Studying Quantum Systems Using Variational Quantum Eigensolvers

Bendik Steinsvåg Dalen[1], Marius Hope[2], and Ruben Pariente[3,4]

[1]Department of Computer Science, Oslo Metropolitan University
[2]Department of Science and Industry Systems, University of South-Eastern Norway
[3]Department of Physics, University of Oslo, Norway
[4]SINTEF AS, Oslo, Norway

Abstract

In this work we investigate the physics of interacting Hamiltonian systems using a combination of exact diagonalization and Variational Quantum Eigensolver (VQE) techniques. We examine two-level and two-qubit systems to compare the exact diagonalization results with the VQE in reproducing ground state energies across various interaction strengths, in particular for the Lipkin model. The main goal is to investigate the capabilities and limitations of the VQE algorithm, particularly examining how the choice of ansatz, classical optimizer, and levels of shot noise impact convergence.

We found that the SPSA optimizer worked well in terms of stability, though other optimizers sometimes converged faster. The separable ansatz was sufficient for 1-qubit systems, but struggled for 2-qubit. The real amplitude and the hardware efficient ansatz both worked well for 2-qubit systems. The number of shots had a large impact on stability, depending on optimizer.

1 Introduction

Quantum computation is expected to provide a powerful platform for the simulation of quantum systems. Parallel to the scientific effort to realise large scale quantum computers, there is much interest in the development of novel quantum algorithms that can outperform classical algorithms. One such algorithm is the Variational Quantum Eigensolver (VQE), a quantum/classical hybrid algorithm for estimating the ground state of a quantum system. The algorithm uses a quantum computer to prepare a quantum state and measure its energy, and a classical optimization step to optimize the energy.

While quantum computers are still in an early phase when it comes to physical implementation, it is common to emulate quantum computers using classical computers. This can be useful both from an educational point of view, as well as for testing and development of new quantum algorithms. One such tool is provided by the open-source software development kit Qiskit [1].

In this work we analyze the performance of the VQE algorithm using the Qiskit package in Python. We apply the algorithm to the Lipkin model [2], which is a common model for benchmarking approximation methods in quantum physics [3, 4]. A secondary goal is to study the nature of interacting 2- and 4-level quantum systems for educational purposes.

This report is organised in the following way. In section 2.a we start with a general discussion of simple models for 2- and 4-level interacting systems. Section 2.c describes the Lipkin model and how it can be efficiently encoded on a quantum computer [3]. Then, section 2.d provides a short introduction to the VQE algorithm. Section 3 describes the methods that are applied in this work. The goal is to develop an understanding from the ground up, starting with basic quantum operations and leading up to the VQE algorithm. Section 4 is devoted to the main results. The VQE results are always compared with the exact results obtained from linear algebra. The results are then discussed both from a physical and an algorithmic point of view. Finally, section **??** concludes the report.

2 Theory

2.a General Models for 2- and 4-level Systems

As a preparation for the Lipkin model, we will briefly discuss some simpler models for interacting 2- and 4-level systems, and how they can be represented in a form that is suitable for quantum computation.

We assume that the interacting system can be described by a Hamiltonian on the form

$$\mathcal{H} = H_0 + \lambda H_I, \tag{1}$$

where H_0 is the Hamiltonian of the non-interacting system and H_I describes the interaction. The strength parameter λ is useful for controlling the strength of the interaction and thus studying its effect on the system.

2.a.i 2-level System

For a 2-level system, the terms in eq. 1 can be written on the form

$$H_0 = \begin{bmatrix} E_1 & 0 \\ 0 & E_2 \end{bmatrix} \quad \text{and} \quad H_I = \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix}, \tag{2}$$

where E_1 and E_2 are the eigenenergies of the noninteracting system and V_{ij} are interaction parameters.

In order to simulate the system on a quantum computer, the Hamiltonian should be rewritten in terms of Pauli matrices. The Pauli matrices can then be realised as elementary gates in a quantum circuit. It can easily be verified that the matrices in eq. 2 can be written in terms of the Pauli matrices as

$$H_0 = \frac{E_1 + E_2}{2} I + \frac{E_1 - E_2}{2} Z \tag{3}$$

$$H_I = \frac{V_{11} + V_{22}}{2} I + \frac{V_{11} - V_{22}}{2} Z + V_x X, \tag{4}$$

where we have set $V_x = V_{12} = V_{21}$ to ensure that the Hamiltonian is symmetric.

2.a.ii 4-level System

A general 4-level interacting system can be written on the form in eq. 1 with

$$H_0 = \begin{bmatrix} E_1 & 0 & 0 & 0 \\ 0 & E_2 & 0 & 0 \\ 0 & 0 & E_3 & 0 \\ 0 & 0 & 0 & E_4 \end{bmatrix} \quad \text{and} \quad H_I = \begin{bmatrix} V_{11} & V_{12} & V_{13} & V_{14} \\ V_{21} & V_{22} & V_{23} & V_{24} \\ V_{31} & V_{32} & V_{33} & V_{34} \\ V_{41} & V_{42} & V_{43} & V_{44}, \end{bmatrix} \tag{5}$$

where the values of V_{ij} are constrained by the requirement that H_I is a Hermitian operator. Written in terms of (tensor products of) Pauli matrices, it can be verified that H_0 can be rewritten as

$$H_0 = E_{II} I \otimes I + E_{ZI} Z \otimes I + E_{IZ} I \otimes Z + E_{ZZ} Z \otimes Z, \tag{6}$$

where

$$\begin{aligned} E_{II} &= \frac{E_1 + E_2 + E_3 + E_4}{4} \\ E_{ZI} &= \frac{E_1 + E_2 - E_3 - E_4}{4} \\ E_{IZ} &= \frac{E_1 - E_2 + E_3 - E_4}{4} \\ E_{ZZ} &= \frac{E_1 - E_2 - E_3 + E_4}{4}. \end{aligned}$$

The problem can be simplified greatly if we consider the situation where $V_x \equiv V_{14} = V_{23} = V_{32} = V_{41}$ and $V_z \equiv V_{11} = V_{44} = -V_{22} = -V_{33}$, and all the other V_{ij}'s are set to zero. In this case, it can be verified that H_I can be written as

$$H_I = \begin{bmatrix} V_z & 0 & 0 & V_x \\ 0 & -V_z & V_x & 0 \\ 0 & V_x & -V_z & 0 \\ V_x & 0 & 0 & V_z \end{bmatrix} = V_x X \otimes X + V_z Z \otimes Z. \tag{7}$$

Having written the Hamiltonian in terms of Pauli matrices, it is now in a suitable form for quantum computation.

2.b Von Neumann Entropy

A quantum state described by a state vector $|\Psi\rangle$ can also be described by the density matrix ρ defined as

$$\rho = |\Psi\rangle\langle\Psi|. \tag{8}$$

For a system composed of two subsystems A and B we can define the density matrix of subsystem A as the partial trace over subsystem B of the total density matrix ρ. In mathematical terms we write

$$\rho_A = \mathrm{Tr}_B(\rho). \tag{9}$$

The von Neumann entropy S of a quantum system described by a density matrix ρ is defined as

$$S(\rho) = -\mathrm{Tr}(\rho \ln(\rho)). \tag{10}$$

The entropy S is a measure of the degree of *mixing* in a quantum system, i.e. the degree of uncertainty about the state of the system. A *pure* state has entropy $S = 0$, while a mixed state has entropy $S > 0$.

For a system composed of two subsystems, an *entangled* state of the total system corresponds to a *mixed* state in each of the subsystems. Therefore, the entropy of a subsystem is a useful measure of the entanglement of the total system. We write the entropy of subsystem A as

$$S_A(\rho_A) = \mathrm{Tr}_A(\rho_A \ln(\rho_A)), \tag{11}$$

where ρ_A is defined in eq. 9.

2.c The Lipkin Model

The Lipkin model [2] was proposed in 1965 as a relatively simple model that exhibits non-trivial many-body effects [3]. It has served as a tool to benchmark various approximation techniques in many-body physics [3, 4].

The model describes N interacting fermions. Each fermion can be in one of two spin states with energy $\pm\epsilon/2$, and the interaction can lead to transitions between states. In second quantisation the Hamiltonian of the Lipkin model can be written as

$$\mathcal{H} = H_0 + H_1 + H_2, \tag{12}$$

where

$$\begin{aligned} H_0 &= \frac{1}{2}\epsilon \sum_{p\sigma} \sigma a^\dagger_{p\sigma} a_{p,\sigma}, \\ H_1 &= \frac{1}{2} V \sum_{p,p',\sigma} a^\dagger_{p\sigma} a^\dagger_{p'\sigma} a_{p'-\sigma} a_{p-\sigma}, \\ H_2 &= \frac{1}{2} W \sum_{p,p',\sigma} a^\dagger_{p\sigma} a^\dagger_{p'-\sigma} a_{p'\sigma} a_{p-\sigma}, \end{aligned} \tag{13}$$

where $a^\dagger$ and a are fermionic creation and annihilation operators, σ is a spin label, p, p' are particle labels, and ϵ, V and W are energy parameters. Looking at the fermionic operators, we see that H_0 is an energy associated with the number of particles in each spin state, H_1 moves two particles from spin state $-\sigma$ to spin state σ, and H_2 exchanges the spin state of two particles.

The Lipkin Hamiltonian can be rewritten in terms of quasispin operators in the following way

$$\begin{aligned} H_0 &= \epsilon J_z, \\ H_1 &= \frac{1}{2} V (J_+^2 + J_-^2), \\ H_2 &= \frac{1}{2} W (J_+ J_- + J_- J_+ - N). \end{aligned} \tag{14}$$

The quasispin operators obey the usual spin commuation relations: $[J_z, J_\pm] = \pm J_\pm$; $[J_+, J_-] = 2J_z$; $[J^2, J_\pm] = [J^2, J_z] = 0$. We also have $J_\pm = J_x \pm i J_y$ and $J^2 = J_x^2 + J_y^2 + J_z^2$. If the system contains N particles, the total quasispin operators are a sum over each particle,

$$J_z = \sum_i^N j_z^i; \quad J_\pm = \sum_i^N j_\pm^i, \tag{15}$$

where $j^i_{z,\pm}$ are quasispin operators for a single particle. For N particles the quasispin number is $J = N/2$, while the z-projection J_z takes on values from $-J$ to J with separation 1.

