Date: 1st March 2024 Project Supervisors:
; Prof. Tal Mor, Mr. Ittay Alffasi

Project in Quantum Computing - 236991
Variational Quantum Eigensolvers

Hadar Tojarieh
Romi Levy

Contents

1 Introduction

The Variational Quantum Eigensolver (VQE) is a hybrid quantum-classical algorithm. It aims to find an upper bound of the lowest eigenvalue of a given Hamiltonian ?. The VQE uses a parameterized quantum circuit to calculate the cost and combine it with an optimizer running on a classical computer. The Subspace VQE, known also as SSVQE, is an algorithm which aims to calculate the k'th excited state energy of a Hamiltonian ?. In this project we implement, using Qiskit ?, VQE algorithm, SSVQE, weighted-SSVQE for the k'th excited state and weighted-SSVQE for all k excited states. Our implementation runs on Qiskit simulator. The optimization process is tested with two optimizaers of Scipy ?, BFGS which is gradient-based and COBYLA which is gradient-free, on different Hamiltonians.
In this report, we explain about the Hamiltonians we use. Then an explanation about the different optimizers that were tried, is given. Afterwards, The algorithms VQE and SSVQE, their results and explanation can be found. We also mention how the evaluation of the expectation value is made, give our explanation on the results, and finally give ideas for further work. This report is attached to our git repository of the project, in which one can find all the our code references.

2 Hamiltonians

Before getting into the core implementation of the project, we would like to present the Hamiltonians we managed to create using Qiskit and OpenFermion ?, for each algorithm. As in the articles, we consider Hamiltonians that can be written as a number of terms which is polynomial in the size of the system. All the transverse ising model Hamiltonians were generated by function we created. When the fully connected transverse ising model

Hamiltonian denoted as:

$$H = \sum_{i=1}^{N} a_i X_i + \sum_{i=1}^{N} \sum_{j=1}^{i-1} J_{ij} Z_i Z_j \tag{1}$$

This formula describes the action on each qubit and the interaction between the different qubits. Where X_i acts with the coefficient of a_i on the i'th qubit, and the interaction between the i'th and j'th qubits is Z_i Z_j with the coefficient J_{ij}. It is important to note that the random coefficients were selected once, namely the transverse ising model Hamiltonian for each number of qubits for all the algorithms is the same one, that's for the need of comparison. Molecular Hamiltonians were generated using OpenFermion while running on Google Colab, *MolecularData* package of OpenFermion. At the begining, we tried all kind of ways to generate mulecular Hamiltonian for HeH, which is the one examined in the SSVQE acrticle. However, that did not managed, mainly because the package *MolecularData* do not contains the information needed for this molecule.
The code for implementing each type of Hamiltonian can be found in the path "final project/hamiltonians", when the "openfermion code" folder contains the code that generates the molecular Hamiltonian LIH (should be run on GoogleColab).
The Hamiltonians for each algorithm:

algorithm	Hamiltonian	qubits number
VQE	LIH	4
	H_2	4
	Transverse Ising Model	4
	Transverse Ising Model	3
	Transverse Ising Model	2
	H_2	2
SSVQE	Transverse Ising Model	4
	LIH	4
	H_2	4

When writing SSVQE, we mean all the three different algorithms.

3 Optimizers

3.1 COBYLA

Constrained optimization by linear approximation (COBYLA) is a numerical optimization method for constrained problems where the derivative of the objective function is not known, invented by Michael J. D. Powell. In mathematical optimization, constrained optimization (in some contexts called constraint optimization) is the process of optimizing an objective function with respect to some variables in the presence of constraints on those variables. That is, COBYLA can find the vector $\vec{x} \in S$ with $S \subseteq R^n$ that has the minimal (or maximal) without knowing the gradient of f. COBYLA is also the name of Powell's software implementation of the algorithm in Fortran. It works by iteratively approximating the actual constrained optimization problem with linear programming problems. During an iteration, an approximating linear programming problem is solved to obtain a candidate for the optimal solution. The candidate solution is evaluated using the original objective and constraint functions, yielding a new data point in the optimization space. This information is used to improve the approximating linear programming problem used for the next iteration of the algorithm. When the solution cannot be improved anymore, the step size is reduced, refining the search. When the step size becomes sufficiently small, the algorithm finishes.

