Investigating the Application of VQE in Quantum Chemistry

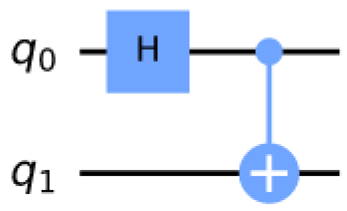
Cyrus Majd

PART I

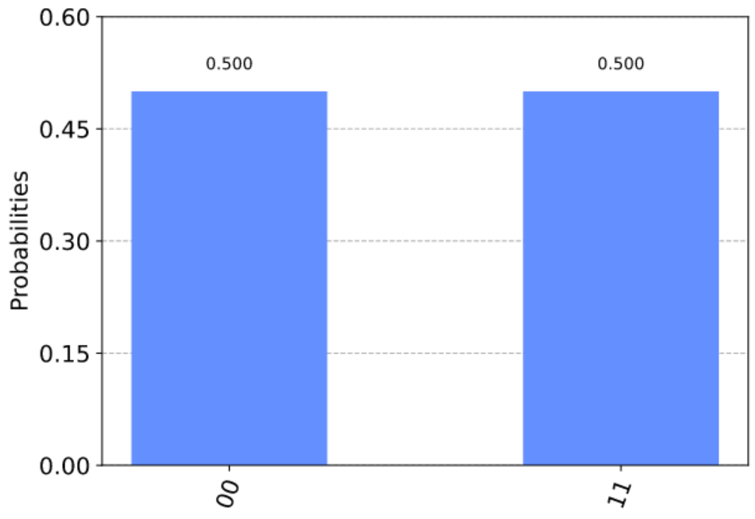
Variational Quantum Eigensolvers, or VQEs, are a specific kind of variational algorithm that can be used on the quantum computers we have today to tackle some of research's most modern problems. In this assignment, we will explore the application of VQE in quantum chemistry via the fabrication and trials of real quantum simulations.

Before we dive deeper into the actual chemistry experiments, let's first try to run a simple experiment on a real quantum computer and observe if we get the same results from a quantum simulator. For this first experiment, we will try to generate a Bell State.

You can think of a Bell State to be the simplest form of entanglement you can run on a quantum computer. It can be achieved by first creating a quantum circuit of two qubits, then applying a hadamard on the first qubit, and then applying a CNOT from that first qubit and onto the second one, as you can see from the figure below.



The purpose of the hadamard is to put q0 into superposition, and then the CNOT simply entangles the two qubits together. The interesting thing about a bell state is its property of either yielding the state |00> or |11> when measured. Importantly, it has a 0% chance of yielding |01> or |10>. Therefore, if we plot all the possibilities of measured values for the bell state, we see a distribution where 50% of the time you observe |00> and 50% of the time you observe |11>.



We can use this knowledge to draft a short program that runs a quantum simulation of a bell state and measures the resulting state 1024 times. The code, found here, will return the results of the simulation in text. After an arbitrary run of the simulation, we get this example output:

```
{'00': 521, '11': 503}
```

Where the values to the left of each colon is the quantum state (|00> and |11>), and the values to the right represent the number of times those states were measured. As you can see, the total number of states is 1024, and the distribution between |00> and |11> is roughly 50-50. There exists some noise in our simulation thanks to sampling error, but of course, the higher number of shots we take from the simulated state, the closer our distribution will get to the expected 50-50 results.

The code then runs the experiment on one of IBM's real quantum computers. In an arbitrary run of the simulation, the program chose to run on IBM's "ibmq_lima" machine (the least busy machine at the time), and yielded these results for the bell state distribution:

```
{'00': 498, '01': 28, '10': 23, '11': 475}
```

Now this is certainly interesting. We haven't changed the code at all, but suddenly when we run it on a real quantum computer, we observe some noise states that should not appear by our calculations. Recall that the purpose of the bell state is to create the quantum state $1\sqrt{2}(|00\rangle+|11\rangle)$, which can mathematically only yield |00> and |11> when measured/collapsed. Hence, we can be sure that the erroneous states |01> and |10> are byproducts of real noise on IBM's Lima machine at the time of this experiment. The noise could be due to multiple factors: from stray microwaves tampering with the quantum processor, to heat radiation, to even slight disturbances with the machinery of the computer.