Going back to the Lipkin Hamiltonian, we see that the terms proportional to V changes the value of J_z with ± 2, while the terms proportional to ϵ and W leaves the value of J_z unchanged. Using the spin states $|J, J_z\rangle$ as basis states, the H_1 terms are off-diagonal, while H_0 and H_2 are on the diagonal. The following relations are used when calculating matrix elements:

$$J_z|J, J_z\rangle = J_z|J, J_z\rangle, \tag{16}$$

$$N|J, J_z\rangle = 2J|J, J_z\rangle, \tag{17}$$

$$J_+|J, J_z\rangle = \sqrt{J(J+1) - J_z(J_z+1)}|J, J_z+1\rangle \quad \text{and} \tag{18}$$

$$J_-|J, J_z\rangle = \sqrt{J(J+1) - J_z(J_z-1)}|J, J_z-1\rangle. \tag{19}$$

We will now look at the specific case of $N = 2$ and $N = 4$ particles.

2.c.i $N = 2$

For $N = 2$ the quasispin number is $J = 1$, and J_z takes the values $\{-1, 0, 1\}$. Since the only allowed change in spin projection is ± 2, there is no transition between $J_z = 0$ and $J_z = \pm 1$. In the basis $|J, J_z\rangle$ we have the nonzero matrix elements $\langle 1, \pm 1|J_z|1, \pm 1\rangle = \pm 1$, $\langle 1, \pm 1|J_\pm^2|1, \mp 1\rangle = 2$, $\langle 1, 0|J_\pm J_\mp|1, 0\rangle = 2$ and $\langle 1, 0|N|1, 0\rangle = 2$. The resulting Hamiltonian matrix is thus

$$\mathcal{H} = \begin{bmatrix} -\epsilon & 0 & V \\ 0 & W & 0 \\ V & 0 & \epsilon \end{bmatrix}. \tag{20}$$

Note that the sign convention on V is different from Ref. [3].

If we set $W = 0$, the problem reduces to a two-level problem that can be represented on a single qubit. We let $|1, -1\rangle \to |0\rangle$ and $|1, 1\rangle \to |1\rangle$, and get the Hamiltonian matrix

$$\mathcal{H} = \begin{bmatrix} -\epsilon & V \\ V & \epsilon \end{bmatrix}. \tag{21}$$

This can be represented in terms of the Pauli basis $\{I, X, Y, Z\}$ as

$$\mathcal{H} = -\epsilon Z + VX. \tag{22}$$

2.c.ii $N = 4$

A system with $N = 4$ particles corresponds to $J = 2$ and $J_z \in \{0, \pm 1, \pm 2\}$. The nonzero matrix elements are: $\langle 2, \pm 2|J_z|2, \pm 2\rangle = \pm 2$, $\langle 2, \pm 1|J_z|2, \pm 1\rangle = \pm 1$, $\langle 2, \pm 1|J_\pm^2|2, \mp 1\rangle = 6$, $\langle 2, 0|J_\pm^2|2, \mp 2\rangle = 2\sqrt{6}$, $\langle 2, \pm 2|J_\pm^2|2, 0\rangle = 2\sqrt{6}$,$\langle 2, 0|J_\pm J_\mp|2, 0\rangle = 6$,$\langle 2, \pm 1|J_\pm J_\mp|2, \pm 1\rangle = 6$ and $\langle 2, \pm 1|J_\mp J_\pm|2, \pm 1\rangle = 4$.

The resulting Hamiltonian matrix is thus

$$\mathcal{H} = \begin{bmatrix} -2\epsilon & 0 & \sqrt{6}V & 0 & 0 \\ 0 & -\epsilon + 3W & 0 & 3V & 0 \\ \sqrt{6}V & 0 & 4W & 0 & \sqrt{6}V \\ 0 & 3V & 0 & \epsilon + 3W & 0 \\ 0 & 0 & \sqrt{6}V & 0 & 2\epsilon \end{bmatrix}. \tag{23}$$

With a simple reordering of rows and columns, the Hamiltonian can be rewritten on the form

$$\mathcal{H} = \begin{bmatrix} -2\epsilon & \sqrt{6}V & 0 & 0 & 0 \\ \sqrt{6}V & 4W & \sqrt{6}V & 0 & 0 \\ 0 & \sqrt{6}V & 2\epsilon & 0 & 0 \\ 0 & 0 & 0 & -\epsilon + 3W & 3V \\ 0 & 0 & 0 & 3V & \epsilon + 3W \end{bmatrix}, \tag{24}$$

where we see that the top three states do not couple with the bottom two states. This reflects the fact that the only allowed transitions in the $|J, J_z\rangle$ basis involves $J_z \pm 2$, and hence the states $\{|2, 0\rangle, |2, \pm 2\rangle\}$ do not couple with the states $|2, \pm 1\rangle$. The two subsystems can therefore be treated independently.

The "subsystem" with *odd* values off J_z is effectively a two-level system that can be represented on a single qubit. We let $|2, -1\rangle \to |0\rangle$ and $|2, 1\rangle \to |1\rangle$, and the Hamiltonian becomes

$$H_o = \begin{bmatrix} -\epsilon + 3W & 3V \\ 3V & \epsilon + 3W \end{bmatrix} = 3WI - \epsilon Z + 3VX, \tag{25}$$

where Z and X are Pauli matrices.

The "subsystem" with *even* values of J_z is a three-level system that can be represented on 2 qubits. Since we use a four-level system to represent a three-level system, one of the basis states is unused. We let $|2, -2\rangle \to |00\rangle$, $|2, 0\rangle \to |01\rangle$ and $|2, 2\rangle \to |11\rangle$ and the Hamiltonian becomes

$$H_e = \begin{bmatrix} -2\epsilon & \sqrt{6}V & 0 & 0 \\ \sqrt{6}V & 4W & 0 & \sqrt{6}V \\ 0 & 0 & 0 & 0 \\ 0 & \sqrt{6}V & 0 & 2\epsilon \end{bmatrix} = WI_1I_2 - WZ_1Z_2 + (W-\epsilon)Z_1 - (W+\epsilon)Z_2 + \frac{\sqrt{6}V}{2}(X_2 + X_1 + Z_1X_2 - X_1Z_2), \tag{26}$$

where $Z_1 \equiv Z \otimes I$, $Z_2 \equiv I \otimes Z$ etc., and Z_1X_2 is the matrix product of Z_1 and X_2 [3]. The decomposition can be obtained using for the diagonal part the decomposition obtained in 2 while for the off diagonal terms we divide in two terms:

$$H_1 = \begin{bmatrix} 0 & \sqrt{6}V & 0 & 0 \\ \sqrt{6}V & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \sqrt{6}VI_1X_2 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \sqrt{6}VI_1X_2\left(\frac{I_1I_2 + Z_1I_2}{2}\right), \tag{27}$$

Noticing that the diagonal matrix is a projector on the states that stabilize Z_1. The second term can be rewritten as follows:

$$H_1 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{6}V \\ 0 & 0 & 0 & 0 \\ 0 & \sqrt{6}V & 0 & 0 \end{bmatrix} = \sqrt{6}VX \otimes \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = \sqrt{6}VX \otimes (\frac{I-Z}{2}), \tag{28}$$

In the result section we are going to restrict the analysis to the W=0 case where the Hamiltonian can be reduced to:

$$H_e = -\epsilon(Z_1 + Z_2) + \frac{\sqrt{6}V}{2}(X_2 + X_1 + Z_1X_2 - X_1Z_2), \tag{29}$$

2.d The VQE Algorithm

The Variational Quantum Eigensolver (VQE) algorithm was first introduced by Peruzzo et al. (2014) [5]. It is a variational hybrid algorithm, that aims to optimize some observable expectation value for a trial wavefunction for producing the ground state of a specific Hamiltonian.

The algorithm is defined hybrid as it combines both quantum and classical elements. An ansatz wavefunction, represented by a parametric circuit, is prepared on a quantum computer (QC) using various gates. Measurements are conducted to compute the expectation value of the Hamiltonian. The classical optimization component involves minimizing the Hamiltonian's expectation value by varying the parameters of the circuit. This process of Hamiltonian measurement and classical optimization iterates until the ground state is attained or other specified stopping criteria are satisfied.

2.d.i Preparing the Hamiltonian

Any Hamiltonian can be expressed as a linear combination of the tensor products of Pauli operators, i.e.

$$\mathcal{H} = \sum_{\alpha} h_\alpha P_\alpha + \sum_{\alpha\beta} h_{\alpha\beta} P_\alpha P_\beta + \dots \tag{30}$$

where the P's are the Pauli operators or the identity matrix, α and β identifies which Pauli operator, and the h's are real numbers. 6

We can then find the expectation value of $\mathcal{H}$ for a given state $|\Psi\rangle$

$$\langle \mathcal{H} \rangle = \langle \Psi | \mathcal{H} | \Psi \rangle = \sum_{\alpha} h_\alpha \langle \Psi | P_\alpha | \Psi \rangle + \sum_{\alpha\beta} h_{\alpha\beta} \langle \Psi | P_\alpha P_\beta | \Psi \rangle + \dots \tag{31}$$

Thus, estimating $\langle \mathcal{H} \rangle$ reduces to a linear combination of the expectation values of simple Pauli strings for any $|\Psi\rangle$.

How one can efficiently represent the wavefunction state on a quantum computer (QC) will depend on the Hamiltonian itself and the hardware limitations of the QC.

The circuit is run several times, and the number of times each state is measured is used to estimate expectation values.