3.2 BFGS

BFGS is an optimizer from the quasi-Newton methods family. In order to understand BFGS, one need to first understand few terms such as gradient-decent, Newton's method and Quasi-Newton methods. Gradient-decent is an iterative method for finding a local minimum of a real-valued, differentiable objective function $f(x)$. We start off at a random initial point and iteratively take steps proportional to the negative gradient of the function f at the current point. As seen in equation (Equation **??**) is a first-order optimization method, as it uses first-order information (ie. the gradient) to find the minimum.

$$p_{k+1} = p_k - \alpha \nabla f(p_k) \tag{2}$$

As for the Newton's method we take into account the objective function's second order behavior in addition to its first order behavior.

$$p_{k+1} = p_k - \frac{\dot{f}(p_k)}{\ddot{f}(p_k)} \tag{3}$$

Generalizing to n dimensions, the first derivative is replaced by the gradient ∇f and the second derivative is replaced by the Hessian H **?**. In n dimensions, our new iterative scheme is thus written as:

$$p_{k+1} = p_k - H(p_k)^{-1} \nabla f(p_k) \tag{4}$$

In each iteration k, Newton's method approximates the function f at the point p_k with a paraboloid, and then proceeds to minimize that approximation by stepping to the minimum of that paraboloid. It sound a lot more accurate than gradient-decent, yet

Newton's method has it drawbacks **?**. The main two we can note are it is sensitive to initial conditions and it's very computationally expensive. Despite its limited practical use, Newton's method is still of great theoretical value. What if we could get a sort of hybrid between gradient descent and Newton's method, where we can have faster convergence than gradient descent, but lower operational cost per iteration than Newton's method? well, quasi-Newton methods. In quasi-Newton methods, instead of computing the actual Hessian, we just approximate it with a positive-definite matrix B, which is updated from iteration to iteration using information computed from previous steps (we require B to be positive-definite because we are optimizing a convex function, and this automatically takes care of the symmetry requirement of the Hessian) **?**. There is however one condition that all quasi-Newton methods must share, and that is the Hessian approximation B must satisfy the quasi-Newton condition.

$$B_{k+1}[p_{k+1} - p_k] = \nabla f(p_{k+1})) - \nabla f(p_k)) \tag{5}$$

At each iteration k+1, the new Hessian approximation B_{k+1} is obtained using only previous gradient information. Solving for $\ddot{f}$ and substituting into Newton's method in one dimension, we get

$$p_{k+1} = p_k - f(\dot{p}_k) \frac{p_k - p_{k-1}}{\dot{f}(p_k) - f(\ddot{p}_{k-1})} \tag{6}$$

The quasi-Newton methods come in, where the above is generalized to multidimensional objective functions.Instead of approximating the second derivative merely by using the finite difference like in the second method, quasi-Newton methods have to impose additional constraints. The common theme still runs through — at each iteration k+1, the new Hessian approximation B_{k+1} is obtained using only previous gradient information. As of today, the most widely used quasi-Newton method is the BFGS method. We described here the basic of an optimizer such BFGS. Of course it has it's own conditions which creates the final algorithm. For further reading, we suggest on "Practical methods of optimization" by R. Fletcher.