While the noise is still impressively low for 1024 samples, the errors add up. More complicated quantum algorithms will have to deal with much more noise, as the operations used will be orders of magnitude more complicated than generating Bell States. Regardless, this experiment proves the fragility of the quantum computers we have today as well as the strides we have made in increasing their accuracies.

PART II

PART A

Now that we have some expectations on how to run a circuit on a quantum computer, along with the errors that come with it, we can investigate some cool chemistry experiments. For this first experiment, let's try to calculate the ground state energy for a LiH molecule. To do this, we first need to know the interatomic bond distance, which can simply be found here. This value was already calculated by some smart chemists. I have no idea how they found it but I won't question their methods.

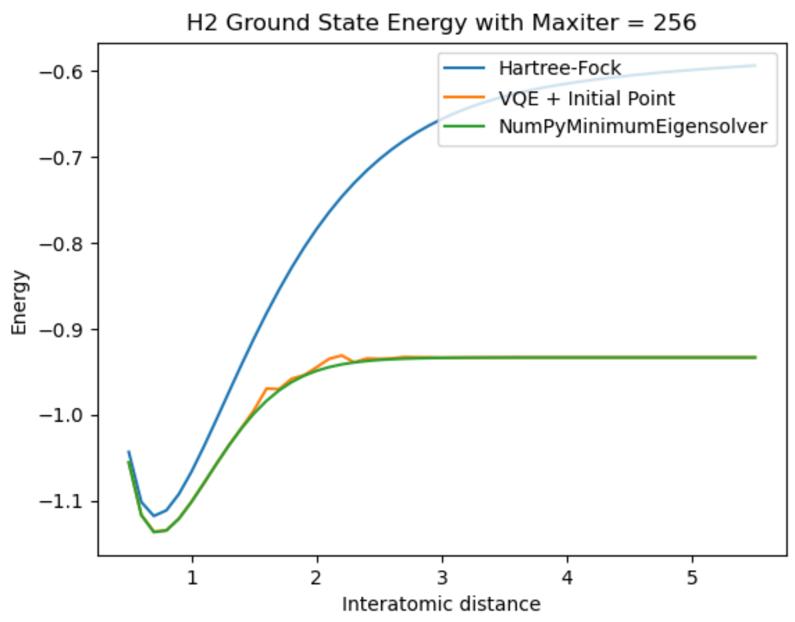
We can use the boilerplate code provided here to run our experiment. In the code, the "atom" parameter in line 7 is a simple string that encodes the bond lengths of the electrons for both the Lithium and the Hydrogen atoms. We set the last electron of Hydrogen to have a bond length of 1.595, as is previously documented and as is shown in the line of code below:

```
atom='Li .0 .0 .0; H .0 .0 1.595'
```

The code runs in two different ways. Firstly, we use a classical optimizer to calculate the chemistry of this molecule and give us a pretty accurate value for the ground state energy. In our experiment, we found this value to be roughly -7.88 Hartrees. After this is done, the code runs a quantum circuit of 4 qubits to calculate the ground state energy via VQE. In our same experiment, it gave a value of roughly -7.86 Hartrees. These are incredibly impressive results. While there is some slight error to acknowledge between our expected value and observed value, this simple experiment proves that it is actually practical for VQE to closely approximate classically computed values. If we changed some parameters to have the simulation be more fine tuned, we could probably achieve an even smaller percent error than we have now.