2.d.ii The Ansatz

For a 1-qubit system, the wavefunction can be expressed as

$$|\psi\rangle = \begin{pmatrix} \cos(\theta/2) \\ e^{i\phi} \cdot \sin(\theta/2) \end{pmatrix}, \tag{32}$$

where θ and ϕ are angles which gives the position on the Bloch sphere. Furthermore, any qubit in a circuit will start in the $|0\rangle$ state. It needs to be rotated into the correct position, which can be done using the rotation gates R_x and R_y. We can then prepare the qubit for the circuit using

$$R_y(\phi)R_x(\theta)|0\rangle = |\psi\rangle . \tag{33}$$

For multi qubit systems, if we apply eq. 33 to each individual qubit, we will only be able to span separable Hilbert states. Thus we also need to entangle them by introducing several 2-qubit gates. The way this is done needs to be wide and adaptable enough for the ground state to be within the solution space, but not so shallow that the procedure halts in a barren plateau. Thus it is problem dependent.

To implement this on a circuit, we need to use a parametric unitary operator, which maps the initial guess $|\psi_0\rangle$ to one state on the Hilbert space that can properly approximate the ground state. The choice of ansatz becomes crucial for the procedure to properly converge into the ground state, and is typically related to the Hamiltonian. More details about the ansatz choice will be described in section 3.b.i.

2.d.iii Optimization

It is known that the eigenvector $|\Psi(\theta)\rangle$ with the lowest eigenvalue, the ground state, also minimises the Rayleigh–Ritz quotient

$$\frac{\langle\psi(\theta)|\mathcal{H}|\psi(\theta)\rangle}{\langle\psi(\theta)|\psi(\theta)\rangle} = E(\theta). \tag{34}$$

Thus we need to vary the parameters of $|\Psi(\theta)\rangle$ to find the lowest energy, using the quantum circuit as a subroutine. This is an optimization problem and can, e.g., be solved using optimization methods such as gradient descent, the Nelder-Mead algorithm, the Adam optimizer, or gradient free methods.

3 Methods

3.a Linear Algebra Methods

The ultimate goal of this work was to run the VQE algorithm using the *Qiskit* package for Python. To shed some light on the underlying methods, we first analyzed the Bell states using standard linear algebra with the *Numpy* package. We set up one- and two-qubit bases, together with elementary quantum gates, and a method for doing measurements. Comparisons are then made with the the output obtained with *Qiskit*. The details can be found in Appendix A.

Having gained familiarity with the basics of the Qiskit package, we moved on to study a general interacting two-level system, i.e. a system in the form of eq. 1 and 2. In order to evaluate the results of the VQE methods, we diagonalised the Hamiltonian using a standard eigenvalue solver in Numpy. This was done for different interaction strengths λ to study the effect of the interaction. The code is shown in Appendix B.

The eigenvalue solver methods were easily extended to the two-qubit system, i.e. eq. 6 and 7, and the Lipkin model, i.e. eq. 25 and 29. For completeness, the code is given in Appendix B.b and B.c respectively.

In order to estimate the entanglement of the general two-qubit system, we calculated the von Neumann entropy of one of the subsystems as a function of the parameter strength λ. The code for the entropy calculation is given in Appendix C.

3.b VQE Methods

In this sections we aim to describe the different VQE methods used for the different simulations. Our primary emphasis lies in describing the diverse ansatz selections utilized within the VQE framework, explicating their dfferences in the characterization of wavefunctions. Furthermore, we give a brief description of the different optimizers employed, outlining their principal differences. In Appendix D the Qiskit code for the VQE class used is reported and commented.

3.b.i Ansatzes

The right ansatz selection is crucial for obtaining a state that properly approximate the ground state. To accomplish this, it is important to carefully choose an ansatz that spans the proper Hilbert space regions containing the solution. In this subsection we want to describe the different types of ansatzes tested for the study of the Hamiltonians previously described.

3.b.i.1 Separable Ansatz The separable ansatz is a type of wavefunction which describes a generic separable and parametric N qubit state. The building blocks of this type of ansatz are the R_x and R_y parametric rotations. If we only apply these gates to an initially separable state, we will produce a parametrically separable state since no two-qubit gates are used to create entanglement between them. Its wavefunction is

$$\left|\psi(\vec{\theta})\right\rangle = (\prod_{i=1}^{p}(\otimes_j^N U_{rot}(\vec{\theta_{ij}})\left|0^{\otimes n}\right\rangle = \otimes_{j=1}^{N}(\prod_{i=1}^{p} R_x(\theta_{ij}^x)R_y(\theta_{ij}^y)\left|0\right\rangle), \tag{35}$$

where N is the number of qubits, $U_{rot}(\vec{\theta_{ij}})$ are unitary rotation gates R_x, R_y and the indices of the parameters θ_{ij} are for the layers and the qubits

An example of a circuit with a separable ansatz can be found in figure 1.

Figure 1: Example of a circuit with a separable ansatz, with $N = 3$ qubits and $p = 1$ layers.

We expect this ansatz to not work well for VQE in general, because the ground state of an Hamiltonian could be an entangled state of the Hilbert space.

3.b.i.2 Hardware Efficient Ansatz The Hardware-Efficient Ansatz (HEA) represents a specialized wavefunction ansatz tailored explicitly to fit the constraints of quantum computing hardware. This ansatz adopts a problem-agnostic parametric circuit structure, constructed through iterative incorporation of parameterized rotational gates and ladders of CNOT gates creating entangled states among distinct qubit subsets.

The ansatz wavefunction, in terms of the number of layers, can be expressed as

$$\left|\psi(\vec{\theta})\right\rangle = (\prod_{i=1}^{p}(\otimes_j^N U_{rot}(\vec{\theta_{ij}})U_{ent}\left|\psi_0\right\rangle = \sum_{i=0}^{2^N}\alpha_i(\vec{\theta})\left|i\right\rangle, \quad a_i \in \mathbb{C}, \tag{36}$$

where N is the number of qubits, p is the number of layers, U_{rot} and the parameters θ_{ij} are the same as for the separable ansatz and the U_{ent} is the unitary representation of the CNOT gates ladders matching the hardware geometry.

An example of a circuit with a hardware efficient ansatz can be found in figure 2.

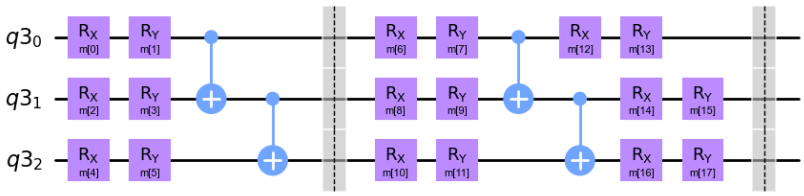


Figure 2: Example of a circuit with a hardware efficient ansatz, with $N = 3$ qubits and $p = 2$ layers.

The choice of the entanglement layers, characterized by the CNOT gates, can be chosen to fit the architecture of the quantum device we want to use. For this specific problem we decided to use the R_x and R_y rotations defined for the separable ansatz as gates for the rotation gates $U_{rot}(\theta ij) = R_x(\theta_{ij}^x)R_y(\theta_{ij}^y)$. The entangling circuit has been chosen to be linear, since it is not needed to fit it to a specific quantum device.

3.b.i.3 Real Amplitude Ansatz The real amplitudes ansatz is a trial wave function used as the circuit ansatz for ground states with real amplitudes. The circuit consists of of alternating layers of R_y and CNOT, which only have real components in the matrix representation,

$$R_y(\theta) = \begin{pmatrix} cos(\theta/2) & -sin(\theta/2) \\ sin(\theta/2) & cos(\theta/2) \end{pmatrix}, \qquad CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{37}$$

For p layers, the trial wave function is

$$\left|\psi((\vec{\theta}))\right\rangle = (\prod_{i=1}^{p}(\otimes_j^N R_y(\theta_{ij}))U_{ent})\left|\psi_0\right\rangle = \sum_{i=0}^{2^N}\alpha_i(\vec{\theta})\left|i\right\rangle, \quad a_i \in \mathbb{R}, \tag{38}$$

where N is the number of qubits, p is the number of layers, the parameters θ_{ij} are similar to the separable ansatz case and the U_{ent} unitary operator represent the CNOT gates as in the Hardware Efficient Ansatz

An example of a circuit with a real amplitude ansatz can be found in figure 3.

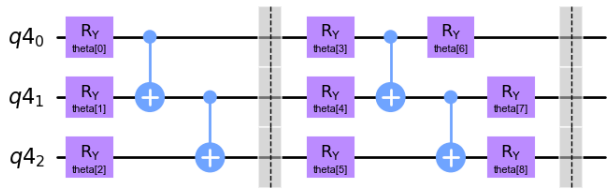


Figure 3: Example of a circuit with a real amplitudes ansatz, with $N = 3$ qubits and $p = 2$ layers.

3.b.ii Optimizers

The VQE is an optimization problem, with the goal of generating an approximation of a wavefunction heuristically, by iteratively learning ansatz parameters. To do so we need to choose a classical optimizer that can give a good approximation of the ground state in a finite number of steps and be stable under finite sampling and random initializations. We tested the performances for VQE of the following 4 different optimizers:

1. **Gradient Descent**: Gradient descent works by calculating the gradient of the objective function with respect to the parameters, and then updating the parameters in the direction of the negative gradient multiplied by a learning rate. This process continues until convergence is reached or a predefined stopping criterion is met.

2. **Adam (Adaptive Moment Estimation)**: Adam is an adaptive gradient optimization algorithm that adjusts the learning rates for each parameter individually, based on the first and second moments of the gradients. It maintains exponentially decaying moving averages of past gradients and squared gradients to adaptively scale the learning rates.

3. **COBYLA (Constrained Optimization BY Linear Approximations)**: COBYLA is a gradient-free optimization algorithm designed for minimizing a function subject to certain constraints. COBYLA is particularly useful for problems where the objective function is computationally expensive, or where gradients are not available or difficult to compute.

4. **SPSA (Simultaneous Perturbation Stochastic Approximation)**: SPSA is another gradient-free optimization algorithm that estimates the gradient of the objective function by using only cost function evaluations, rather than analytical gradients. It is well-suited for optimization problems with noisy measurements, high-dimensional parameter spaces, or when gradient information is not available or too costly to compute.

