum information theory being a relatively 'recent' field, researchers have developed a broad collection of algorithms based on different methodologies and for disparate applications, for example [15, 16]. Some noteworthy applications are: machine learning [17], portfolio optimization in finance [18], computational chemistry [19], and others. This research is focused on algorithms for linear systems of equation (for example [20]), which is a ubiquitous problem in engineering. Of particular interest for the scope of this research is their application to differential equations.

1.1 Problem statement and approach

The objective of this research is to gain preliminary insights into possible implementations of computational fluid dynamics on quantum computers. In particular, investigations are focused on variational quantum algorithms for linear systems of equations, the compatibility of current discretization methods with these algorithms, and scaling prospects.

One dimensional Poisson's equation was chosen as a test-case for this work as a simplified proxy for NS, also because of its simplicity and wide applicability. Overall, in this research, rather than devising a custom quantum algorithm for Poisson's equation, this equation is firstly discretized and its linear system is solved using the Variational Quantum Linear Solver (VQLS), an existing quantum algorithm for linear systems of equations.

Therefore, the first question to be addressed is what discretization scheme is more appropriate and compatible with VQLS. Depending on the discretization method used, the resulting linear system matrix will be different. Thus, it is reasonable to expect some matrices will be easier and more compatible to solve with VQLS than others. Therefore, this work focuses on finding the best possible implementation of these matrices on VQLS and compares first order finite element method and hybrid finite element method discretizations. For each linear system, conclusions are drawn about expected efficiency and implementation issues.

[2]This claim is not universally accepted among researchers, see `https://www.ibm.com/blogs/research/2019/10/on-quantum-supremacy/` and [14]

While preliminary efficiency insights can already be drawn from the implementation of this matrices on VQLS, in this research, Poisson's equation is solved using VQLS on a quantum simulator to evaluate whether a solution can be achieved, understand challenges and scaling prospect. To this extent, VQLS implementation of Poisson's equation with Dirichlet boundary condition on a uniform domain was experimented, so that numerical results could be compared with analytical solutions. Generally, for each numerical experiment iterations to solution and accuracy of solution are measured so that different solution techniques (such as different optimizers) can be compared.

1.2 Outline

Chapter 2 contains a brief literature review, necessary to understand the content of this research and upon which this analysis is based. Chapter 3 explains the preprocessing necessary to solve discretized differential equations using a variational linear solver and some preliminary feasibility assessments of each technique. Mainly, this consists of decomposing the matrix one wants to invert into unitary matrices, which are composed using quantum logic gates. Chapter 4 contains the results of numerical experiment of the selected test case on a quantum simulator. Here convergence behaviour, issues and potential solutions are tested. Finally, Chapter 5 summarizes the main conclusion of this research and suggests the most logical next steps following the completion of this work. Quantum computing fundamentals that are often referenced in the text are briefly explained in the appendix.

Chapter 2

Theoretical Background

As explained in the introduction, fluid dynamics equations can be approximated by a linear system of equations. Therefore, this chapter focuses on the theory of variational quantum algorithms and their application to linear equations. Sections 2.1 briefly introduces variational quantum algorithms, Section 2.2 elaborates on how to implement these algorithms. In Section 2.3, existing applications of variational algorithms to linear systems of equations are discussed.

2.1 Introduction to variational quantum algorithms

Variational quantum algorithms were firstly introduced by Peruzzo in 2014 when he developed a Variational Quantum Eigensolver (VQE), which is a hybrid quantum-classical algorithm that allows finding the eigenvalues of a Hamiltonian [16]. As the name suggests, the algorithm utilizes quantum and classical resources at the same time to decrease the necessary coherence time, which, in simple words, is the time information can be encoded in a qubit before it loses information to the environment.

Motivation

Quantum computers available today are often named Noisy Intermediate-Scale Quantum (NISQ), where Intermediate refers to the scale of their size (less than a few hundred qubits) and Noisy to the fact the qubits are not perfectly controlled [21]. A Noisy qubit implies that only a finite amount of elementary computations (gates) can be performed before the signal to noise ratio drops (loss of coherence). Because of these limitations, algorithms like the famous quantum linear solver HHL [20] that could solver linear system of equations up to exponentially faster on a quantum computer are most likely decades away. For example, Quantum phase estimation (QPE), which is a fundamental block of the aforementioned algorithm, might require millions of gates [22], which is substantially more than what is achievable by a NISQ device today.

2.2 Implementation of variational quantum algorithms

This section explains the basic building blocks of variational quantum algorithms. More details about the underling theory (where concepts such as expectation value are discussed) can be consulted in Appendix A.2. At first, Subsection 2.2.1 shows the general workflow of a variational algorithm. Then, Subsection 2.2.2 is dedicated to the first step of the variational algorithm, which is the preparation of a state vector. Subsection 2.2.3 investigates the scaling of the state preparation parameters and connected challenges. Subsection 2.2.4 explains different techniques used to measure the expectation value in variational algorithm. Lastly, in 2.2.5 an overview of different optimization techniques is presented.

2.2.1 Algorithm Workflow

Consider a general problem in which one wants to minimize the expectation value of an Hamiltonian H. In general, a Hamiltonian can be derived from physical modelling of the problem or algebraic manipulation of a system of equations. Jarrod R. McClean, one of the inventors of Variational quantum algorithms, describes their workflow as follows [23]: given a state vector $|\psi\rangle$ that can be parametrized as a function of classical parameters $\boldsymbol{\theta}$:

1. a guess state $|\psi(\boldsymbol{\theta})\rangle$ is prepared;
2. the expectation value of the Hamiltonian $\langle H\rangle_{|\psi\rangle}$ is measured;
3. a classical optimizer is used to modify $\boldsymbol{\theta}$ and determine the new 'guess' $|\psi(\boldsymbol{\theta})\rangle$;
4. step 2 and 3 are repeated until convergence to a solution is reached.

'Reaching convergence' means the algorithm finds $\boldsymbol{\theta}$ that allows preparing the state $|\psi(\boldsymbol{\theta})\rangle$ that minimizes the energy of the Hamiltonian: i.e the solution. In the following subsection, each of the steps and the challenges it entails will be explored.

2.2.2 State preparation ansatz

As explained above, the first step of a variational algorithm is to prepare a state $|\psi(\boldsymbol{\theta})\rangle$. It is noteworthy that the quantum state depends on classical parameters, so its state is stored classically, hence the hybrid nature of variational solver. In other words, the vector of classical parameters $\boldsymbol{\theta}$ could be regarded as an input or solution guess for an optimization process, and the quantum computer a tool to evaluate the cost function.

In a quantum circuit, qubits are initialized in a predetermined state at the beginning of the computation (usually, the zero state $|0\rangle$). At this stage, the qubits hold no information, they are not entangled, neither they are in a superposition state. Thus, a sequence of unitary gates $V(\boldsymbol{\theta})$ named "ansatz" is applied to reach the desired quantum state:

$$\psi(\boldsymbol{\theta}) = V(\boldsymbol{\theta})\,|\mathbf{0}\rangle \tag{2.1}$$

In the most general case, the sequence of unitary gates V depends on two sets of parameters:

$$V(\boldsymbol{\theta}) = U_{k_L}(\theta_L)...U_{k_i}(\theta_i)...U_{k_0}(\theta_0) \tag{2.2}$$

where k is a discrete parameter that determines the type of gate and their qubit position, whereas α is a continuous parameter that determines the action of the gate itself. For instance, one gate of the sequence could be a $R_{y_2}(\theta_j)$ which is a rotation along the y axis of the Bloch sphere, [24]. In this case, the k index indicates the gate type (rotation around y) and the target qubit 2, whereas θ indicates the magnitude of the rotation. In general, $U_{k_i}(\theta_j)$ can be any unitary gate, including multi-qubit gates (in this case multiple i indices could be necessary depending on the notation adopted).

It is easily understandable that increasing the number of gates allows searching throughout the Hilbert space more thoroughly. On the other hand, as the number of gates grows, the parameters that have to be optimized to achieve a solution increase. In other words, a longer gate sequence allows for a more accurate solution but increases the complexity of optimization of the algorithm. This is the famous "curse of dimensionality" that afflicts many optimization problems. Although different, one could imagine that increasing the number of unitaries has a similar effect of increasing the number of points in a numerical discretization. A finer grid often results in a more accurate representation of the function but also a higher computational cost.

Now that the functioning of an ansatz has been explained, one might wonder how to choose the type of gates. Naively, one could think that a random set of gates would provide an unbiased and optimal choice. However, generally, it is convenient to use a structured guess. In other words, using insights about the physics of the problem allows for searching only the portion of the Hilbert space that is relevant for the solution, which means resources are used most efficiently. In general, some ansatzes used in the literature are:

1. Unitary coupled cluster (UCC) ansatz [25]
2. Hardware efficient ansatz [1]
3. Adaptive Derivative-Assembled Pseudo-Trotter ansatz (ADAPT-VQE) [26]
4. Quantum circuit structure learning [27]

Hardware efficient Ansatz

The idea of hardware efficient ansatz it to build a simple ansatz with limited hardware requirement. In other words, the structure of the ansatz is dictated by the available hardware rather than problem specific knowledge. In particular, it uses gates that are native to the hardware, as well as it requires limited connectivity, in the sense that two-qubits gates are applied only to neighbouring qubits. Kandala et al. [1] proposes the following structure: entanglers U_{ENT} alternated to single qubit rotations $U^{q,i}(\boldsymbol{\theta}) = R_z^q(\theta_1^{q,i})R_x^q(\theta_2^{q,i})R_z^q(\theta_3^{q,i})$.

Where q identifies the qubit and i the depth position (in plain words, depth is the position along the circuit "wire" or the order a logical operation is performed). A parallelism can be drawn with euclidean geometry, where any rotation about the origin can be expressed as a composition of three rotation, when only one Euler angle is changed at the time. Similarly, these three rotations allow to reach any point of the Bloch sphere. Thus, the ansatz final structure is expressed as:

$$|\phi(\boldsymbol{\theta})\rangle = \prod_{q=1}^{N} \left[U^{q,d}(\boldsymbol{\theta})\right] U_{\text{ENT}} \prod_{q=1}^{N} \left[U^{q,d-1}(\boldsymbol{\theta})\right] \cdots U_{\text{ENT}} \prod_{q=1}^{N} \left[U^{q,0}(\boldsymbol{\theta})\right] |00\ldots0\rangle \tag{2.3}$$

which is visualized in Figure 2.1:

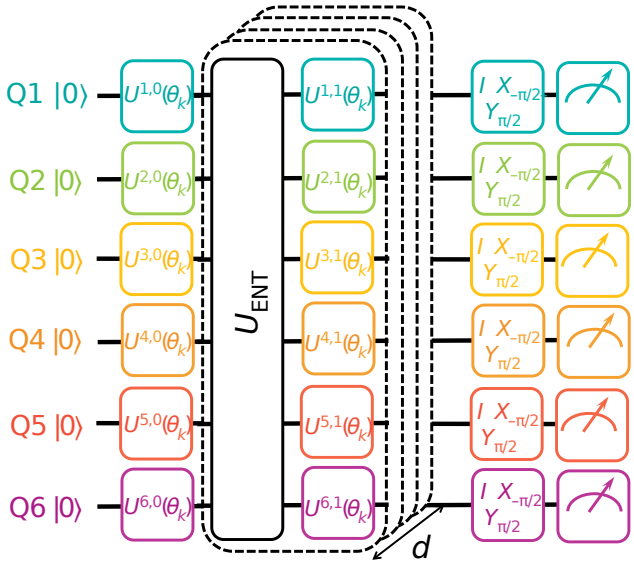


Figure 2.1: Illustration of hardware efficient ansatz [1]

While other ansatzes' structures are inspired by the physics of the problem, a hardware efficient ansatz does not require this knowledge. This peculiarity makes it suitable for linear system of equations where this additional information is not available. Several applications of this ansatz can be found in literature, where depending on the application, one chooses the rotations and entangling gates that are most suitable.

For example, if one is only interested in real numbers, only rotations around y are necessary $U^{q,i}(\theta) = R_y^{q,i}(\theta^{q,i})$. This reduces the number of necessary parameters by a factor of three, compared to the previous ansatz. For instance, in [28], R_y gates are used for rotations and CNOT as U_{ENT}. Similarly, [2] uses the same rotations but CZ gates as entanglers. In [29], $U^{q,i}(\boldsymbol{\theta}) = R_x^{q,i}(\theta_1^{q,i})R_y^{q,i}(\theta_2^{q,i})$, and U_{ENT} =CNOT.

2.2.3 Scaling of the ansatz

In summary, a generic quantum state is prepared by a set of unitary gates $V(\boldsymbol{\theta})$ which is called an ansatz. The reason behind this name is that, in general, it is not possible to know the structure of the circuit which would lead the solution a priori. Therefore, the algorithm starts with a guess that depends on parameters that can be optimized. The following paragraphs are dedicated to explaining two of the core problems of using ansatzes: the scaling of the number of unitaries and the problem of the vanishing gradient. It is important to answer this question to evaluate the limits of the variational approach and understand if these algorithms would be able to tackle real problems once the hardware scales up.

Number of unitaries scaling

Given that a physical state can be approximated by a sequence of gates, it is important to understand how long this sequence is or in other words, what is the number of unitaries necessary. The length of this gate sequence is important for several reasons: firstly, for practical application, the gate length must not exceed the coherence time capabilities of the hardware. Secondly, an increasing number of gates increases the runtime of the algorithm. Thirdly, as the number of gates increases, the number of optimization parameters increases as well, which makes the optimization harder.

When discussing the scaling of unitaries, one fundamental theorem that cannot be overlooked is Solovay-Kitaev. Let us, consider a set of unitary gates G in an d dimensional subset of $SU(d)$, that respects:

1. All gates $g \in G$ are contained in $SU(d)$
2. For each g its adjoint $g^\dagger$ is also in G
3. G is universal for $SU(d)$. In other words, given any unitary $U \in SU$ there exist a set of gates $S = g_1...g_l$ that approximates U with precision $\epsilon > 0$.

Then, **Solovay-Kitaev** loosely states that for any gate $U \in SU(d)$ there is a sequence S in G that approximates U with precision ϵ and length $O(log^c(1/\epsilon))$, where c is a constant. [30]

Furthermore, it is important to specify what does it mean for a gate S to approximate another gate U with precision ϵ. Formally, we define it as $d(U, S) = sup_{\|\psi\|=1}\|(U - S)\psi\| < \epsilon$. According to Dawson et al. [30], $c \approx 3.97$. However, as reported by the authors, there are several other proofs with different bounds.

To explain why this theorem is so important, consider the case when one wants to perform a quantum Fourier transform (QFT). This is a relevant example as QFT is a basic building block of some of the most famous quantum algorithms, for example, Shor's factoring or HHL [15, 20]. A QFT requires single-qubit rotations of size $e^{2\pi i/2^k}$ to be successfully implemented

[24]. However, it is easily understandable that these types of gates might not be native to quantum hardware. For instance, a quantum computer might be able to perform only $\pi/8$ rotations, Hadamard, and identity gates. Thus, the question that arises in these cases is: how expensive would it be to approximate the required gate with those that are available?

It should not be surprising that this question is crucial for the success of the algorithm: if an exponential number of gates are required to approximate a specific one, then the exponential gains in terms of computational speed-up promised by the algorithm might be lost. Luckily, the SK theorem cited above guarantees that SU is filled *quickly* by the available gates, which means that the number of necessary gates scales logarithmically. Providing a proof for this theorem goes beyond the scope of this thesis, but the interested reader can consult [30] or Appendix number 2 of [24]. It is interesting to mention that the proof is based on recursion: starting from an approximation with ϵ_0 it is possible to increase the precision by recursively approximating.

However, in a variational algorithm, we are not only interested in how efficiently a single-qubit gate can be approximated, but also multi-qubit gates. Interestingly, Solovay-Kitaev was written for a general qudit (in plain words, the superposition of d qubits). This means that the above theorem can be extended to a multi-qubit gate just by setting the dimension of the space $d = 2^n$ where n is the number of qubits.

The proof for this case is similar to the single-qubit gate. The curious reader is invited to look at [30] (Section 5). The main difference lies in the initial sequence necessary to approximate a gate with ϵ_0 error, which is the base for the recursion process. Because in this case $SU(d)$ is a manifold in $d^2 - 1$ dimensions, approximating every gate within ϵ_0 would require $O(1/\epsilon_0^{d^2-1})$ sequences. Nielsen and Chang also provide an interesting bound taking the ratio of surface to volume of a sphere in $2^n + 1$ dimensions. [24]

It is necessary to remark that SK only states that given a certain sequence, any other universal set approximates it efficiently. Thus, the double exponential scaling does not depend on Solovay Kitaev, it is just a basic mathematical consequence of the exponential increase in degrees of freedom of a quantum state.

For these reasons, if one has physical insights or other information about the behaviour of the solution, it is possible to decrease the size of the ansatz. In other words, it is possible to explore only the portion of the Hilbert space which is expected to contain the solution. Although this effectively allows decreasing the size of an ansatz it leaves with the most important question of this section: *what is an appropriate ansatz for a given problem?* In other words, given that a more compact ansatz explores only a small portion of all the possible states, how does one choose it such that the probability of the solution being in the ansatz range is maximized? This is a topic of ongoing research and several structures that have been proposed in the literature are reported in the following paragraphs.

In general, the utility of SK in practical applications (variational algorithms) is limited because it gives no information about the optimal shape or type of gates for an ansatz. In [2], Bravo-Pietro et al. investigate the scaling of the depth of a variational quantum algorithm for a condensed matter system. The ansatz utilized in the paper is reported in Figure 2.2.

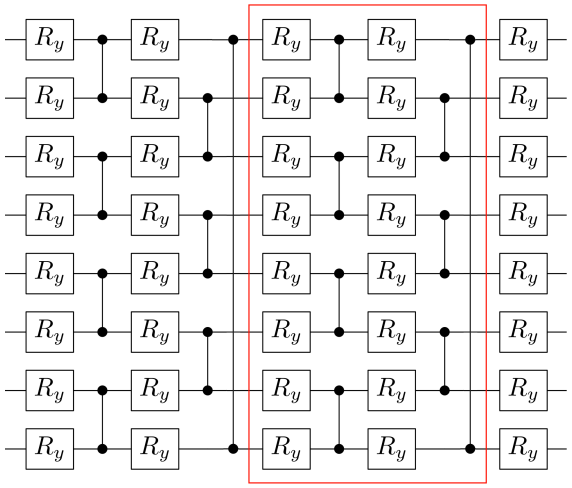


Figure 2.2: Variational ansatz from [2]. The ansatz is made of layers (a layer is identified in the red square). Every layer is composed of R_y rotations followed by CZ gates.

The CZ gates have the crucial function of entangling the qubits, whereas R_y is a single qubit rotation. In the aforementioned paper, the VQE with this specific ansatz is benchmarked with two quantum spin chains, the Ising model in the transverse field and the XXZ chain. For these particular problems, the authors found an exponential accuracy increase with increasing ansatz depth. Thus, this is a perfect example of what was mentioned above: the chosen ansatz does not explore the whole Hilbert space, but the solution happens to be within its range, and therefore it can be implemented extremely efficiently.

The key takeaways of this paragraph are that in general, it is impossible to build an ansatz capable of representing every state that also scales efficiently: it is necessary to make a trade-off between the power of the ansatz and its computational cost. Moreover, this means that most likely **it is impossible to build an efficient all-purpose ansatz**. Only problem-tailored ansatzes are likely to provide an advantage with respect to classical computing.

The problem of vanishing gradient

Along with the scaling of the number of gates, there is another common issue that characterizes variational solvers using hardware efficient ansatzes: the vanishing of the gradient. This issue shares some similarities with gradient-based training issues of artificial neural

networks. This paragraph is largely based on a work of McClean [31]. To clearly describe the problem let's recall the basics of a variational algorithm, which is to minimize the cost or expectation value of a Hamiltonian:

$$C(\boldsymbol{\theta}) = \langle H \rangle_{|\psi(\boldsymbol{\theta})\rangle} = \langle 0| \, V(\boldsymbol{\theta})^\dagger H V(\boldsymbol{\theta}) \, |0\rangle \tag{2.4}$$

A randomly parametrized circuit has the structure:

$$V(\boldsymbol{\theta}) = \prod_{l=1}^{L} V_l(\theta_l) W_l \tag{2.5}$$

where $V_l(\theta_l) = \exp(-i\theta_l U_l)$ and U_l is an Hermitian operator. One simple example is when V_l is a Pauli gate. Similarly, W_l is an unitary operator. In this case, the derivative of the cost function is computed using chain rule as:

$$\frac{\partial C(\boldsymbol{\theta})}{\partial \theta_k} = \langle 0| \, i V_-^\dagger U_k V_+^\dagger H V - i V H V_+ U_k V_- \, |0\rangle = i \, \langle 0| \, V_-^\dagger [U_k, V_+^\dagger H V_+] V_- \, |0\rangle \tag{2.6}$$

In the expression above, $V_- = \prod_{l=1}^{k-1} V_l(\theta_l) W_l$ and $V_+ = \prod_{l=k}^{L} V_l(\theta_l) W_l$. According to McClean's work [31], the problem with randomly parametrized circuits is that the average value of the gradient of the cost function $\boldsymbol{\nabla} C(\boldsymbol{\theta})$ tends to 0 and the probability that it deviates from its average value by ϵ decreases exponentially with the number of qubits. According to the authors, this is due to the phenomena of concentration of measure for high dimensional spaces. In particular, this is formalized by Levy's Lemma: consider a set of points in a hypersphere ϕ of dimension d and area $S[\{\phi\}]$, and f such that $|\boldsymbol{\nabla} f| < 1$ then

$$\frac{[S\{\phi | f(\phi) - \langle f \rangle \geq \epsilon\}]}{S[\{\phi\}]} \leq 4 \exp\left(-\frac{(d+1)\epsilon^2}{9\pi^3} \right) \tag{2.7}$$

that is to say, the points where f is not close to its average with respect to the totality of the points decreases exponentially with the number of dimensions [32].
Thus, McClean proves and numerically verifies that the gradient does indeed vanish experientially in the number of gates (that is $\propto L$) which allows drawing the conclusion that randomly initialized circuits of sufficient depth have little utility for hybrid quantum-classical algorithms, as they get progressively less functional for more qubits or deeper ansatzes.

Luckily, several solutions have been proposed in the literature. Firstly, it should be pointed out that the loss of utility, in particular, affects random circuits. Therefore, this issue does not affect ansatzes inspired by the physics of the problems, such as Unitary Coupled Clusters [33]. However, for some applications (for example solving liner problems) there are no physical insights available. For those cases, some solution provided in the literature are the following:

- Gradually adding layers: the ansatz is initialized as a low depth circuit and gradually expanded.

- Introducing a local cost function [28]
- Using Hamiltonian Morphing: this method is based on the Adiabatic theorem and consists of evolving the problem's Hamiltonian from the Identity to the desired final state.

Gradually adding layers: Grant et al. [34] suggest dividing a hardware efficient ansatz into M blocks. Every block is divided into two sub-blocks as following:

$$V_m(\boldsymbol{\theta}_m) = \prod_{l=1}^{L} V_l(\theta_{l,1}^m) \prod_{l=1}^{L} V_l(\theta_{l,2}^m). \tag{2.8}$$

When initializing the circuit, on every block $\theta_{l,1}$ are chosen randomly, whereas $\theta_{l,2}$ are chosen so that $V_m(\boldsymbol{\theta}_m) = I$, that is to say, one sub-block is the inverse of the other. One of the blocks is chosen with a random initialization so that the gradient of that block can be computed, whereas all the other blocks are equivalent to the identity. The reason behind this choice is that this reduces the effective depth of the circuit, and thus the likelihood of the gradient vanishing. Then, the blocks are progressively optimized. Although it cannot be proven algebraically that the single block initialization will not result in a gradient plateau, this method was validated numerically by the authors. A similar procedure was successfully implemented in [35] as well.

Hamiltonian Morphing is clearly explained by Xi et al. in [29]. When solving a linear system $A|x\rangle = |b\rangle$ with variational algorithms, one builds a Hamiltonian that measures the projection of $A|x\rangle$ on the subspace perpendicular to b. The idea behind this method is to evolve the aforementioned Hamiltonian from an initial state (identity) to a final state that represents the problem considered. The authors also mentioned that According to the Adiabatic theorem, if A is positive definite, ∂t small enough and the ansatz powerful enough, this method is guaranteed to find a solution [36]. The simplest parametrization possible for the linear system matrix is $A(t) = (1 - t/T)I + t/TM$, which means A evolves linearly over time. Thus, the ansatz is parametrized as $V(\boldsymbol{\theta}(t))$. At every time step n, the ansatz is initialized with the values $\boldsymbol{\theta}((n-1)dt)$ that were obtained minimizing the cost function at the previous time step, and are minimized again for the evolved system.

2.2.4 Expectation value Measurement

As explained in the previous sections, a variational algorithm works by preparing a quantum, state, then by measuring its expectation value with respect to a Hamiltonian and successively minimizing the obtained value. While sub-Sections 2.2.2, 2.2.3 were dedicated to the state preparation, this section is dedicated to the next step of the algorithm, which is determining how the expectation value $\langle H \rangle_{|\psi\rangle}$ is measured.

In this framework, the expectation value is effectively a cost function that the algorithm attempts to minimize. Therefore, it is necessary to build a circuit that can measure this

value, and more importantly, assess how expensive this procedure is and its feasibility on a quantum computer. In other words, some evaluation procedures might require deeper circuits and fewer repetitions, whereas others have less stringent requirements but have a longer runtime. In general, evaluation procedures can be divided into direct and indirect measurements. When a direct measurement is performed, the quantum state collapses in the measurement basis because the qubit that encodes the information is measured, whereas indirect measurements allow it to "stay intact" [37] since an ancillary qubit is measured and the register is left untouched. One example of common indirect measurement is the Hadamard test, which allows the determination of the expectation value of a unitary $\langle U \rangle_{|\psi\rangle}$ using an ancillary qubit.