4 VQE

The available quantum computers in the near term, called Noisy Intermediate Scale Quantum computers (NISQ), are too small to support error correction and have high levels of noise. Therefore, implementing long quantum circuits on such devices will not likely provide satisfyingly accurate results. A class of algorithms, called the variational algorithms, on the other hand are suited to be performed on a quantum computer since they can be implemented using shallow circuits.
One of these variational algorithms called VQE. The Variational Quantum Eigensolver

(VQE) is a flagship algorithm for quantum chemistry using near-term quantum computers. It is an application of the Ritz variational principle, where a quantum computer is trained to prepare the ground state of a given molecule. This algorithm is a hybrid algorithm as it divides the task of finding the smallest eigenvalue between quantum and classical processors. Previously, the common method for preforming such a task was the quantum phase estimation algorithm (QPE), that can efficiently find the eigenvalue of a given eigenvector but requires fully coherent evolution. This new and innovative method is done by taking an advantage of the natural preparation of variational quantum states and measuring expectation values on quantum computers, which are difficult to do on classical computers due to the exponential growth of the Hilbert space (quantum qubits state space) with system size (the number of qubits). Moreover, the algorithm uses classical processors to do tasks they are capable of performing easily. More specifically, VQE uses a classical optimizer to minimize the expectation value obtained from the quantum computer by varying the variational qubits state. The inputs to the VQE algorithm are a Hamiltonian (molecular or modeled) and a parameterized circuit preparing the quantum state of the molecule. Within VQE, the cost function is defined as the expectation value of the Hamiltonian computed in the trial state. The ground state of the target Hamiltonian is obtained by performing an iterative minimization of the cost function. The optimization is carried out by a classical optimizer which leverages a quantum computer to evaluate the cost function and calculate its gradient at each optimization step. In our implementation we used both BFGS and COBYLA optimizers, on which we expand in section 3. The ansatz circuit we used for this algorithm is an ansatz circuit from section 3.5 of the Qiskit tutorial. In the article, they experimentally demonstrate the feasibility of this approach with an example from quantum chemistry.

4.1 QEE (Quantum Expectation Estimation)

QEE computes the expectation value of a given Hamiltonian H for an input state $|\psi\rangle$. When considering Hamiltonians that can be written as a number of terms which is polynomial in the size of the system, this encompasses a wide range of physical systems, including the electronic structure Hamiltonian of quantum chemistry and the quantum Ising Model, which are discussed in this project. Thanks to the linearity of expectation, the expectation value of a Hamiltonian can be calculated as a sum of expectations of its components. Therefore, the evaluation of $\langle H\rangle$ reduces to the sum of a polynomial number of expectation values of simple Pauli operators for a quantum state $|\psi\rangle$, multiplied by some real constants. A quantum device can efficiently evaluate the expectation value of a tensor product of an arbitrary number of simple Pauli operators, therefore with an n-qubit state we can efficiently evaluate the expectation value of this $2^n X 2^n$ Hamiltonian.

Figure 1: linear entanglement ansatz circuit of 4 qubits and random parameters values

Figure 2: full entanglement ansatz circuit of 3 qubits and random parameters values

4.2 Full and Linear Entangled Ansatz Circuits

This implementation can be found in the notebook "linear-entanglement-and-full-entanglement-ansatz-circuits" in "vqe" folder. Examples for those circuits can be found in Figures 1,2. Of course, that for two qubits the entanglement does not matter. All the other circuits are displayed in the notebook.

4.3 COBYLA runs

For each graph in the notebook, one can find the running time of the optimization - this is relevant for all the algorithms. Graphs can be found in the appendix. The approximation error we defined for all the experiments is Equation **??**, when λ is the exact eigenvalue and $\tilde{\lambda}$ is the approximated eigenvalue.

$$ApproximationError = \frac{||\lambda| - |\tilde{\lambda}||}{|\lambda|} \tag{7}$$

The presented approximation error are for 50 iterations. Those of the 100 iterations can be found in the notebooks. We decided that an experiment converge to the correct solution if the approximation error is smaller than 20 % of the size of the correct solution. The threshold choice is was decided by trial and error , in accordance with the experiments carried out.