4 Results

4.a Two-Level System with Eigenvalue Solver

Figure 4 shows the exact energies and state probabilities as a function of the interaction strength λ for the general two-level system in eq. 1 and 2, calculated with the eigensolver from the Numpy package. There is an avoided level crossing at $\lambda = 2/3$. At this point, the ground state goes from being dominated by the basis state $|0\rangle$ to the basis state $|1\rangle$. The energies and state probabilities are plotted in the interval $\lambda \in [0, 4/3]$ for symmetry reasons.

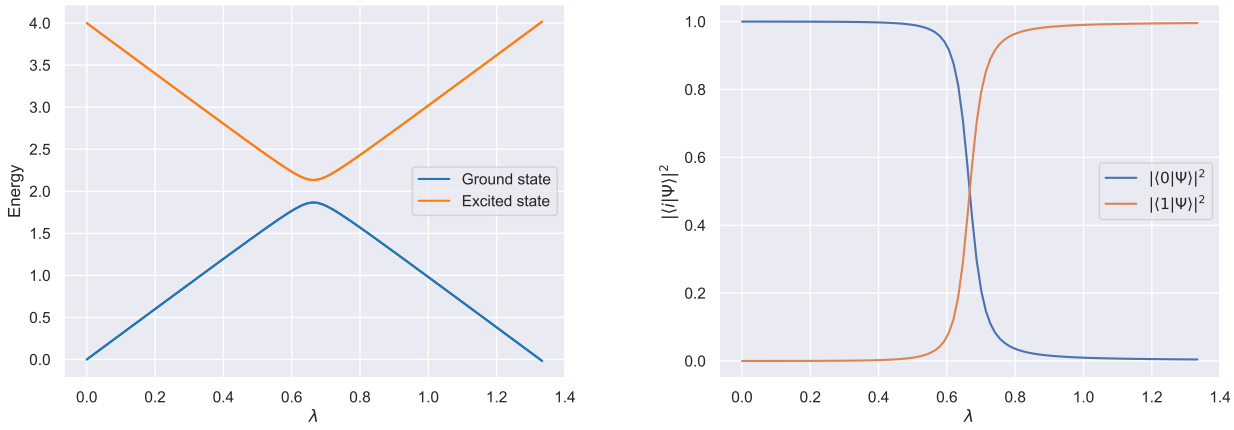


Figure 4: Left: Energy spectrum of eq. 1 as a function of λ. The values of the parameters in eq. 2 are $E_1 = 0$, $E_2 = 4$, $V_{11} = -V_{22} = 3$, $V_{12} = V_{21} = 0.2$. There is an avoided level crossing at $\lambda = 2/3$. Right: Basis state probabilities for the ground state as a function of λ. At weak interactions the ground state is dominated by $|0\rangle$. At the avoided level crossing, the state probabilities are equal, while strong interaction leads to a ground state dominated by $|1\rangle$.

4.b Two-Level System VQE

In this section, we aim to examine how the choice of optimizer and the number of measurement shots affect the performance of the Variational Quantum Eigensolver (VQE). We will compare the energy obtained at each step of the VQE process with the exact ground state previously calculated, as described in the methods section.

4.b.i Optimizers Analysis

To gain deeper insights into how the selection of classical optimizers may impact VQE performance, we conducted a comparative analysis of their efficacy relative to a consistent Hamiltonian problem, employing identical ansatz and initial parameters.

For this purpose we simulated the VQE, with exact expectation value calculations, for a two-level Hamiltonian. This was compared to the exact diagonalization result using the separable ansatz with one-qubit, with $\theta, \phi = \pi/2$. The results can be seen in Figure 5.

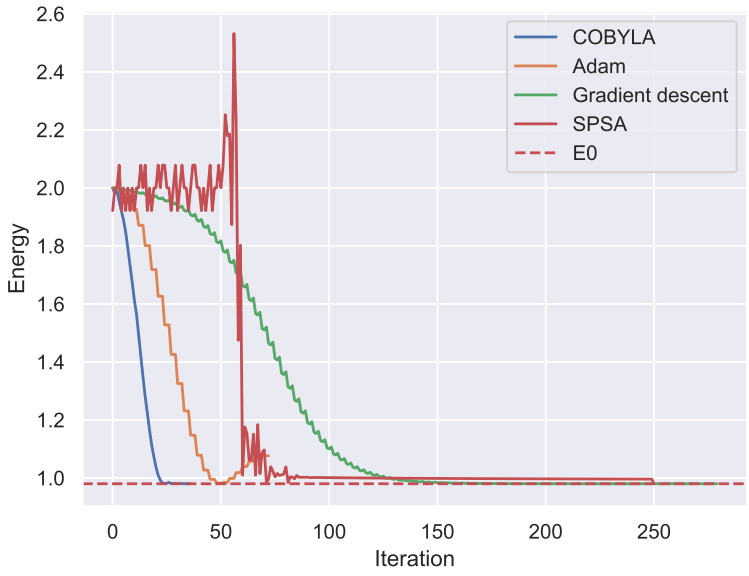


Figure 5: Comparison of VQE performances with respect to the ground state energy E_0, with different optimizers for the two-level system, with $E_1 = 0$, $E_2 = 4$, $V_{11} = -V_{22} = 3$, $V_{12} = V_{21} = 0.2$ and $\lambda = 1$

All four of the optimizers we examined demonstrate the ability to converge toward the exact ground state

energy E_0 within a finite number of iterative steps. The main difference lies in the number of evaluations of the cost function required for convergence. Both COBYLA and Adam exhibit efficient approximation of the ground state, reaching it in fewer than 50 iterations.

4.b.ii Shot Noise Analysis

When executing a VQE on an actual quantum device, numerous sources of noise can potentially degrade its performance. In the preceding section, we examined the impact of optimizer selection by calculating the expectation value exactly at each step. In this section, our main goal is to analyze how different optimizers perform at varying levels of shot noise. To accomplish this, we conducted VQE experiments on the same problem with different optimizers while varying the number of shots, and subsequently compared the results with their exact values.

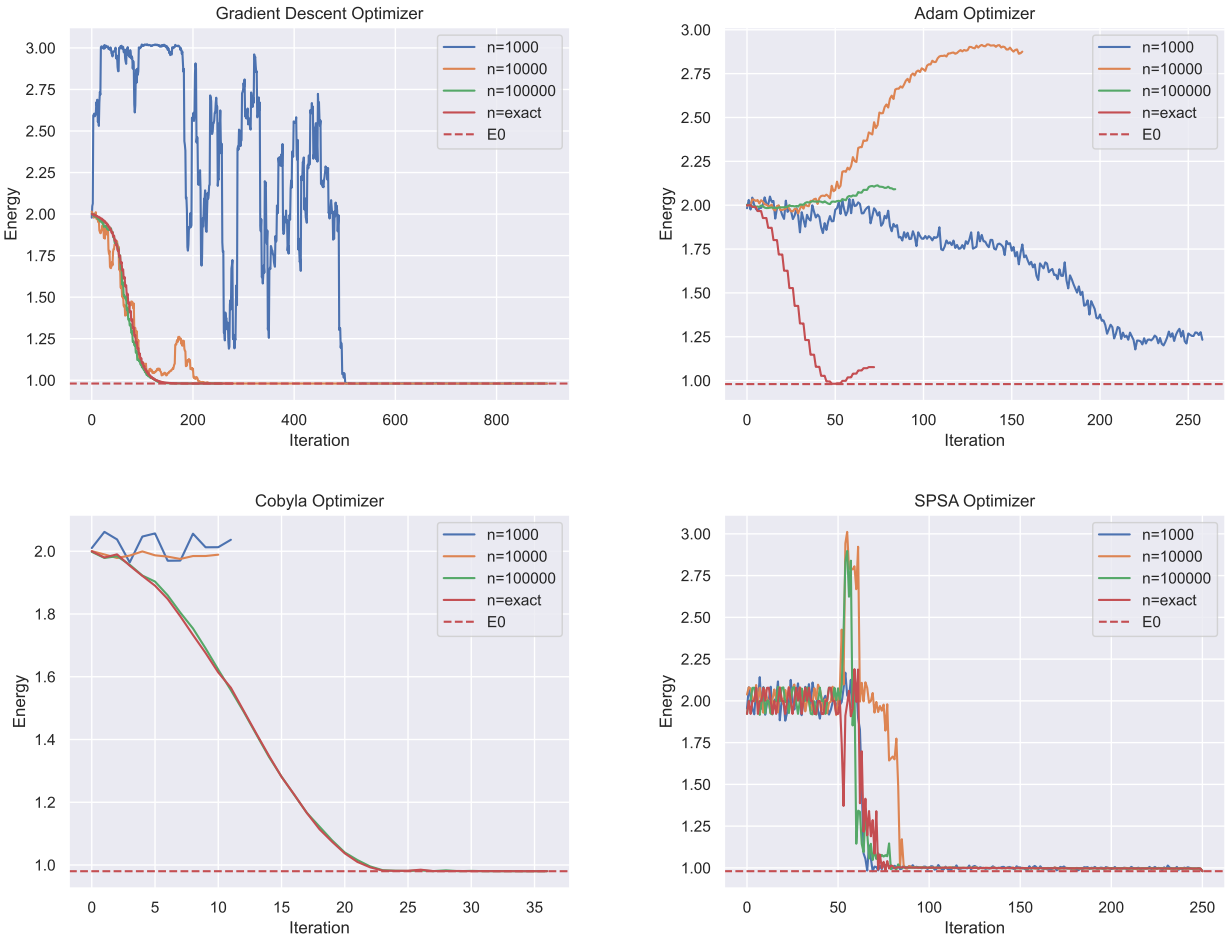


Figure 6: Comparison of VQE performance with respect to the shot noise with different optimizers. Here we used $E_1 = 0$, $E_2 = 4$, $V_{11} = -V_{22} = 3$, $V_{12} = V_{21} = 0.2$ and $\lambda = 1$.

As depicted in the four plots in Figure 6, shot noise can significantly affect the convergence of the VQE procedure, depending on the specific choice of optimizer. The SPSA optimizer, which does not rely on gradient calculations at each step, exhibits better performance compared to the others. In particular, it demonstrates high stability under shot noise, converging to the ground state even with as few as 1000 shots.