One of the main differences between the two approaches is that an indirect measure allows reusing the quantum state (for example, iterative quantum phase estimation [38]). These algorithms run in $O(1/\epsilon)$, whereas, for example, VQE requires $O(1/\epsilon^2)$ using direct measurements [16]. Thus, the latter can be summarized by saying that direct measurement techniques are generally less efficient in terms of runtime, but they are easier to implement on a NISQ. A short comparison of the measurement subroutines is reported in Table 2.1

Table 2.1: Summary of measurement techniques. The columns indicate what is being measured and the row how it is being measured. A more in-depth description of this techniques can be consulted in [39]

	Expectation value measurements	Overlap Measurement
Indirect measurement	Hadamard Test	Swap Test
Direct measurement	Quantum expectation estimation	Destructive Swap Test

Indirect measurements: Hadamard Test

The Hadamard test is a quantum subroutine that allows to measure the expectation value of a unitary with respect to a state $\langle U \rangle_{|\psi\rangle}$. Its circuit structure is reported below

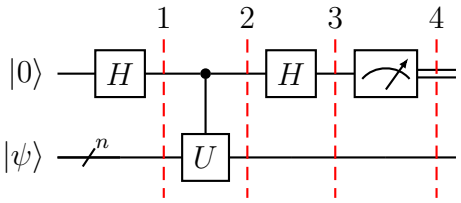


Figure 2.3: Circuit that computes the Hadamard Test. Dashed lines represent steps that are referenced in the text

In Figure 2.3, the underling idea is that the probability of measuring 0 in the first

(ancilla) qubit is equal to $P(0) = \frac{1}{2}(1 + \mathrm{Re}\,\langle U\rangle)$ and the probability of measuring 1 is $P(1) = \frac{1}{2}(1 - \mathrm{Re}\,\langle U\rangle)$. Thus, taking their difference will return the real part of the expectation value $P(0) - P(1) = \mathrm{Re}\,\langle U\rangle$. After the application of the Hadamard gate, in step 1, the circuit is initially in the state:

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |\psi\rangle = \frac{|0\rangle \otimes |\psi\rangle + |1\rangle \otimes |\psi\rangle}{\sqrt{2}} \tag{2.9}$$

where the first term is the ancilla qubit and it is obtained directly by applying the Hadamard gate to $|0\rangle$, whereas the second term is the state $|\psi\rangle$ that is being evaluated. Then, the controlled unitary U is applied, obtaining state number 2:

$$\frac{|0\rangle \otimes |\psi\rangle + |1\rangle \otimes U\,|\psi\rangle}{\sqrt{2}} \tag{2.10}$$

Finally, another Hadamard gate is applied to the ancilla qubit to obtain state number 3:

$$\begin{aligned}
&\frac{|0\rangle \otimes |\psi\rangle + |1\rangle \otimes U\,|\psi\rangle}{\sqrt{2}} \to \frac{(|0\rangle + |1\rangle) \otimes |\psi\rangle}{2} + \frac{(|0\rangle - |1\rangle) \otimes U\,|\psi\rangle}{2} = \\
&= \frac{1}{2}\left[|0\rangle \otimes (|\psi\rangle + U\,|\psi\rangle) + |1\rangle \otimes (|\psi\rangle - U\,|\psi\rangle)\right] = \\
&= \frac{1}{2}\left[|0\rangle \otimes (I + U)\,|\psi\rangle + |1\rangle \otimes (I - U)\,|\psi\rangle\right]
\end{aligned} \tag{2.11}$$

According to the third postulate of quantum mechanics ([24] p. 87), a projective measurement is performed by multiplying the state with the measurement operator and by its complex conjugate. So, given a measurement operator M_m and a state $|\psi\rangle$, the probability of obtaining m as a measurement outcome is

$$P(m) = \langle\psi|\, M_m^\dagger M_m\,|\psi\rangle \tag{2.12}$$

Usually, one measures in the computational basis, thus the operators for a one-qubit measurements are $M_0 = |0\rangle\langle 0|$ and $M_1 = |1\rangle\langle 1|$. So, when taking measurements of (2.11) with (2.12), the term multiplying $|1\rangle$ naturally drops out when measuring $P(0)$ and vice versa:

$$\begin{aligned}
P(0) &= \frac{1}{4}\langle\psi|\,(I + U^\dagger)(I + U)\,|\psi\rangle = \frac{1}{4}\langle\psi|\,(I + U + U^\dagger + U^\dagger U)\,|\psi\rangle = \\
&= \frac{1}{4}\langle\psi|\,(I + U + U^\dagger + I)\,|\psi\rangle = \frac{1}{4}\left[2\langle\psi|\psi\rangle + \langle\psi|\,U\,|\psi\rangle + \langle\psi|\,U^\dagger\,|\psi\rangle\right]
\end{aligned} \tag{2.13}$$

Because $\langle\psi|$ is a unit vector, $\langle\psi|\psi\rangle = 1$. Moreover, the conjugate transpose has the propriety $(AB)^\dagger = B^\dagger A^\dagger$ so that $(\langle\psi|\,U\,|\psi\rangle)^\dagger = |\psi\rangle^\dagger U^\dagger \langle\psi|^\dagger = \langle\psi|\,U^\dagger\,|\psi\rangle$. Since summing a number with its complex conjugate yields the real part, the final result is:

$$P(0) = \frac{1}{2}(1 + \mathrm{Re}(\langle\psi|\,U\,|\psi\rangle) \tag{2.14}$$

In a similar manner, $P(1) = \frac{1}{2}(1 - \mathrm{Re}(\langle\psi| U |\psi\rangle)$, which means

$$\langle\psi| U |\psi\rangle = P(0) - P(1) \tag{2.15}$$

To summarize, the expectation value can be obtained through indirect measurements using a simple quantum subroutine. However, this routine is easily understood from a theoretical point of view but it is not as easily implemented: this is due to the controlled U gate, which acts on n qubits and it is inherently difficult to build.

2.2.5 Classical Optimization

Because a variational quantum algorithm translates a linear system of equations into an optimization problem, classical optimization is a key step necessary for the success of a variational algorithm. There is a wide variety of optimization algorithm available: one relevant criterion to classify them is if they require knowledge about the derivative of the cost function, or if the optimization problem is constrained or unconstrained.

We define as 0-th order methods those which do not require information about the derivative of the cost (or objective) function. Examples are Nelder-Mead, Genetic Algorithms, Particle Swarm ([40] Sections 7.1, 7.2). The obvious advantage of these methods is that they can be used in problems where the objective function derivative is not known. In principle, one can compute the gradient of a cost function with finite differences, but the cost associated with it increases with the number of dimensions. In general, one could say that the advantage of these methods is that they can always be applied treating the cost function as a black box. As a consequence, these methods are extremely versatile and often very robust. On the other hand, it is generally impossible to guarantee that the optimization approached a minimum and a stopping criterion has to be set arbitrarily.

On the other hand, one example of a first-order method is a gradient descent algorithm, when one computes the gradient of the objective function and moves in that direction, hence the name steepest descent.

Classical Optimization for variational quantum algorithms

According to McClean, because the objective function is stochastic by nature, it is difficult to use gradient-based methods for its optimization [23]. An additional hurdle for gradient-based methods is the noise in the objective function that derives from its calculation. Thus, the author has benchmarked a few gradient-free methods for a computational chemistry problem: Nelder–Mead, TOMLAB/GLCLUSTER, TOMLAB/LGO, and TOMLAB/MULTIMIN [41]. For this test case, TOMOLAB library algorithms were shown to perform far better. However, the author remarks that these methods need further testing in larger design spaces, and a tailored algorithm for stochastic objective functions could be

beneficial.

Another brief review and performance comparison can be found in [35]. In this case, the authors compare Powell's algorithm, Constrained Optimization BY Linear Approximation (COBYLA), Bound Optimization BY Quadratic Approximation (BOBYQA), Nelder- Mead, Broyden-Fletcher-Goldfarb-Shanno (BFGS), and conjugate gradient (CG). This review is more interesting because, in this case, derivative-free methods are compared to gradient-based ones (BFGS, CG). Powell's method consists of performing a single variable optimization along a search direction per every variable. The direction of the 'search' is a linear combination of the search vectors from the previous steps. For example, consider the case with two design variables x_1, x_2. The first two iterations will move $\alpha_1 x_1$ and $\alpha_2 x_2$. Then, the third search direction will be defined by $\alpha_1 x_1 + \alpha_2 x_2$, and so on. [42]. COBYLA and BOBYQA are trust-region algorithms [43]. This means that the optimizer builds an approximation of the objective function (often linear or quadratic) which is deemed acceptable only within a certain region, called the trust region. BFGS is a quasi-Newton method, which consists of using previous function and gradient evaluation to estimate the Hessian matrix without the need to compute it ([40] section 6.3).

In the aforementioned study, La Rose et al. found that 0th order methods outperform the gradient-based ones. This was attributed to the noise in the computation. Moreover, Powell was found to outperform the other algorithms because it achieved a lower value of the cost function, whereas COBYLA was the one that required fewer function evaluations (13 times less than Powell's) and therefore shorter run time. However, the authors highlight that the effectiveness of a particular algorithm might be dependent on the ansatz or the type of problem at hand. Therefore, it is unknown whether these results can be generalized to other variational algorithms test cases.

The results above seem to be in contrast with [44], where the authors proved that for some problems gradient descent algorithms (where the gradient can be directly measured) significantly outperform 0th order methods. On the other hand, this theoretical study does not take noise into account which likely caused gradient-based methods to fail in [35]. In [28] and [29] gradient descent was used when solving linear equations with variational algorithms.

2.3 Application of Variational algorithms to Linear equations

To the best of the author's knowledge, there are only two instances in literature of variational algorithms applied to linear equations: Bravo-Prieto et al. [28] and Xu et al. [29]. In particular, the first was implemented on a quantum computer for a simple test case as well as a simulator, whereas the latter was tested on a simulator only using the Quantum Exact Simulation Toolkit (QuEST) package [45]. In both works, the underlying idea is to transform

a system of equation into a Hamiltonian whose expectation value is the optimization cost. In particular, solving a linear system consist in finding $\boldsymbol{x}$ such that $A\boldsymbol{x} = \boldsymbol{b}$. When implemented into a quantum computer, the above becomes $A|x\rangle = |b\rangle$, where $|x\rangle = \boldsymbol{x}/||\boldsymbol{x}||$. As it was explained above, the solution or 'guess' vector is parametrized according to a vector $\boldsymbol{\theta}$, such that $|x\rangle = V(\boldsymbol{\theta})|0\rangle$ (see section 2.2.2). While the error of the solution guess, which is the distance of $|x\rangle$ with respect to $|x_0\rangle$ cannot be directly measured, one can measure the projection of $A|x\rangle = |\psi\rangle$ on the subspace orthogonal to $|b\rangle$. If the projection is 0, then $A|x\rangle$ is linearly dependent on $|b\rangle$, which means $|x\rangle$ is the solution. Thus, the cost function is set up as

$$\widehat{C}(\boldsymbol{\theta}) = \mathrm{Tr}\big(A|x\rangle\langle x|A^\dagger(I - |b\rangle\langle b|)\big) = \mathrm{Tr}(|\psi\rangle\langle\psi|(I - |b\rangle\langle b|)) \tag{2.16}$$

moreover, because the trace is invariant under cyclic permutations $\mathrm{Tr}(ABC) = \mathrm{Tr}(BCA) \neq \mathrm{Tr}(ACB)$

$$\widehat{C}(\boldsymbol{\theta}) = \mathrm{Tr}\big(\langle x|A^\dagger(I - |b\rangle\langle b|)A|x\rangle\big) = \langle x|H|x\rangle \tag{2.17}$$

where $H = A^\dagger(I - |b\rangle\langle b|)A$. One further remark is that the cost could be low also when $A|x\rangle$ is small, which is not desirable since the objective is to find when $A|x\rangle$ is orthogonal to $(I - |b\rangle\langle b|)$. Thus, it is useful to normalize the cost function starting from (2.17) and $|\psi\rangle = A|x\rangle$:

$$C(\boldsymbol{\theta}) = \frac{\widehat{C}(\boldsymbol{\theta})}{\langle\psi|\psi\rangle} = \frac{\langle\psi|(I - |b\rangle\langle b|)|\psi\rangle}{\langle\psi|\psi\rangle} = \frac{\langle\psi|\psi\rangle - \langle\psi|b\rangle\langle b|\psi\rangle}{\langle\psi|\psi\rangle} = 1 - \frac{|\langle b|\psi\rangle|^2}{\langle\psi|\psi\rangle} \tag{2.18}$$

Variational Quantum Linear Solver

The Variational Quantum Linear Solver is a hybrid quantum algorithm developed by Bravo-Prieto et al. to use the techniques from variational quantum computing and use them to solve linear systems of equations. On a high level, the workflow of this algorithm is similar to any variational algorithm as explained in Section 2.2.1.

The authors tested the algorithm using both a hardware efficient ansatz and a Quantum Alternating Operator Ansatz (QAOA) [45]. QAOA evolves the initial superposition state $H^{\otimes n}|\mathbf{0}\rangle$ using two Hamiltonians which are applied alternatively. The two Hamiltonians are usually called driver (H_D) and mixer (H_M): the authors chose $H_D = A^\dagger(I - |b\rangle\langle b|)A$ and $H_M = \prod_{i=1}^n X_i$ were X_i is the Pauli X gate on the ith qubit. The time evolution is obtained by exponentiating the Hamiltonians which yields a unitary operator $U_M(\theta_j) = e^{-iH_M\theta_j}$. Thus, the state preparation sequence is obtained alternating the unitary operators:

$$V(\boldsymbol{\theta}) = e^{-iH_M\theta_{2p}}e^{-iH_D\theta_{2p-1}}\dots e^{-iH_M\theta_2}e^{-iH_D\theta_1} \tag{2.19}$$

to handle to problem of the vanishing gradient (see Section 2.2.3), a local cost function C_L can also be introduced, as

$$C_L = \frac{\langle x|H_L|x\rangle}{\langle\psi|\psi\rangle} \tag{2.20}$$

where

$$H_L = A^\dagger U \left(\mathbb{I} - \frac{1}{n} \sum_{j=1}^{n} |0_j\rangle \langle 0_j| \otimes \mathbb{I}_{\bar{j}} \right) U^\dagger A \tag{2.21}$$

and $|0_j\rangle$ is the 0th state on the jth qubit and $\mathbb{I}_{\bar{j}}$ is the identity on all the others qubits. U is the unitary that prepares the state $|b\rangle = U|0\rangle$. Because one can show that $C_L \leq C$, $C_L = 0 \leftrightarrow C = 0$, C_L is a valid cost function.

As shown in equation (2.18), to estimate the cost it is necessary to compute $\langle\psi|\psi\rangle$. Assuming A is Hermitian (it is always possible to transform it into an Hermitian by creating $\begin{bmatrix} 0 & A \\ A^\dagger & 0 \end{bmatrix}$), one can decompose it into the linear combination of unitary matrices $A = \sum_{l=1}^{L} c_l A_l$, where A_l themselves can be expressed using quantum gates. Thus,

$$\langle\psi|\psi\rangle = \langle \mathbf{0}| V^\dagger A^\dagger A V |\mathbf{0}\rangle = \sum_{l=1}^{N} \sum_{l'=1}^{N} c_l c_{l'}^* \langle \mathbf{0}| V^\dagger A_{l'}^\dagger A_l V |\mathbf{0}\rangle \tag{2.22}$$

which means that $L(L-1)/2$ terms need to be evaluated. Equation (2.22) can be evaluated with an Hadamard test. To successfully compute equation (2.18), one also needs:

$$|\langle b|\psi\rangle|^2 = |\langle \mathbf{0}| U^\dagger A V |\mathbf{0}\rangle|^2 = \sum_{l=1}^{L} \sum_{l'=1}^{L} c_l c_{l'}^* \langle \mathbf{0}| U^\dagger A_l V |\mathbf{0}\rangle \langle \mathbf{0}| V^\dagger A_{l'}^\dagger U |\mathbf{0}\rangle \tag{2.23}$$

for this, one can use the Hadamard test, or the Hadamard overlap test, that implies using double the amount of qubits but requires less connectivity because it is not necessary to apply a controlled V and U (see [28] for more details). In a similar way, the cost of the local cost function is estimated as:

$$C_L = \sum_{l=1}^{L} \sum_{l'=1}^{L} c_l c_{l'}^* \langle \mathbf{0}| V^\dagger A_{l'}^\dagger U (|0_j\rangle \langle 0_j| \otimes \mathbb{I}_{\bar{j}}) U^\dagger A_l V |\mathbf{0}\rangle \tag{2.24}$$

and can be measured through Hadamard or Hadamard overlap test. Because of the heuristics component of this algorithm, it is not possible to derive a theoretical runtime bound, but rather a heuristic one. The metric chosen by the author is the run per success, which are the iteration necessary to achieve the required tolerance. When using a Hardware efficient ansatz, the algorithm seems to be scaling linearly and sub-linearly in κ and ϵ. On the other hand with QAOA the scaling is sub-exponential in κ and logarithmic in $1/\epsilon$.

Chapter 3

Pre-processing and feasibility of VQLS

The Variational Quantum Linear solver (VQLS) is the quantum algorithms chosen for practical implementation attempts for two main reasons: the promise of a short term practical implementation and the possibility of obtaining a full solution of the linear system. The main hurdles of practical implementations of this algorithm are preparing the solution state efficiently using an ansatz, decomposing the linear system matrix into a linear combination of unitaries, and efficiently preparing the right-hand side vector $|b\rangle$.

In particular, this chapter focuses on the decomposition of the linear system matrix and contains a methodology explanation, decomposition results for different discretization schemes and a study of the shot noise. Section 3.1 introduces the concept of unitary decomposition and a general methodology to decompose a matrix given an set of quantum logic gates forming a basis. Then, Section 3.2 shows decomposition result for first order finite element discretization in different bases, listing number of terms and exact gate sequences, including generalization to higher dimension. Section 3.3 explains why higher order discretizations could be beneficial and are worth pursuing. Section 3.4 extends the analysis by looking at the decomposition in unitaries of a hybrid finite element discretization matrix. Finally Section 3.5 provides analytical and numerical estimation of the noise in cost function evaluation due to finite sampling of a quantum circuit.

3.1 Introduction to matrix decomposition

One way to apply quantum computing to differential equations is to look for discretization schemes whose resultant matrix A can be efficiently decomposed, and then use VQLS to tackle those linear systems. In general, all operations on a quantum computer must be unitary and any matrix A can be represented as a linear combination of unitary matrices. Therefore, to input a linear system in a quantum computer one just needs to find those unitaries and the combination of gates corresponding to them.

As a direct consequence, given n qubits, only linear systems of size $N = 2^n$ can be solved with this approach. Moreover, this implies overall runtime and efficiency of the VQLS is tightly related to the number of terms that make up the decomposition of the matrix A. In particular, as shown in (2.22) and (2.23), each equation requires at least $L(L-1)/2$ distinct quantum circuits evaluations, where L is the number of unitaries in $A = \sum_{l=1}^{L} c_l A_l$. Therefore, if L is exponential in n, it would most likely compromise the quantum advantage, since at every step of the minimization one has to run an exponential number of distinct circuits. (see Section 3.2 for more details)

Method

Decomposing an arbitrary matrix into a linear combination of unitaries is somewhat conceptually similar to decomposing a vector along a basis using Gram–Schmidt. In that case, one chooses a linearly independent set of vectors and projects a vector along each of these directions. Similarly, a set of unitaries that covers the entire space spanned by a matrix of dimension $N \times N$ is necessary to perform the decomposition. Naturally, a basis for this space must have N^2 unitaries, each corresponding to a "degree of freedom" of the matrix that has to be decomposed.

In principle, any set of orthogonal unitaries is suitable as a basis. However, a natural choice is to use Pauli matrices. Although these are not necessarily hardware-native gates, they are widely used and recognized. Moreover, these gates can be readily inputted in several providers hardware (either because a built-in decomposition exists or they are native), whereas hardware-native gates often depend on the hardware provider. For example, native gates for *ibmq_athens* are CNOT, R_z, $\sqrt{X}$, X [46].

For a 2×2 matrix, Pauli gates (X, Y, Z) and the identity I form a complete set. These matrices are unitary, and one can easily prove that they are orthogonal by verifying the products $\mathrm{Tr}\big(XY^\dagger\big) = \mathrm{Tr}(XY) = 0$. For all other $2^n \times 2^n$ matrices, all the possible combinations of tensor product of the four gates mentioned above form a basis so that any matrix A is decomposed as:

$$A = \sum_{j_1, j_2, \ldots, j_n} h_{j_1 j_2 \ldots j_n} \cdot \sigma_{j_1} \otimes \sigma_{j_2} \otimes \ldots \otimes \sigma_{j_n} \tag{3.1}$$

where

$$h_{j_1 j_2 \ldots j_n} = \frac{1}{2^n} \mathrm{Tr}\big((\sigma_{j_1} \otimes \sigma_{j_2} \otimes \ldots \otimes \sigma_{j_n}) A\big). \tag{3.2}$$

Because the initial set has four matrices, all possible combinations of tensor products will have $4^n = N^2$ orthogonal matrices, which is enough to form a basis.

It is worth mentioning that, although this decomposition method is helpful for case study implementations of a matrix, in practical applications, this decomposition quickly becomes

costly, so that it is probably more expensive than solving the linear system classically. The main reason is that a decomposition requires taking N^2 tensor products operations, matrix products, and trace calculation. Based on these considerations, most likely, any practical implementation of a variational linear solver will require a priori knowledge of the decomposition of a matrix in a set of unitaries.

3.2 First order finite element discretization matrix decomposition

This section is dedicated to decomposing a first-order finite element discretization matrix of a one dimensional Laplacian. This numerical scheme was chosen as a test case because of its simplicity and wide applicability. Firstly, the matrix is decomposed in the Pauli gates basis and a pattern that allows predetermining its decomposition without expensive calculations is explained. Secondly, a more efficient decomposition using multi-qubit gates is introduced. The Laplacian operator with Dirichlet boundary conditions is discretized as:

$$L = \begin{bmatrix} 2 & -1 & 0 & \dots & & \\ -1 & 2 & -1 & 0 & \dots & \\ 0 & -1 & 2 & -1 & & \\ \vdots & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ 0 & 0 & 0 & \dots & -1 & 2 \end{bmatrix} \tag{3.3}$$

3.2.1 Decomposition in Pauli Basis

For $N = 4$, it is trivial to compute $L = 2II - 1IX - 0.5XX - 0.5YY$, where I is the identity X, Y, Z are Pauli gates, and the tensor product sign between the matrices is omitted for a more concise notation. In a similar way, for $N = 8$, $L = 2III - 1IIX - 0.5IXX - 0.25XXX - 0.25YYX - 0.25YXY - 0.5IYY + 0.25XYY$. Figure 3.1 illustrates these results

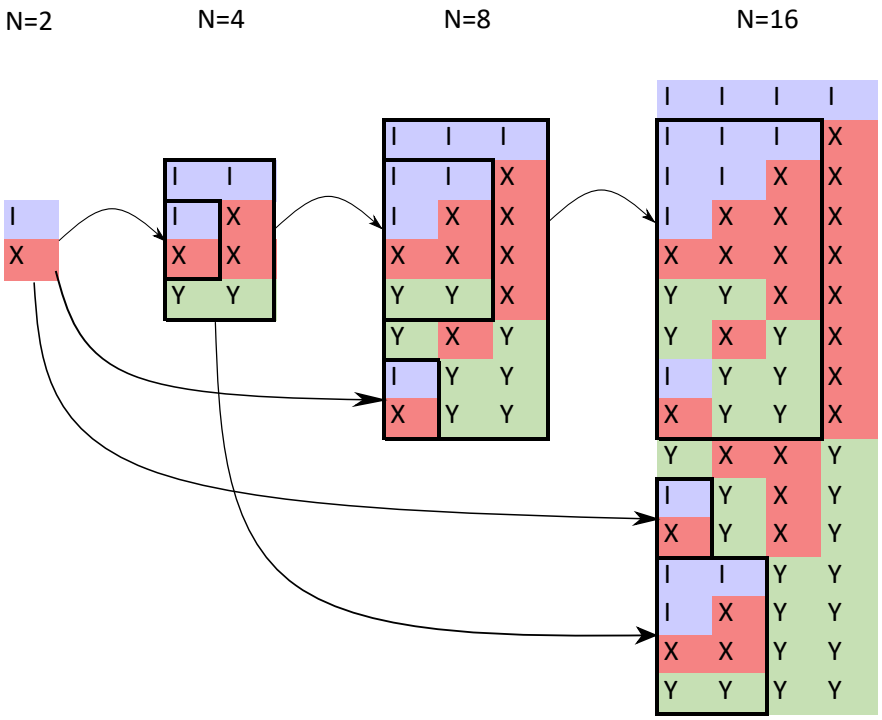


Figure 3.1: Illustration of the sequence of Pauli gates (without coefficients) that results in the operator of (3.3). For every N, each row is a tensor product of the gates listed and the overall matrix is obtained by linear combination of the rows unitaries. Every gate instruction can be created in a recursive manner starting from the previous.

It is noteworthy that in this basis **the number of unitaries forming the decomposition is exponential in the number of qubits**, and more precisely, it is exactly N. This was verified numerically up the computational power available ($N = 1024$), but because (3.3) is kept unchanged in its structure and just increased in size, it is reasonable to expect this result to hold for any size.

Moreover, as illustrated in Figure 3.1 it is possible to find a pattern: a sequence of Pauli is recursively assembled starting from $N = 2$. More precisely, every new power of 2 is formed by adding a row of identities, multiplying the previous power by the X matrix (to the right), then the 2^{n-2} sequence by $Y \otimes Y$, the 2^{n-3} by $Y \otimes X \otimes Y$, all the way to $Y \otimes X \otimes ... \otimes X \otimes Y$. So in the example above, $N = 16$ is formed multiplying $N = 8$ by X, $N = 4$ by $Y \otimes Y$, $N = 2$ by by $Y \otimes X \otimes Y$ and $N = 1$ (which degenerates from a matrix to just a coefficient) by $Y \otimes X \otimes X \otimes Y$. Naturally, also the coefficients multiplying the sequence of Pauli gates follow the same pattern so that the resultant decomposition formula is:

$$D_n = 2I + \frac{1}{2}(D_{n-1} - 4I_{n-1}) \otimes X - \sum_{k=0}^{n-2} \frac{1}{2^{k+2}} D_{n-2-k} \otimes Y \bigotimes^{k} X \otimes Y \tag{3.4}$$

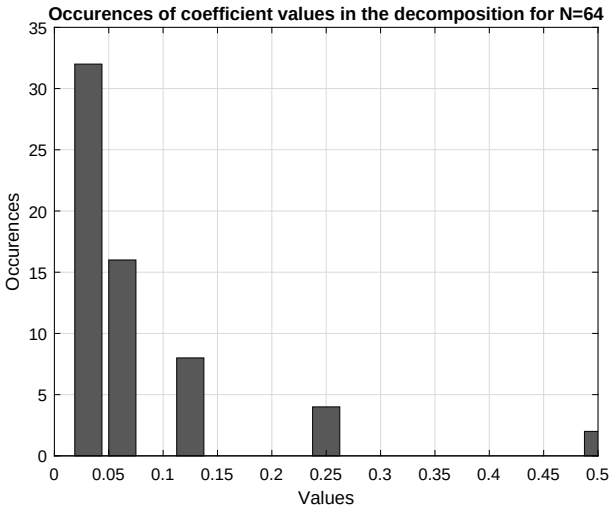
where $Y \bigotimes^{k} X \otimes Y$ means $Y \otimes X \otimes X ... \otimes X \otimes Y$ with X products repeated k times.

In principle, because the conjugate gradient method can solve sparse linear systems in $\mathcal{O}(Ns\kappa)$, using Paulis as a basis implies that VQLS cannot meaningfully outperform a classical algorithm in this setting. As mentioned above, every cost function evaluation in VQLS takes at least $2L(L-1) \approx \mathcal{O}(L^2)$, where L is the number of unitaries decomposing A. In this case, this would mean every function evaluation of the optimization procedure requires $\mathcal{O}(N^2)$ distinct circuit evaluations. Although a possible quantum advantage would depend on the speed of the iteration itself and the optimization procedure, it is clear that this exponential scaling of unitaries is a significant hurdle.

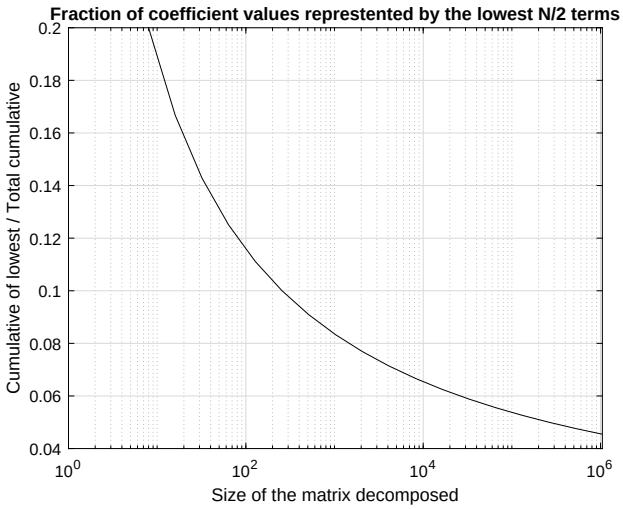
However, these results are problem-specific and only valid for this decomposition obtained using Pauli matrices and are not representative of the overall efficiency of VQLS. For example, using another basis might generate a more efficient decomposition, or other discretization schemes might yield more suitable matrices. Moreover, the discretization matrix changes for higher-dimensional problems and could be better suited for this application. Finally, it is worth investigating whether it is possible to neglect the smallest terms of this decomposition without significantly impacting the accuracy of the calculations.