	Approximation Error for Linear Ansatz	Approximation Error for Full Ansatz	Did the experiment converge to the correct solution
LIH molecule on 4 qubits	0.0219243405377	0.0086331652706	yes for both ansatzs
H_2 molecule on 4 qubits	0.2412778	0.3011833	yes for both ansatzs
Transverse Ising Model on 4 qubits	0.7108275083	0.52354214	no for both ansatzs
Transverse Ising Model on 3 qubits	0.0998286199	0.22287919	yes for both ansatzs
H_2 molecule on 2 qubits	0.0138039787	0.0400587076	yes for both ansatzs
Transverse Ising Model on 2 qubits	0.0458911025	0.44459380	yes for Linear, no for Full

Results. It's easy to see that with this optimizer, we reach convergence in different scales. When it comes to the Transverse Ising Model on 4 qubits, indeed according to the threshold we defined it did not converged, however, when looking on the 100 iterations run, one can see, it indeed properly converge. Another interesting point we noticed is that the more the number of qubits is smaller, the graph is more convergent, namely the number of iterations needed to reach the eigenvalue is far more low.

4.4 BFGS runs

Graphs can be found in the appendix.

	Approximation Error for Linear Ansatz	Approximation Error for Full Ansatz	Did the experiment converge to the correct solution
LIH molecule on 4 qubits	0.036833673	0.057195832	yes for both ansatzs
H_2 molecule on 4 qubits	0.720886133	0.5603278975	no for both ansatzs
Transverse Ising Model on 4 qubits	0.96026641	0.8426952170	no for both ansatzs
Transverse Ising Model on 3 qubits	0.37586623	0.61006320	no for both ansatzs
Transverse Ising Model on 2 qubits	0.646771588	0.7593974855	no for both ansatzs

Results. BFGS looks like a failure in compare to COBYLA results. All the graphs seem to not converge to any value, but keep more or less in the same initial point, with tiny ripples around it. Now, one might think we changes something in the ansatz, or in the molecule hamiltonians, or in the initial values we give it. Namely, in relation to the previous runs, the only thing we changed is the optimizer it self. The LIH molecule on 4 qubits is is the outstanding between the results, however we think it is because the initial point was randomly chosen close to the exact eigenvalue. We do not think there any significance to the value that it does converge to, because it stays on the initial state which was randomly chosen. Why BFGS does not seems to converge to the right solution? we do not know. But, we have few assumptions. for non-convex nonlinear optimization problems, it is known that the BFGS algorithm may not be convergent **?**, our optimization might be non-convex nonlinear. The specific ansatz are not tailored to the problem specifically and are being used with no information before the optimization, maybe by changing the ansatz to be problem-specific the optimization will get better form.

4.5 Other Gradient-Free and Gradient-Based Optimizers

When we noticed the radical difference in the convergence of all the graphs between COBYLA and BFGS we tried to run other optimizers from the same "optimizers family" of each. We tried both families of optimizers - the gradient-free family and the gradient-based family. Let us explain the difference between them. Any optimization method basically tries to find the nearest/next best parameters form the initial parameters that will optimize the given function (this is done iteratively with the expectation to get the best parameters). The optimization method that uses gradient to get this parameters is called gradient based method; the other ones would be gradient free optimization. In gradient based optimization, there is a high possibility that we might stuck in a local minima/maxima when traversing parameters. This is because the gradient at any local minima/maxima is zero. Thus, making the difference between current and next parameter also zero. Gradient free methods are free from such faults and, hence, are quite useful. The results can be found in git, under the folder of "test vqe with additional optimizers". From the gradient-based optimizers, we has CG (Conjugate Gradient) and L-BFGS-B (The L-BFGS-B algorithm is an extension of the L-BFGS algorithm which is limited memory BFGS. It's goal is to handle simple box constraints (aka bound constraints) on variables) **?**. From the gradient-free optimizers we have powell and nelder-mead. The results of the optimizers of each family are quite the same as the original optimizers we tried from each family.

