Simulating Basic Hydrogen Molecules, with Quantum Computing

Alex Zbinden

Abstract

For this project I wanted to go the route of something that deals with quantum itself. Therefore I thought a reasonable route would be to go with Chemistry, and more specifically, simulating simple molecules. Since this idea is very complicated and can quickly become complex even for quantum computers, the goal is to keep this project as simple as possible, whilst still producing impressive results. Jumping off of this, we will simulate H_2, or two Hydrogen Atoms that form a bond. This will be done using a minimal basis, so this will only use 2 qubits. We will use Suzuki-Trotter product formulas to achieve this basis, and then construct a quantum circuit to measure how our quantum state changes over time using Hamiltonians.

In terms of evaluation, we will evaluate by comparing variables like energy and bond lengths, and the correspondence between quantum states. The goal was to use Qizkit to run these simulations, however Pennylane is what I ended up finding more useful. For analyzing the data, we will use not only basic 2D plots, but pseudo 4-Dimensional animations as well to see how these molecules are acting in 'space' (given this simulation is done in a vacuum).

Introduction

To start, I gathered a lot of information about how quantum computers actually simulate chemistry. That is, how do we use entanglement and superposition to our advantage? The answer is quite simple, although taking me some time to fully understand. The idea was that we encode our 'data' into waves, and put these into superposition to simulate the randomness that we find at the atomic level. It is quite nice since chemistry at the atomic level really goes hand in hand with quantum level computing. They both deal with particles at an incredibly small scale!

With that, I decided to focus on a few of the many aspects of a molecule that one can measure. So for this project I exclusively focused on the change of Bond Lengths, and the change of Energy of these bonds over time. Of course there is much more to measure, especially when you simulate quantum physics along with this to get more exact measurements. That would have been much too far beyond the scope of this class though, and so I decided to keep things simple.

Simulating Hydrogen Bond Energy Levels

To start this project off, I focused on getting my energy measurements, since this was by far the more difficult measurement to be created and coded. In the beginning, I knew I would have to use Hamiltonian encoding, which I admittedly knew nothing about at the time. After reading a few papers, I realized that Hamiltonian encoding is essentially using Euler's Formula to simulate my energy trotterization, which uses the beautiful idea of imaginary numbers in a polar form to simulate and encode the energy levels. To put it simpler, if we take the Bloch sphere, we can encode our data by choosing a random angle, and plugging it into Euler's formula to get a 3-Dimensional vector which represents a wave (our simulated data). Once the energy levels were simulated, they were merely just quantum state vectors of the Hamiltonians, so from there I had to do an extra step. This extra step is getting the Eigenvalues from the Hamiltonian matrices. Luckily for small molecules like H_2, there was a simple built-in function from pennylane that did not require me to build a separate Eigensolver function, so this step was relatively easy.