Magnitude of coefficients and possibility of neglecting them

Figure 3.2a shows the occurrence of different coefficient values for the decomposition mentioned above. The most numerous bucked (with $N/2$ or 2^{n-1} coefficients) is the one corresponding to the lowest coefficient value: $1/2^{n-1}$. This implies that the smallest terms of the decomposition are also the most frequent and become increasingly smaller for larger matrices. For example for $N \approx 10^6$ the smallest term is $2^{-19} \approx 2 \cdot 10^{-6}$. In principle, one might think that this would make these terms negligible, effectively making the decomposition $N/2$ instead of N terms and thus more efficient. However, this is not necessarily the case: although each term might be negligible, to understand their weight, one should compute their cumulative value, i.e. the product of the value of a single term times its occurrence. Because every bucket of value $1/2^i$ has exactly 2^i terms, the result is that the sum of the absolute value of all the contributions of every bucket will always sum to 1, which means they all are equally important.



(a) List of occurrences of the absolute value of the coefficient for $n = 6$. The numbers 1 and 2 occur just one time and are not displayed for ease of illustration. In general, all the coefficients are $1/2^i$ where i are integers $i < n$. A coefficient of value $1/2^i$ occurs 2^i times.



(b) The cumulative values of the bottom $N/2$ coefficients of the decomposition (represented by the leftmost bar in figure 3.2a) are plotted as a fraction of the cumulative value of all the coefficients forming the decomposition.

Figure 3.2

In this framework, to measure for the weight of the smallest values with respect to the overall system it is necessary to compute the ration $\sum_{i=1}^{N_2} |c_i^{(min)}| / \sum_{i=1}^{N} |c_i|$, where $|c_i^{(min)}|$ are the smallest coefficients (in absolute value) of the set c_i. Figure 3.2b shows the aforementioned ration as a function of the system size N. Although the weight of the smallest coefficients decreases with the system size, it is still about 4% for extensive systems such as $N = 10^6$. Note that this is not the error one commits on the solution of the final system, rather just in the representation of the matrix.

Assuming the aforementioned error is acceptable, to evaluate what would happen if one neglects the bottom $N/2$ altogether it is interesting to look at the sparsisty plot of the resultant matrix, represented in Figure 3.3.

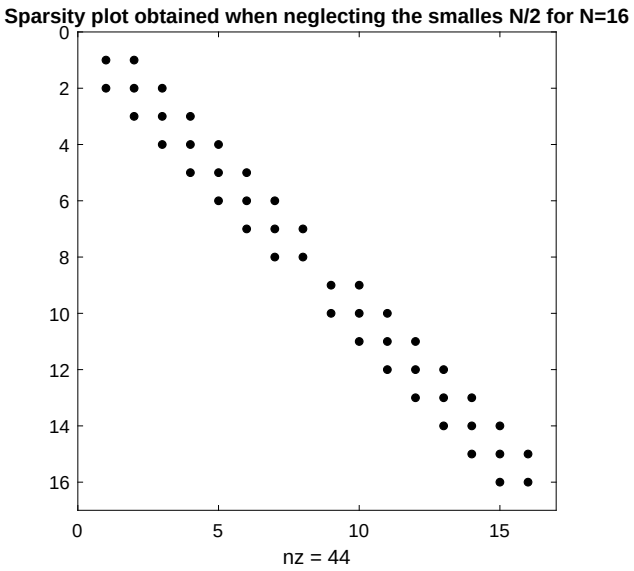


Figure 3.3: Sparsity plot of L if the smallest $N/2$ terms are neglected

As shown in Fig. 3.3 the $N/2$ smallest terms are responsible for implementing the entries $A_{N/2,N/2+1}$ and $A_{N/2+1,N/2}$ in the minor diagonal, and neglecting would result in two uncoupled systems.

To summarize, this analysis suggests a limited possibility of improvements in efficiency by neglecting the smallest term of the matrix decomposition.

3.2.2 Decomposition with high entanglement

As introduced above, Pauli matrices are among the most straightforward bases one can employ because they do not include controlled gates or multi-controlled gates. Such decompositions are better suited for near-term quantum computers because they pose less stringent connectivity and noise requirements. However, for the sake of completeness but also to evaluate possibilities on future hardware, it is interesting to look at more advanced decompositions of the tridiagonal matrix in (3.3). In [3], the author proposed a decomposition routine partially based on [47] for the Hamiltonian simulation of a tridiagonal matrix. The same method is adapted here for variational algorithms.

First, it is necessary to introduce the notion of a graph, which is a pair $G = (V, E)$ where $V = \{v_1, ..., v_n\}$ is a set of vertices and E are pair of vertices $E = \{(v_1, v_2), ..., (v_{n-1}, v_n)\}$, simply called edges. Given a graph with a set of vertices $V = \{v_1, ..., v_n\}$, an adjacent matrix A is a symmetric matrix whose entries are:

$$a_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

Essentially, A is non-zero only at the position i, j that have a corresponding (v_i, v_j) entry in the edges set. Naturally, every symmetric matrix of zeros and ones which is null along the main diagonal can be considered an adjacent matrix of a specific graph. In this case, the vertices of the graph are also the row or column indices equivalently. Alternatively, one can pick a matrix to decompose and find the graph associated with it.

By definition, every couple of matrix elements within the same row or column connects to the same vertex. To proceed with the decomposition, one can employ a graph colouring rule, which consists of labelling graph edges following a specific logic. In this case, we choose that all the edges sharing a vertex must have a different colour. As explained above, all matrix entries with the same colour or row will share a vertex, which means this rule allows to group all those entries that do not have the same column index. Now, every group of edges with the same colour forms a different matrix of the decomposition set.

The power of this rule is that it ensures that for every matrix of the decomposition, all the rows and columns that are not the zero vector will be linearly independent and with norm one because they are 1-sparse always at "different locations". To ensure full rank, we can add a 1 in the main diagonal in correspondence of the null rows. Thus, we obtain matrices with orthonormal rows, which implies they are unitary and form a proper gate decomposition.

When applied to (3.3), this procedure results in:

$$L = 2I_N - L_1 - L_2 = 2I_N - \begin{bmatrix} 0 & 1 & & & & \\ 1 & 0 & & & & \\ & & \ddots & & & \\ & & & \ddots & & \\ & & & & 0 & 1 \\ & & & & 1 & 0 \end{bmatrix} - \begin{bmatrix} 0 & & & & & \\ & 0 & 1 & & & \\ & 1 & 0 & & & \\ & & \ddots & \ddots & & \\ & & & 0 & 1 & \\ & & & 1 & 0 & \\ & & & & & 0 \end{bmatrix} \tag{3.6}$$

In this decomposition, L_1 is trivial to implement because it is block-diagonal. In particular, it is formed by the repetition of the X Pauli matrix along the main diagonal:

$$L_1 = I_{2^{n-1}} \otimes X \tag{3.7}$$

which means the circuit that implements it is just an Pauli X on the first qubit (rightmost in the tensor product). On the other hand, L_2 cannot be directly implemented because it is not unitary: this is easily verified since one propriety of a unitary matrix is that all rows are linearly independent, and in this case, the first and last row are null. Thus, the suggested modification is to add a 1 in positions $(1, 1)$ and (N, N). For this point, in the text, L_2 will refer to the modified version just mentioned. For a 4×4 matrix, its implementation is easy

and corresponds to the swap matrix

$$\text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

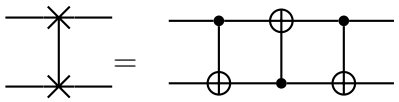


Figure 3.4: Implementation of L_2 for $N = 4$

However, for higher dimensions more complex routines are necessary. In general, the procedure of flipping the central element as in (3.4) for an arbitrary matrix size 2^i is equivalent of devising a gate that flips the binary registers:

$$011...1 \rightarrow 100...0 \tag{3.9}$$

(where the leftmost bit is the most significant one). To do so, one needs to flip all the qubits if the most significant is 0, which corresponds to the chain of nCNOT in Figure 3.5, so that $01...1 \rightarrow 00...0$. Then, the most significant bit is flipped if and only if all the others are 0, which corresponds to the multi controlled gate in the middle of Figure 3.5, and results in $00...0 \rightarrow 10...0$. In this case, the following chain of gates does not affect the statevector because the control is now 1.

For any other state that is not $011...1, 100...0$, it is trivial to prove this gate accounts to identity: if the first bit is 0, but the others are not all 1, the multi controlled nCNOT will not be applied, and the two sets of nCNOT will reduce to identity. If the first qubit is 1 and the others are not all 0, no gate is active.

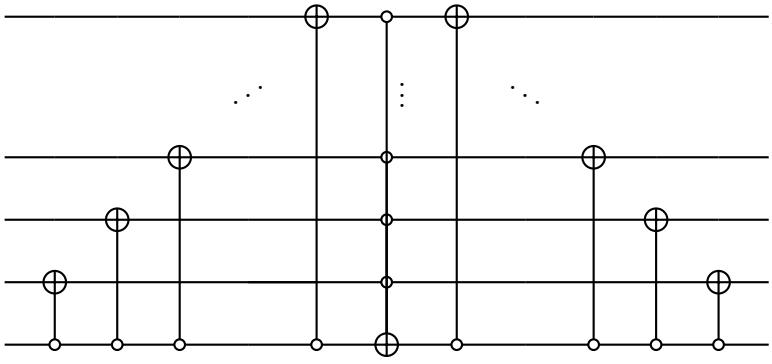


Figure 3.5: Detailed circuit representation of C_{n-1}

To implement all the other diagonal terms, Vazquez [3], who devised a similar unitary, suggests arranging this gate along the diagonal as illustrated in Figure 3.6.

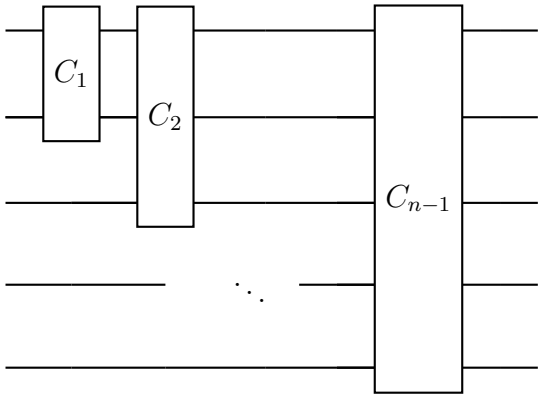


Figure 3.6: Circuit implementation of L_2 [3]

In this structure, every block flips some elements along the diagonal. In particular, C_1 and $n-2$ adjacent empty wires is equivalent to $I_{n-2} \otimes \text{SWAP}$, which means SWAP is identically repeated along the diagonal and effectively 'flips' every other pair. The functioning of the other block is similar: Every C_i 'flips' progressively less pairs until C_{n-1} that is responsible for the elements in the minor diagonal $(2^{n-1}, 2^{n-1}+1)$ and $(2^{n-1}+1, 2^{n-1})$ only. Note that these are the same entries requiring the sum of 2^{n-1} distinct unitaries when using Pauli as a base.

For example, for $n = 3$, applying C_1 and C_2 separately yields

$$C_1 = \begin{bmatrix} 1 & & & & & & & \\ & 0 & 1 & & & & & \\ & 1 & 0 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 0 & 1 & \\ & & & & & 1 & 0 & \\ & & & & & & & 1 \end{bmatrix} \quad \text{and} \quad C_2 = \begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 0 & 1 & & & \\ & & & 1 & 0 & & & \\ & & & & & 1 & & \\ & & & & & & 1 & \\ & & & & & & & 1 \end{bmatrix} \tag{3.10}$$

and their joint application is L_2. Note that, as was already mentioned, L_2 has one unwanted non-zero term at positions $(1,1)$ and (N,N), which is necessary to guarantee it is a valid unitary (full rank). However, this implies (3.6) would not be satifyied unless an additional

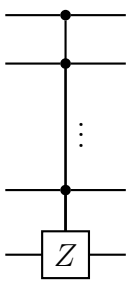
term is added to correct it. This is done via the unitaries

$$L_3 = \begin{bmatrix} 1 & 0 & & & \\ 0 & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & -1 \end{bmatrix}, \qquad L_4 = \begin{bmatrix} -1 & 0 & & & \\ 0 & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} \tag{3.11}$$

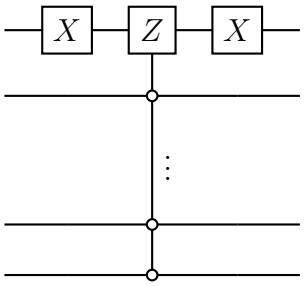
so that

$$I_N - \frac{1}{2}(L_3 + L_4) = \begin{bmatrix} 1 & 0 & & & \\ 0 & 0 & & & \\ & & \ddots & & \\ & & & 0 & \\ & & & & 1 \end{bmatrix} \tag{3.12}$$

The circuits that implement L_3 and L_4 are relatively straightforward and correspond to:



(a) Circuit implementation of L_3. It requires a Z on the most relevant qubit controlled on all other qubits ($n-1$ times).



(b) Circuit implementation of L_4. It requires a Z on the least relevant qubit controlled on all other qubits ($n-1$ times)

Figure 3.7

Moreover, these gates are a powerful and straightforward way of changing the boundary conditions as they can arbitrarily change the $(1,1)$ and (N,N) elements.

Therefore, the resultant equation for (3.3) is:

$$L = 3I - L_1 - L_2 - \frac{1}{2}(L_3 + L_4) \tag{3.13}$$

which implies that it is possible to express (3.3) efficiently as a linear combination of only 5 unitaries for any n. It directly follows that VQLS could be efficient when solving for a one-dimensional Poisson equation, in line with Bravo-Prieto et al. claim [28] (based on [47]) that every q-sparse (at most q non-null row entries) Hamiltonian has an efficient implementation.

However, it is important to mention that, although this decomposition would greatly increase the efficiency of VQLS, it goes against its raison d'etre because it requires interconnectivity between qubits and relatively deep circuits. In particular, every C_i requires $2i+1$ gates, so in total L_2 requires

$$\sum_{i=1}^{n-1} 2i+1 = n^2 - 1$$

which is still polynomial in the number of qubits. On the other hand, a system of 10 qubits would already have a gate count of 99 for this unitary, which could be a problem because of noise in real hardware. Moreover, this gate count neglects that in real hardware, multi-controlled gates need to be decomposed, which increases the gate count even further. In particular, a decomposition for $C^n\text{NOT}$ requires $(12n-10)\text{CNOT} + (20n-16)\{\text{one qubit gate}\}$ and $(n-1)\{\text{ancilla qubits}\}$ [3, 24], which adds a significant overhead both in terms of gates count but also connectivity requirements.

Following the logic of variational approaches, one could decompose the circuit of Figure 3.6 in a separate block whose linear combination still yields L_2. This would require more terms (hence a less efficient algorithm) but could make the implementation easier because the decomposed gate sequence will at most be $2n-1$ long instead of n^2-1, a considerable improvement. The sum of those unitaries yields:

$$\sum_{i=1}^{n-1} C_i = \begin{bmatrix} (n-1) & & & & & & & \\ & (n-2) & 1 & & & & & \\ & 1 & (n-2) & & & & & \\ & & & \ddots & & & & \\ & & & & (n-2) & 1 & & \\ & & & & 1 & (n-2) & & \\ & & & & & & & (n-1) \end{bmatrix}. \tag{3.14}$$

This is because, as explained above, every matrix is responsible for 'flipping a different pair' (3.10), which means that every unitary will have a 1 along the main diagonal at all other 'non-flipped' locations. Since there are $n-1$ unitaries, this results in a $n-2$ along the diagonal and a $n-1$ at $(1,1)$, $(2^n, 2^n)$ where all unitaries are equal to 1. Then

$$L_2 = \sum_{i=1}^{n-1} C_i - (n-2)I \tag{3.15}$$

and renaming $C_i = L_{2,i}$ yields a new decomposition of L

$$L = (n+1)I - L_1 - \sum_{i=1}^{n-1} L_{2,i} - \frac{1}{2}(L_3 + L_4) \tag{3.16}$$

where the decomposition accounts to a total of $n+3$ terms, which is still polynomial in n, hence efficient.

In conclusion, it is possible to efficiently decompose L in only $n+3$ distinct terms, however, this requires a deep circuit, up to a total $(14n-12)\text{CNOT}+24(n-1)\{\text{one qubit gate}\}$ for L_i. Moreover, a complete implementation of VQLS requires two L_i in the same circuit, the ansatz and $|b\rangle$, which sum up to an even deeper circuit. In addition to depth requirements, this implementation also requires n ancillas and high qubit interconnectivity. Therefore, it is deemed unlikely that such implementation will be possible in the short term.

As a reference, Figure 3.8 compares the circuit depth for VQLS using the aforementioned unitaries and HHL. Although gate count for VQLS seems to scale exponentially, this test case was run using Qiskit built-in transpiler, which probably does not make use of the routine in [24] (Section 4.3) to efficiently transpile the C^nNOT gates. Nonetheless, the obtained circuit is still approximately four times shallower than HHL for the same test case. Because compared to HHL a variational algorithm requires many iterations to convergence, such a depth is hardly justifiable which suggests the use of ancillary qubits in the decomposition will be necessary.

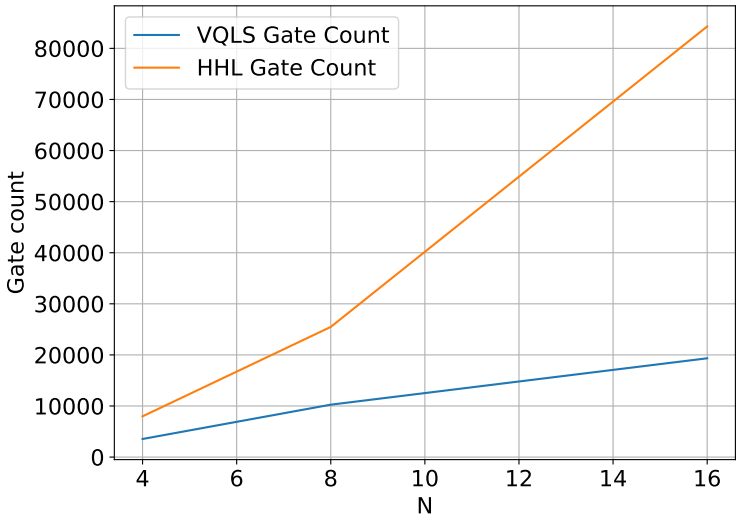


Figure 3.8: Depth requirement comparison of HHL [4] (section 4.1.1) and VQLS as implemented by the Qiskit compiler. VQLS is measured in the worst-case scenario (i.e. with the deepest of the unitaries applied). Circuits were transpiled based on *Ibmq_16_Melbourne* layout, qiskit version 0.26.2, and default transpiling optimization level 1 (light optimization)

3.2.3 Generalization to higher dimensions

An interesting propriety of a higher dimension discrete Laplacian operator with Dirichlet boundary conditions is that it can be obtained starting from a one-dimensional discretization

matrix and using what is commonly known as Kroner sum. In particular, a two dimensional Laplacian with Dirichlet Boundary conditions can be computed as:

$$L^{(2)} = D_{xx} \oplus D_{yy} = D_{xx} \otimes I + I \otimes D_{yy} \tag{3.17}$$

similarly,

$$L^{(3)} = D_{xx} \oplus D_{yy} \oplus D_{zz} = D_{xx} \otimes I \otimes I + I \otimes D_{yy} \otimes I + I \otimes I \otimes D_{zz} \tag{3.18}$$

where D_{xx}, D_{yy}, D_{zz} stands for the one-dimensional discrete Laplacian along the x, y, z axes whereas I is identity matrix of appropriate size (e.g. same size of $D_x x$ when multiplied to $D_y y$ and vice versa). This result is interesting because it implies that the efficiency of unitary decompositions increases with the increase in the number of dimensions. As shown before, a one-dimensional matrix requires N terms for its decomposition in a Pauli Base, whereas according to equations (3.17, 3.18) a 2D and 3D Laplacians will have respectively $2N$ and $3N$ terms with matrices size of N^2 and N^3. In other words, if by N we refer to the size of the resultant matrices, than $L^{(2)}$ requires $2\sqrt{N}$ unitaries, whereas $L^{(3)}$ requires $3\sqrt[3]{N}$. This decomposition is valid for an arbitrary number of dimensions, and it becomes progressively more efficient when increasing the dimension numbers. Possibly, VQLS could find applications for problems where a Laplacian is discretized in a large number of dimensions.

Because with VQLS each cost of one function evaluation using Pauli basis requires $\mathcal{O}(L^2)$ circuits, a two dimensional case will require a total of $\mathcal{O}(N)$ circuits, whereas a $L^{(3)}$ will require $\mathcal{O}\left(\sqrt[3]{N^2}\right)$. Similarly a k-dimensional problem requires $\mathcal{O}\left(\sqrt[k]{N^2}\right)$, which is sub-exponential and potentially advantageous. Finally, it is necessary to remark that solving a system with VQLS requires this amount of function evaluation per *every step* of the optimization problem, which will add a significant overhead to the total cost.

In conclusion, although solving for a higher number of dimensions results in more efficient decompositions, using Pauli as a base, the number of terms is sub-exponential and not polynomial in n. On the other hand, this generalization is also valid for higher entanglement basis to add a helpful generalization feature to an already efficient decomposition. Thus, it is still an open question whether this can provide any advantage, which would depend on the efficiency of the ansatz and the ease of optimization.

3.3 Generalization to higher order finite element schemes

Provided the problem is sufficiently smooth, finite element methods allow for higher-order basis functions than the linear discussed in the previous chapter. Roughly speaking, higher-order methods generate less sparse discretization matrices, and therefore harder to solve efficiently. However, this is not necessarily true for the variational quantum linear solver. Section 4.2 shows that as problem dimensionality rises (larger N) so does the difficulty in

obtaining a meaningful solution. Therefore, it interesting to investigate decomposition for higher-order basis functions and evaluate whether those can lead to better results.

Assuming an exponentially deep ansatz is necessary to obtain a precise solution, the cost of a variational solver will be $\propto kN^m \cdot hL^2$, where k, m, h are problem dependent constant and L is the number of terms in the decomposition of A. Therefore, based on these coefficients, there could be a tangible advantage in higher-order methods, which (with comparable precision) could potentially decrease N while increasing L (fuller matrices generally require more terms to be decomposed).

3.4 Hybrid finite element method

While the previous section is dedicated to understanding the most efficient decomposition technique given a specific discretization matrix, the focus is shifted towards the matrix in this section. In particular, the hybrid finite element discretization matrix is analyzed similarly to the first-order matrix from the previous chapter to understand if this numerical technique is more suitable for a quantum computing implementation.

Hybrid formulations allow dividing the domain into several interlinked smaller systems of equations so that one can solve many local systems instead of a large one [48]. In particular, these subdomains connect by local variables that ensures continuity, often called Lagrange multipliers. Thus, if one solves for the Lagrange multipliers, the result is a set of independent linear systems where the Lagrange multiplier acts as a boundary condition for the local system. In the framework of spectral element methods, the proprieties of a local system are determined by the polynomial degree chosen to approximate that element of the domain.

In the simplest case, one can solve Poisson's equations on a one-dimensional domain divided in K non-overlapping uniform elements of size $h = \frac{b-a}{K}$. Using linear basis functions results in the following matrices for one local domain:

$$\mathbb{M} = \frac{h}{6}\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \quad E = \begin{bmatrix} -1 & 1 \end{bmatrix} \tag{3.19}$$

The resultant discretization of a local domain is therefore obtained as :

$$\mathbb{A}_k = \begin{bmatrix} M & E^T \\ E & 0 \end{bmatrix} = \begin{bmatrix} h/3 & h/6 & -1 \\ h/6 & h/3 & 1 \\ -1 & 1 & 0 \end{bmatrix} \tag{3.20}$$

Then, one assembles these K local systems in a block diagonal matrix to obtain:

$$\mathbb{A} = \begin{bmatrix} \mathbb{M}_1 & E^T & & & & & & \\ E & 0 & & & & & & \\ & & \mathbb{M}_2 & E^T & & & & \\ & & E & 0 & & & & \\ & & & & \ddots & \ddots & & \\ & & & & \ddots & \ddots & & \\ & & & & & & \mathbb{M}_K & E^T \\ & & & & & & E & 0 \end{bmatrix} \tag{3.21}$$

Because the matrix $\mathbb{A}$ is block-diagonal, the K local systems are not connected by any equations, and if one solves this system, it is not possible to guarantee a continuous solution, but only disjoint local solutions. Thus, we introduce the connectivity matrix, which substantially contains the equation connecting the elements. In this example, one needs to ensure the right velocity component of one sub-domain is equal to the leftmost of the adjacent one. In other words, if the unknowns of an element are u_1, u_2, and p, then $u_{2,K-1} - u_{1,K} = 0$. For example, for $K = 2$ two points are connecting the three elements, which results in

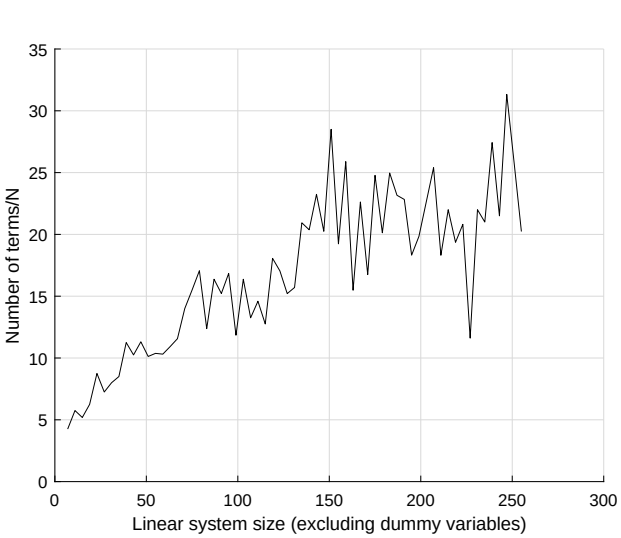
$$\mathbb{E}_{\mathbb{N}} = \begin{bmatrix} 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \end{bmatrix} \tag{3.22}$$

Thus, the final system is assembled as:

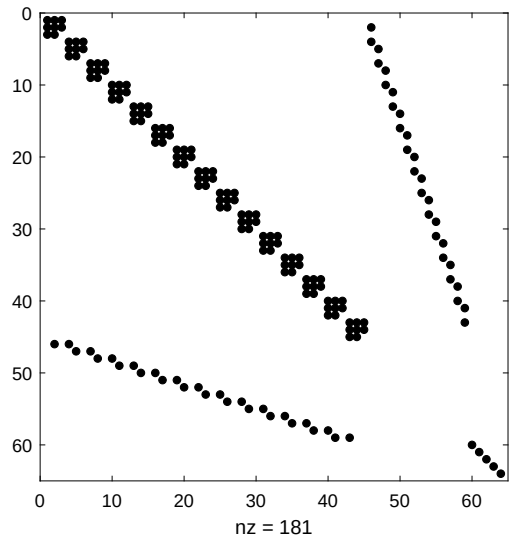
$$\begin{bmatrix} \mathbb{A} & {\mathbb{E}_{\mathbb{N}}}^T \\ \mathbb{E}_{\mathbb{N}} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{F} \\ \mathbf{0} \end{bmatrix} \tag{3.23}$$

where $\mathbf{X}$ is the unknowns vector, $\boldsymbol{\lambda}$ contains the variables connecting the elements (Lagrange multipliers), and $\mathbf{F}$ is just the right-hand side of the system with the known terms.

Naturally, the first question is whether this matrix has an efficient decomposition in the Pauli basis. When using a first-order discretization, each block has dimension 3, which means the resultant matrix is always $3K + K - 1 = 4k - 1$ in size, which by definition is never a multiple of 2. On the other hand, only matrices of size $N = 2^n$ can be decomposed with the mentioned method. Therefore, to match the necessary matrix size, the proposed solution was to add rows of dummy variables where the matrix has ones along the identity to reach the closest power of 2 and 0 on the right-hand side.



(a) Number of terms of the decomposition divided by matrix size N generated by this discretization in the Pauli basis.



(b) Sparsity plot of the resultant matrix of this discretization scheme.