5 SSVQE

5.1 Introduction

The SSVQE uses a simple technique to find the excited energy states. All quantum computing transformations are unitary. Unitary transformations maintains the angles between vectors, so orthogonal states before transformations remains as orthogonal states after transformation. In general the SSVQE takes k ($k \geq 2$) orthogonal states as inputs to a parameterized quantum circuit, and minimizes the expectation value of the energy in the space spanned by those states. This method automatically imposes the orthogonality condition on the output states (thanks to the conservation of orthogonality under the unitary transformation). By that the algorithm convergent to the subspace spanned by the eigenstates that belong to the k-minimal eigenvalues. Later on the algorithm is maximizing the expectation value of the energy in the minimal-energy subspace gained from the previous part. By that the Algorithm convergent to the maximal value among the k minimal eigenvalues, meaning to the k'th energy level.
To sum up the algorithm is as follows:
Let $\{|\varphi_i\rangle\}_{i=0}^{k}$ be a set of orthogonal states,
U(θ) be a parameterized quantum circuit (the ansatz),
s be an arbitrary index $s \in \{0, ..., k\}$
and V(ϕ) be another parametrized quantum circuit that only acts on the space spanned by $\{|\varphi_i\rangle\}_{i=0}^{k}$

Figure 3: ansatz circuit of 4 qubits and random parameters values from SSVQE article

Then the algorithm:
1. Minimize this cost function:

$$L_1(\theta) = \sum_{i=0}^{k} \langle\varphi_j| U^\dagger(\theta) H U(\theta) |\varphi_j\rangle \tag{8}$$

denote the optimal θ by θ^* .
2. Maximize this new cost function:

$$L_2(\phi) = \langle\varphi_s| V^\dagger(\phi) U^\dagger(\theta^*) H U(\theta^*) V(\phi) |\varphi_s\rangle \tag{9}$$

5.2 SSVQE Ansatz Circuit

You can find the implementation of ansatz in Figure 1 of SSVQE article in the notebook called "ssvqe ansatz circuit". This is a variational quantum circuit used in the simulations performed in the article. The parameters Φ, θ are optimized to minimize L. D1 and D2 denote the number of repetition of a circuit in each bracket. In the SSVQE, to check if the experiment converged to the correct result we are not going to define a threshold, because all graphs did not converged.

5.3 Runs

In the following experiments k = 3. Graphs can be found in the appendix.

	Approximation Error	Did the experiment converge to the correct solution
LIH molecule on 4 qubits	0.05920872077	no
H_2 molecule on 4 qubits	0.00892716	yes
Transverse Ising Model on 4 qubits	0.194443998	no

Results. The only one that did hit the exact eigenvalue is H2 and that is also because the initial random state was close to the correct value. All the others have the typical behaviour with this optimizer, on which we gave our assumptions why it behaves that way.

6 Weighted SSVQE for finding the k-th excited state

6.1 Introduction

In this algorithm a weight is added to a specific original input state, let this state be signed as $|\varphi_k\rangle$. All the original input states goes into a parameterized quantum circuit: U(θ). Let w be value for which holds $w \in (0, 1)$.
Then the algorithm minimizes the next cost function:

$$L_w(\theta) = w \langle\varphi_k| U^\dagger(\theta)HU(\theta) |\varphi_k\rangle + \sum_{i=0}^{k-1} \langle\varphi_i| U^\dagger(\theta)HU(\theta) |\varphi_i\rangle \tag{10}$$

Note that eq. 4 is almost like eq. 2 except that this time we choose an arbitrary state which gets a lower weight then the others. When the cost function reaches its global optimum, the circuit U(θ) becomes a unitary which maps $|\varphi_k\rangle$ to the k-th excited state $|E_k\rangle$ of the Hamiltonian. Therefore, by minimizing the cost function, we can find the k-th excited state by a single optimization process. Note that the overall time required for the optimization might increase, due to the more complicated landscape of the cost function.

6.2 Runs

Hyper parameters:
$k = 4$
$w = 0.5$
After we saw COBYLA gives better result in the VQE section, we decided to try to run weighted SSVQE with COBYLA as well, those are not attached to the appendix and can be found in found in "weighted ssvqe" folder in "algorithms". All the other graphs can be found in the appendix.

	Approximation Error	Did the experiment converge to the correct solution
LIH molecule on 4 qubits	0.0400354812	yes
H_2 molecule on 4 qubits	0.45586612	no
Transverse Ising Model on 4 qubits	0.95923841	no

Results. The only one that did hit the exact eigenvalue is LIH and that is also because the initial random state was close to the correct value. All the others have the typical behaviour with this optimizer, on which we gave our assumptions why it behaves that way.