Conversely, the COBYLA optimizer appears to be unable to optimize with 1000 and 10 000 shots, struggling to discern near parameter points, although it converges smoothly with 100 000 shots.

Gradient descent seems to converge adequately with 10 000 shots. However, the impact of noise on optimization becomes apparent when considering the 1000 shots scenario.

The Adam optimizer seems to be the worst, not being able to converge for any chose of number of shots. A better analysis with respect to the shot noise should be conducted, with a proper analysis of varying the learning rate and the other hyperparameters.

For our purposes, we opted for the SPSA for the VQE with a finite number of shots calculations, because it seems to be the more stable.

4.b.iii VQE Energy Spectrum Calculations

In the following section we want to compare the VQE energy spectrum calculations with the exact calculations results obtained in fig. 4, while varying the interaction parameter $\lambda \in [0, 4/3]$. In Figure 7 we show the result of the VQE calculation for the ground and excited state. In the simple case of a 2 level systems Hamiltonian $\mathcal{H}$ we can express the excited state as the ground state of $-\mathcal{H}$, giving

$$E_1 = max \langle\psi| \mathcal{H} |\psi\rangle = -min \langle\psi| - \mathcal{H} |\psi\rangle . \tag{39}$$

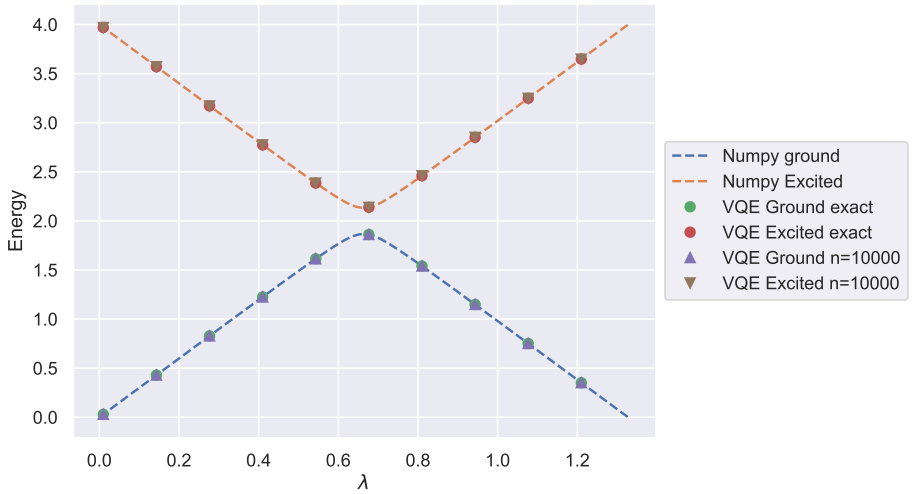


Figure 7: Comparison of the exact energy spectrum for different values of λ (continuous line) with the VQE performance (dots). This was done for both the exact calculations and with 10 000 shots, using the SPSA optimizer, and $E_1 = 0$, $E_2 = 4$, $V_{11} = -V_{22} = 3$ and $V_{12} = V_{21} = 0.2$.

As expected from the previous shot noise analysis we can see that we can reproduce the expected energy spectrum for every choice of λ using 10 000 shots and the SPSA optimizer.

4.c Two-Qubit System with Eigenvalue Solver

The left plot in Figure 8 shows the energy spectrum for the two-qubit system as a function of the interaction strength λ. There is a level crossing between the ground and first excited state at around $\lambda = 0.4$. The middle plot in Figure 8 shows the von Neumann entropy for one of the one-qubit subsystems in the ground state as a function of λ. There is a sharp increase in the entropy around the level crossing. The plot on the right hand side in Figure 8 shows the probabilities for each of the basis states in the ground state. For weak interactions the ground state is dominated by $|00\rangle$. At the level crossing the ground state becomes dominated by $|01\rangle$, and to a lesser degree $|10\rangle$. Stronger interactions increases the degree of mixing between $|01\rangle$ and $|10\rangle$.

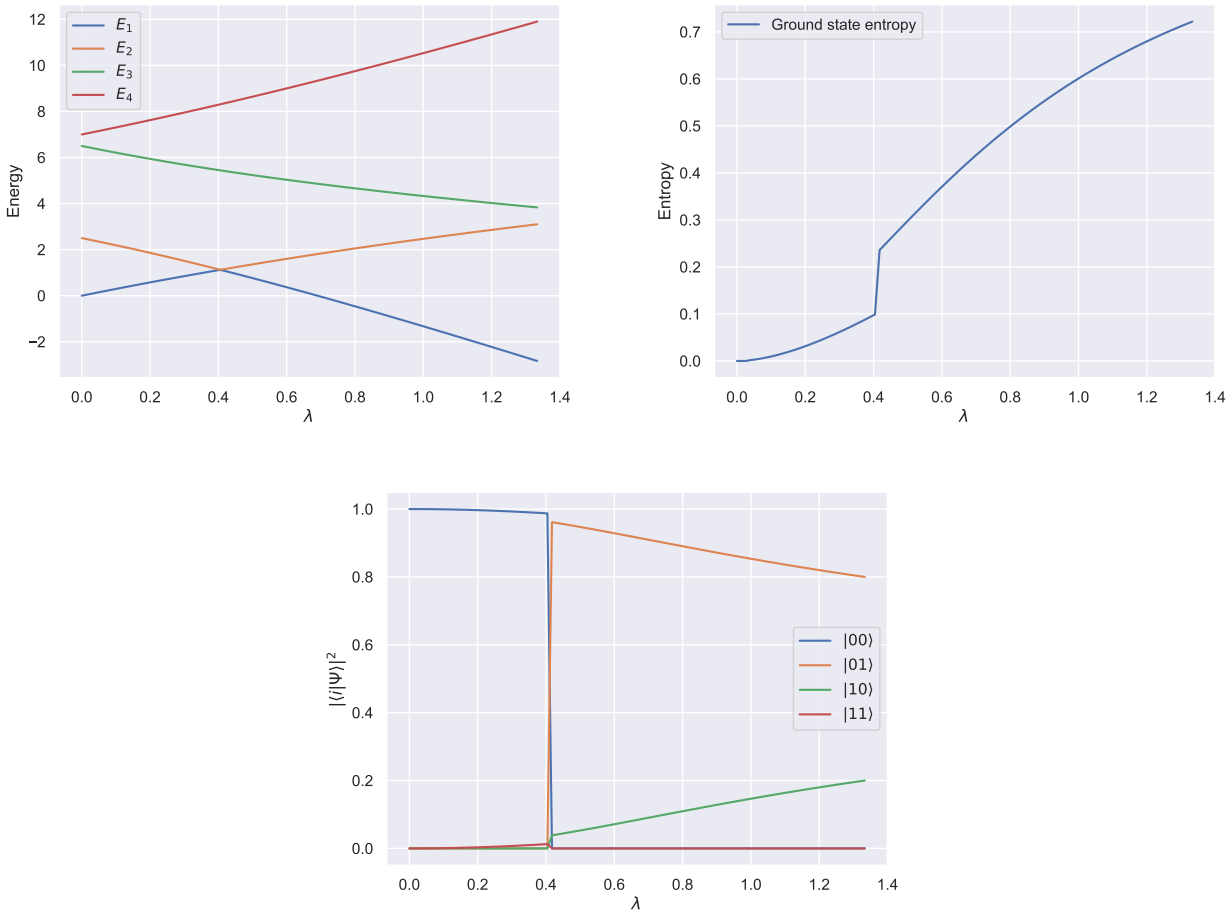


Figure 8: Left: energy spectrum of eq. 1 as a function of λ. The values used in eq. 6 and 7 are $E_1 = 0$, $E_2 = 2.5$, $E_3 = 6.5$, $E_4 = 7.0$, $V_x = 2.0$ and $V_z = 3.0$. Middle: Ground state entropy of one of the subsystems as a function of λ. Right: Basis state probabilities in the ground state as a function of λ.

4.d Two-Qubit System VQE

In this section, our goal is to evaluate the performance of VQE for a two-qubit system. In contrast to the one-qubit scenario with the Separable Ansatz, we can only describe a subset of the two-qubit Hilbert space. Therefore, in the initial section, we aim to understand how the selection of the ansatz wavefunction influences the calculations of the ground state.

4.d.i Ansatz Choice Analysis

To understand how the choice of the ansatz wavefunction influence the ground state calculations we compared three different types of ansatz on the same two-level Hamiltonian with the same optimizer and compare the results with the exact diagonalization results. To ensure a fair comparison, we restricted each ansatz to a single layer and plotted the VQE energy obtained at each optimization step.

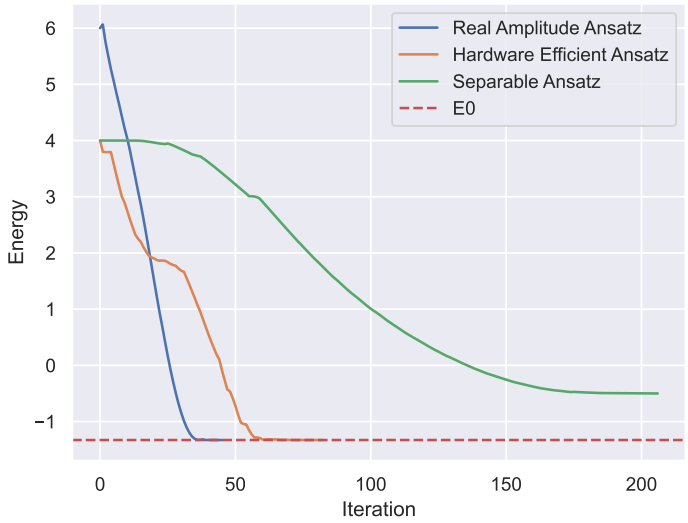


Figure 9: Comparison of the VQE performances with respect to the ground state E_0 for different Ansatz choices using the COBYLA optimizer. The values used in eq. 6 and 7 are $E_1 = 0$, $E_2 = 2.5$, $E_3 = 6.5$, $E_4 = 7.0$, $V_x = 2.0$, $V_z = 3.0$ and $\lambda = 1$.