Figure 3.9: Number of decomposition terms and sparsity plot for hybrid finite element method on a one dimensional domain and linear basis functions

This initial assessment suggests that when decomposing this discretization matrix in the Pauli basis, the result has an exponential number of terms, hence inefficient. Moreover, as shown in Figure 3.9a, the number of terms is not a fixed multiple of the system size, nor seems to follow a discernible pattern, which implies the a-priori determination of its decomposition in Pauli is challenging to obtain. Possibly, this oscillating behaviour can be explained by the fact that the matrix is not self-identical: as system size grows, one has to add a different amount of dummy variables to match the closest power of 2.

Interestingly, in the global system above one can solve directly for the vector of the Lagrange multipliers as

$$\boldsymbol{\lambda} = (\mathbb{E}_{\mathbb{N}}\mathbb{A}^{-1}{\mathbb{E}_{\mathbb{N}}}^{T})^{-1}(\mathbb{E}_{\mathbb{N}}\mathbb{A}^{-1}\mathbf{F}). \tag{3.24}$$

The advantage of this decomposition is that it allows to parallelize and solve the problem efficiently. Firstly, $\mathbb{A}$ is block-diagonal, which means it can be easily inverted, and secondly, once (3.24) is solved, one only needs to solve smaller and decoupled local systems to get the final solution. Once again, it is interesting to look at the shape and possible decompositions of $\mathbb{E}_{\mathbb{N}}\mathbb{A}^{-1}{\mathbb{E}_{\mathbb{N}}}^{T}$ to investigate possible speed-ups using quantum algorithms. For this specific case and discretization, the result of this product is precisely the matrix already the same of (3.3), which is a second-order finite difference discretization of Poisson.

In conclusion, in these settings hybrid finite element method underperforms first order finite element method when decomposed in the Pauli basis, with respectively $\propto N^2$ and $\propto N$ unitaries. Given the exponential number of terms in this decomposition, this numerical

scheme does not seem to be compatible with VQLS and it is unlikely it could lead to any advantage with respect to classical methods. Future works could extend the validity of this analysis to different unitary bases and higher order basis functions to asses the performance of this method more comprehensively.

3.5 Accuracy on the function evaluation

One of the building blocks of variational algorithms is the classical optimization procedure in between successive cost function evaluations of the quantum circuit. Optimizing entails several challenges, for example, the 'curse of dimensionality', which means optimizing gets progressively more troublesome in higher dimensions and the noise in the objective function evaluation.

In particular, the latter has two primary sources: errors stemming from the execution on physical hardware and errors due to finite sampling of circuit output. Imperfections in logical operations on current hardware cause the first, whereas the second derives from the fact that the result of a quantum circuit is always probabilistic, and its accuracy depends on how many times one samples the circuit (number of shots). Note that this has nothing to do with hardware, and it is an intrinsic propriety of the functioning of quantum algorithms. Thus, one could ask what is an appropriate number of runs for a quantum circuit, assuming no hardware-related noise.

For simplicity and without loss of generality, assume that we want to measure a single qubit. After N_r runs, the raw output is just a sequence of 0s and 1s, and their average is what one is generally seeking. Clearly, it is just s/N_r where s denotes the amount of 1 obtained, whereas f will denote the total number of 0s. However, one might be interested in other properties, such as the degree of confidence in this solution. In other words, the probability density function given a set of data (s, f), which is a perfect application of Bayes theorem [49]:

$$P(p \mid (s,f)) = \frac{P((s,f) \mid p)P(p)}{P(s,f)}. \tag{3.25}$$

Here $P(p \mid (s,f))$ denotes the conditional probability, or the probability of p given that (s,f) happened. $P(s,f)$ is trivially obtained as the probability of measuring 1 s times (which is p^s), multiplied the probability of measuring 0 f times for all possible combinations.

$$P((s,f) \mid p) = \binom{s+f}{s} p^s (1-p)^f. \tag{3.26}$$

Because there is no preferential value of p (i.e 0 and 1 are equally probable without considering the dataset) we assume a constant priori distribution which yields $P(p) = 1$. On the other hand, $P(s,f)$ is just the probability of obtaining that data set, regardless of a specific probability. This is computed as the integral of all possible probabilities times their

conditioned probabilities

$$\int_0^1 P((s,f) \mid t)P(t)dt = \int_0^1 P((s,f) \mid t)dt \tag{3.27}$$

where the latter simplification is because above we assumed $P(p) = 1 \ \forall \ p$. Plugging in (3.25), (3.26) yields

$$\int_0^1 P((s,f) \mid t)dt = \int_0^1 \binom{s+f}{s} t^s(1-t)^f dt \tag{3.28}$$

and simplifying the binomial:

$$P(p \mid (s,f)) = \frac{p^s(1-p)^f}{\int_0^1 t^s(1-t)^f dt}. \tag{3.29}$$

Note that the above is simply the beta distribution with $(\alpha, \beta) = (s+1, f+1)$. Thus, one can express its variance as a function of sample average and size [50]

$$\mathrm{Var}(X) = \frac{\mu(1-\mu)}{1+N_r}. \tag{3.30}$$

Variance is maximized when $\mu = 1/2$ and thus this can be considered the worst case scenario. Then, the standard deviation for a large number of shots and in worst case scenario is

$$\sigma = \frac{1}{4\sqrt{1+N_r}} \approx \frac{1}{4\sqrt{N_r}}. \tag{3.31}$$

This result indicates that when running a quantum circuit the standard deviation of the probability distribution that encodes the solution varies with the inverse square root of the number of shots. For example, if one wants to evaluate the cost function with a precision ϵ, a fair assumption would be that 2σ on the probability distribution is smaller than the requested precision, yielding

$$N_r \geq \frac{1}{2\epsilon^2}. \tag{3.32}$$

Note that a less arbitrary threshold can be set by computing the cumulative β distribution and solving numerically to find the exact number of N_r one needs for a specific confidence level.

When solving a linear system with a variational quantum algorithm, the cost function measures the orthogonality between $A\boldsymbol{x}$ and $\mathbf{b}$ (see (2.16)). Similarly, classical solvers of linear equations such as the conjugate gradient method solve for a residual $||A\boldsymbol{x} - \boldsymbol{b}||$ that can get close to machine precision. In these cases, the error on the solution vector depends on the eigenvalues of A, which are generally unknown. This analysis suggests that a variational quantum algorithm can never achieve this level of precision: for instance, if one wants to

solve for a cost function $C(\boldsymbol{\theta}) = 10^{-10}$ (which is larger than typical machine precision 10^{-16}), this would require $\mathcal{O}(10^{20})$ circuit runs per function evaluations. Thus, quantum variational solvers are limited to applications where a lower precision on the residual estimation is allowed.

Similarly, (3.30) can be applied to a Hadamard test, one of the most commonly used routines in variational algorithms to measure the expectation value of a unitary and fundamental to the VQLS. In this case, one is generally interested in $P(0) - P(1) = 1 - 2P(1)$ or using the notation above $1 - 2s$. For a random variable X and a constant c, $\mathrm{Var}(cX) = c^2\mathrm{Var}(X)$ which implies that the variance associated with a Hadamard test is in fact four times that of a single qubit measurement $P(1)$ as in (3.30):

$$\mathrm{Var}(X) = \frac{4\mu(1-\mu)}{1+N_r}. \tag{3.33}$$

Clearly, in the most general case μ is not known before running the experiment, however, it is possible to estimate the variance a priori computing the worst case scenario $\mu = \frac{1}{2} \rightarrow \mathrm{Var}(X) = \frac{1}{1+N_r}$, a typical instance $\mu_{avg} = \int_0^1 \mu(1-\mu)d\mu = 1/6 \rightarrow \mathrm{Var}(X) = \frac{5}{9(1+N_r)}$.

These results have been numerically validated with Qiskit *QuasmSimluator*[1] by running a simple circuit that computes the inner product between two vectors. This is done by assembling two unitaries U and $\tilde{U}$ such that $U|0\rangle = |\psi\rangle$ and $\tilde{U}|0\rangle = |\tilde{\psi}\rangle$, and running a Hadamard test as in Figure 3.10, whose result is $\langle\tilde{\psi}|\psi\rangle$.

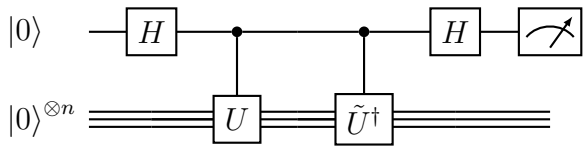


Figure 3.10: Circuit implementation of the Hadamard test that computes $\langle 0|\tilde{U}^\dagger U|0\rangle$

Figure 3.11 shows good agreement between numerical data and analytical predictions.

[1]Statevector method:"A dense statevector simulation that can sample measurement outcomes from ideal circuits with all measurements at end of the circuit." `https://qiskit.org/documentation/stubs/qiskit.providers.aer.QasmSimulator.html`

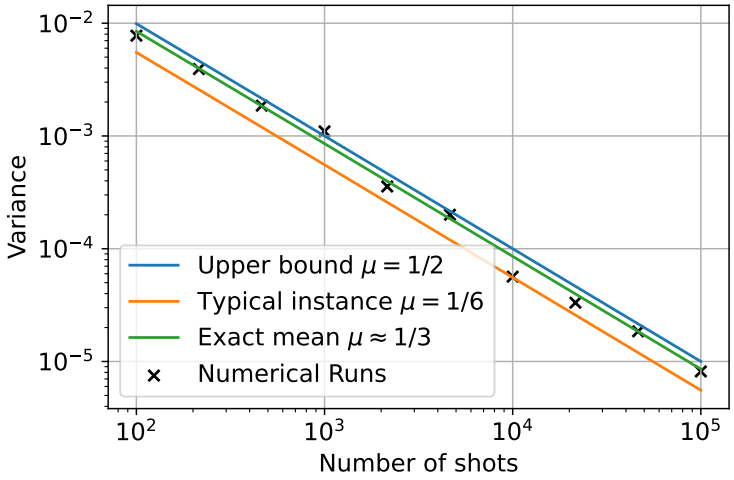


Figure 3.11: Numerical runs are obtained measuring the variance of a sample of 100 runs per each different number of shots. Upper bound, typical instance and "Exact mean" are obtained plugging different values in (3.33), respectively $\mu = 1/2$, $\mu = 1/6$ and $\mu \approx 1/3$ which is the actual average value for this specific run. The slope of the logarithmic fit is -1, as suggested by (3.33).

3.5.1 Shot noise in the VQLS algorithm

A similar discussion can be done concerning the VQLS algorithm. However, when considering a normalized cost function, its value is computed as the division of two different factors:

$$C(\boldsymbol{\theta}) = 1 - \frac{|\langle b|\psi\rangle|^2}{\langle\psi|\psi\rangle}, \tag{3.34}$$

where both the numerator and the denominator are the results of the sum of independent measurements. Since the variance of the sum of independent distribution is the sum of the single variances, it is reasonable to expect a decrease in precision proportional to the number of terms L in the decomposition of A. Thus, this section aims to understand the relation between noise and L in VQLS and its implications for the overall algorithm.

Variance of $\langle\psi|\psi\rangle$

This paragraph is dedicated to understanding the variance of $\langle\psi|\psi\rangle$ and providing an upper bound for it, based on the results for a single noiseless measurement as in (3.30). $\langle\psi|\psi\rangle$ is computed as the sum of $L(L-1)/2$ independent terms, where L is the number of unitaries in the decomposition of the linear system matrix A. To compute it we recall

equation (2.22):

$$\langle\psi|\psi\rangle = \langle\mathbf{0}|\, V^\dagger A^\dagger A V\, |\mathbf{0}\rangle = \sum_{l=1}^{L}\sum_{l'=1}^{L} c_l c_{l'}^*\, \langle\mathbf{0}|\, V^\dagger A_{l'}^\dagger A_l V\, |\mathbf{0}\rangle\,.$$

Obviously, $\langle\mathbf{0}|\, V^\dagger A_{l'}^\dagger A_l V\, |\mathbf{0}\rangle = \langle\mathbf{0}|\, V^\dagger A_l^\dagger A_{l'} V\, |\mathbf{0}\rangle$ and $\langle\mathbf{0}|\, V^\dagger A_l^\dagger A_l V\, |\mathbf{0}\rangle = 1$, which is why only $L(L-1)/2$ quantum circuit are necessary. Thus, (2.22) is simplified as

$$\langle\psi|\psi\rangle = \sum_{l=1}^{L}\sum_{l'=l+1}^{L} 2c_l c_{l'}^*\, \langle\mathbf{0}|\, V^\dagger A_{l'}^\dagger A_l V\, |\mathbf{0}\rangle\,. \tag{3.35}$$

Two basic proprieties of the variance are that for a constant c, and a random variable X, $\mathrm{Var}(cX) = c^2\mathrm{Var}(X)$ and that $\mathrm{Var}(X+Y) = \mathrm{Var}(X) + \mathrm{Var}(Y)$ if the two variables are uncorrelated. Using these proprieties, as well as (3.33), resulting variance is:

$$\mathrm{Var}(\langle\psi|\psi\rangle) = \sum_{l=1}^{L}\sum_{l'=l+1}^{L} 4c_l^2 c_{l'}^{*\,2}\mathrm{Var}(X_{ll'}) = \sum_{l=1}^{L}\sum_{l'=l+1}^{L} 16c_l^2 c_{l'}^{*\,2}\frac{\mu_{ll'}(1-\mu_{ll'})}{(1+N_r)}. \tag{3.36}$$

Clearly, there the variance of the sum will depend on the output of a single experiment $\mu_{ll'} = \langle\mathbf{0}|\, V^\dagger A_{l'}^\dagger A_l V\, |\mathbf{0}\rangle$. However, as proposed in the previous section, this problem is bypassed by computing an upper bound $\mu_{ll'} = 1/2\ \forall\ l, l'$, so that $\mu_{ll'}(1-\mu_{ll'}) = 1/4$:

$$\mathrm{Var}(\langle\psi|\psi\rangle) = \frac{4}{1+N_r}\sum_{l=1}^{L}\sum_{l'=l+1}^{L} c_l^2 c_{l'}^{*\,2} \tag{3.37}$$

and a typical instance $\mu_{ll'} = 1/6\ \forall\ l, l'$:

$$\mathrm{Var}(\langle\psi|\psi\rangle) = \frac{20}{9(1+N_r)}\sum_{l=1}^{L}\sum_{l'=l+1}^{L} c_l^2 c_{l'}^{*\,2}. \tag{3.38}$$

Once again, these results are shown to be in good agreement with numerical runs of $\langle\psi|\psi\rangle$ in Figure 3.12.

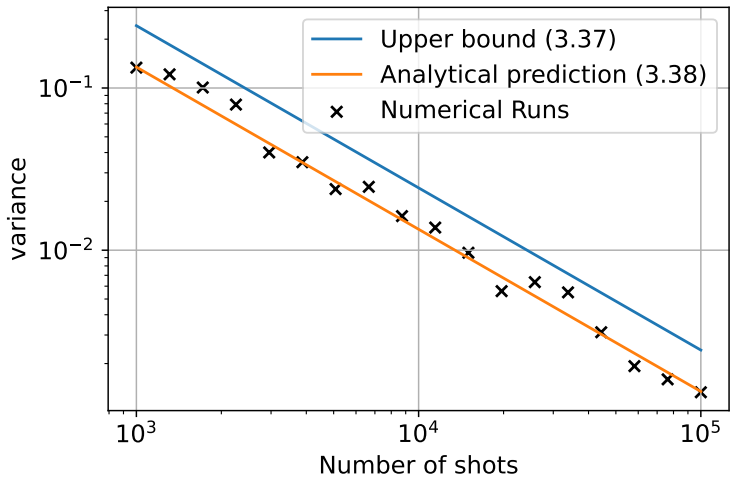


Figure 3.12: Numerical validation of (3.37) and (3.38), respectively upper bound and analytical prediction in the legend. Variance is computed with $N = 80$ separate measurement for each data point. Test case is for 3 qubits and a polynomial decomposition of A ($L \propto n$)

Variance of $|\langle b|\psi\rangle|^2$

In a similar way, the variance of $|\langle b|\psi\rangle|^2$ can be computed as a function of L and the number of shots. For this discussion $|\langle b|\psi\rangle|^2$ is computed as:

$$|\langle b|\psi\rangle|^2 = |\langle \mathbf{0}| U^\dagger A V |\mathbf{0}\rangle|^2 = \left(\sum_{l=1}^{L} c_l \langle \mathbf{0}| U^\dagger A_l V |\mathbf{0}\rangle\right)^2 \tag{3.39}$$

In principle, $|\langle b|\psi\rangle|$ is trivial, and can be obtained in a similar manner as (3.38):

$$\mathrm{Var}(\langle b|\psi\rangle) = \frac{5}{9(1+N_r)} \sum_{l=1}^{L} c_l^2. \tag{3.40}$$

However, here it is necessary to find the variance of a squared random variable, which in the most general case is:

$$\mathrm{Var}(X^2) = E[X^4] - E[X^2]^2 \tag{3.41}$$

which requires the knowledge of the expectation values of the squared distribution. To simplify the discussion, one can assume each measurement to be a symmetric beta distribution ($\mu = 1/2$). Under this hypothesis, it can be proven that for a large sample size, a symmetric beta $f_X(x) = \frac{\Gamma(2b)x^{b-1}(1-x)^{b-1}}{\Gamma(b)\Gamma(b)}$ can be approximated with a normal distribution (see Appendix C). Then, since the sum of normal distributions is still a normal distribution, summing each measurement yields

$$f_{\langle b|\psi\rangle} = \mathcal{N}(\mu, \sigma) = \mathcal{N}\left(\sum_{l=1}^{L} c_l, \sqrt{\frac{1}{(1+N_r)} \sum_{l=1}^{L} c_l^2}\right) \tag{3.42}$$

Luckily, moments of a normal distribution are known: $E[X^4] = \mu^4 + 6\mu^2\sigma^2 + 3\sigma^4$, $E[X^2] = \mu^2 + \sigma^2$ [51], so that:

$$\mathrm{Var}(X^2) = 4\mu^2\sigma^2 + 2\sigma^4 \tag{3.43}$$

In this case, analytical prediction does not seem to match numerical data precisely, perhaps because approximating β with a normal distribution is accurate only when $\mu = 1/2$, which is not necessarily the case for all measurements. Further studies could improve the accuracy of this estimation by devising a proof that does not rely on approximating the β with a normal distribution.

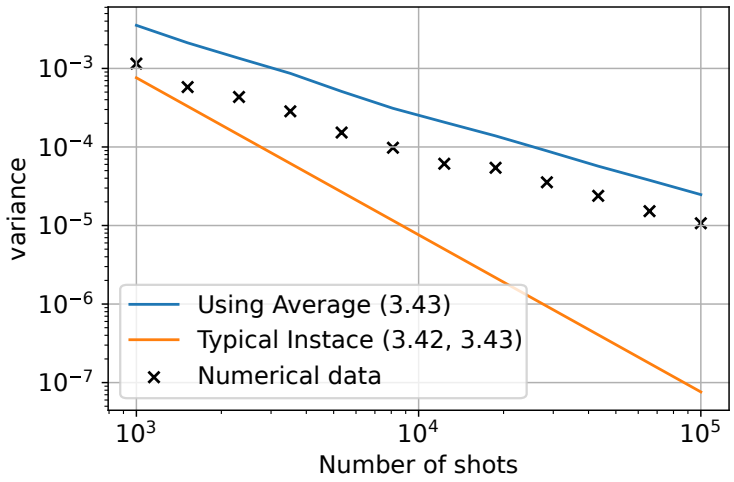


Figure 3.13: Numerical runs obtained measuring the variance of a sample of 150 runs for each number of shots point. Typical instance is obtained using (3.42) and the average as from (3.43) whereas a "Using average" is obtained plugging in the actual experiment average in (3.43)

Numerical Variance of $C(\boldsymbol{\theta})$

Finally, the variance associated with the whole cost function (3.34) is evaluated numerically at a random point of the domain, using the same test case introduced in Sections 4.1, 4.2, for which the number of unitaries is linear in the number of qubits ($L \propto n$) and $\sum_{l=1}^{L} c_l = 0$. Figure 3.14 suggest $\mathrm{Var}(C) \propto kN_r^{-1}$, showing a similar trend as $\mathrm{Var}(\langle\psi|\psi\rangle)$, with a coefficient $k \approx 1.54, 0.31, 0.33, 0.15$ for respectively 2, 3, 4, 5 qubtis configuration of the same run.

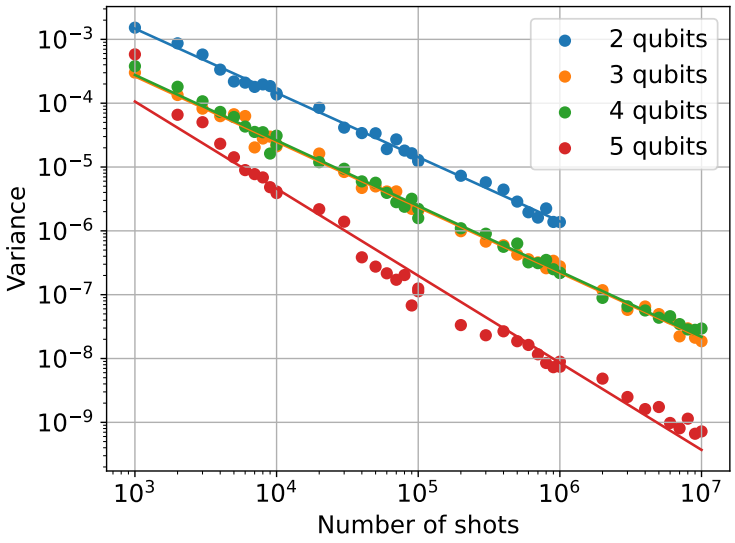
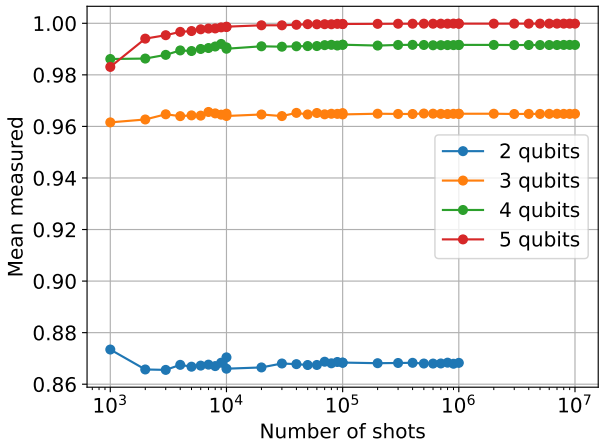


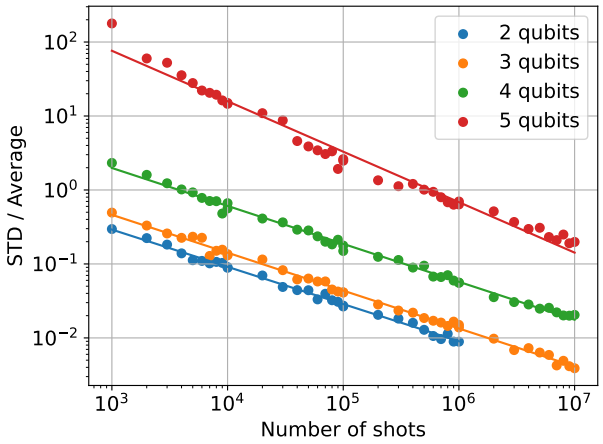
Figure 3.14: Numerical runs are obtained measuring the variance of a sample of 80 runs at each number of shot point. 2, 3, 4 qubits fits very precisely $\mathrm{Var}(C) \propto kN_r^{-1}$, whereas for 5 qubits $\mathrm{Var}(C) \propto kN_r^{-1.36}$, which is more likely due to under-sampling at low number of shots. The test case utilized is explained in sections 4.1 and 4.2.

However, as opposed to $\mathrm{Var}(\langle\psi|\psi\rangle)$, the variance seems to decrease for a higher qubit number, whereas in (3.38) $\mathrm{Var}(\langle\psi|\psi\rangle) \propto L^{\propto}n^2$. Perhaps, this scaling should be ascribed to (3.43), which claims $\mathrm{Var}(|\langle b|\psi\rangle|^2)$ also depends on the average of the distribution measured. In fact, for this test case $C(\boldsymbol{\theta})$ is larger for higher qubits, hence $\frac{|\langle b|\psi\rangle|^2}{\langle\psi|\psi\rangle}$ is smaller (Figure 3.15a). Similarly, this relation also explain why 3 and 4 qubits seems to coincide: this is a mere coincidence as the standard deviation measurement depends on the average of the random point in which it was measured and it may cause lines to randomly align.

To avoid the cost function value at the measurement point perturbing the measurement, Figure 3.15b shows the ratio of the standard deviation over the value measured, which is a more representative metric for the purposes of this section. This ratio is conceptually similar to measuring noise to signal ratio. Worryingly, noise to signal increases for higher qubits, potentially threatening the scaling of VQLS.



(a) Cost function value at measurement point



(b) Ratio between standard deviation and measured a value

Figure 3.15

One reasonable hypothesis for the decrease in signal-to-noise ratio would be that while standard deviation scales with $\sum_{l=1}^{L} c_l^2$, which monotonically increases with L, $\mu_{\langle b|\psi\rangle} \propto \sum_{l=1}^{L} \mu_l c_l$, assuming μ_l uniformly distributed and L large will tend to $\mu_{avg} \sum_{l=1}^{L} c_l \to 0$, given that for the decomposition of A of this test case, $\sum_{l=1}^{L} c_l = 0 \; \forall \; n$. These results might also explain the gradient vanishing measured in Section 4.3.

Finally, whilst results for $\mathrm{Var}(\langle\psi|\psi\rangle)$ and $\mathrm{Var}(|\langle b|\psi\rangle|^2)$ are valid for all problems using the aforementioned VQLS evaluation routines, these results for $\mathrm{Var}(C(\boldsymbol{\theta}))$ are problem-specific. They depend on the matrix decomposition considered and the cost function routine utilized because they have been computed numerically for this test case rather than derived analytically.

Chapter 4

Numerical Experiments of VQLS

Whilst the previous chapters focus on the theoretical background and the pre-processing necessary to apply VQLS to FEM discretization of one dimensional Poisson equation, this chapter is dedicated to its implementation on a quantum computer simulator and discussion of results.

Section 4.1 defines the test case for the numerical experiments of this chapter, focusing on the differential equation studied and the discretization used. Instead, Section 4.2 epxlains the details of the implementation on a quantum simulator, including results and issues. Specifically, sub-section 4.2.1 explains the simulator setup, including ansatz, matrix decomposition and optimization algorithm utilized. Sub-section 4.2.2 gives an overview of the results of the experiments, showing solution accuracy, iterations and scaling challenges. Sub-section 4.2.3 and 4.2.4 illustrates different techniques to improve the convergence of VQLS and evaluates their performance, respectively focusing on the ansatz and on the cost function. Sub-section 4.2.5 briefly compares the effectiveness of classical optimizers. Finally, Section 4.3 summarizes the main challenges and issues observed through these experiments, including the interplay of shot noise and vanishing gradients.

4.1 Definition of the test case

As a proof of concept, the selected test case is the one dimensional Poisson equation with Dirichlet boundary conditions because of its simplicity, relevance and ease of implementation

$$\begin{aligned} -\Delta u &= k, \quad u \in (0,1) \\ u(0) &= u(1) = 0 \end{aligned} \tag{4.1}$$

where k is set to be constant throughout the domain. This allows to use the discretization strategies derived in Section 3.1, prove their functioning and verify if they converge. Once

again, first-order FEM discretization is used

$$A = \begin{bmatrix} 2 & -1 & 0 & \dots & & \\ -1 & 2 & -1 & 0 & \dots & \\ 0 & -1 & 2 & -1 & & \\ \vdots & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ 0 & 0 & 0 & \dots & -1 & 2 \end{bmatrix} \tag{4.2}$$

whereas k is represented by the constant vector $\mathbf{b} = (1/N^2, ..., 1/N^2)$. The implementation of $\mathbf{b}/||\mathbf{b}||$ on a quantum computer is straightforward and is obtained as realized through the operator $H^{\otimes n} |\mathbf{0}\rangle$.