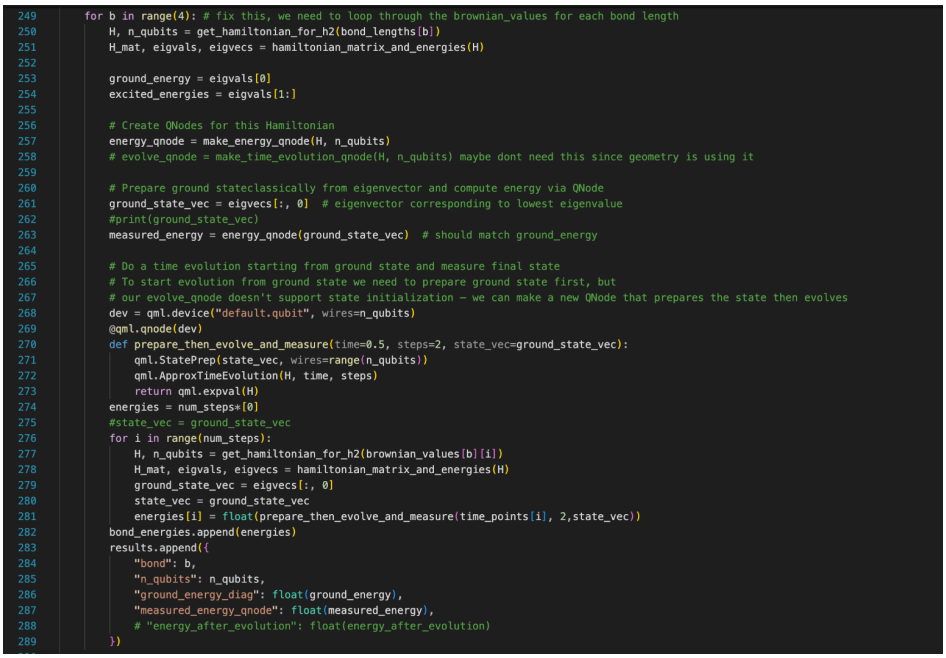
To explain this pipeline more simply, we begin with the data we started with, which is a bond length, compute a Hamiltonian for the bond, gather the eigenvalues from the Hamiltonian Matrix, and this gives us our ground state energy. Once we have the ground state, we compute time evolution on the ground state, which takes in a certain amount of 'time steps' and simulates the energy level after that amount of time. I then run this multiple times over to grasp a simulated 'real time' analysis of the changes in energy. Below we see this in code:

```python
def get_hamiltonian_for_h2(bond_length):
    symbols = ["H", "H"]
    coords = np.array([[0.0, 0.0, 0.0],
                       [0.0, 0.0, bond_length]])
    H, n_qubits = qchem.molecular_hamiltonian(symbols, coords)
    return H, n_qubits


def hamiltonian_matrix_and_energies(H):
    # sparse_hamiltonian returns a sparse matrix
    sparse_H = H.sparse_matrix()
    H_dense = sparse_H.toarray()
    eigvals, eigvecs = np.linalg.eigh(H_dense)
    # eigvals sorted ascending, ground state is eigvals[0]
    return H_dense, eigvals, eigvecs

def make_energy_qnode(H, n_qubits):
    dev = qml.device("default.qubit", wires=n_qubits)

    @qml.qnode(dev)
    def energy_for_state(statevector):
        """
        Prepares the provided statevector and returns expectation <H>.
        statevector must be length 2**n_qubits and normalized.
        """
        qml.StatePrep(statevector, wires=range(n_qubits))
        return qml.expval(H)

    return energy_for_state

def make_time_evolution_qnode(H, n_qubits):
    dev = qml.device("default.qubit", wires=n_qubits)

    @qml.qnode(dev)
    def evolve(t):
        qml.BasisState(np.array([1] + [0]*(n_qubits-1)), wires=range(n_qubits))
        qml.ApproxTimeEvolution(H, t, 1)
        return qml.expval(H)

    return evolve

```

So here we see our functions where we get our Hamiltonian, find our Eigenvalues, and then use Time Evolution to get our levels.

There is still one problem though, and that is that we need to exponentiate this returned value one more time to get our actual values. This is easily fixed with a simple helper function in our main loop.

Here is the function, and we also loop through this to simulate many time steps as well.



```
for b in range(4): # fix this, we need to loop through the brownian_values for each bond length
    H, n_qubits = get_hamiltonian_for_h2(bond_lengths[b])
    H_mat, eigvals, eigvecs = hamiltonian_matrix_and_energies(H)

    ground_energy = eigvals[0]
    excited_energies = eigvals[1:]

    # Create QNodes for this Hamiltonian
    energy_qnode = make_energy_qnode(H, n_qubits)
    # evolve_qnode = make_time_evolution_qnode(H, n_qubits) maybe dont need this since geometry is using it

    # Prepare ground stateclassically from eigenvector and compute energy via QNode
    ground_state_vec = eigvecs[:, 0]  # eigenvector corresponding to lowest eigenvalue
    #print(ground_state_vec)
    measured_energy = energy_qnode(ground_state_vec)  # should match ground_energy

    # Do a time evolution starting from ground state and measure final state
    # To start evolution from ground state we need to prepare ground state first, but
    # our evolve_qnode doesn't support state initialization — we can make a new QNode that prepares the state then evolves
    dev = qml.device("default.qubit", wires=n_qubits)
    @qml.qnode(dev)
    def prepare_then_evolve_and_measure(time=0.5, steps=2, state_vec=ground_state_vec):
        qml.StatePrep(state_vec, wires=range(n_qubits))
        qml.ApproxTimeEvolution(H, time, steps)
        return qml.expval(H)
    energies = num_steps*[0]
    #state_vec = ground_state_vec
    for i in range(num_steps):
        H, n_qubits = get_hamiltonian_for_h2(brownian_values[b][i])
        H_mat, eigvals, eigvecs = hamiltonian_matrix_and_energies(H)
        ground_state_vec = eigvecs[:, 0]
        state_vec = ground_state_vec
        energies[i] = float(prepare_then_evolve_and_measure(time_points[i], 2,state_vec))
    bond_energies.append(energies)
    results.append({
        "bond": b,
        "n_qubits": n_qubits,
        "ground_energy_diag": float(ground_energy),
        "measured_energy_qnode": float(measured_energy),
        # "energy_after_evolution": float(energy_after_evolution)
    })
```

Simulating Hydrogen Bond Lengths over Time

Next we want to simulate the hydrogen bond lengths with respect to time. We will also use a loop within the main loop, similar to the one above, to get our lengths over time. One thing to note, which I found out later on, is that our energies depend on the ground states that we calculate from our bond lengths. In essence, our energies depend on the bond lengths, so to get this to work I had to create the brownian motion simulation, then decode the values from there, and calculate the ground state energy to be able to run the energy of the molecule at that time stamp.