As illustrated in the plot in Figure 9, both the real amplitude ansatz and the hardware efficient ansatz successfully converge to the ground state energy E_0. However, the separable ansatz becomes stuck at a higher energy value. This discrepancy arises from the inherent limitation of the separable ansatz, which can only describe separable two-qubit states. In contrast, the ground state for the chosen parameters is an entangled state.

The convergence of the real amplitude ansatz to the value E_0 suggests that the ground state in this interaction regime is a real linear combination of the computational basis states.

4.d.ii VQE Energy Spectrum Calculations

In the following section we report the VQE energy spectrum calculations, while varying the interaction parameter $\lambda \in [0, 4/3]$. This is compared to the state with the lowest eigenvalues in Figure 8. In all the simulations we used the SPSA optimizer with 10 000 shots.

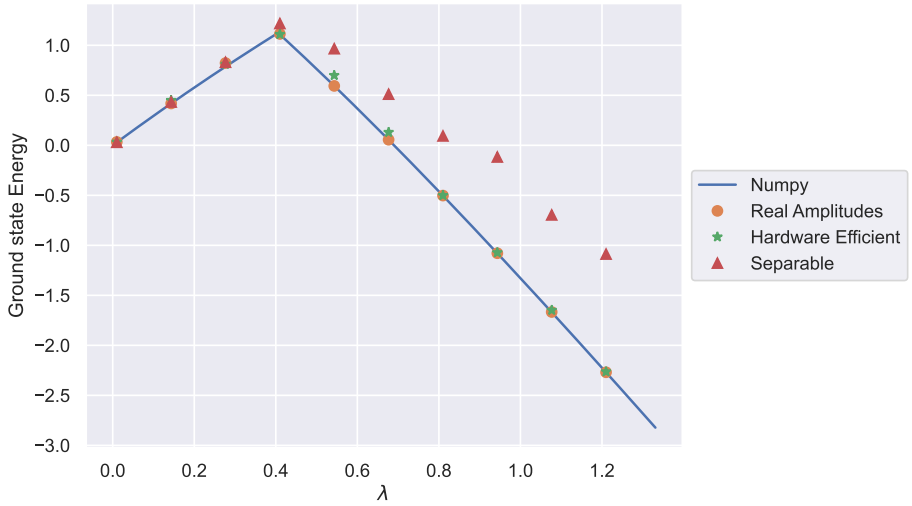


Figure 10: Comparison of the exact energy spectrum for different values of λ (continuous line) with the VQE performance (dots). This was done for several different ansatz, using the SPSA optimizer with 10 000 shots. The values used in eq. 6 and 7 are $E_1 = 0$, $E_2 = 2.5$, $E_3 = 6.5$, $E_4 = 7.0$, $V_x = 2.0$ and $V_z = 3.0$.

In the plot in Figure 10 we compare the energy spectrum calculations for the two-level Hamiltonian using the different ansatz for various $\lambda \in [0, 4/3]$. They are also compared with the exact diagonalization results. As we can see, the separable ansatz work well for low λ values where the entropy value is low and the states are quasi-separable. Increasing the value of λ results in higher entropies and entangled states and so the distance between

the estimated separable VQE energy and the exact ones increase.

Conversely, the real amplitude ansatz and the hardware efficient ansatz exhibit consistent performances across all regimes, with only a single outlier for the real amplitude ansatz, likely attributed to the finite number of shots employed in the calculation.

4.e Lipkin Model with Eigenvalue Solver

Figure 11 shows the energy spectrum of the Lipkin model with $N = 4$ particles, as a function of the interaction strength λ, calculated with the eigenvalue solver in Numpy.

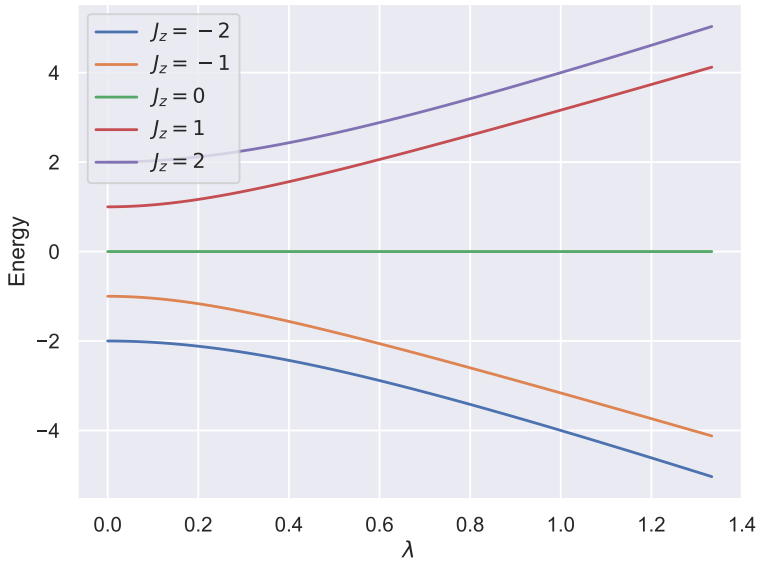


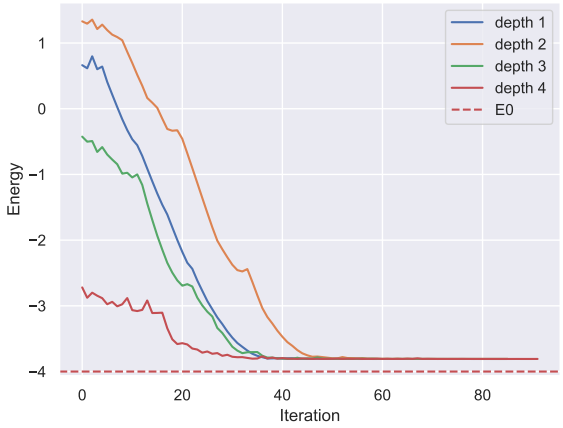
Figure 11: Exact calculation of the energy spectrum of the Lipkin model with $N = 4$ particles. The values used in eq. 25 and 29 are $\epsilon = V = 1$. The "subsystems" of odd and even value of J_z do not couple, and are solved separately.

4.f Lipkin Model VQE

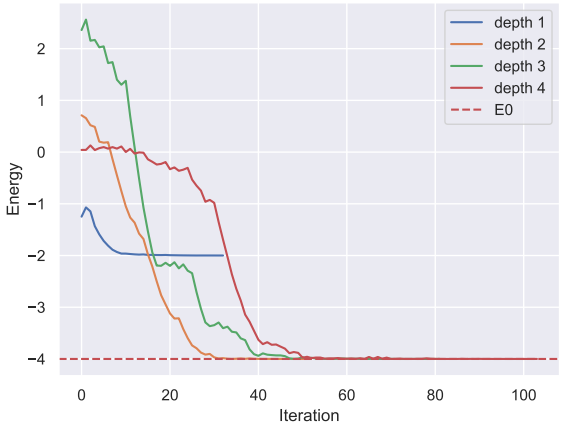
In this section, our aim is to analyze the performance of VQE for the N=4, J=2 Lipkin model, as described in Section 2.c.ii, and compare it with exact diagonalization results in the W=0 case. This model can be decomposed into two different Hamiltonians that can be expressed with 2 and 1 qubits respectively.

4.f.i Ansatz Depth Analysis

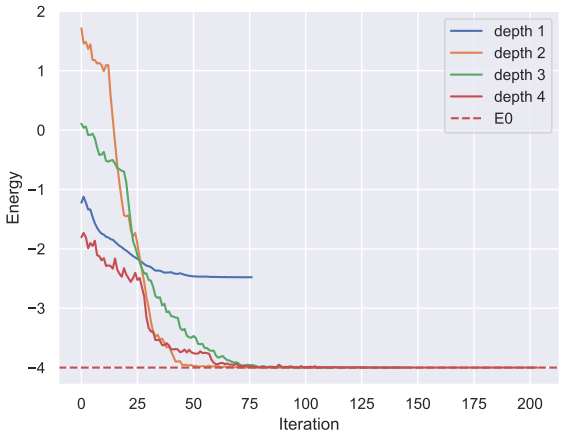
For the two-level Hamiltonian problem (section 4.d), we only needed one layer of the real amplitude ansatz or the hardware efficient ansatz to fully describe the ground state of the Hamiltonian. For the Lipkin model, as shows in the plots in Figure 12, more layers are needed to properly converge to the ground state energy E_0 for this specific choice of parameters.



(a) Separable ansatz



(b) Real amplitude ansatz



(c) Hardware efficient ansatz

Figure 12: Depth VQE performance analysis for different ansatz, using the COBYLA optimizer. The Hamiltonian studied is the $N = 4, J = 2$ one, with $E = 1, V = 1$ and $W = 0$.

We can see that adding more layers for the separable ansatz does not improve the estimated ground state energy. This can be attributed to the fact that only two rotations (such as R_x and R_y) on the Bloch sphere are necessary to describe a generic one-qubit state $|\psi\rangle$ starting from the initial state $|0\rangle$.

On the other hand, the real amplitude ansatz and the hardware efficient ansatz fail to accurately determine the energy value with just one layer. However, with two layers, both ansatz types are able to appropriately describe the ground state for this particular selection of parameters.

4.g VQE Energy Spectrum Calculations

In this section the VQE energy spectrum calculations are reported for the two sectors of the Lipkin model with $N = 4, J = 2, W = 0$. This is compared with the exact diagonalization results. The following results compare the different 2-qubit ansatzes previously studied in the even sector using 10 000 shots and the SPSA optimizer. For the odd sector, the Hamiltonian can be expressed as a one-qubit operator. Hence, we simulated the VQE with 10 000 shots and SPSA with the separable ansatz.