4.2 Implementation on a Quantum Computer simulator

This section illustrates the implementation of VQLS applied to Poisson's equation on a Quantum computer simulator. The main objectives are to understand whether this technique is feasible, if it converges to a meaningful result, implementation hurdles and prospects.

4.2.1 Setup

Given a specific linear system one wants to solve, it is now necessary to input the required data in a Quantum computer accordingly. In particular, one has to represent a guess vector $\mathbf{x}$ as well as the linear system matrix A. In variational algorithms, the inital guess (or solution) vector is implemented through an ansatz (see Section 2.2.2), whereas A needs to be decomposed in a set of unitaries (see Sections 3.1, 3.2). In particular, for this test case, the setup used is as follows:

- **Hardware efficient ansatz**: in the most general case, the shape of a solution vector cannot be determined a priori. In other words, there are no physical insights that allow to determine a specific ansatz configuration and restrict the portion of the solution space that should be explored.
  - An exponentially deep ansatz has 2^n control parameters (where n is the number of qubits) so that it can represent an arbitrary vector of dimension 2^n. It allows to achieve a numerically exact solution but scales poorly and is not feasible for a higher number of qubits because the circuit becomes exponentially deep
  - An ansatz where the number of input variables is smaller than 2^n is easier to optimize and implement, but might not be able to represent the solution vector accordingly

- **Polynomial matrix decomposition**: The linear system matrix (4.2) is decomposed using the efficient high-entanglement decomposition presented in Section 3.2.2. Clearly, this decomposition cannot be implemented on real quantum hardware, however, because of its efficiency and scaling, it could serve as a proof of concept to understand future capabilities on more robust quantum computers.

- **Powell optimization algorithm** [42] was utilized for most of the runs unless indicated otherwise. This choice is based on literature, where Powell was indicated as the best optimization technique for variational algorithms among those investigated [35] as well as preliminary comparisons on the current test case.

4.2.2 Solution overview

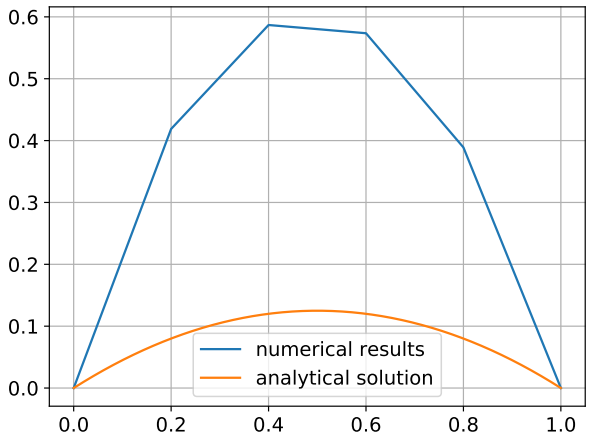
By definition, the solution vector obtained by VQLS is normalized to 1 ($|| \, |x\rangle \, || = 1$), however this is not necessarily true for an arbitrary solution vector $\mathbf{x}$. Therefore, it is necessary to change its norm to match the exact one. Luckily, this is a simple process and can be done by computing the ratio between *one* component of $\mathbf{b}$ and $A \, |x\rangle$. Given a random component i

$$\mathbf{x} = r \, |x\rangle \quad \text{with} \quad r = \frac{b_i}{\psi_i} \tag{4.3}$$

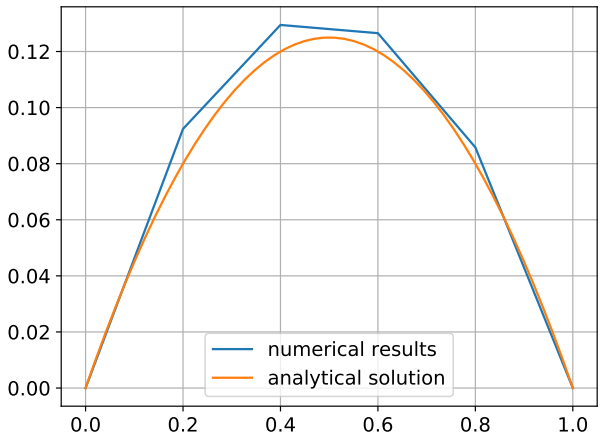
and $\psi = A \, |x\rangle$. In this specific case, the right hand side vector was set as $b_i = 1/N^2$, and given the sparsity of A the ration is given by the simple expression

$$\begin{aligned} r &= \frac{1}{N^2(-\psi_{i-1} + 2\psi_i - \psi_{i+1})} \\ &= \frac{1}{N^2(-A_{ii-1} \, |x\rangle_{i-1} + 2A_{ii} \, |x\rangle_i - A_{ii+1} \, |x\rangle_{i+1})} \end{aligned} \tag{4.4}$$

which is straightforward to compute.



(a) Un-normalized solution as for VQLS output

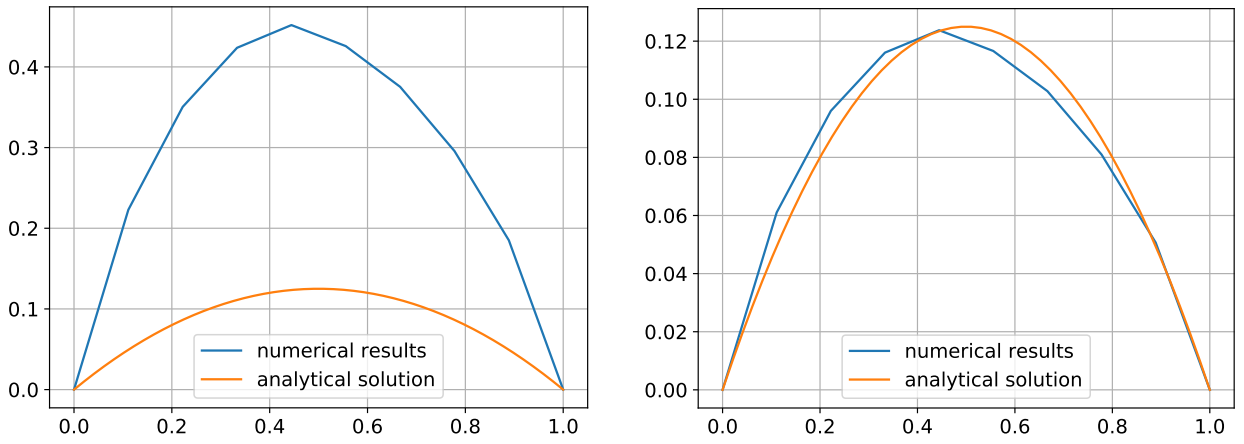


(b) Normalized solution rescaling VQLS output with (4.4)

Figure 4.1: Numerical results for Poisson's equation with $n = 2$ qubits (internal points $N = 4$), number of shots=10^6 and $c \approx 0$, exponentially deep ansatz.

In Figure 4.1 it can be seen that VQLS is able to produce meaningful solutions for small problem instances. Moreover, the re-normalization procedure is not perfectly accurate: although the cost achieved is close to 0, an inaccurate rescaling of the solution implies a slightly inaccurate final result. Even if rescaling is more accurate if (4.4) is averaged over more terms, this source of error on the final solution should be taken into account and evaluated in futures studies.

Similarly, Figure 4.2 shows the solution obtained for a three-qubit configuration ($n = 3$). While a finer discretization generally implies more accurate results for a convergent discretization, using VQLS with exponentially deep ansatz results in a higher number of optimization variables, which hinders convergence and affects the accuracy of the solution obtained.

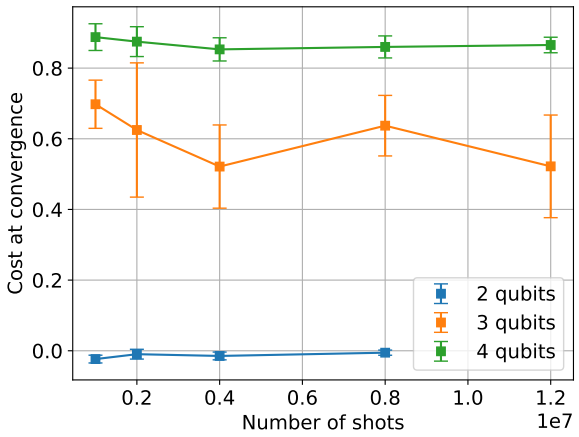


(a) Un-normalized solution

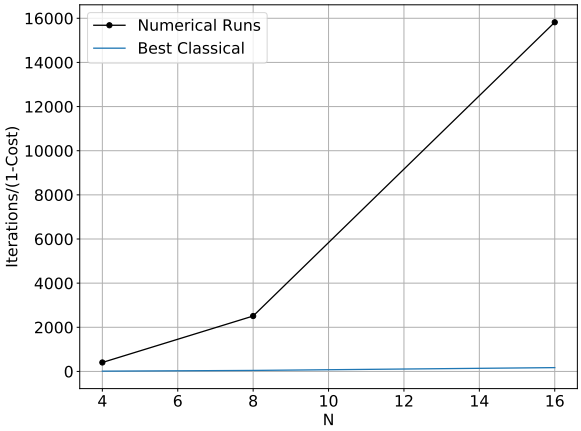
(b) Normalized solution

Figure 4.2: Numerical results for Poisson's equation with $n = 3$ qubits (internal points $N = 8$), number of shots=$1.2 \cdot 10^7$, $c \approx 0.1$, exponentially deep ansatz.

Given that VQLS can converge to a solution, it is interesting to understand how quickly it converges, if it scales favourably and could outperform classical methods in the future. As the number of qubits rises, so does the number of variables that needs to be optimized, making convergence of the optimization loop harder. In particular, as show in Figure 4.3a, runs with two qubits (4 optimization variables) converged for all data points evaluated. On the other hand, only a few runs converged to a solution with three qubits, whereas four qubits never converged, hence, the high average cost function. This analysis seems to suggest that VLQS is prone to vanishing gradient and dimensionality curse and these problems onset already at very low n.



(a) Average minimum cost function achieved, averaged over 4 runs per number of shots data point with an exponentially deep ansatz



(b) Iteration to solutions averaged over 4 runs per data point as a function of $N = 2^n$ data point, with an exponentially deep ansatz

Figure 4.3: Averaged convergence performance with exponential ansatz

Figure 4.3b shows how iterations-to-success scales with respect to the linear system size N. Because many runs did not converge to zero, in Figure 4.3b the iteration number is divided by $1 - C_{final}$ which is the cost achieved at the end of the optimization procedure. Somehow, this measure tries to account for the heuristic of the optimization procedure where, for example, some runs might get stuck in local minima and fail to converge. When counting iterations to success, using those runs would be misleading as they did not reach a solution. However, increasing the number of iteration obtained by dividing for "distance to success" can give a qualitative measure of scaling prospects. For example, if $C_{final} = 0.5$ on average, then the average expected number of iterations to success is double the average $N_{iterations}$. This scaling is by no means guaranteed and should only be considered a qualitative measure. Furthermore, it becomes progressively less reliable as $C_{final} \to 1$ because those runs might never converge or require far more trials than predicted.

As a comparison, Figure 4.3b reports the scaling of the best known classical algorithm, the conjugate gradient method, which scales with $\sqrt{\kappa}N$ given that A is positive definite. It is noteworthy that VQLS scales more poorly than represented because one iteration requires several circuit queries to compute the cost function and complete one iteration. However, this becomes less relevant as n rises, given that the decomposition studied scales logarithmically in N, polynomial in n. Furthermore, it is important to mention that this scaling is qualitative and not rigorous. The number of iterations also depends on the optimization technique employed and has some degree of randomness but is not mathematically bounded. Moreover, it is crucial to remark these results are preliminary as a more solid analysis of scaling prospects would require a more extensive n range.

4.2.3 Ansatz optimization strategies

In this subsection, several techniques to improve convergence by acting on the ansatz are discussed. In general, the overarching idea is to ease optimization by reducing the dimensionality of the optimization problem. This can be done by either reducing the ansatz's depth or optimizing it in a stepwise manner.

Shallow ansatz configuration

Because convergence strongly depends on the problem dimensionality, it is relevant to investigate if shallower ansatzes reduce dimensionality without excessively compromising accuracy. Ideally, one would like a tailored ansatz that only explores a small portion of the vector space containing the solution, though this is generally problem-dependent and difficult to find out. In our test case, one can blindly try to shrink the hardware efficient ansatz to a single rotation layer and evaluate the result.

This shallow configuration is appealing because it reduces implementation complexity (does not require 2-qubit gates) and gate count.

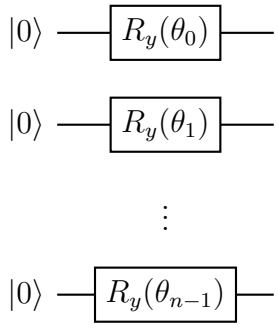
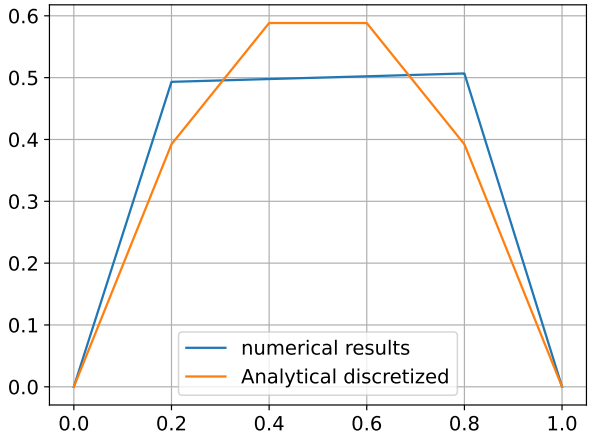
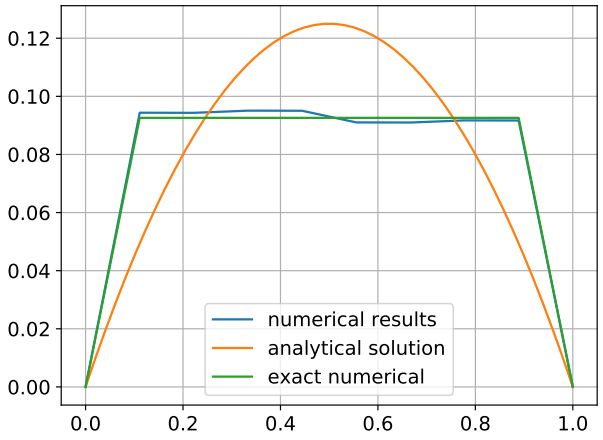


Figure 4.4: Single layer rotation ansatz

Consequently, degrees of freedom of the solution guess are reduced, and there is no guarantee that this configuration will allow achieving meaningful results. In fact, Figure 4.5 shows poor results obtained using this ansatz.



(a) 2 qubits, compared to analytical solution discretized and normalized

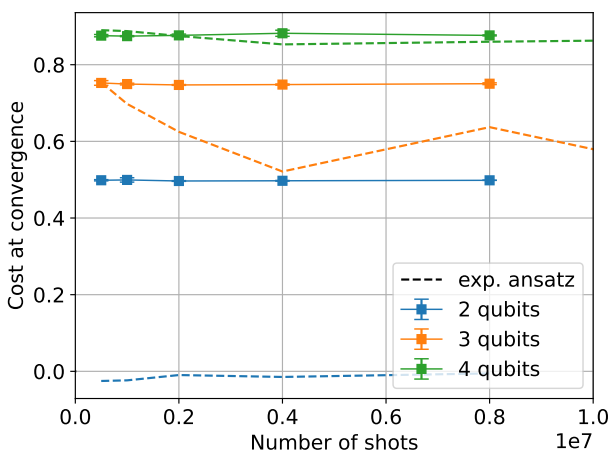


(b) 3 qubits (rescaled manually), compared to exact analytical solution

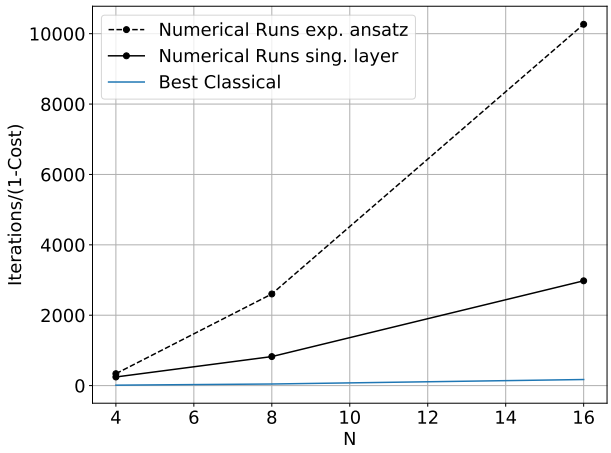
Figure 4.5: Numerical results with a single rotation ansatz, $8 \cdot 10^6$ shots. "Exact numerical" was obtained using a statevector simulation to make sure poor results were only due to the ansatz and not poor convergence in the optimization process

The performance of single-layer and exponential ansatzes for this test case are compared in Figure 4.6, both in terms of total number of iterations and best cost achieved. First of all, in Figure 4.6a one immediately notices that a single layer ansatz consistently underperforms, but the gap decreases as the number of qubits rises. Since for a single layer ansatz the number of optimization parameters grows linearly in n, whereas it is exponential for the exponentially deep ansatz, it is no surprise that the latter's performance deteriorates faster because of increasing problem dimensionality. Nevertheless, this comes at a slight tangible advantage since they both fail to provide a meaningful solution. Moreover, a single layer ansatz seems to be more resilient to noise, showing little to no variation with decreasing number of shots. In these cases, although noise is still persistent, for the number of shots examined, precision is enough to reach the best solution achievable with the given ansatz. While not shown in Figure 4.6a, noise would start to affect convergence at a lower number of shots.

Similarly, it is noteworthy in 4.6b a shallower ansatz leads to a smaller number of iterations, though that comes at the expense of precision.



(a) Average minimum cost function achieved



(b) Iteration to solutions as a function of $N = 2^n$

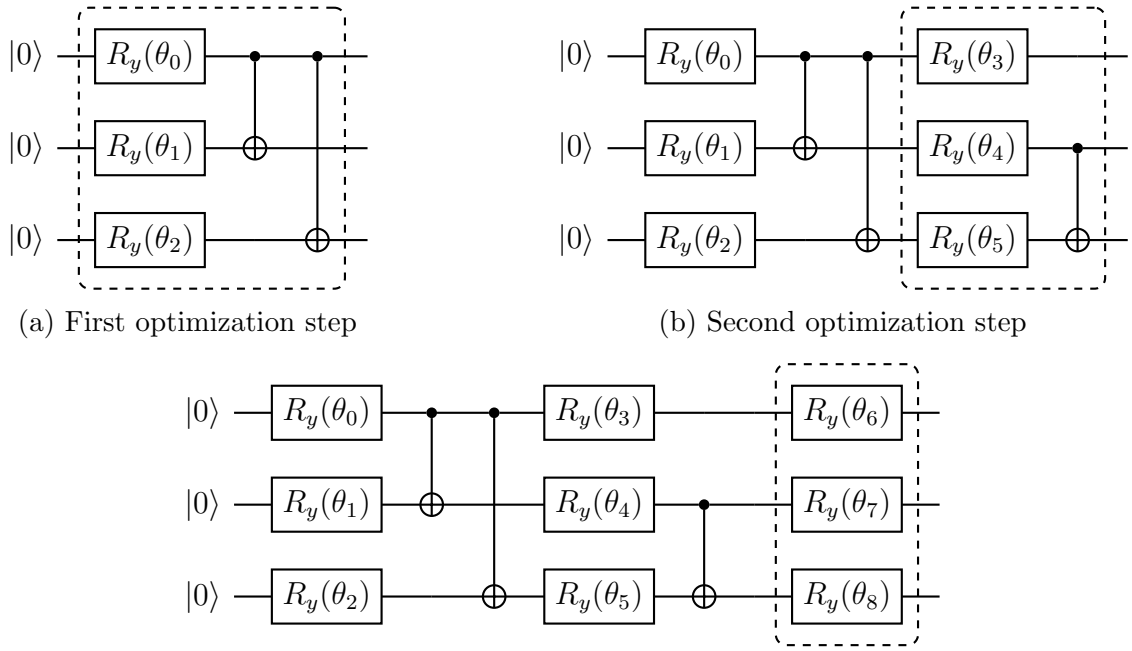
Figure 4.6: Comparison of convergence performance between single layer and exponentially deep ansatz (dotted lines)

Because the effectiveness of an ansatz is problem-specific, this analysis only shows it is not appropriate for this test case and that it is generally not effective in all situations. However, there could be cases when a single layer or a polynomial number of layers is sufficient to represent the solution vector.

Given that the necessary ansatz depth is not known a priori and that optimization becomes progressively difficult as the number of parameters increases, instead of attempting to guess the required depth, one could start with a shallow ansatz and progressively add layer optimizing at each step. In this case, one can check if $C \approx 0$ at each stage and stop optimizing when appropriate, hopefully with fewer layers than exponential. Moreover, this approach was also proposed in the literature to overcome the issue of vanishing gradient and help the optimization procedure [34, 35]. Hence, it seems a reasonable way forward to tackle both problems.

Progressive layer optimization: one layer at the time

When optimizing layer by layer, one key question is the degree of independence between each layer. In other words, it would be convenient if independently optimizing one layer at a time could achieve a solution because it would consist of sequential optimization with fewer variables (polynomial in n) rather than one optimization with many variables. To address this question, this paragraph illustrates results obtained optimizing only each ansatz layer at the time independently.



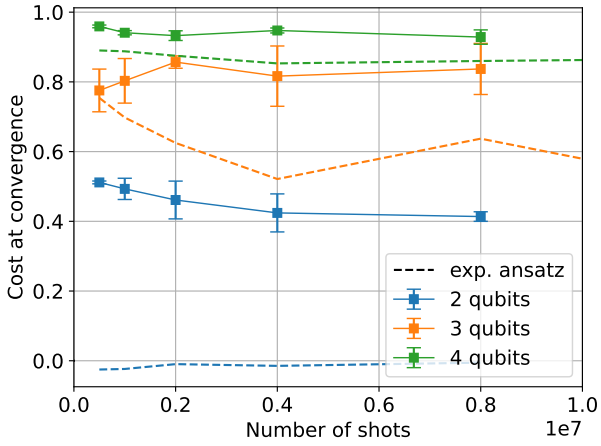
(a) First optimization step

(b) Second optimization step

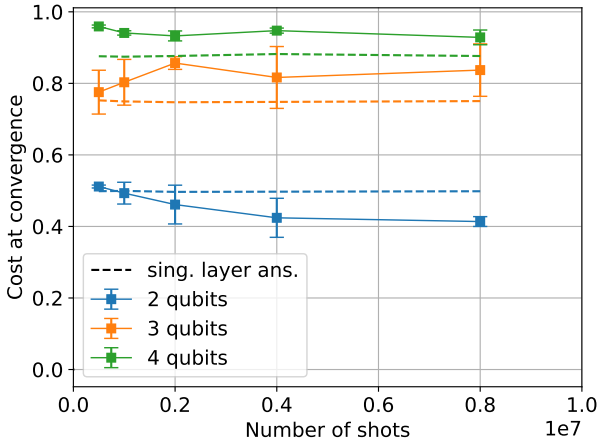
(c) Third optimization step

Figure 4.7: Example of gradual optimization for three qubits. At each step, only the boxed parameters are optimized. Once a local minimum is achieved, the optimizer moves to the next set of parameters and those from previous steps (e.g. $\theta_0, \theta_1, \theta_2$ in step 2) are left untouched. New parameters are initialized as $\theta_i = 0$ so that $R_y(\theta_i) = I$.

The optimization procedure is illustrated in Figure 4.7, whereas convergence results are displayed in Figure 4.8. In particular, as shown in Figure 4.8a, optimizing layer by layer seems less effective compared to exponential ansatzes, especially for fewer qubits. Similarly to a single rotation ansatz, the gap progressively shrinks as the number of qubit rises, since both runs fail to converge to meaningful results. For this test case, these results indicate that optimizing independently single ansatz layers does not provide good results. Intuitively, this makes sense, as each entry of the solution vector depends on all ansatz layers and there are no "local" effects, in the sense that none of the statevector entries depends on one ansatz parameter only.



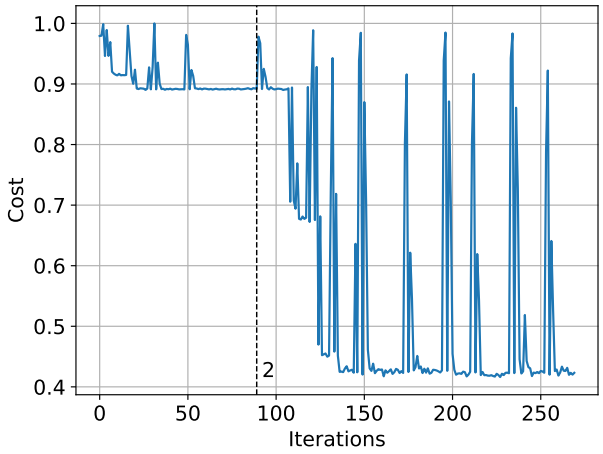
(a) Comparison of gradual optimization vs exponentially deep ansatz



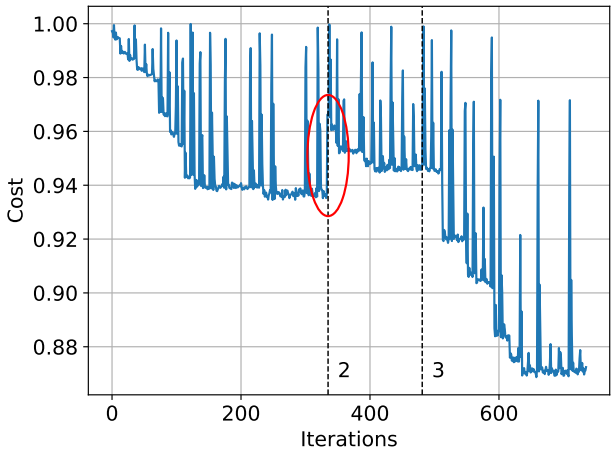
(b) Comparison of gradual optimization ansatz vs single layer of rotation ansatz

Figure 4.8: Comparison of convergence performance between different ansatzes as a function of the number of shots and number of qubits

On the other hand, in Figure 4.8b it is interesting to see how for 3 and 4 qubits, gradual optimization performs worse than a single rotation layer. Intuitively, this should not be the case: even if optimizing one layer at a time might not work because of qubits independence, it should be at least equal to single rotations.



(a) 2 qubits



(b) 3 qubits

Figure 4.9: Cost function as a function of the number of iterations. Dotted lines indicate where a new ansatz layer is added, with the optimizer moving to the next set of parameters.

Perhaps, poor performance is caused by the addition of new layers to the cost function. As shown in Figure 4.9b, after the second layer is added, there is a sudden jump of the cost function (marked by a red ellipse). The reason for this discontinuity is shown in Figure 4.7:

adding a new layer "perturbs" the ansatz, leading to a different statevector than the result of the optimization. In particular, although $R_y(\theta)$ are initialized with $\theta = 0$ to avoid this issue, the entangler gates that follows are sufficient to alter the statevector optimized in the previous step. As further confirmation, when adding the third layer in Fig. 4.9b and the second in Fig. 4.9a, no discontinuity in the cost function appears because those are last layers that are not followed by an entangler gate (see Figure 4.7c). The latter could also explain why gradual optimization outperforms a single layer only for two qubits when this type of discontinuity does not happen.