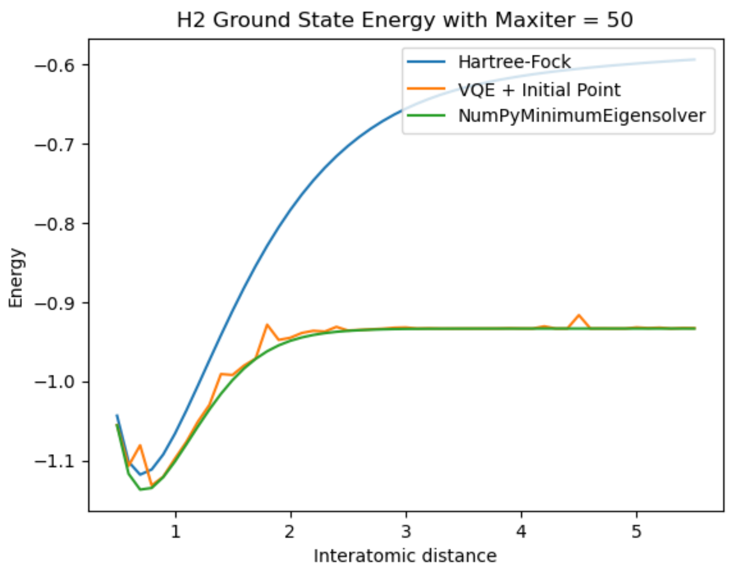
PART B

Here, we want to simulate the ground state energy of molecular hydrogen (H2). Like before, we use the boilerplate code found here. Our objective is to create a dissociation curve, which will show us an optimal energy to interatomic distance ratio for the molecule. In our code, the "Start" parameter signals the beginning of the x-axis (interatomic distance in angstroms), the "By" parameter is the range of the x-axis from the previously defined start, and the "Steps" parameter basically controls the resolution of the graph. It is important to note that the total execution time of the program scales linearly with the “steps” parameter, so increasing this value makes the simulation last a long time. We run the experiment with nearly all default parameters except for steps, which we scale to 50 for the sake of generating a nice looking graph. This is our result:

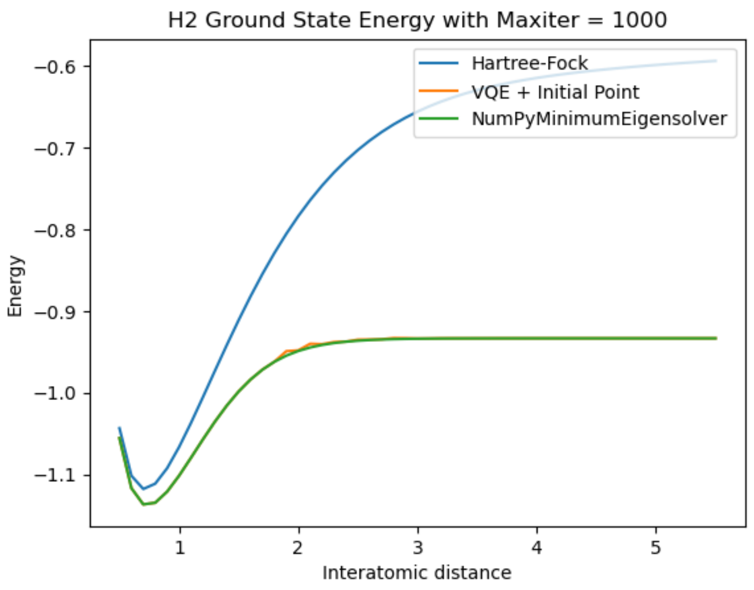


The optimization curve we want to achieve is the NumByMinimumEigensolver, or the classical optimizer, and the optimizer we have is VQE + Initial Point. Hartree-Fock is another optimization algorithm for reference. As you can see, our approximation is pretty darn close to the classically calculated answers, which is great news. The ground state energy can be determined by finding the y value at the minimum of the graph, so in this scenario, the ground state energy is around -1.14 Hartrees. The hydrogen’s bond length is simply the x-axis value at that point, which is found to be around .75 angstroms. This is extremely accurate to already known data on the hydrogen’s bond length, which is .7414 according to this site.

But what makes our optimizer so good? A theory is that the prediction may depend on the max_trials parameter when we define a new object of the SPSA optimizer engine on line 58. If we lower the value of this parameter from 256 to say, 50, we get this graph:



As you can tell, the graph is much more jaggedy and uncertain. The results are still generally accurate, but this may be thanks to the high resolution caused by the increased “steps” parameter. Maxiter, or max_trials, basically controls the number of times the optimization algorithm runs. Naturally, if the number is low, the optimizer has less chances to produce an accurate graph. If the number is high, the optimizer finds more chances to produce better results. This can be seen if we simply increase the value of max_trials to 1000, which is four times that of the default value of 256.