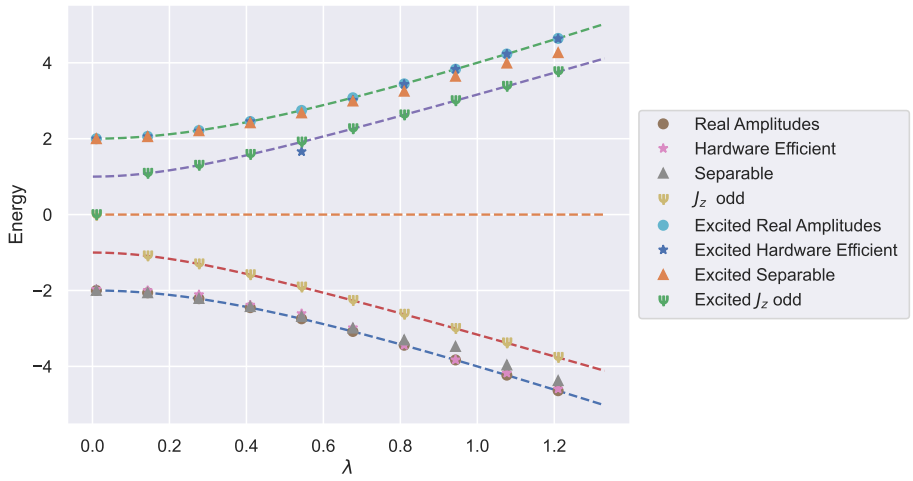


Figure 13: Ground and excited state VQE energy spectrum calculations for the odd and even sectors of the Lipkin model with $N = 4, J = 2, W = 0$, using 10 000 shots and the SPSA optimize. Compared with the exact diagonalization results.

From the plot in Figure 13 we can see that we can closely reproduce the ground state and the excited state for every value of λ, in the even section using either the real amplitude ansatz or the hardware efficient ansatz. As expected, the separable ansatz fails to properly describe the ground state and excited state as the interaction strength increases.

In the odd sector, as anticipated, the separable ansatz adequately captures both the ground state and the excited state. The outliers observed in the plot can be attributed to the optimization landscape affected by shot noise and the random parameter initialization employed in the simulations

5 Conclusions

In this work we have studied the physics of diverse interacting Hamiltonian systems through the combination of exact diagonalization and the Variational Quantum Eigensolver (VQE) algorithm.

For the two-level system, our exact diagonalization successfully described the energy spectrum varying the interaction parameter leading to avoided level crossings. The VQE simulations compared favourably to these results, which showcased the efficacy of different optimizers and ansatz choices in reproducing the ground state energies accurately. The shot noise analysis showed how the accuracy of Hamiltonian expectation value can deeply influence the convergence depending on the choice of the optimizer. Notably, the SPSA optimizer exhibited robustness against shot noise, being able to converge even for 1000 shots.

Extending our analysis to the two-qubit case has shown us the critical role of ansatz selection in accurately approximating ground states. From the exact calculations of the entropy, we successfully explained the increase in the deviance of the separable ansatz VQE result in the stronger interaction regime, with respect to the exact energies. Additionally, we demonstrated the capability of VQE in reproducing energy spectra across various interaction strengths using the real amplitude and the hardware efficient ansatz.

In the case of the Lipkin model, our simulations revealed sector-specific behaviors, with distinct ansatz requirements for even and odd sectors. We have successfully mapped the even and odd Hamiltonian subspaces to 2-qubit and 1-qubit Hamiltonians, and decomposed them in term of Pauli operators. Through the VQE simulations we have shown that the depth of the ansatz is an important parameter for properly describing the ground state wavefunction. Finally we successfully reproduced the ground and excited state energy spectra in every regime.

In conclusion, our study revealed the utility of VQE in ground state calculations for quantum systems, but it also showcased some common limitations. While our investigation has provided valuable insights, there remain further exploration to better understand the VQE capacities and limitations. Future studies can focus on understanding the VQE performance for bigger systems or less localized Hamiltonian. A deeper examination of optimizer performance under diverse sources of noise could enhance the understanding of VQE performance on real quantum devices. Moreover, the development of tailored ansatz designs for specific problems can improve the capabilities of VQE in solving a wider range of quantum systems.

References

[1] Qiskit contributors. Qiskit: An open-source framework for quantum computing, 2023.

[2] H. J. Lipkin, N. Meshkov, and A. J. Glick. Validity of many-body approximation methods for a solvable model: (I). Exact solutions and perturbation theory. *Nuclear Physics*, 62(2):188–198, February 1965.

[3] Manqoba Q. Hlatshwayo, Yinu Zhang, Herlik Wibowo, Ryan LaRose, Denis Lacroix, and Elena Litvinova. Simulating excited states of the Lipkin model on a quantum computer. *Physical Review C*, 106(2):024319, August 2022.

[4] Michael J. Cervia, A. B. Balantekin, S. N. Coppersmith, Calvin W. Johnson, Peter J. Love, C. Poole, K. Robbins, and M. Saffman. Lipkin model on a quantum computer. *Physical Review C*, 104(2):024305, August 2021.

[5] 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(1):4213, July 2014.

A Analyzing Bell states with Numpy and Qiskit

We want to analyze the Bell states. They are defined as

$$|\Phi_+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$
$$|\Phi_-\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}}$$
$$|\Psi_+\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}}$$
$$|\Psi_-\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}}$$

The goal is to compare the results from Qiskit with basic linear algebra with Numpy.

A.a Setting up Bell states with Numpy

The following code shows how to generate Bell states with Numpy, starting from one- and two-qubit basis states and elementary gates.

```
import numpy as np

# One-qubit basis
def qubit_0():
    return np.array([1, 0], dtype=np.complex128)
def qubit_1():
    return np.array([0, 1], dtype=np.complex128)

# Two-qubit basis
def qubit_00():
    return np.kron(qubit_0(), qubit_0())
def qubit_01():
    return np.kron(qubit_0(), qubit_1())
def qubit_10():
    return np.kron(qubit_1(), qubit_0())
def qubit_11():
    return np.kron(qubit_1(), qubit_1())

# One-qubit gates Identity and Hadamard
I = np.array([[1, 0], [0, 1]])
H = (1/np.sqrt(2))*np.array([[1, 1], [1, -1]])

# Two-qubit gates CNOT and Hadamard
CNOT= np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 0, 1], [0, 0, 1, 0]])
H1 = np.kron(H, I)
H2 = np.kron(I, H)

# Bell states
def bell_phi_plus():
    #Apply Hadamard and CNOT on |00>
    return CNOT@H1@qubit_00()
def bell_phi_minus():
    #Apply Hadamard and CNOT on |10>
    return CNOT@H1@qubit_10()
def bell_psi_plus():
    #Apply Hadamard and CNOT on |01>
    return CNOT@H1@qubit_01()
def bell_psi_minus():
    #Apply Hadamard and CNOT on |11>
    return CNOT@H1@qubit_11()
```

A.b Setting up Bell states with Qiskit

Next, we generate the Bell states with the Qiskit package. The states are represented as quantum circuits. The following code snippet shows how this can be done.

```
import qiskit as qk

n_qubits = 2
n_cbits = 2

def create_qc(n_q, n_c):
    qreg = qk.QuantumRegister(n_q)
    creg = qk.ClassicalRegister(n_c)
    return qk.QuantumCircuit(qreg,creg)

def qk_bell_phi_plus():
    qc = create_qc(n_qubits, n_cbits)
    #Apply Hadamard and CNOT on |00> (default)
    qc.h(0)
    qc.cx(0,1)
    return qc

def qk_bell_phi_minus():
    qc = create_qc(n_qubits, n_cbits)
    #Apply Hadamard and CNOT on |10>
    qc.x(0) # Flip first qubit, so we have |10>
    qc.h(0)
    qc.cx(0,1)
    return qc

def qk_bell_psi_plus():
    qc = create_qc(n_qubits, n_cbits)
    #Apply Hadamard and CNOT on |01>
    qc.x(1) # Flip second qubit, so we have |01>
    qc.h(0)
    qc.cx(0, 1)
    return qc

def qk_bell_psi_minus():
    qc = create_qc(n_qubits, n_cbits)
    #Apply Hadamard and CNOT on |11>
    qc.x(qc.qubits) # Flip both qubits, so we have |11>
    qc.h(0)
    qc.cx(0,1)
    return qc
```

A.c Measurements in Numpy

We can simulate a measurement by picking a random outcome. The possible outcomes are the basis states, and the probability distribution corresponds to the squared amplitude of the basis in the state vector. The following code shows how one can perform n_{shots} measurements with Numpy. The output is a dictionary with the number of occurrences of each outcome.

```
from collections import Counter

def measure(state, n_shots=1):
    probabilities = np.abs(state**2)
    outcomes = np.arange(len(state))

    # Perform 'measurement' n_shots times
    measured_outcomes = np.random.choice(outcomes, p=probabilities, size = n_shots)

    # Organise the results in a dict
    output = {}
    counts = Counter(measured_outcomes) # Aggregate outcomes in Counter object
    n_qubits = int(np.log2(len(state))) # Number of qubits to get the right binary format
```

```
    for k in counts.keys():
        output[format(k, f"0{n_qubits}b")] = counts[k]
    return output
```

A.d Measurements in Qiskit

The following code shows how one can perform a measurement (of all qubits in the circuit) with Qiskit. The function takes the quantum circuit as input. Then a measurement block is added to the circuit, and a simulation is run n_{shots} times. The output is a dictionary with the number of occurrences of each measurement outcome, similar to the output from the above measurement function.

```
import qiskit_aer
def qk_measure(qc, n_shots=1):
    qc.measure(qc.qubits, qc.clbits) # Add measurement to all qubits in circuit
    backend = qiskit_aer.Aer.get_backend('qasm_simulator') # Initialize backend
    job = backend.run(qc,shots=n_shots)
    return job.result().get_counts(qc)
```

A.e Comparison

We now want to compare the output from Numpy with the output from Qiskit. We measure the qubits in the Bell state $|\Phi_+\rangle$:

```
# Numpy
measure(bell_phi_plus(), 1000)
> Output: {'11': 518, '00': 482}

# Qiskit
qk_measure(qk_bell_phi_plus(), 1000)
> Output: {'11': 480, '00': 520}
```

We see that the average value of each basis state is not equal to the expectation value at 1000 shots, but by increasing the number of shots the average value will approach the expectation value.