7 Weighted SSVQE

7.1 Introduction

This algorithm is an extension of the algorithm described in the previous section to find all of the k-th excited states of a given Hamiltonian which requires only a single optimization procedure.
The algorithm is as follows:
Let $\{|\varphi_i\rangle\}_{i=0}^{k}$ be a set of orthogonal states,
U(θ) be a parameterized quantum circuit (the ansatz),
and w be a weight vector such that $w_i > w_j$ when $i < j$
Then the algorithm minimizes this cost function:

$$L_w(\theta) = \sum_{i=0}^{k} w_i \langle\varphi_i| U^\dagger(\theta) H U(\theta) |\varphi_i\rangle \tag{11}$$

Note that eq. 5 is almost like eq. 4 except that this time each state gets a weight, in a way that $|\varphi_i\rangle$ has greater weight from $|\varphi_j\rangle$ if $i < j$.
In this algorithm a weight is added to each of the original input states that goes into a parameterized quantum circuit U(θ). Then the algorithm minimizes the expectation value of the energy in the space spanned by those weighted states. When the cost function reaches its global optimum, thanks to the way the weight vector is constructed, the circuit U(θ) becomes a unitary which maps each of the $|\varphi_j\rangle$ to the j-th excited state

$|E_j\rangle$ of the Hamiltonian. In our case we started from the famous orthogonal basis: the standard basis. So, Weighted SSVQE transforms $|0...0\rangle$ to the ground state, and another orthogonal basis state $|0...1\rangle$ to the first excited state and so on. Therefore, by minimizing the cost function, we can find all of the k-th first excited states by a single optimization process. Note that the overall time required for the optimization might increase, due to the more complicated landscape of the cost function.

7.2 Runs

In the following experiments k = 4, namely we looked for the first four energies. Graphs can be found in the appendix. In this section we decided that an experiment converge to the correct solution if the Mean Approximation Error is smaller than 20 % of the size of mean of the correct solutions.

	Mean Approximation Error	Did the experiment converge to the correct solution
LIH molecule on 4 qubits	0.03635983	yes
H_2 molecule on 4 qubits	0.6640126	no
Transverse Ising Model on 4 qubits	0.2029963	yes

Results. The results are similar to the results in the previous sections. A phenomena we can see over here is degenerate energy levels. An energy level is degenerate if it corresponds to two or more different measurable states of a quantum system.

8 Evaluation of Expectation Value via Pauli Strings

Each algorithm has a different cost function on which we preformed the optimization. Yet, each iteration, during the optimization, required computation of an expectation value. One can find the code which implement this computation in each notebook under the title "Expectation Value", the implementation is exact in all notebooks. For the classical minimization part, we used the python package Scipy. The objective function to be minimized is the cost function defined for each algorithm. The input to be evaluated is a list of circuit parameters, when the initial guess is of course random. Each parameter is

in the range $[0, 2\pi]$. The optimization aims to minimize the returned value from the cost function, by finding the right values for the circuit parameters for this minimum value. It's important to note that the goal is to converge to global minimum. Each classical iteration of the chosen optimizer requires creating the ansatz circuit and decomposing the Hamiltonian into Pauli matrices to calculate the expectation value for an input state.
Let n be the number of qubits in the system,
m be a polynomial in the number of qubits n,
$\sigma_{j_l} \in \{\sigma_x, \sigma_y, \sigma_x, I\}$
and $\forall i : P_i = \sigma_{j_1} \bigotimes \sigma_{j_2} \bigotimes ... \bigotimes \sigma_{j_n}$
Then in our case the Hamiltonian can be written as:

$$H = \sum_{i=1}^{m} \alpha_i P_i \tag{12}$$

That means that the Hamiltonian can be written as a sum of Pauli strings which is polynomial in the number of qubits. Each Pauli string represents the Hamiltonians acts on the l'th qubit by σ_{j_l}.
This linearity can be exploit to compute $\langle H \rangle$ using $\langle P_i \rangle$ for any i, and the right Pauli coefficients α_i. The above code be found in each notebook under the title "Expectation value from probabilities".
The algorithm goes as follows:
1. decompose $\langle H \rangle$ into linear combination of $\langle P_i \rangle = \langle \sigma_{j_1} \bigotimes \sigma_{j_2} \bigotimes ... \bigotimes \sigma_{j_n} \rangle$ and its matches Pauli coefficients α_i respectively .
2. For each Pauli Matrices string, compute the $\langle \sigma_{j_1} \bigotimes \sigma_{j_2} \bigotimes ... \bigotimes \sigma_{j_n} \rangle$ using the ansatz circuit:
a. "reduce" the string into only σ_zs.
b. as part of each reduction of each Pauli matrix that is not σ_z, append the appropriate transformation to the ansatz circuit, by transformations (**??**) and (**??**).

$$\sigma_x = H \sigma_z H \tag{13}$$

$$\sigma_y = S H \sigma_z H S^\dagger \tag{14}$$

Where $S = \begin{pmatrix} 1 & 0 \\ 0 & \text{i} \end{pmatrix}$
c. run the ansatz circuit and get the measurements probabilities distribution (computational basis).
d. determine the sign of each probability in the final sum up in stage e by:

$$s_i = (-1)^k \tag{15}$$

where is the number of places where the j^{th} matrix is σ_z (and not I), and the j_{th} bit is lit.
e. sum up all the probabilities with the proper signs.
3. sum expectation values of all the Pauli Matrices strings with the proper Pauli coefficients α_i.

9 Running on Real Hardware - IBM-Q

All the results in this project, came from running on a simulator, specificly, "qasm simulator". During the project we had several trials for running our code on IBM-Q, with no success. Each trial took between full day or two full days, and eventually always raised the next error:
""ibmqjobfailureerror: 'unable to retrieve result for job ****. job has failed: internal error. error code: 9999.'"" This error indicates on internal error when running it remotly. We tried to look at the list of API error codes of IBM-Q
https://quantum-computing.ibm.com/lab/docs/iql/manage/errors
which guided us to contact IBM Quantum via Slack for help. We indeed tried to contact them, however got no response.

10 Code Orientation

For running the code of our project and getting the result in real-time, one should clone our repository in GitHub: https://github.com/Tojarieh97/VQE
The environment we worked with is Jupyter Notebook, thus it is very easy to run every section by yourself. We created a conda environment in which all the needed python packages for this project are installed and one can simply create an environment of his own with the same packages exactly, using the 'environment.yml' file. The structure of the project goes as follows: under *finalproject*:
hamiltonians folder contains all the Hamiltonian implementation, code that one should run on Google Colab, and all the final Hamiltonians we used in for the projects runs. *algorithms* folder contains folder for each algorithm we discuss on, which contains notebooks within for each kind of run. *ssvqeansatz* contains the implementation of the ansatz circuit which appear in SSVQE article.
In each folder, you can find *ReadMe* which will give you a brief about the folder you are currently in. In addition, in each notebook there are markdowns about the process.

11 Discussion

There is no doubt that the COBYLA preformed much better than the BFGS, both in a matter of time and convergence. After trying to run VQE with additional gradient-free and gradient-based optimizers and indeed got the same result as the COBYLA and BFGS, respectively, we believe the difference in the results related to the optimizer, rather then our specific implementation. As written in the Prospective Research section, we suggest to further research this direction. Two articles we find relevant for this discussion and that we found as proper literature in the subject are **?**, **?**.

12 Prospective Research

During the project we raised a few ideas for future research, let us denote them.
1. Instead of running the software on real quantum computer, one can use a simulator which mimic real noisy of real hardware.
2. Using grid-search on the ansatz parameters for better optimization.
3. Research about the connection of lack of convergence to gradient-based optimizers, vs gradient-free optimizers which looks like converge better.