As you can see, the VQE + Initial Point curve is much smoother and more refined than the previous ones we had. While we can get ever more accurate predictions, however, the computation time of this algorithm increases rapidly with the increase of this max_trials parameter, and the optimizer gives diminishing returns at some point too. It eventually becomes more profitable to run the experiment with moderate values of both “steps” and “max_trials” and deal with an error rate of ~3-5% than to run the experiment with high values for both parameters and wait a really long time just to have a ~1-2% error rate.

PART C

Now, using the knowledge we obtained from the previous experiments, we want to simulate the ground state energy of the molecule LiH. To do this, we can just modify the code we had for simulating H2. The first step is to change the “atoms” string on line 20 to run the correct simulation for LiH and not H2. We change it to this:

```
atoms = 'Li .0 .0 -{0}; H .0 .0 {0}'
```

The special formatting characters (the {0} parts) are used for the optimizer to change those values. Because we aren’t supposed to know the molecular bond length for these two atoms in the molecule, we will have to optimizer find it out for us.

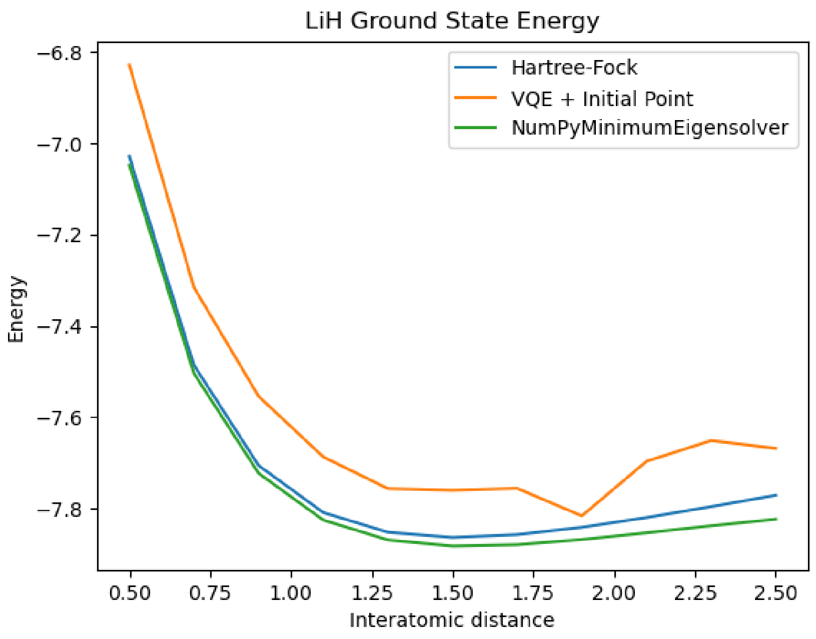
We also need to change the values of “start” and “by” for our new molecule. Because we know that the molecular bond length of LiH should be 1.5949 based on this website. So we change these values to start = .5 and by = 2, so we have a graph whose x axis starts at .5 and has a range of 2. We can also change the number of steps in the simulation to 10. Because LiH is a difficult molecule to simulate, increasing the steps parameter here can really hurt our runtime. Setting steps to 20, for example, will double the execution time as opposed to setting it to 10, and arguably doesn’t give us that much greater insight to the curve of the graph. We also remove the max_trials parameter from the SPSA optimizer object instantiation and use the default value, since it’s good enough for our experiment and, again, leads to a reasonable program execution time. We also want to add two parameters to the Hamiltonian object we create for the qubit operator on line 44. We can add the following:

```
        freeze_core=True,
        orbital_reduction=None)
```

These are basically two easy ways to optimize the runtime of the program further by reducing the number of things the VQE algorithm has to simulate. For example, freeze_core simply tells the program to ignore all of the electrons in the inner orbitals of the Li and H atoms, since those electrons won't be the ones interacting with the other atom's electrons anyway. It's semi-cheating the simulation, but not really.