As another example, we demonstrate how we can separate the states $|\Phi_+\rangle$ and $|\Phi_-\rangle$. It is not possible to distinguish the states by measuring them directly, as can be seen by the following code:

```
# Numpy
measure(bell_phi_plus(), 1000)
> Output: {'00': 495, '11': 505}
measure(bell_phi_minus(), 1000)
> Output: {'00': 509, '11': 491}

# Qiskit
qk_measure(qk_bell_phi_plus(), 1000)
> Output: {'11': 515, '00': 485}
qk_measure(qk_bell_phi_minus(), 1000)
> Output: {'00': 495, '11': 505}
```

Instead, we can apply the Hadamard gate to both qubits, in which case $H^{\otimes 2}|\Phi_+\rangle = |\Phi_+\rangle$ and $H^{\otimes 2}|\Phi_-\rangle = |\Psi_+\rangle$

```
# Numpy
measure(H1@H2@bell_phi_plus(), 1000)
> Output: {'11': 458, '00': 542}
measure(H1@H2@bell_phi_minus(), 1000)
> Output: {'10': 508, '01': 492}

# Qiskit
s1 = qk_bell_phi_plus()
s1.h(s1.qubits)
qk_measure(s1, 1000)
> Output: {'11': 489, '00': 511}
s2 = qk_bell_phi_minus()
s2.h(s2.qubits)
qk_measure(s2, 1000)
```

```
> Output: {'01': 488, '10': 512}
```

We see that the simulations behave as expected, and that the output is similar for the Numpy and Qiskit approach.

B Exact methods using eigenvalue solvers

B.a Two-level system

We solve the eigenvalue problem by using the method **numpy.linalg.eigh**, which takes a matrix as input and returns eigenvalues and eigenvectors. The following code shows two methods for setting up the Hamiltonian of the two-level system for a given set of parameters and solving the problem for a range of values of the interaction strength λ.

```
import numpy as np
# One-qubit gates
I = np.array([[1, 0], [0, 1]])
Z = np.array([[1, 0], [0, -1]])
X = np.array([[0, 1], [1, 0]])

def hamiltonian_general_tls(lam):
    # Basic params
    E1 = 0
    E2 = 4
    V11 = 3
    V22 = -V11
    Vx = 0.2

    #H_0 params
    H_0_I = (E1 + E2)/2
    H_0_Z= (E1 - E2)/2
    #H_I params
    H_I_I = (V11 + V22)/2
    H_I_Z = (V11-V22)/2
    H_I_X = Vx
    #H_0 and H_I
    H_0 = H_0_I*I + H_0_Z*Z
    H_I = H_I_I*I + H_I_Z*Z + H_I_X*X

    return H_0 + lam*H_I

def solve_tls_vs_lambda():
    #Array of different interaction strengths
    lam = np.linspace(0, 1.3, 100)
    #Arrays to hold eigenvecs and eigenvals
    eig_vals = np.zeros((2, len(lam)))
    eig_vecs = np.zeros((2, 2, len(lam)), dtype = np.complex128)
    for i in range(len(lam)):
        H = hamiltonian_general_tls(lam[i])
        val, vec = np.linalg.eigh(H)
        eig_vals[:, i] = val
        eig_vecs[:, :, i] = vec
    return lam, eig_vals, eig_vecs
```

B.b Two-qubit system

The code is easily adjusted for the two-qubit system.

```
import numpy as np
# One-qubit gates
I = np.array([[1, 0], [0, 1]])
Z = np.array([[1, 0], [0, -1]])
X = np.array([[0, 1], [1, 0]])
# Two-qubit gates
II = np.kron(I, I)
```

```
ZI = np.kron(Z, I)
IZ = np.kron(I, Z)
ZZ = np.kron(Z, Z)
XX = np.kron(X, X)

def hamiltonian_two_qubits(lam):
    #Basic params
    E1 = 0.0
    E2 = 2.5
    E3 = 6.5
    E4 = 7.0
    Vx = 2.0
    Vz = 3.0

    #H_0 params
    E_II = (E1 + E2 + E3 + E4)/4
    E_ZI = (E1 + E2 - E3 - E4)/4
    E_IZ = (E1 - E2 + E3 - E4)/4
    E_ZZ = (E1 - E2 - E3 + E4)/4
    #H_0 and H_I
    H_0 = E_II*II + E_ZI*ZI + E_IZ*IZ + E_ZZ*ZZ
    H_I = Vx*XX + Vz*ZZ

    return H_0 + lam*H_I

def solve_two_qubit_vs_lambda(lam_max=1):
    #Array of different interaction strengths
    lam = np.linspace(0, lam_max, 100)
    #Arrays to hold eigenvecs and eigenvals
    eig_vals = np.zeros((4, len(lam)))
    eig_vecs = np.zeros((4, 4, len(lam)), dtype = np.complex128)
    for i in range(len(lam)):
        H = hamiltonian_two_qubits(lam[i])
        val, vec = np.linalg.eigh(H)
        eig_vals[:, i] = val
        eig_vecs[:, :, i] = vec
    return lam, eig_vals, eig_vecs
```

B.c Lipkin model

The following code solves the Lipkin model with the eigenvalue solver method, with $\epsilon = 1$.

```
import numpy as np

def H_e(V):
    return np.array([[-2, np.sqrt(6)*V, 0, 0], [np.sqrt(6)*V, 0, 0, np.sqrt(6)*V], [0, 0, 0, 0], [0,
        np.sqrt(6)*V, 0, 2]])
def H_o(V):
    return np.array([[-1, 3*V], [3*V, 1]])

def solve_lipkin_energies_vs_lambda(lam_max=1):
    #Array of different interaction strengths
    lam = np.linspace(0, lam_max, 100)
    #Arrays to hold eigenvals for H_e and H_o
    eig_vals_e = np.zeros((4, len(lam)))
    eig_vals_o = np.zeros((2, len(lam)))
    for i in range(len(lam)):
        val_e = np.linalg.eigvalsh(H_e(lam[i]))
        eig_vals_e[:, i] = val_e
        val_o = np.linalg.eigvalsh(H_o(lam[i]))
        eig_vals_o[:, i] = val_o

    return lam, eig_vals_e, eig_vals_o
```

C Calculation of von Neumann Entropy

The following code contains methods to calculate the von Neumann entropy of a subsystem of the two-qubit system.

```
import numpy as np

def rho_vs_lambda(state_vs_lambda):
    #Initialize matrix to hold density matrix for each lambda.
    #Density matrix is 4x4 for the two-qubit system
    rho = np.zeros((4, 4, len(state_vs_lambda[0, :])), dtype = np.complex128)
    for i in range(len(state_vs_lambda[0, :])):
        #Density matrix is outer product of state |psi>, i.e. rho = |psi><psi|.
        #Second factor is bra-vector, i.e. conjugated.
        rho[:, :, i] = np.outer(state_vs_lambda[:, i], state_vs_lambda[:, i].conj())
    return rho

def get_rho_A(rho):
    #Calculate reduced density matrix, like rho_A = tr_B(rho).
    b0 = np.kron(I, qubit_0())
    b1 = np.kron(I, qubit_1())
    return b0.conj()@rho@b0.T + b1.conj()@rho@b1.T

def entropy_vs_lambda(rho_vs_lambda):
    #Initialize 1D vector to hold entropy value at different lambda
    entropy = np.zeros(len(rho_vs_lambda[0,0, :]), dtype = np.complex128)
    for i in range(len(rho_vs_lambda[0,0, :])):
        #Get the reduced density matrix of subsystem A
        rho_A = get_rho_A(rho_vs_lambda[:, :, i])
        #Get eigenvalues of rho_A
        vals = np.linalg.eigvalsh(rho_A)
        #Check that value is not 0 before adding to entropy, since log(0) is not defined.
        for val in vals:
            if val > 0.0001:
                entropy[i] -= val*np.log2(val)
    return entropy
```

The entropy can be calculated by first solving the system with the function **solve_two_qubit_vs_lambda** from appendix B.b, before creating the density matrix and calculating the entropy vs lambda, as is shown in the following code snippet.

```
lam, eig_vals, eig_vecs = solve_two_qubit_vs_lambda(4/3)
rho = calculate_rho_vs_lambda(eig_vecs[:, 0, :])
entropy= entropy_vs_lambda(rho)
```

D VQE Methods

The following code is the customize qiskit class we have used to every simulation. The class is defined to be applied to a generic VQE problem and is characterize by the following attributes:

- **Estimator**: A base Qiskit class used to estimate expectation values of quantum circuits and observables (Estimator or Sampler).
- **Circuit**: A parameterized quantum circuit serving as the ansatz for the problem.
- **Optimizer**: Either a customized or a SciPy classical optimizer, essential for minimizing the objective function.
- **Callback**: A callback function facilitating the retrieval of intermediate information such as energy, gradients, and parameters at each step of the optimization process.
- **Initial Parameters**: Optional initial parameters for the ansatz.
- **Observables**: The Hamiltonian provided as a sum of Pauli strings, defined using sparse Qiskit Operator Flow (Opflow) operators.

```
# Define a custome VQE class to orchestra the ansatz, classical optimizers,
# initial point, callback, and final result
class QiskitVQE(MinimumEigensolver):

    def __init__(self,estimator, ansatz, optimizer,ham,initial_parameters=None,callback=None):
        self._estimator = estimator
        self._circuit = ansatz
        self._optimizer = optimizer
        self._callback = callback
        self._initial_parameters=initial_parameters
        self._obs=ham


    def compute_minimum_eigenvalue(self):

        # Define objective function to classically minimize over
        def objective(x):
            job = self._estimator.run(self._circuit,self._obs,[x])
            H=job.result().values[0]

            if self._callback is not None:
                self._callback([H,x])
            return H

        # Select an initial point for the ansatzs' parameters
        if self._initial_parameters is None:
            x0 = np.pi/2 * np.random.rand(self._circuit.num_parameters)

        else:
            x0=self._initial_parameters

        # Run optimization
        res = self._optimizer.minimize(objective, x0=x0)

        job = self._estimator.run(self._circuit,self._obs,[res.x])
        H=job.result().values[0]

        return H
```