Some strategies exist to mitigate this issue: for example, to avoid a gradient plateau, Grant et al. [34] suggested initializing the ansatz as a series of blocks, each containing a gate sequence and its inverse so that each block reduces to identity before being optimized. This structure would also prevent the issue mentioned above but requires either changing the ansatz structure or deeper ansatzes. This is because, with the current design, two different entangler gates alternate (see Figure 2.2 for example), which would require at least a depth of 4 layers to reduce a single block to identity. Alternatively, one could start the optimization procedure with the whole ansatz implemented with all $R_y(0)$ and progressively optimize the rotation gates while the entanglers are left untouched. However, this solution comes with two drawbacks: firstly, the ansatz depth is fixed instead of progressively increasing only if necessary. Secondly, although this approach helps to reduce problem dimensionality, if the ansatz is deep, it might experience vanishing gradients because of the large number of entangler gates at initialization.

Finally, Figure 4.10 shows the iterations to the local minimum. Noteworthy is that gradual optimization requires fewer iterations on absolute value but underperforms when taking the average cost into account.

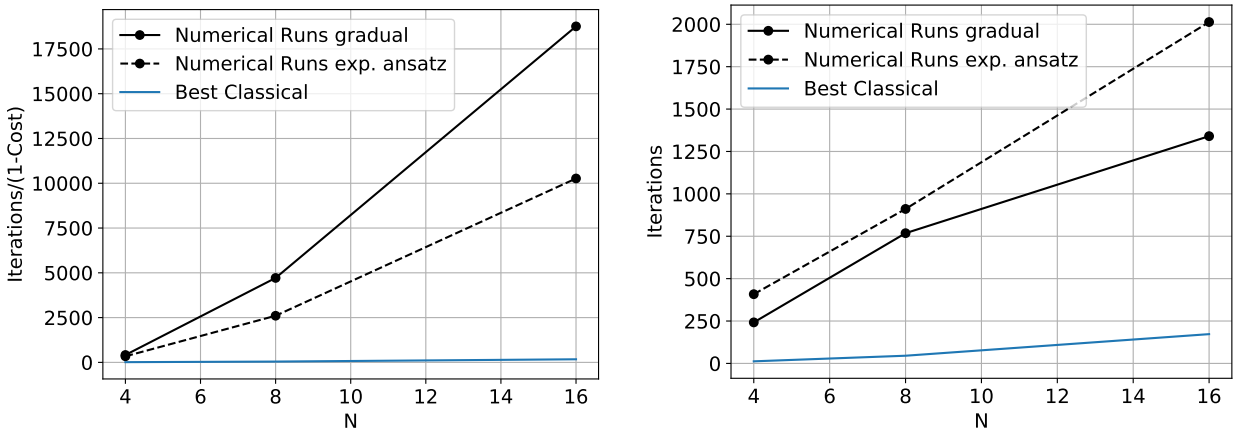
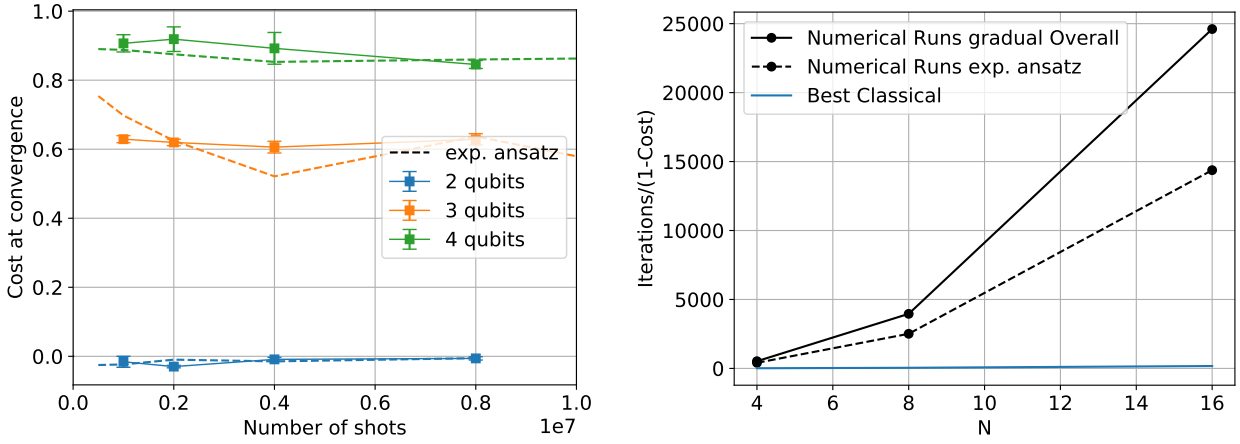


Figure 4.10: Number of iterations to local minimum

Overall optimization after gradual optimization

In previous paragraphs, it was shown that, for this test case, gradual layer optimization performs quite poorly, failing to outperform both a single layer ansatz and an exponential setup. However, it is still interesting to investigate whether a progressive layer optimization could provide a more advantageous starting point for the overall optimization of an exponential ansatz.

The results for this test case are reported in Figure 4.11. In general, this approach seems to yield no advantage, with a very similar final cost compared to a simple optimization of an exponential ansatz and a minor increase in noise resilience (shown in Figure 4.11a). Moreover, the standard deviation associated with the final results is smaller compared to previous test cases. Although this is advantageous in those cases when the optimizer can achieve a good final result, it also implies this approach could end up blocked in local minima more often, as shown in the three qubits test case. As shown in Figure 4.11b gradual optimization approach accounts for a higher iteration count coherently with the fact that the optimization procedure requires more steps, which implies this optimization procedure is not advantageous.



(a) Cost at convergence as a function of number of shots and number of qubits

(b) Iterations normalized by quality of results

Figure 4.11: Result of overall optimization using as a starting point the results of gradual optimization vs exponential ansatz baseline case. Powell was used for all data points.

Gradual optimization of exponentially deep ansatz

The previous section showed a gradual (layer by layer) optimization of the ansatz to underperform an exponentially deep ansatz. Even when an overall optimization is performed after a gradual procedure, similar results are achieved but with more iterations. One cause could be the discontinuity in the cost value introduced by adding a layer and the interdependence of all optimization parameters, which prevents achieving a good solution when only a subset

is optimized.

To test these hypotheses, in this subsection the same gradual optimization procedure is applied with a different approach: optimization starts with an exponentially deep ansatz already at the first iterations, and non-active layers initialized as $\theta_i = 0$. The first step is illustrated in Figure 4.12, where $\theta_0, \theta_1, \theta_2$ are optimized, successively $\theta_3, \theta_4, \theta_5$ and so on.

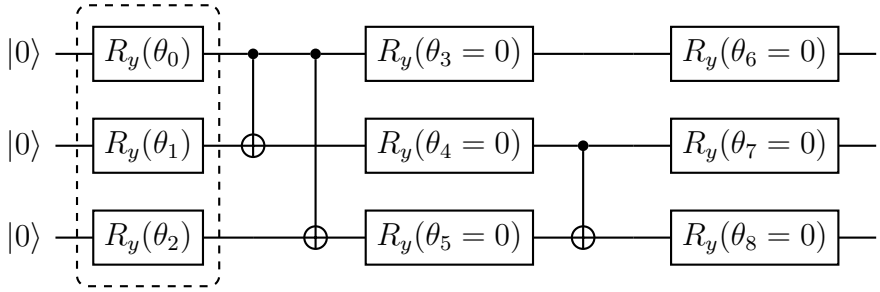
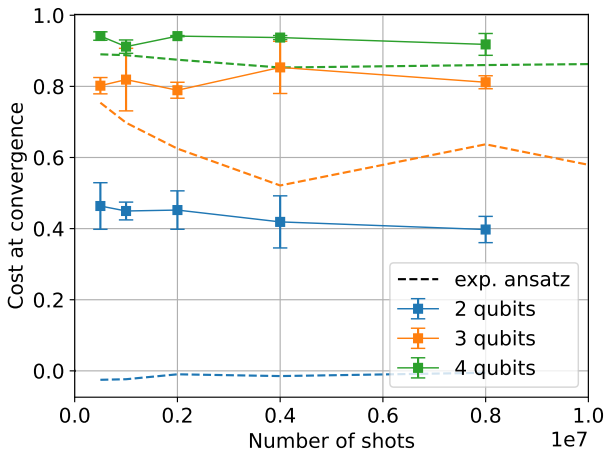
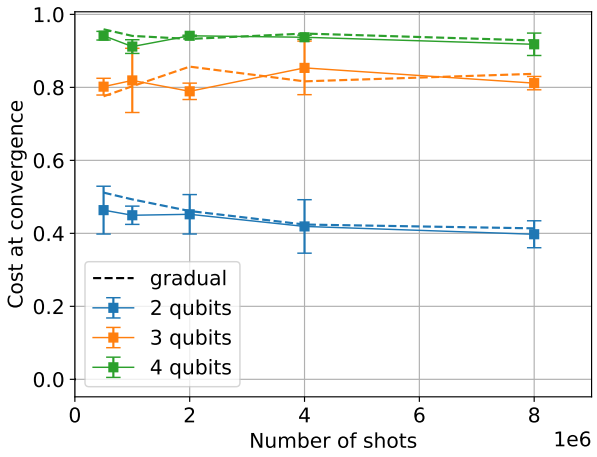


Figure 4.12: Gradual optimization of an exponentially deep ansatz

Compared to 4.7, this ansatz aims to retain the advantages of gradual optimization (smaller problem dimensionality) while avoiding the cost function discontinuities caused by adding new layers as in Fig. 4.9b. Moreover, this also allows testing whether these discontinuities are responsible for poor performance or if the main cause is the interdependence of optimization parameters. One main drawback is that one has to assume a priori that the ansatz is exponentially deep. In contrast, in the ansatz of Figure 4.7 it is possible to stop at a shallower depth in case convergence is achieved



(a) Gradual optimization of exponentially deep (Fig. 4.12) vs exponentially deep ansatz

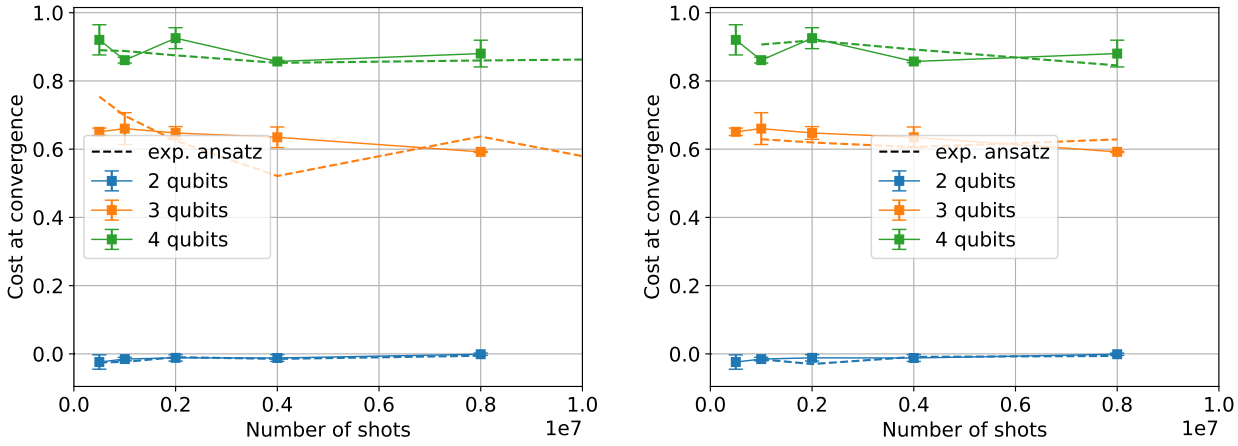


(b) Gradual optimization of exponentially deep (Fig. 4.12) vs layer by layer (Fig. 4.7c)

Figure 4.13: Cost at convergence as a function of number of shots and number of qubits for different ansatz

Similarly to 4.8a, in 4.13a, gradual optimization of an exponentially deep ansatz underperforms the exponentially deep ansatz baseline case for all qubits configurations. In 4.13b it is noteworthy that between the two gradual optimization techniques, there is no tangible difference in the quality of the final results, suggesting the aforementioned issue of cost function discontinuities does not play a relevant role when evaluating minimum cost achieved.

Finally, likewise Figure 4.11, using the results above as a starting point for an overall optimization is not advantageous when compared to the optimization of an exponential ansatz with a random starting point (see Figure 4.14).



(a) Gradual optimization of exponential ansatz vs exponentially deep ansatz

(b) Gradual optimization of exponential ansatz vs gradual optimization layer by layer

Figure 4.14: Cost at convergence as a function of number of shots and number of qubits

Concluding Remarks on Ansatz optimization strategies

Overall, different optimization strategies of the ansatz did not lead to meaningful advantages compared to the baseline case. Moreover, for this test case and possibly most FE systems of equations, an exponentially deep ansatz seems necessary to achieve a precise solution. This issue poses a serious threat to the scalability of this algorithm, as exponentially deep ansatzes are harder to optimize and lead to circuits beyond current hardware capabilities because of their depth.

Future efforts could focus on devising an ansatz that is better suited to the problem at hand. Generally speaking, this is not possible for linear systems of equations as the solution vector can be of arbitrary shape depending on $\mathbf{b}$. However, perhaps in some practical application, knowledge of the problem's physics could allow better tailoring of the ansatz.

4.2.4 Cost function improvement strategies

As seen in the previous sections, VQLS has a troublesome convergence when solving for this test case. Moreover, different optimization strategies of the ansatz did not lead to any better convergence. Thus, in this section, the focus is shifted on the objective function itself to understand if modifications to its formulation can lead to better convergence behaviour.

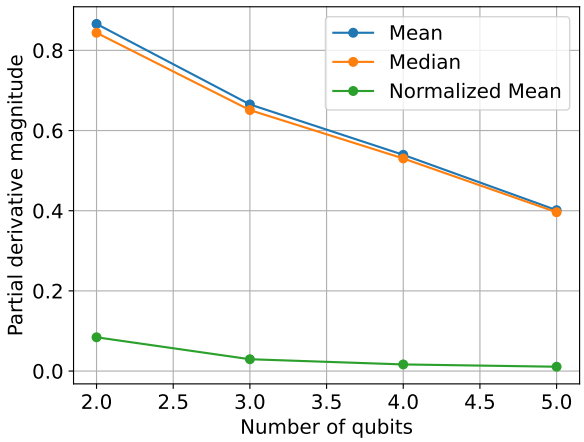
Non-normalized cost function

When VQLS was introduced, Bravo-Prieto et al. [28] proposed different formulations of the cost function, each with different proprieties. In particular, a non-normalized and normalized cost function are defined respectively as:

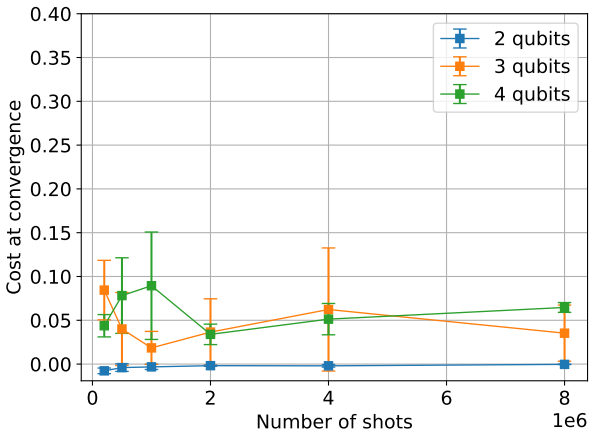
$$\hat{C}(\boldsymbol{\theta}) = \langle\psi|\psi\rangle - |\langle b|\psi\rangle|^2, \quad C(\boldsymbol{\theta}) = 1 - \frac{|\langle b|\psi\rangle|^2}{\langle\psi|\psi\rangle} \tag{4.5}$$

where $\psi = A|x(\boldsymbol{\theta})\rangle$. Essentially these function both measure if $A\mathbf{x}$ is linearly dependent of b. The main difference between these two formulations is that $\hat{C}(\boldsymbol{\theta})$ becomes small both when $\langle\psi|\psi\rangle = |\langle b|\psi\rangle|^2$ (which is the solution), but also when the norm of ψ is small, which is not necessarily the solution point, whereas $C(\boldsymbol{\theta}) \to 0$ only at the solution point.

Figure 4.15a shows average and mean partial derivative resulting from a random sampling across the domain for a non Normalized cost function, notably larger in magnitude compared to the normalized case. Moreover, similar values of median and mean suggest a less skewed distribution, indicating a "smoother" cost function as opposed to plateaus and steep wells for the normalized one. These hypotheses are validated in Figure 4.15b, which displays the results of the non-Normalized cost function, showing markedly lower cost function values achieved compared to normalized cost function runs from previous sections.



(a) Partial derivative average and mean values for a non normalized cost function

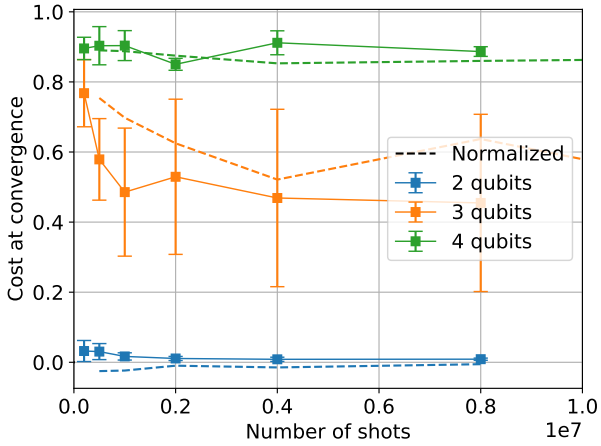


(b) $\hat{C}(\boldsymbol{\theta})$ at convergence as a function of the number of shots and the number of qubits

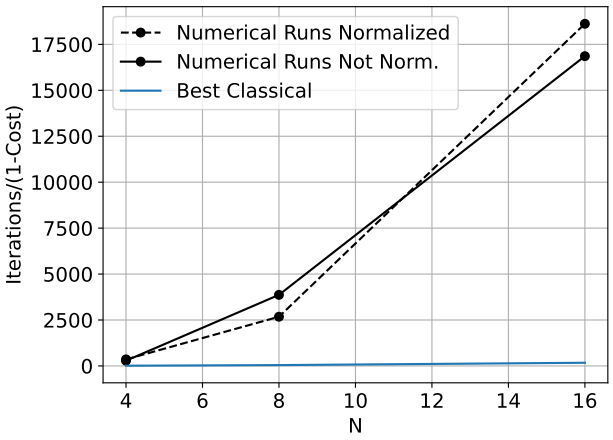
Figure 4.15: Cost landscape and cost at convergence for a non normalized cost function

However, as mentioned above, small cost function values do not necessarily imply the algorithm converged to a valid solution. To check for this, $C(\boldsymbol{\theta})$ is calculated using the solution's parameters and compared to results obtained optimizing a normalized cost function in Figure 4.16a. Firstly, it is noteworthy overall convergence improved, especially for the three-qubits configuration. In particular, the average cost function is lower but with a higher standard deviation. The latter is because if $\hat{C}(\boldsymbol{\theta}) \approx 0$ either solution is achieved and $C(\boldsymbol{\theta}) = 0$, or the norm of $\langle\psi|\psi\rangle$ is small, which results in neatly distinct outcomes of the normalized cost function. This also implies that although average values are similar, minimum values achieved are markedly smaller when optimizing $\hat{C}(\boldsymbol{\theta})$.

On the other hand, for four qubits, results are in line with the normalized case despite the small cost achieved in Figure 4.15b, suggesting in this case minimization of $\langle\psi|\psi\rangle$ is most likely the reason for $\hat{C}(\boldsymbol{\theta}) \approx 0$. In general, iterations to convergence do not depend on the cost function choice when using the same optimization algorithm.



(a) Cost at convergence as a function of number of the shots and qubits



(b) Iterations to convergence

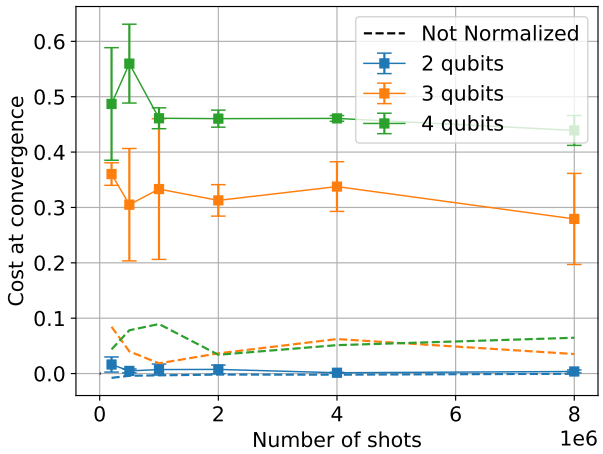
Figure 4.16: Result of overall optimization of $\hat{C}(\boldsymbol{\theta})$ (non Normalized) vs $C(\boldsymbol{\theta})$ (normalized). Powell was used for all data points.

Linear combination of cost functions

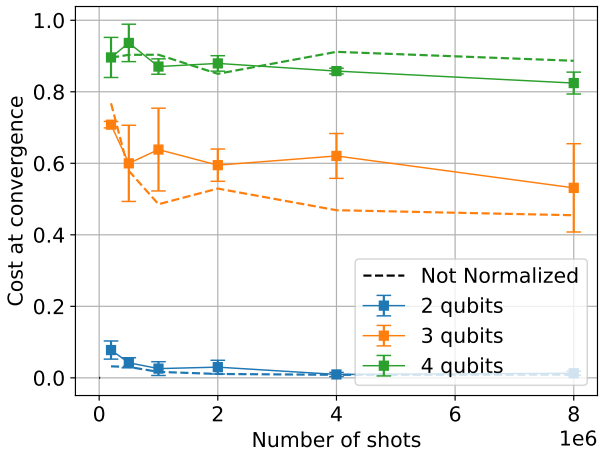
As introduced above, the normalized cost function is harder to optimize but only converges to the "real" solution. In contrast, a non-normalized converges easier but not always to the right results. Thus, it is interesting to explore if a linear combination of these functions could retain ease of optimization while converging to the actual solution. The cost function is then defined as:

$$\tilde{C}(\boldsymbol{\theta}) = \alpha\hat{C}(\boldsymbol{\theta}) + \beta C(\boldsymbol{\theta}). \tag{4.6}$$

Figure 4.17, 4.18 show results obtained with $\alpha, \beta = 1/2$. In 4.17a it is noteworthy that adding $\hat{C}(\boldsymbol{\theta})$ to $\tilde{C}(\boldsymbol{\theta})$ makes the overall cost landscape harder to optimized because all runs converged to a higher costs. Consequently, in 4.17b the linear combination approach is shown to perform worse compared to $\hat{C}(\boldsymbol{\theta})$. On the other hand, adding $C(\boldsymbol{\theta})$ seem to help avoiding minimization of $|\langle\psi|\psi\rangle|$ because for $\tilde{C}(\boldsymbol{\theta})$ the difference between Figure 4.17a and 4.17b is less marked, implying $\tilde{C}(\boldsymbol{\theta})$ converged to that result less often.



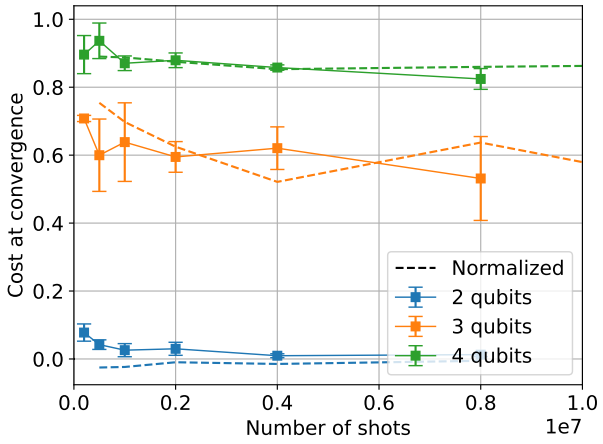
(a) $\tilde{C}(\boldsymbol{\theta})$ and $\hat{C}(\boldsymbol{\theta})$ as resulting from the optimization process



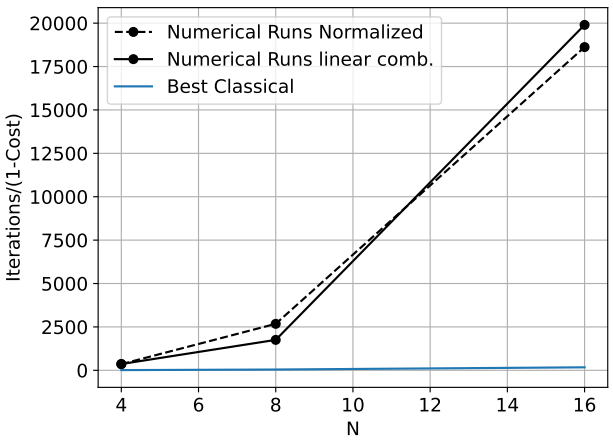
(b) Normalized cost for $\tilde{C}(\boldsymbol{\theta})$ and $\hat{C}(\boldsymbol{\theta})$.

Figure 4.17: Cost at convergence as a function of number of shots and number of qubits for $\tilde{C}(\boldsymbol{\theta})$ (linear combination with $\alpha, \beta = 1/2$) vs $\hat{C}(\boldsymbol{\theta})$ (referred as "not Normalized" in the plots). Powell was used for all data points.

As shown by Figure 4.18, optimizing $\tilde{C}(\boldsymbol{\theta})$ seems to provide a limited advantage compared to the baseline $C(\boldsymbol{\theta})$ case, both in terms of iterations number and quality of results. Using this linear combination transform the cost landscape such that there is a local minimum in $|\langle\psi|\psi\rangle| = 0$ and a global minimum at the solution point. Perhaps, optimization is more likely to get stuck in local minima rather than reaching the global, negatively impacting performance.



(a) Cost at convergence as a function of the number of shots and qubits



(b) Iterations to convergence

Figure 4.18: Result of overall optimization of $\tilde{C}(\boldsymbol{\theta})$ (linear combination with $\alpha, \beta = 1/2$) vs $C(\boldsymbol{\theta})$ (normalized). Powell was used for all data points.

Concluding Remarks on cost function improvement strategies

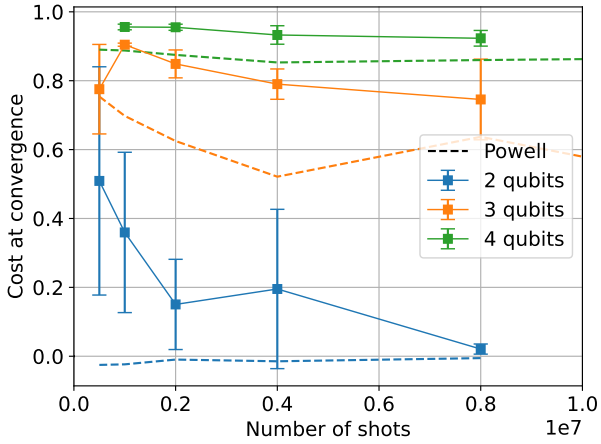
In general, there seems to be a noticeable advantage in using a non-normalized cost function compared to a normalized one, though this advantage wears off at four qubits. Although this method does not guarantee a meaningful solution when converged, better results are achieved with a suitable number of repetitions. On the other hand, a linear combination of these two cost functions did not seem to yield any advantage with respect to a non Normalized cost function.

As opposed to ansatz optimization strategies tested, acting on the cost function itself seemed effective in improving results. Therefore, it is recommended future studies focusing on this aspect of VQLS when studying convergence in these settings. In particular, the authors suggested the usage of a Local cost function which they claim improves convergence [28]. Testing this type of cost function would be the most logical next step following this work.

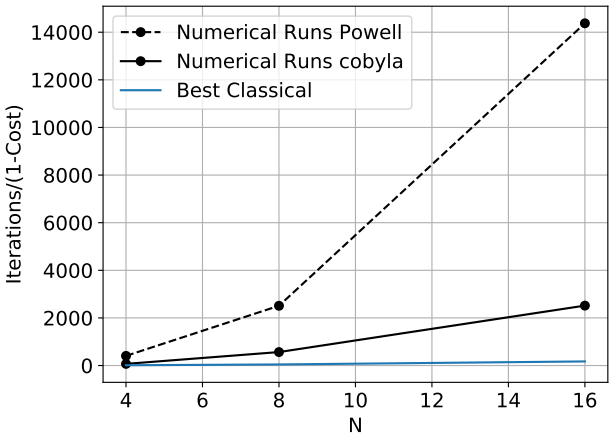
4.2.5 Comparison of classical optimization algorithms

In [35], La Rose et al. compare the effectiveness of some classical optimization algorithms applied to variational algorithms. Among the algorithms studied, Powell [42] achieved the best results (i.e. lowest cost function), whereas COBYLA was the algorithm requiring fewer iterations. Because these results can be problem-specific, these algorithms are compared for this test case in this section. The reader can consult a more comprehensive review of classical optimization for variational quantum linear solver at [52].