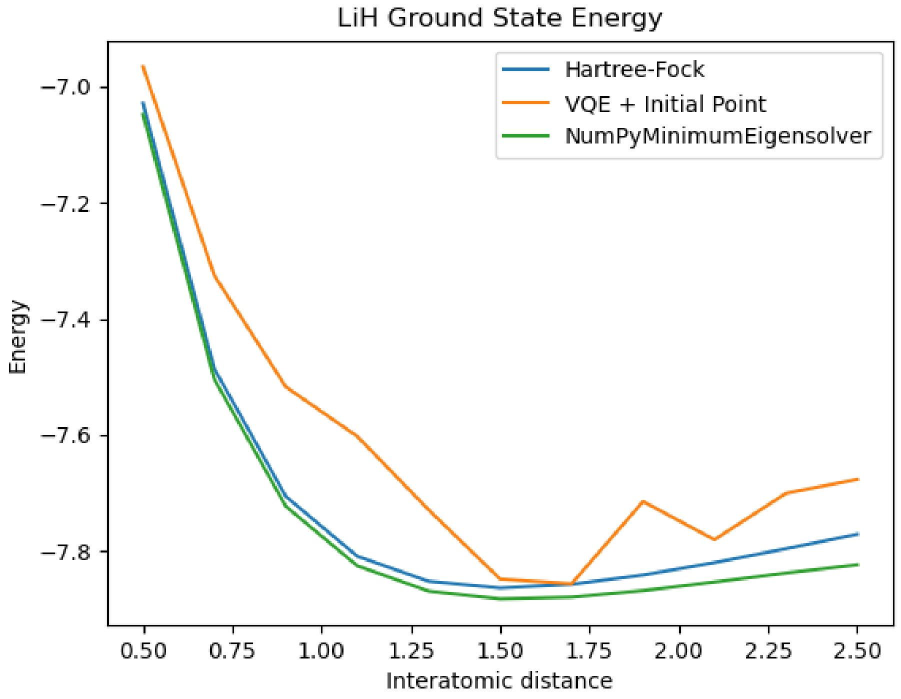
Another important parameter to change is the "reps" parameter, when we create the variable "var_form". "Reps" in this project is analogous to the value of "P" in QAOA. Increasing the number of reps increases the runtime of the algorithm but in turn yields more accurate results by giving the VQE optimizer more "layers" to mess around with. We change this value to 5 because it seems to work well with our simulations.

Now we run the simulation, and get this result:

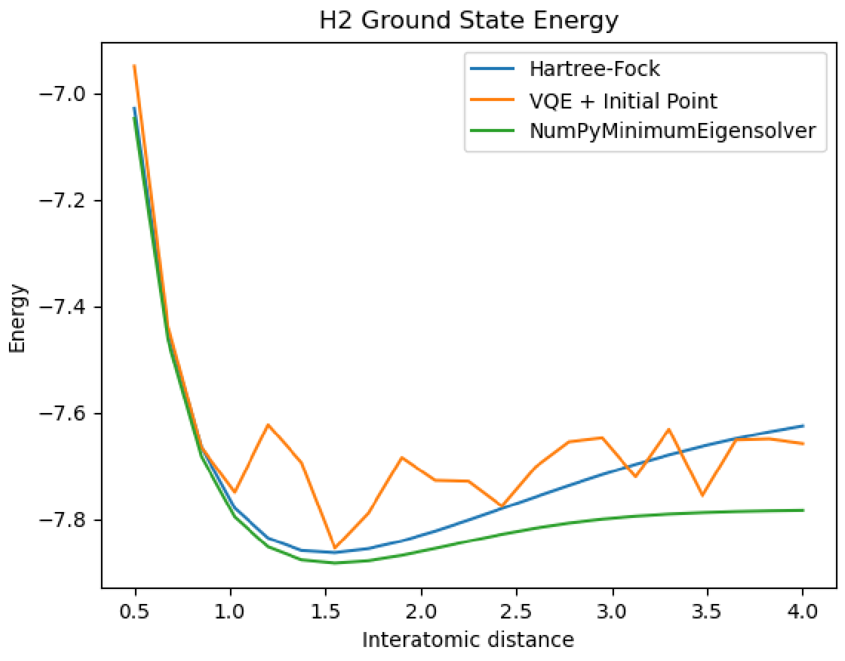


This simulation is pretty good, but of course it isn't as close to numpy's simulator as we would like it to be. It's important to note that the y axis scale is very sensitive, however, so our results aren't actually too inaccurate either.