For this test case, Figure 4.19a shows Powell achieves better results than COBYLA for all data points. Moreover, it appears COBYLA is more susceptible to noise, with results quickly deteriorating for fewer shots, particularly for the two qubits case. On the other hand, COBYLA requires substantially fewer iterations, as shown in Figure 4.19. Given the difficulty of obtaining a meaningful solution, Powell is a more appropriate choice for this test case. In other settings, one could try to run COBYLA at first and then switch to Powell only if necessary.



(a) Cost at convergence as a function of the number of shots and qubits

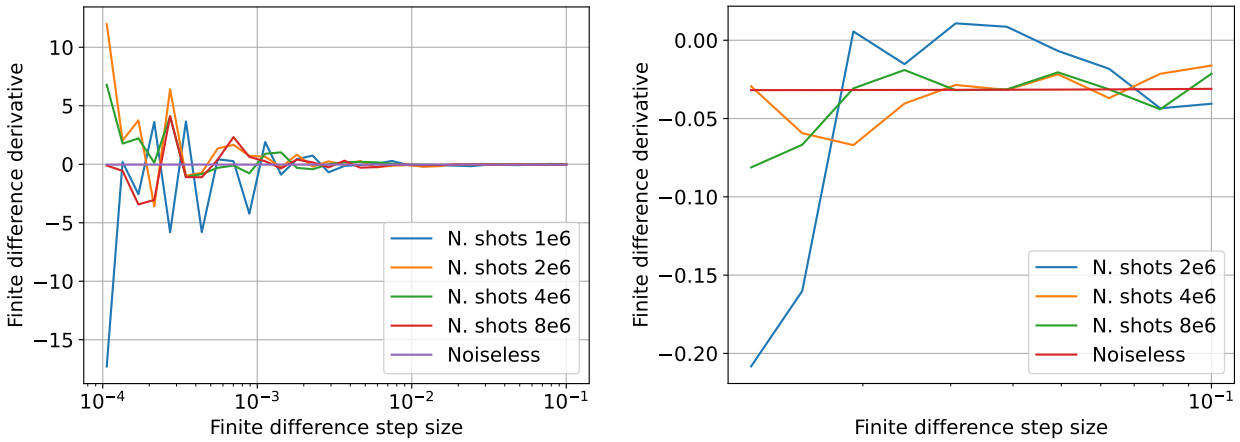


(b) Iterations normalized by quality of results

Figure 4.19: Comparison of Powell and Cobyla optimizers

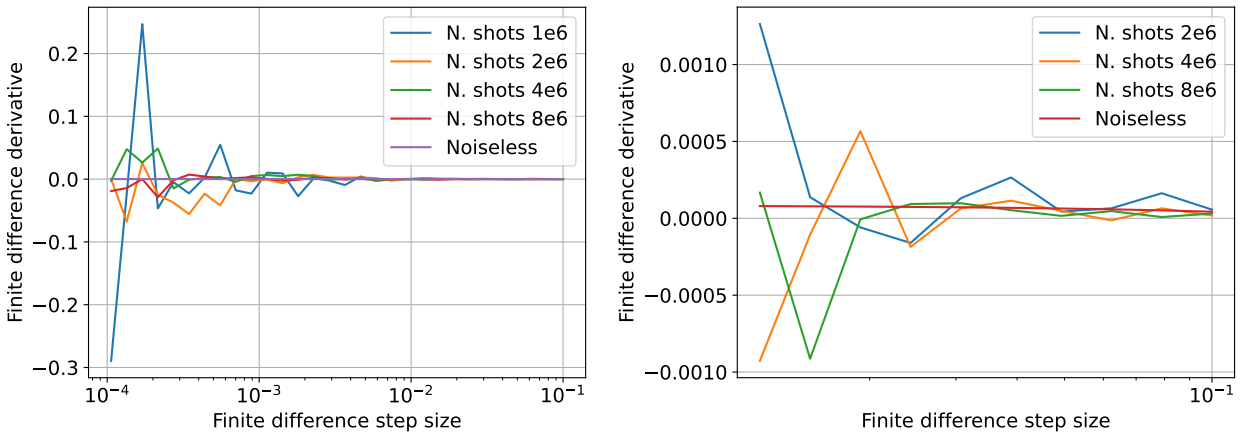
Comments on gradient based methods

Because 0-th order methods struggle to find meaningful solutions, it is interesting to investigate whether gradient-based algorithms could bring any advantage. To answer this question, Figure 4.20, 4.21 show the results of partial derivative computation as a function of step size and the number of shots, obtained using central finite difference. Clearly, in both cases, noise makes a reliable determination of a partial derivative unfeasible to the point where often not even a correct sign for the derivative can be determined. As predictable, finite difference measurement becomes increasingly less reliable for higher n mostly because partial derivatives are, on average, smaller in magnitude.



(a) Overall (b) Detail

Figure 4.20: Finite difference partial derivative as a function of step size and number of shots for a 2 qubit configuration



(a) Overall (b) Detail

Figure 4.21: Finite difference partial derivative as a function of step size and number of shots for a 3 qubit configuration

In [28], Bravo Prieto et al. suggest a quantum subroutine to compute the gradient. In principle, because this routine requires two circuit evaluations per variable, it seemed to bring a limited advantage compared to finite differences. However, because this analysis hints finite difference is unfeasible, this implementation could lead to tangible benefits. Therefore, for future works it is suggested similar studies are carried out on this routine to assess its feasibility.

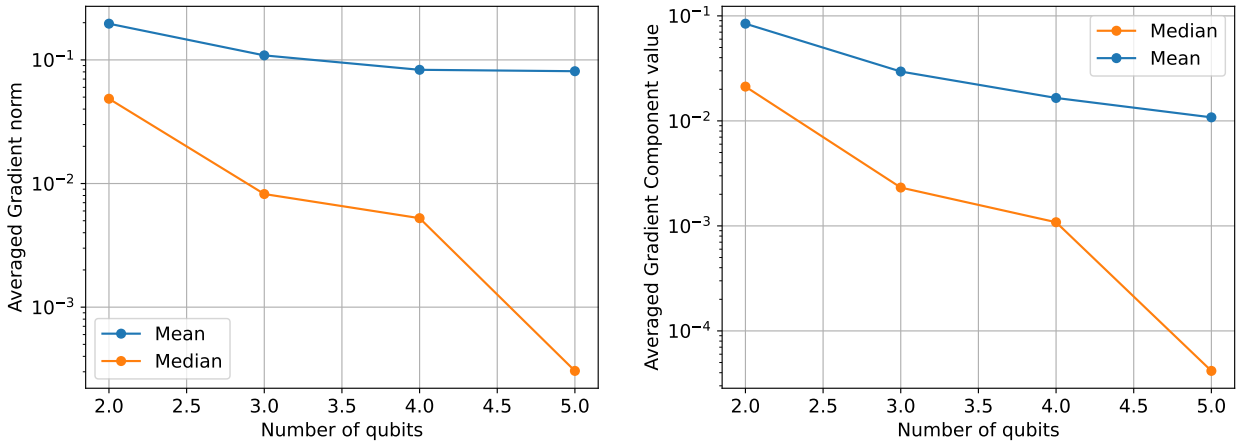
4.3 Implementation challenges

In general,three factors prevent convergence:

- **Numerical noise** on cost function evaluation, which depends on the number of shots
- **Vanishing gradients** for the cost function as the number of qubit rises. This problem is also reported in the literature for hardware efficient ansatzes [31]
- **Curse of dimensionality**: optimization generally becomes more difficult as the number of variables increases because the optimization domain gets progressively larger and harder to explore with each new variable

Partially, these phenomena can be evinced by results plots such as Figure 4.3a. For most test cases, increasing the number of shots allows for more accurate cost function evaluation, leading to easier convergence, as shown by the decreasing average cost at convergence. Furthermore, it is interesting to point out these gains are less prominent for higher qubit numbers.

The issue of vanishing gradient for this test case is illustrated in Figure 4.22, which shows the average norm of the gradient sampled at different points across the domain. Figure 4.22a shows average and median gradient decreasing markedly as a function of the number of qubits. Interestingly, this issue seems to onset at a relatively low number of qubits, threatening the scaling of VQLS for this application. To evaluate optimization difficulties, perhaps a better metric is the median value of the partial derivatives rather than the overall gradient norm, as represented in Figure 4.22b. This metric is more illustrative because each partial derivative is necessary to define the direction of the gradient vector, which is as crucial as its magnitude when optimizing. Unfortunately, partial derivatives vanish even faster with exponential ansatzes because the gradient dimensionality scales with 2^n. For this test case, random sampling found median values of partial derivatives for 3 and 4 qubits to be $\mathcal{O}(10^{-3})$, which could explain the difficulty in convergence.

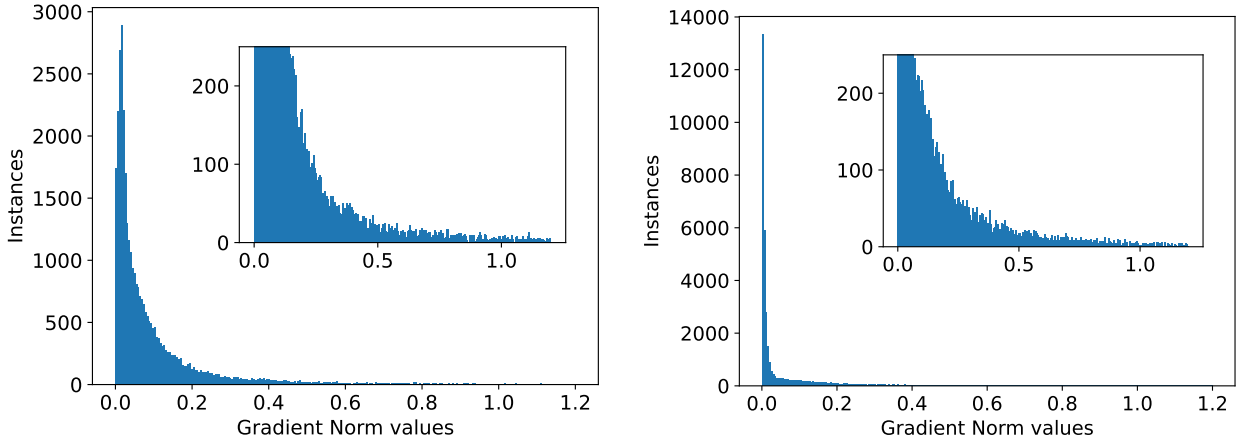


(a) Gradient norm

(b) Average value of partial derivatives

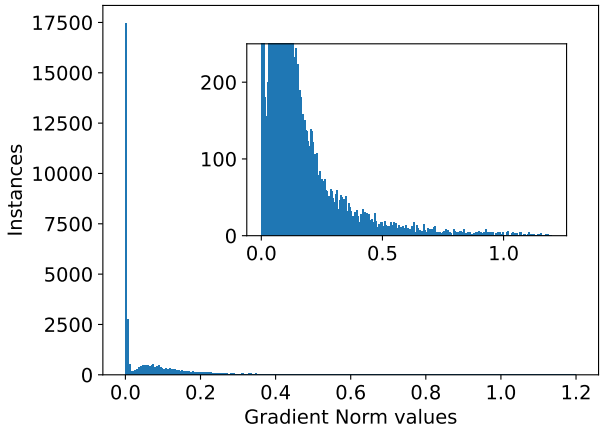
Figure 4.22: Average gradient norm and components value as obtained by random sampling of the optimization domain at $3.6 \cdot 10^4$ points with an exponentially deep ansatz

The discrepancy between mean and median in Figure 4.22 suggest the distribution of gradients values is not symmetrical. For example, it could be flat with steep, sudden wells. Moreover, a low average is not necessarily enough to prove large plateaus in the cost landscape because a cost function could be concave with a constant but slight gradient. Thus, to get a more accurate representation, Figure 4.23 shows occurrences of different gradient norms magnitude obtained with random sampling. These results appear to prove the gradient does indeed suffer from large plateaus as the lowest value bucket has the highest frequency. As the number of qubits rises, this problem is more marked: while for two qubits, gradient norm smoothly tapers off from smaller to higher values, the transition becomes increasingly shaper for higher n. For five qubits (Fig. 4.23d), almost all occurrences happen at the lowest value, while stronger gradients (possibly close to the solution point) occur rarely.

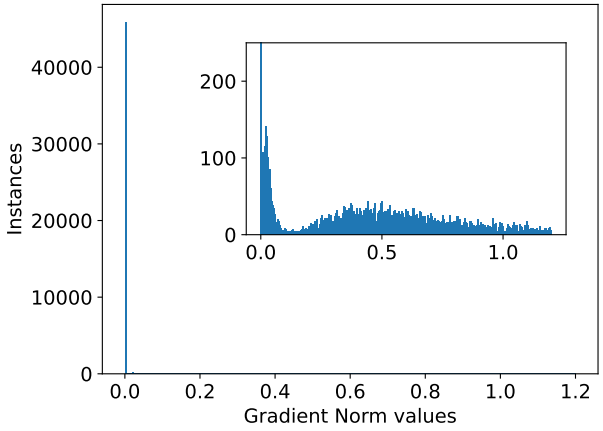


(a) 2 qubits

(b) 3 qubits



(c) 4 qubits

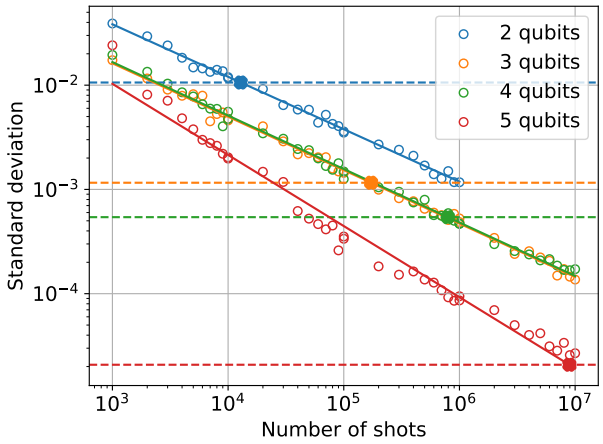


(d) 5 qubits

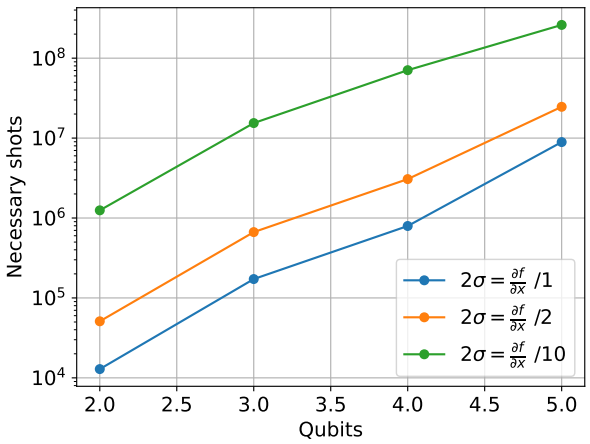
Figure 4.23: Gradient norm instances from random sampling of the cost function. Each figure is obtained sampling $3.6 \cdot 10^4$ points. ($5 \cdot 10^4$ points for 5 qubits)

Shot noise and average gradient

Finally, Figure 4.24 relates shot noise on the cost function with the median partial derivative across the domain. In particular, Figure 4.24a shows the shots required to ensure with 95% confidence that the sign of the partial derivative measured is correct. On the other hand, Figure 4.24b shows the scaling of the necessary shots as a function of qubits given a precision requirement. Three arbitrary precision levels are suggested, up to 95% confidence that the error is within 10% of the value of the median partial derivative. Noteworthy is is that, given a precision level, the number of shots required is already high at a low qubit number and scales exponentially with the number of qubits, quickly becoming unfeasible.



(a) Standard deviation as a function of the number of shots compared with double the median value of the partial derivative (dotted line). Intersection point determines at which number of shots 2σ is equivalent to the partial derivative median



(b) The number of shots necessary to meet the required level of precision as a function of qubit number. "Precision levels" are based on the accuracy one wants to achieve on a partial derivative, i.e. $2\sigma = \frac{\partial f}{\partial x}$ implies 95% confidence of estimating at least the sign of the gradient correctly

Figure 4.24: Relationship between shot noise and median partial derivative across the domain for a normalized cost function

Concluding remarks on implementation challenges

It is necessary to remark that these results are problem-specific. In particular, they could vary based on:

- Ansatz utilized: ansatzes that are not exponentially deep entail higher partial derivatives value for the same gradient magnitude
- Cost function structure: Other cost functions might suffer less plateau and therefore larger partial derivative
- Test-case specific: the cost landscape is expected to vary for different matrix and matrix decomposition
- Optimizer specific: 0-th order methods and noise specific optimizer might be able to cope with higher noise.

In particular, it is hard to judge a priori how much noise 0th-order optimization methods could tolerate. However, this problem is deemed relevant even if the optimizer does not directly measure the gradient because the cost itself plateaus. Nonetheless, it is suggested that future research carries out a similar analysis for other applications and different cost

function structures.

Finally, these results are for a quantum simulator: noise levels on current hardware are much larger than shot noise, and such precision levels are practically unfeasible. Thus, this analysis concludes that this test case is unlikely to succeed both on current hardware at a small scale and on future hardware at a larger scale.

Chapter 5

Conclusion and recommendations

Fluid dynamics is notoriously expensive to solve computationally, and while researchers made notable progress through simplified models of Navier Stokes equations, significant challenges remain. In this research, instead of acting on the equations themselves, it is attempted to speed up calculations by changing the computational medium from a classical to a quantum computer. Although quantum computers are still in their infancy and yet to achieve any results of practical use, it is theoretically proven that this technology could drastically outperform existing computers in some settings (for example [15]). Moreover, possible applications and algorithms are currently being investigated and discovered, contributing to a flourishing research branch. While many algorithms are decades away from implementation, a class of algorithms, called variational solvers, has been purposely designed to work with current hardware and achieve early results.

In this work, instead of trying to devise a new quantum algorithm for fluid dynamics, the feasibility of casting current discretization methods into existing quantum algorithms is investigated. In particular, different discretization techniques are used to transform differential equations into linear equations, and their feasibility is tested on the Variational Quantum Linear Solver (VQLS), a recently proposed quantum algorithm for solving linear systems of equation.

Matrix Decomposition Results

In particular, the studied test case is a 1D Laplacian with Dirichlet boundary conditions, discretized using finite element method with linear basis function (tri-diagonal) and hybrid finite element method. These matrices were decomposed into unitaries, an essential step to implement linear systems in quantum computers. The potential for near term applications was evaluated by finding single qubits gates decompositions (Pauli) of the aforementioned matrices.

In these settings, the hybrid finite element matrix is inefficiently decomposed with $\mathcal{O}(N^{m>1})$

terms, without a clear pattern that allows determining the gate sequence a priori. These hurdles are most likely caused by "irregularities" in the matrix structure, which needs additional dummy variables along the diagonal to match the closest 2^n. Using the same base, a trigonal is also inefficient and requires exactly $N = 2^n$ distinct unitaries and presents a clear pattern that allows a priori determination of the gate sequence for any N. Noteworthy is that the decomposition of a tridiagonal matrix can be extended to an arbitrary number of dimensions d (only with Dirichlet boundary conditions), and increases in efficiency while doing so, with $\mathcal{O}\left(d\sqrt[d]{N}\right)$ unitaries. Thus, solving for higher dimensions could prove advantageous.

Both cases are deemed unlikely to bring any quantum advantage. While a classical solver can find a solution in $\mathcal{O}(\kappa N)$ steps (κ is the condition number), this decomposition implies each step of the solution process will require at least $\mathcal{O}(N^2)$ distinct quantum circuit evaluations. Although these evaluations could be completely parallelized, this would require many parallel quantum processors outside the realm of near-term applications.

On the other hand, it is possible to efficiently decompose a tridiagonal matrix in $n + 3$ gates using multi-controlled quantum gates. Given the highly efficient decomposition, this method could achieve quantum advantage, although that would depend on scaling of optimization iterations and ansatz depth. Noteworthy is that using multi controlled gates allows extending this decomposition to other boundary conditions with a minor overhead. However, C^{m}NOT gates require more advanced hardware than currently available. Nonetheless, it is shown this decomposition requires a far shallower circuits compared to HHL, suggesting easier implementation.

To summarize, it was impossible to find a gate decomposition both efficient and with limited hardware requirements: a simple decomposition requires exponentially many unitaries, whereas an efficient one requires complex multi controlled gates.

Practical implementation of VQLS

To validate these decompositions, assess feasibility and find some preliminary performance estimates, VQLS was run using the high entanglement decomposition of the aforementioned tri-diagonal matrix. While this implementation is not yet feasible on current quantum hardware, this approach has been tested on a quantum simulator to assess the potential of future capabilities, given that simpler bases are unlikely to ever outperform classical computations. In general, numerical runs highlighted substantial difficulties in obtaining a meaningful result. While for $n = 2$ qubits VQLS successfully converged most cases, $n = 3$ was far less likely to succeed, and $n = 4$ never converged to meaningful results. In all cases, the total number of iteration was far larger compared to best classical methods. Thus, several strategies were attempted to ease optimization by acting on the ansatz, the cost function and the

optimization procedure.

Firstly, to reduce the problem's dimensionality, an ansatz with a single layer of R_y gate was attempted. This approach does indeed reduce optimization complexity and lead to lower iteration count. However, the quality of the solution deteriorates, showing a shallow (hardware efficient) ansatz is not capable of achieving satisfactory results for this test case and leads to errors hardly acceptable in typical fluid dynamics applications. Furthermore, gradual ansatz parameters optimization was attempted but resulted in no improvement in overall results, indicating high interconnection between ansatz parameters does not allow independent optimization. Overall, while an efficient matrix decomposition was found, ansatz optimization remains one of the main hurdles preventing successful implementation. If no problem-specific ansatz can be devised, optimization complexity and circuit depth increase exponentially, making a quantum advantage unlikely.

Secondly, a tangible improvement was detected when using a non normalized cost function with respect to a normalized one. Although this does not guarantee a solution when the cost is null, this can be easily verified and re-attempted until a solution is found. On the other hand, a linear combination of non-normalized and normalized cost functions did not bring any advantage compared to the normalized case. Thirdly,a brief comparison of classical optimizers (Powell and Cobyla) showed Cobyla requires far fewer iterations but achieves worse results and is more susceptible to numerical noise. Sensibility analysis showed incompatibility of numerical differentiation schemes because of numerical noise, making analytical gradient measurement indispensable if gradient-based methods are chosen.

To summarize, at the current state of the art, VQLS can solve simple FE problems for $n = 2$, but fails to scale to larger problems successfully. The main issues preventing a solution were large plateaus in the cost function, shot noise in cost evaluation and curse of dimensionality. In particular, random sampling of the domain revealed a worrisome exponential decline of gradient and median partial derivative value, which would explain the difficulties encountered in optimizing. When comparing noise with median partial derivative, it is shown numerically that for this test case, maintaining the same level of precision in median derivative measurement would require an exponential increase of the number of shots for increasing qubit number. In a real hardware setting, vanishing gradient is even more problematic as one has to add hardware noise to shot noise, which would make this test case quickly unfeasible as n scales up. However, these results are limited to this test case, and different cost function formulations might help avoid this issue.

5.1 Recommendations for future work

While this research identified several roadblocks to the practical implementation of VQLS for discretized differential equations, future research will be necessary to either make these results more conclusive or find alternative solutions.

Matrix Decomposition

When considering Pauli Decomposition of a tridiagonal matrix, although a rigorous pattern was inferred, a proof by induction would help to formalize the result that a tridiagonal matrix always require N unitaries. In addition, the central matrix element that is responsible for the mapping $011...1 \rightarrow 10...0$, was shown to require exactly $N/2$ Pauli in Figure 3.3 whereas just one unitary when $C^{n-1}NOT$ gates are used (see Fig. 3.5). One could speculate that, because all bits are flipped at once and only for this specific string, this type of binary mapping always requires either $n-1$ control or an exponential number of single-qubit gates. If a rigorous proof of this idea can be devised, it would imply that an efficient single-qubit gate decomposition does not exist for the matrix analyzed.

Secondly, given the methods explained in this research, it is relatively straightforward to devise an efficient decomposition (with high entanglement) of higher-order FE discretization schemes. In this case, it would be interesting to evaluate differences in cost function ease of optimization and if the increase in accuracy outweighs the increment in the number of unitaries in the matrix decomposition. Similarly, one could attempt to use a high entanglement decomposition on the hybrid discretization matrix to evaluate if this leads to a better-behaved cost function. Furthermore, increasing the number of unitaries while decreasing total iterations might be advantageous because unitary evaluations can be parallelized, whereas the optimization processes generally proceed sequentially.

Shot noise

While this research provided an analytical prediction of shots noise for $\langle\psi|\psi\rangle$, a similar prediction for $|\langle b|\psi\rangle|^2$, perhaps due to excessive approximation, does not match numerical data as well. Future studies could improve the accuracy of this estimation. More importantly, $\mathrm{Var}(C(\boldsymbol{\theta}))$ was only determined numerically. Based on the matrix decomposition and its coefficients, an analytical law would be highly beneficial to give a more general assessment to the problem of shot noise for variational algorithms.

Practical implementation of VQLS

All the results obtained suggest that, from an implementation point of view, a local cost function as proposed by [28] is the most logical next step following this research. The vanishing gradient in the cost function was the biggest obstacle to successful implementation, and Bravo Prieto et al. found this formulation helps alleviate this issue. Moreover, while different ansatz optimization techniques did not lead to significant improvements, using a non Normalized cost function did improve convergence, suggesting acting on the cost function is an effective lever. Similarly, Hamiltonian morphing as introduced in [29] could be an effective solution.

Furthermore, because the exponential scaling of the necessary number of shots is mostly driven by the gradient vanishing rather than a precision loss for higher qubits, evaluating a local cost function (or other different formulations) would be crucial to understanding if this problem persists.

In addition, if the ansatz is exponentially deep, it is reasonable to expect scaling to a high number of qubits (hence thousands of optimization variables) would require gradient measurement to converge to a solution successfully. Therefore, implementation and testing of pre-existing routines for analytical gradient measurement [28], especially to evaluated feasibility and resilience to noise, is a logical next step.

Finally, devising an efficient ansatz (i.e. precise but not exponentially deep) is a fundamental step for successful practical applications and scalability. However, while there are examples of efficient ansatzes in the literature, in the most general case, a solution for a discretized FE can have an arbitrary shape, which makes tailoring an ansatz very difficult since reducing the number of variables decrease its degrees of freedom.

Other prospects

While many aspects are still yet to be evaluated, the author's preliminary assessment is that approximating differential equations with classical discretization methods to solve them with VQLS does not seem a successful approach. In the literature, Lubash et al. [53, 54] showed a variational circuit capable of solving differential equations that rely on specific quantum routines to approximate a derivative operator. Therefore, the author suggests comparing these methods to the results presented in this research when planning future studies.

Appendix A

Quantum computing fundamentals and theoretical backgroud

This appendix briefly explains some fundamental concepts of quantum computing and is mostly based on [24] where a more in-depth discussion can be consulted.

A.1 Fundamentals

In quantum computing a commonly used notation is Dirac notation or bra-ket notation. In simple terms, a ket $|x\rangle$ is a column vector and bra $\langle x|$ a row vector such that:

$$|x\rangle = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}, \quad \langle x| = |x\rangle^\dagger = \begin{bmatrix} x_1^* & x_2^* & \dots & x_N^* \end{bmatrix} \tag{A.1}$$

where † indicates the conjugate transpose and * the complex conjugate. It directly follows that the matrix multiplication $\langle a|b\rangle$ is equivalent to a inner product between two vectors (resulting in a scalar) and $|a\rangle\langle b|$ the outer product (resulting in a matrix).