We can try running the simulation again with a slightly higher max_trials value, but our graph remains a little off:



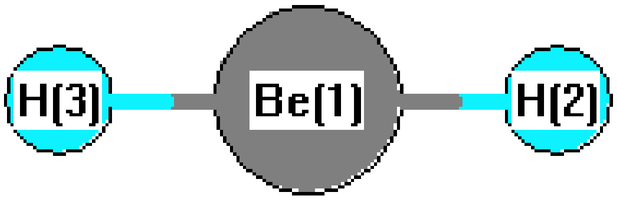
Admittedly, these results aren't too bad. Creating more accurate graphs would have taken multiple hours or even days, and we all know that ain't nobody got time for that……well, unless you're me I guess. BEHOLD THE STEPS = 20 GRAPH:



Disregarding the fact that I ran this extremely long simulation and forgot to change the title of the graph in the code so now the graph is slightly inaccurate, look at those amazing approximation ratios! With a higher steps and max_trials value, we could generate a graph that shows that LiH has a ground state energy of around -7.8 Hartrees with an interatomic bond distance of around 1.58 angstroms. These are extremely close to the real-world -7.85 Hartrees and 1.59 angstroms figures observed in the real molecule and recorded on this site. In other words, it works and I am happy now.

PART III

For this final part of the assignment, I decided to try and simulate beryllium hydride (BeH2). I thought to basically use what I have learned so far about how all of the parameters affect the final graph to generate a near-perfect VQE approximation of the numpy simulator, which includes changing the values of steps, max_trials, and reps. I reused most of the code that I had for simulating LiH, but the only challenge was to find the correct atom string for BeH2. Consulting the information on this site, I found that the BeH2 molecule looked like this:



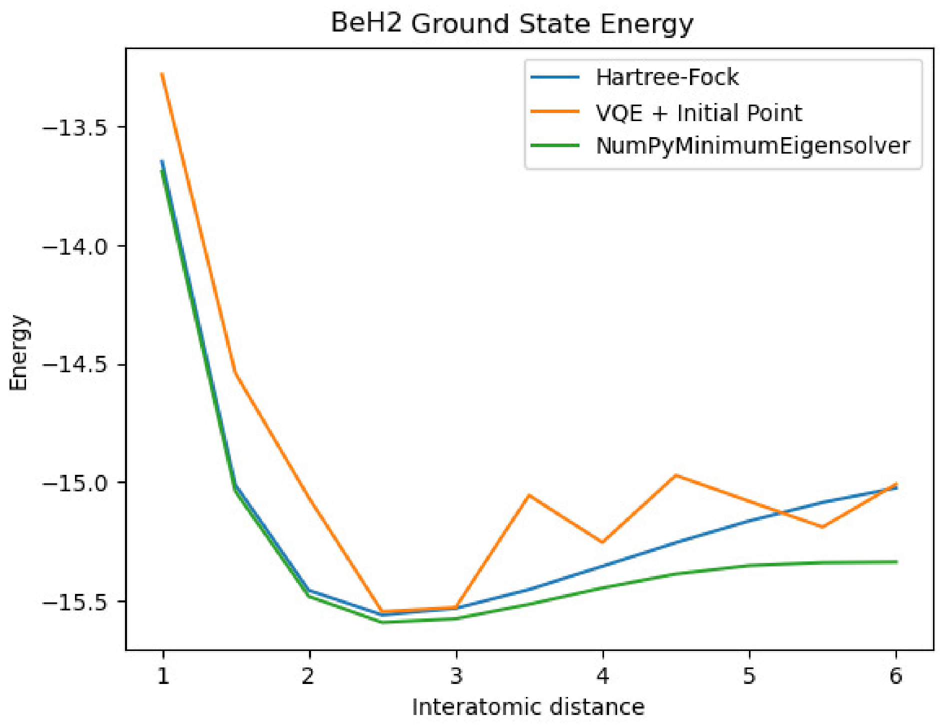
This implies that in the “atom” string, I need to put an H on either side of the Be atom. With this knowledge, I have my atom string as:

```
atoms = 'H .0 .0 0; Be .0 .0 .0; H .0 .0 0'
```