Qubits and superposition

Similarly to classical computation, a qubit (or quantum bit) is the fundamental unit of information. The key difference between a classical and quantum bit is that while a the first can only be either 1 or 0, a qubit can be in a state of "superposition" which means it is possible to form a linear combination of the two states. Therefore, the state of an arbitrary qubit $|\psi\rangle$ can be described as:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \alpha\begin{bmatrix}1\\0\end{bmatrix} + \beta\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}\alpha\\\beta\end{bmatrix} \tag{A.2}$$

where $|0\rangle$ and $|1\rangle$ are the computational basis states. In other words, if a qubit is in a superposition state it is somehow "both 0 and 1". However, once the qubit is measured, it collapses in either of the two states with probability $|\alpha|^2$ for 0 and $|\beta|^2$ for 1. Note that probabilities have to sum to 1, which means $|\alpha|^2 + |\beta|^2 = 1$.

Superposition is at the core of quantum computing and is the driver behind the surprising computing capabilities of this technology. This becomes evident once multiple qubits are considered: for example, let us compare two classical bits and two qubits. Since a classical bit can only be either 0 or 1, a two components vector is enough to represent all the possible states of the classical register 00, 01, 10, 11. However, because a qubit can exist in a superposition of 0 and 1, a vector representing the state of two qubits must have four components to accommodate a linear combination of all possible states.

$$|00\rangle = \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \quad |01\rangle = \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \quad |10\rangle = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \quad |11\rangle = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}. \tag{A.3}$$

In fact, describing a superposition of these 4 possible states requires 4 coefficients

$$|\psi\rangle = \alpha\,|00\rangle + \beta\,|01\rangle + \gamma\,|10\rangle + \delta\,|11\rangle = \begin{bmatrix}\alpha\\\beta\\\gamma\\\delta\end{bmatrix}. \tag{A.4}$$

It is clear the vector that describes a qubit register quickly scales up in the number of qubits: the state of three bits can be described by a vector of length three, whereas, since $2^3 = 8$ possible combinations exist for a three bits register, a quantum state can only be fully described by a vector of length 8, which would take into account the possible superposition of these states $(c_1\,|000\rangle + c_2\,|001\rangle + c_3\,|010\rangle + c_4\,|011\rangle + c_5\,|100\rangle + c_6\,|101\rangle + c_7\,|110\rangle + c_8\,|111\rangle)$.

Effectively, the state vector size doubles every time a qubit is added (because so does the number of bits combinations), explaining why a relatively small number of qubits can encode large registers. Formally, the state of a qubit register is computed by the tensor product of the qubits, so that $|00..0\rangle$ is a short notation for $|0\rangle \otimes |0\rangle \otimes \ldots \otimes |0\rangle$.

Quantum circuits and gates

The sequence of logical operations making up a quantum algorithm is usually visualized through quantum circuits, as shown in Figure A.1. The horizontal lines are "wires" along which the logical operations are performed sequentially and each wire corresponds to a qubits. In this example, qubits are initialized as $|0\rangle$ state (as shown in the left-hand side of the circuit) so that the overall initial state is $|00\rangle$.

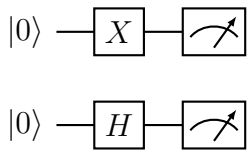


Figure A.1: Example of quantum circuit

Qubits are manipulated through quantum gates, represented as $\boxed{X}$ and $\boxed{H}$. These are unitary matrices (meaning their adjoint is also their inverse $UU^\dagger = I$) that alter the state of a qubit. In this example

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \text{and} \quad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{A.5}$$

The way these operations are performed is matrix multiplication. For example, X (sometimes called NOT) is responsible for flipping a qubit for $|0\rangle$ to $|1\rangle$ and vice versa:

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle. \tag{A.6}$$

similarly, H allows to create the aforementioned superposition state

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{A.7}$$

In this work, the convention used is that the top wire represents the rightmost bit of the tensor product defining the quantum state. Therefore, before measurement, the circuit in Figure A.1 results in the quantum state

$$|\psi\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ 1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}. \tag{A.8}$$

Measurements

In Figure A.1, $\boxed{}$ represents a measurement operation after which the superposition of a qubit is destroyed, and the qubit collapses in one of the two states of the basis used to measure (in the simplest case, either 0 or 1). Since superposition is destroyed by measurement, this operation cannot be carried out in the middle of the circuit but is always the last operation that allows for output readout.

Often, one measurement gives little information about the state of a qubit: it is necessary to repeat the circuit many times and the output of each experiment to obtain an

accurate measurement. For example, if the results of a series of experiments is 800 times 1 and 200 times 0, one can estimate the probabilities $P(1) = 800/1000 = 4/5 = |\beta|^2$ and $P(0) = 1/5 = |\alpha|^2$.

In Figure A.1 one would measure $P(1) = 1$ for the first wire because its state is $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$. On the other hand, the second qubit will have $P(0) = P(1) = 1/2$, so that the system as a whole will collapse with probabilities $P(10) = P(11) = 1/2$, which is in line with (A.8), (A.4).

Entanglement

Quantum logic gate can also be multi-qubit, meaning they affect more than one qubit. Figure A.2 shows a circuit with a Hadamard (single qubit gate mentioned before) and a CNOT (controlled not) which essentially is a multi-qubit gate that flips the target if and only if the control is 1.

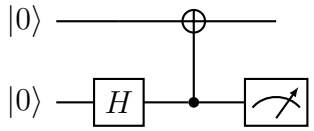


Figure A.2: Quantum circuit with entangled qubits

This means that a CNOT will map $|01\rangle \to |11\rangle$ and $|11\rangle \to |01\rangle$, but will keep unchanged $|00\rangle$, $|10\rangle$. Therefore, in Figure A.2:

$$|00\rangle \xrightarrow{H} \frac{|00\rangle + |10\rangle}{\sqrt{2}} \xrightarrow{\text{CNOT}} \frac{|00\rangle + |11\rangle}{\sqrt{2}}. \tag{A.9}$$

The final state of (A.9) is called Bell State and is a famous example of entangled qubits. What makes this state peculiar is that the outcome of one qubit will affect the other even if the other is left untouched. In other words, since the system can only be $|00\rangle$ or $|11\rangle$, if one of the qubit is measured and collapses in 0 the other must be also 0 and vice-versa. Note that what is so peculiar is not that the two qubits have the same measurement outcome, but that one affects the other even if the interaction happened "in the past": for instance, adding a X gate to the first qubit before measurement would imply that one qubit is 1 when the other is 0 and vice-versa, but the entanglement would be equally valid.

A.2 Theoretical background of variational quantum solvers

In this section some general concepts regarding the theoretical framework underlying variational solvers will be introduced, such as operators and measurements, Hamiltonian and the variational principle of quantum mechanics. This summary is largely based on a paper from McClean et al. [23] and a book from Feynman [55], which clearly explains most of these concepts.

Operators and measurements

In physics, an operator can be used to describe the evolution of a system: substantially, it maps a physical system from one state to another. As any pure quantum state can be represented by a normalized vector $|\psi\rangle$ in the Hilbert space, and a linear operator is represented by a matrix $\hat{A}$, this mapping is simply expressed as:

$$|\phi\rangle = \hat{A}\,|\psi\rangle \tag{A.10}$$

This might sound very abstract, but a simple example could be the rotation of the state vector. Moreover, a state vector can be decomposed as a linear combination of base states as:

$$|\psi\rangle = \sum_i C_i\,|i\rangle = \sum_i \langle i|\psi\rangle\,|i\rangle \tag{A.11}$$

where C_i are complex numbers, and $\sum_i |i\rangle$ form an orthonormal basis. Because $|\psi\rangle$ is normalized, $\sum_i C_i^2 = 1$. Note that as C_i^2 represents the probability of the state vector being in the basis state $|i\rangle$, it makes perfect sense that those coefficients sum up to unity.

Now, it is necessary to introduce the concept of expectation value for a random variable with finite outcomes as:

$$\langle A\rangle = \sum_i p_i a_i \tag{A.12}$$

where a_i are the different outcomes and p_i the different probabilities associated with them. In simple terms, the expectation value is just a weighted average of the possibles outcomes of a variable. One example could be that a_i correspond to some energy state and p_i the probability that the system is in that state.

Suppose we want to perform a measurement on the state vector $|\psi\rangle$. If one wants to know the expectation value of any of its proprieties, it is necessary to compute equation (A.12). In general, every a_i in (A.12) will correspond to a specific state $|i\rangle$. As introduced above, the probability of $|\psi\rangle$ being in the state $|i\rangle$ is given by the projection $\langle i|\psi\rangle$ squared. Thus, we can rewrite (A.12) as:

$$\langle A\rangle = \sum_i C_i^2 a_i = \sum_i C_i^* C_i a_i = \sum_i \langle\psi|i\rangle\,\langle i|\psi\rangle\,a_i \tag{A.13}$$

where $|i\rangle$ for a orthonormal basis and $*$ indicates the complex conjugate. Then, there is a matrix $\hat{A}$ for which

$$\hat{A}\,|i\rangle = a_i\,|i\rangle \tag{A.14}$$

that is to say the vectors $|i\rangle$ are its eigenvectors and a_i its eigenvalues. Remember that a_i is a scalar, so with a suitable reformulation we can write

$$\langle A\rangle = \langle\psi|\sum_i \hat{A}\,|i\rangle\,\langle i|\psi\rangle = \langle\psi|\,\hat{A}\sum_i |i\rangle\,\langle i|\psi\rangle$$

that can be simplified as:

$$\langle A \rangle = \langle \psi | \hat{A} | \psi \rangle \tag{A.15}$$

which was obtained using the completeness relation

$$\sum_i |i\rangle \langle i| = \mathbb{I} \tag{A.16}$$

To summarize, in (A.15) we proved that the expectation value of a propriety of a physical system can be computed as a multiplication of the state vector with a suitable matrix. In general, a quantity that can be measured (also called observable), can be expressed by an operator, and the result of that measurement is computed as in (A.15).

Hamiltonian

The Hamiltonian is a recurrent concept in classical mechanics and quantum physics. It is used to describe a physical system and it is generally associated with the total energy of the system. In quantum mechanics and many classical mechanics applications, the Hamiltonian is just the sum of the potential and kinetic energy of the particles composing the system. Moreover, it is possible to rewrite a linear system as the minimization of a Hamiltonian (see Section 2.3, (2.17)).

Variational principle of quantum mechanics

Let us consider a Hamiltonian M on a system Q and a quantum system S with n qubits. The Hamiltonian can be directly derived from a physical system or used to encode an optimization problem. Given any quantum state $|\psi\rangle$, the variational principle of quantum mechanics states that the expectation value of H is always larger than its smallest eigenvalue λ_1.

$$\langle H \rangle_{|\psi\rangle} = \langle \psi | H | \psi \rangle \geq \lambda_1 \tag{A.17}$$

Of course, $\langle H \rangle_{|\psi\rangle} = \lambda_1$ when $|\psi\rangle$ is the corresponding eigenvector of H. Before proceeding to the proof, it is interesting to point out why this matters: because the expectation value is always larger than the smallest eigenvalue, we know that it is sufficient to find $|\psi\rangle$ that minimize $\langle H \rangle_{|\psi\rangle}$ to find the corresponding eigenvector In summary, this result because it is the theoretical basis upon which variational solvers are based.

Equation (A.17) can be easily proven by recalling (A.15) and the start of its proof in (A.13):

$$\langle H \rangle_{|\psi\rangle} = \langle \psi | H | \psi \rangle = \sum_i C_i^2 \lambda_i \geq \lambda_1 \sum_i C_i^2 = \lambda_1 \tag{A.18}$$

where the fact that $\sum_i C_i^2 = 1$ has been used. Furthermore the inequality is justified by the fact that $\lambda_1 < \lambda_2 < ... < \lambda_n$ by definition, thus $\sum_{i=1}^{n} \lambda_i \geq n\lambda_1$.

Appendix B

Glossary of quantum gates

This appendix contains a brief explanation of the quantum logic gates mentioned in the text. For a more comprehensive review of the basics, please refer to [24].

Quantum gates are logic operations acting on qubits and are used as building blocks for quantum circuits. Any valid quantum gate is reversible and can described by an unitary matrix with respect to a computational basis. Quantum gates can be single or multi qubit depending on how many qubits they act on.

Well known quantum gates are Pauli Gates X, Y, Z, that for 2×2 matrix space when identity is added.

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

Each gate performs a specific manipulation on a qubit, for example X flips a qubit from $|0\rangle$ to $|1\rangle$ and vice versa:

$$X \otimes |0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle . \tag{B.2}$$

Another commonly used gate is Hadamard, which can casts a qubits into superposition state, for example $H|0\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$,

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{B.3}$$

A generalization of single qubit gates are rotation gates, which allow for an arbitrary rotation along one of the three axis of the Bloch Sphere.

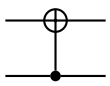
$$\begin{aligned} R_x(\theta) &= \exp(-iX\theta/2) = \begin{bmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{bmatrix} \\ R_y(\theta) &= \exp(-iY\theta/2) = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix} \\ R_z(\theta) &= \exp(-iZ\theta/2) = \begin{bmatrix} \exp(-i\theta/2) & 0 \\ 0 & \exp(i\theta/2) \end{bmatrix} \end{aligned} \tag{B.4}$$

In particular, R_y is often used in variational circuit ansatzes because it allows for an arbitrary rotation within the real numbers domain.

Other gates sometimes referenced in the literature are the continuously parametrized $U_1(\lambda), U_2(\lambda, \phi)$, and $U_3(\lambda, \phi, \theta)$, defined as:

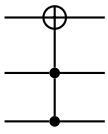
$$U_1(\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix}, \quad U_2(\lambda, \phi) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i(\lambda+\phi)} \end{pmatrix}, \quad U_3(\lambda, \phi, \theta) = \begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i(\lambda+\phi)}\cos(\theta/2) \end{pmatrix} \tag{B.5}$$

Another fundamental class of gates are controlled gates. These are acting on more than one qubit where some are used as control. Below is the example of a CNOT, or controlled X gate which acts as X on the target qubit (indicated by $\bigoplus$) when the control is $|1\rangle$.



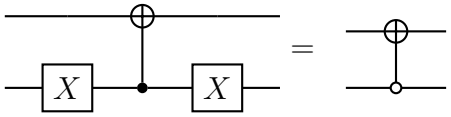
$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{B.6}$$

An example of multi controlled gate is the CCNOT, also known as Toffoli gate, which is essentially a CNOT with two controls



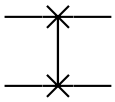
$$\text{CCNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{B.7}$$

Clearly, one can generalize a CCNOT to an arbitrary number of control qubits. This gate is named C^{m}NOT in this text, where m is the number of control qubits.
On the other hand, a CNOT with negative control is just a CNOT that acts when the control is $|0\rangle$ instead of $|1\rangle$:



$$\text{nCNOT} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.8}$$

Lastly, noteworthy is also the SWAP gates that, as the name suggests, swaps out the states of two qubits:



$$\text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.9}$$

Appendix C

Proof of convergence of Beta distribution to Gaussian

This section contains a simple proof that a symmetric beta distribution converges to the normal distribution for the limit of the number of sample $b \to \infty$.[1]

Let the random variable X be described by the beta $\beta(b, b)$ distribution with probability density function

$$f_X(x) = \frac{\Gamma(2b)x^{b-1}(1-x)^{b-1}}{\Gamma(b)\Gamma(b)} \quad 0 < x < 1$$

where b is a real, positive parameter half the sample size and Γ represents the gamma function. The mean of X is $E[X] = 1/2$ and the variance of X is $V[X] = 1/4(2b+1)$. Before taking the limit, it is convenient to subtract the mean and divide by the standard deviation. This is easily done with the transformation $Y = g(X) = 2\sqrt{2b+1}(X - 1/2)$, which is a one-to-one transformation from $\mathcal{A} = \{x \mid 0 < x < 1\}$ to $\mathcal{B} = \{y \mid -\sqrt{2b+1} < y < \sqrt{2b+1}\}$ with inverse $X = g^{-1}(Y) = Y/2\sqrt{2b+1} + 1/2$ and Jacobian

$$\frac{dX}{dY} = \frac{1}{2\sqrt{2b+1}}$$

Thus, the probability density function of Y is

$$\begin{aligned} f_Y(y) &= \frac{1}{2\sqrt{2b+1}} f_X\left(\frac{y}{2\sqrt{2b+1}} + \frac{1}{2}\right) \\ &= \frac{1}{2\sqrt{2b+1}} \cdot \frac{\Gamma(2b)}{\Gamma(b)^2}\left(\frac{1}{2} + \frac{y}{2\sqrt{2b+1}}\right)^{b-1}\left(\frac{1}{2} - \frac{y}{2\sqrt{2b+1}}\right)^{b-1} \\ &= \frac{1}{2\sqrt{2b+1}} \cdot \frac{\Gamma(2b)}{\Gamma(b)^2}\left(\frac{1}{4} - \frac{y^2}{4(2b+1)}\right)^{b-1} \quad -\sqrt{2b+1} < y < \sqrt{2b+1} \end{aligned}$$

[1] This proof is reported almost exactly as written in the lecture notes of Professor Robin Ryder (Ceredame at Université Paris Dauphine) and the author of this thesis takes no credit for this work, which is reported only for the sake of reader's understanding. Original notes are freely available at `http://www.math.wm.edu/~leemis/chart/UDR/PDFs/BetaNormal.pdf`

By applying Stirling's approximation of Gamma $\Gamma(z) = \sqrt{2\pi/z}(z/e)^z(1 + O(1/z))$ one obtains

$$\begin{aligned}
f_Y(y) &= \frac{1}{2\sqrt{2b+1}} \cdot \frac{\sqrt{\frac{2\pi}{2b}}\left(\frac{2b}{e}\right)^{2b}}{\frac{2\pi}{b}\left(\frac{b}{e}\right)^{2b}} \left(1 + O\left(\frac{1}{b}\right)\right)\left(\frac{1}{4} - \frac{y^2}{4(2b+1)}\right)^{b-1} \\
&= \frac{b}{\sqrt{2b}\sqrt{2b+1}} \cdot \frac{2^{2b}}{2\sqrt{2\pi}} \left(1 + O\left(\frac{1}{b}\right)\right)\left(\frac{1}{4} - \frac{y^2}{4(2b+1)}\right)^{b-1} \\
&= \frac{1}{\sqrt{2\pi}} 4^{b-1} \left(1 + O\left(\frac{1}{b}\right)\right)\left(\frac{1}{4} - \frac{y^2}{4(2b+1)}\right)^{b-1} \\
&= \frac{1}{\sqrt{2\pi}} \left(1 - \frac{y^2}{2b+1}\right)^{b-1} \left(1 + O\left(\frac{1}{b}\right)\right) \\
&= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{y^2}{2}\right)\left(1 + O\left(\frac{1}{b}\right)\right) \quad -\sqrt{2b+1} < y < \sqrt{2b+1}
\end{aligned}$$

Thus, in the limit $b \to \infty$, $f_Y(y)$ converges to the probability density function of a standard normal random variable.

Bibliography

[1] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M. Chow, and Jay M. Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246, 2017.

[2] Carlos Bravo-Prieto, Josep Lumbreras-Zarapico, Luca Tagliacozzo, and José I. Latorre. Scaling of variational quantum circuit depth for condensed matter systems. pages 1–12, 2020.

[3] Almudena Carrera Vázquez, Stefan Wörner, and Ralf Hiptmair. Quantum Algorithm for Solving Tri-Diagonal Linear Systems of Equations. 2018.

[4] Yael Ben-Haim Sergey Bravyi Lauren Capelluto Almudena Carrera Vazquez Jack Ceroni Richard Chen Albert Frisch Jay Gambetta Shelly Garion Leron Gil Salvador De La Puente Gonzalez Francis Harkins Takashi Imamichi David McKay Antonio Mezzacapo Zlatko Minev Ra Abraham Asfaw Luciano Bello. Learn Quantum Computation Using Qiskit, 2020.

[5] Joel H. Ferziger, Milovan Peric, and Anthony Leonard. Computational Methods for Fluid Dynamics. *Physics Today*, 50(3):80–84, mar 1997.

[6] P. R. Spalart. Strategies for turbulence modelling and simulations. *International Journal of Heat and Fluid Flow*, 21(3):252–263, 2000.

[7] Pierre Sagaut. *Large Eddy Simulation for Incompressible Flows. An Introduction*, volume 12. 2001.

[8] René Steijl and George N. Barakos. Parallel evaluation of quantum algorithms for computational fluid dynamics. *Computers and Fluids*, 173:22–28, 2018.

[9] René Steijl. Quantum Algorithms for Fluid Simulations. *Advances in Quantum Communication and Information*, pages 1–15, 2019.

[10] Guanglei Xu, Andrew J. Daley, Peyman Givi, and Rolando D. Somma. Turbulent mixing simulation via a quantum algorithm. *AIAA Journal*, 56(2):687–699, 2018.

[11] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics*, 22(5):563–591, 1980.

[12] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6-7):467–488, 1982.

[13] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G.S.L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[14] Edwin Pednault, John A. Gunnels, Giacomo Nannicini, Lior Horesh, and Robert Wisnieff. Leveraging Secondary Storage to Simulate Deep 54-qubit Sycamore Circuits. pages 1–39, 2019.

[15] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[16] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man Hong Yung, Xiao Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(May), 2014.

[17] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, 2017.

[18] Román Orús, Samuel Mugel, and Enrique Lizaso. Quantum computing for finance: Overview and prospects. *Reviews in Physics*, 4(January), 2019.

[19] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C. Benjamin, and Xiao Yuan. Quantum computational chemistry. *Reviews of Modern Physics*, 92(1):15003, 2020.

[20] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15):1–4, 2009.

[21] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2(July):79, 2018.

[22] N. Cody Jones, James D. Whitfield, Peter L. McMahon, Man Hong Yung, Rodney Van Meter, Alán Aspuru-Guzik, and Yoshihisa Yamamoto. Faster quantum chemistry simulation on fault-tolerant quantum computers. *New Journal of Physics*, 14, 2012.

[23] Jarrod R. McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2), 2016.

[24] Michael a. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. 2011.

[25] Andrew G. Taube and Rodney J. Bartlett. New perspectives on unitary coupled-cluster theory. *International Journal of Quantum Chemistry*, 106(15):3393–3401, 2006.

[26] Harper R. Grimsley, Sophia E. Economou, Edwin Barnes, and Nicholas J. Mayhall. An adaptive variational algorithm for exact molecular simulations on a quantum computer. *Nature Communications*, 10(1), dec 2019.

[27] Mateusz Ostaszewski, Edward Grant, and Marcello Benedetti. Quantum circuit structure learning. *arXiv*, pages 1–11, 2019.

[28] Carlos Bravo-prieto, Ryan Larose, M Cerezo, Yiğit Subaşı, Lukasz Cincio, and Patrick J Coles. Variational Quantum Linear Solver: A Hybrid Algorithm for Linear Systems. pages 1–16.

[29] Xiaosi Xu, Jinzhao Sun, Suguru Endo, Ying Li, Simon C. Benjamin, and Xiao Yuan. Variational algorithms for linear algebra. 2(2):1–10, 2019.

[30] Christopher M. Dawson and Michael A. Nielsen. The Solovay-Kitaev algorithm. *Quantum Information and Computation*, 6(1):081–095, 2006.

[31] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1):1–6, 2018.

[32] Sandu Popescu, Anthony J. Short, and Andreas Winter. Entanglement and the foundations of statistical mechanics. *Nature Physics*, 2(11):754–758, 2006.

[33] Yudong Cao, Jonathan Romero, Jonathan P. Olson, Matthias Degroote, Peter D. Johnson, Mária Kieferová, Ian D. Kivlichan, Tim Menke, Borja Peropadre, Nicolas P.D. Sawaya, Sukin Sim, Libor Veis, and Alán Aspuru-Guzik. Quantum Chemistry in the Age of Quantum Computing. *Chemical Reviews*, 119(19):10856–10915, 2019.

[34] Edward Grant, Leonard Wossnig, Mateusz Ostaszewski, and Marcello Benedetti. An initialization strategy for addressing barren plateaus in parametrized quantum circuits. *Quantum*, 3:214, 2019.

[35] Ryan LaRose, Arkin Tikku, Étude O'Neel-Judy, Lukasz Cincio, and Patrick J. Coles. Variational quantum state diagonalization. *npj Quantum Information*, 5(1), 2019.

[36] Tameem Albash and Daniel A. Lidar. Adiabatic quantum computation. *Reviews of Modern Physics*, 90(1):15002, 2018.

[37] Kosuke Mitarai and Keisuke Fujii. Methodology for replacing indirect measurements with direct measurements. *Physical Review Research*, 1(1):1–3, 2019.

[38] Miroslav Dobšíček, Göran Johansson, Vitaly Shumeiko, and Göran Wendin. Arbitrary accuracy iterative quantum phase estimation algorithm using a single ancillary qubit: A two-qubit benchmark. *Physical Review A - Atomic, Molecular, and Optical Physics*, 76(3):1–4, 2007.

[39] Enrico Cappanera. Quantum linear solvers for computational fluid dynamics: a literature review. *Literature Study, Delft University of Technology*, 2021.

[40] Panos Y. Papalambros and Douglass J. Wilde. *Principles of Optimal Design*. Cambridge University Press, jan 2017.

[41] Luis Miguel Rios and Nikolaos V. Sahinidis. Derivative-free optimization: A review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293, 2013.

[42] M J D Powell. A fast algorithm for nonlinearly constrained optimization calculations BT - Numerical Analysis. pages 144–157, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg.

[43] Mjd Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. *NA Report NA2009/06*, page 39, 2009.

[44] Aram Harrow and John Napp. Low-depth gradient measurements can improve convergence in variational hybrid quantum-classical algorithms. pages 1–45, 2019.

[45] Tyson Jones, Anna Brown, Ian Bush, and Simon C. Benjamin. QuEST and High Performance Simulation of Quantum Computers. *Scientific Reports*, 9(1):1–11, 2019.

[46] IMB Quantum. https://quantum-computing.ibm.com/, 2021.

[47] Andrew M. Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A. Spielman. Exponential algorithmic speedup by a quantum walk. *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 59–68, 2003.

[48] Daniele Boffi, Franco Brezzi, and Michel Fortin. *Mixed Finite Element Methods and Applications*, volume 44 of *Springer Series in Computational Mathematics*. Springer Berlin Heidelberg, Berlin, Heidelberg, sep 2013.

[49] Maurice George Kendall, Alan Stuart, and John Keith Ord. *Kendall's advanced theory of statistics*. London, 6th ed. edition, 1994.

[50] Rodney Coleman, N. L. Johnson, S. Kotz, and N. Balakrishnan. Continuous Univariate Distributions. *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, 159(2):349, 1996.

[51] Athanasios Papoulis. *Probability, Random Variables and Stochastic Processes*. McGraw-Hill Higher Education, fourth edition, 2017.

[52] Aidan Pellow-Jarman, Ilya Sinayskiy, Anban Pillay, and Francesco Petruccione. A comparison of various classical optimizers for a variational quantum linear solver. *Quantum Information Processing*, 20(6), 2021.

[53] Michael Lubasch, Jaewoo Joo, Pierre Moinier, Martin Kiffner, and Dieter Jaksch. Variational quantum algorithms for nonlinear problems. *Physical Review A*, 101(1):10301, 2020.

[54] Michael Lubasch, Jaewoo Joo, Pierre Moinier, Martin Kiffner, and Dieter Jaksch. Supplemental Material: Variational quantum algorithms for nonlinear problems. *arXiv*, 1(c):1–7, 2019.

[55] Feynman. *The Feynman Lectures on Physics, Vol. III: The New Millennium Edition: Quantum Mechanics*. 2011.