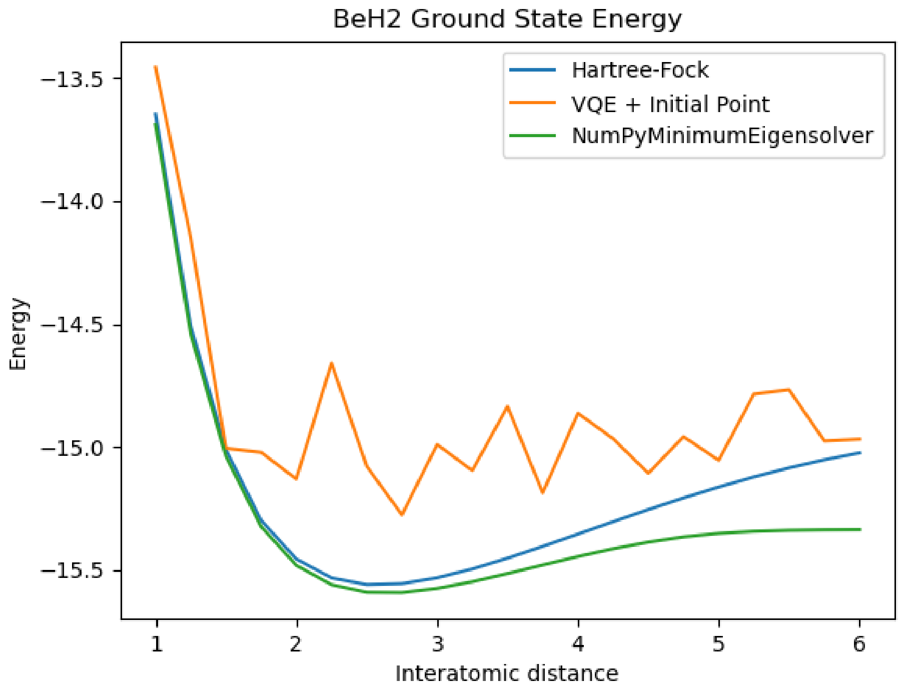
But now we need to figure out which electron distances should be variable and which ones should be fixed. Using information from https://cccbdb.nist.gov/geom2x.asp, https://cccbdb.nist.gov/nure2x.asp, https://cccbdb.nist.gov/expgeom2x.asp?casno=7787522, and https://github.com/qiskit-community/qiskit-community-tutorials/blob/master/chemistry/beh2_reductions.ipynb, I quickly figured out what the correct string for the atoms variable should be

```
    atoms = 'H .0 .0 -{0}; Be .0 .0 .0; H .0 .0 {0}'
```

Because we are measuring the lengths of the bonds from each hydrogen atom to the single beryllium, and also because the diagram shows the two hydrogen atoms being on opposite sides of the beryllium atom, one of these z values has to be negative (and I just chose the left hydrogen atom to be negative). Running the code with this new atoms string yields this result:



This is great! It shows a nice curve with steps = 10 and it shows that VQE has a clear minimum at around 2.5 angstrom, with an energy of -15.5 Hartrees. This is close to the actual values of BeH2, which points to an interatomic distance of around 2.65 angstrom. I was really excited to see what the graph would look like with a higher steps value, and consequently ran the experiment with steps = 20. To my surprise, the graph seemed to be a little less accurate this time.



I think that the reason why this happened was simply because the max_trials parameter was too low for the VQE algorithm to really do its magic for BeH2. Unfortunately, due to the complexity of the molecule, increasing these parameters led to insanely long computation times, and I could not reliably create a nice graph. Again, the diminishing returns that come with higher steps, reps, and max_trials parameters values make it difficult to gauge the efficiency of simulating with these high values.

The code to all of my experiments can be found in the same zip file as this PDF is in.

In summary, the experiment was still a success. The results we yielded from the first graph have an extremely low % error from the real results, once again proving the practicality of VQE algorithms in quantum chemistry. Definitely, as more work and research is being done on these algorithms and on actual quantum computers, these tasks will become quicker, more reliable, and more practical than running classical computations. VQE is an excellent example of how even moden quantum algorithms can yield impressive results, despite quantum computing still being in its infancy.