To start with this section, we want to make sure our data is normalized, so that is the first step in our process of simulating brownian motion. Next we use amplitude encoding to encode our data. This returns a state of our data in quantum form, which we know is now lying somewhere on the Bloch sphere. With these quantum states we can then perform random action on it. To do this we just grab a random angle and apply these rotations to our data, and repeat this process for the number of steps we want to apply. From there our quantum state is now much different than before, and of course is randomized. I also had plans to take this angle and show it in 3D space,

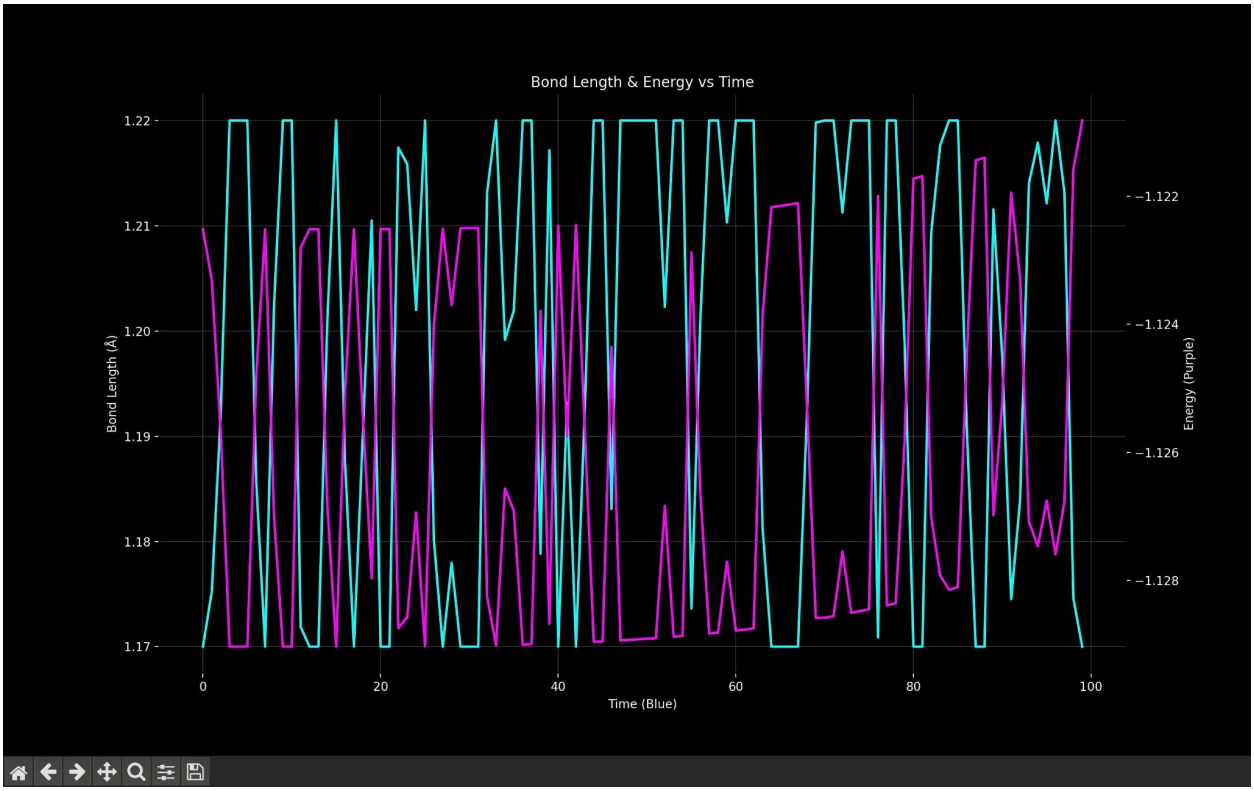
but this did not end up working and was more of a visual idea I had, and would not have changed the outcome whatsoever. Either way, we return the states from our random walk function, which ends up being our values we need (the bond lengths over some number of steps in time).

From doing this function where we apply the random walk, I found it interesting to see all the applications of random walks. Brownian Motion and the Random Walk are the exact same thing, and there are also so many other applications of the random walk, which is mindblowing. It really does show that the world around us, whether on the atomic scale or not, has a lot of randomness which we cannot necessarily predict!

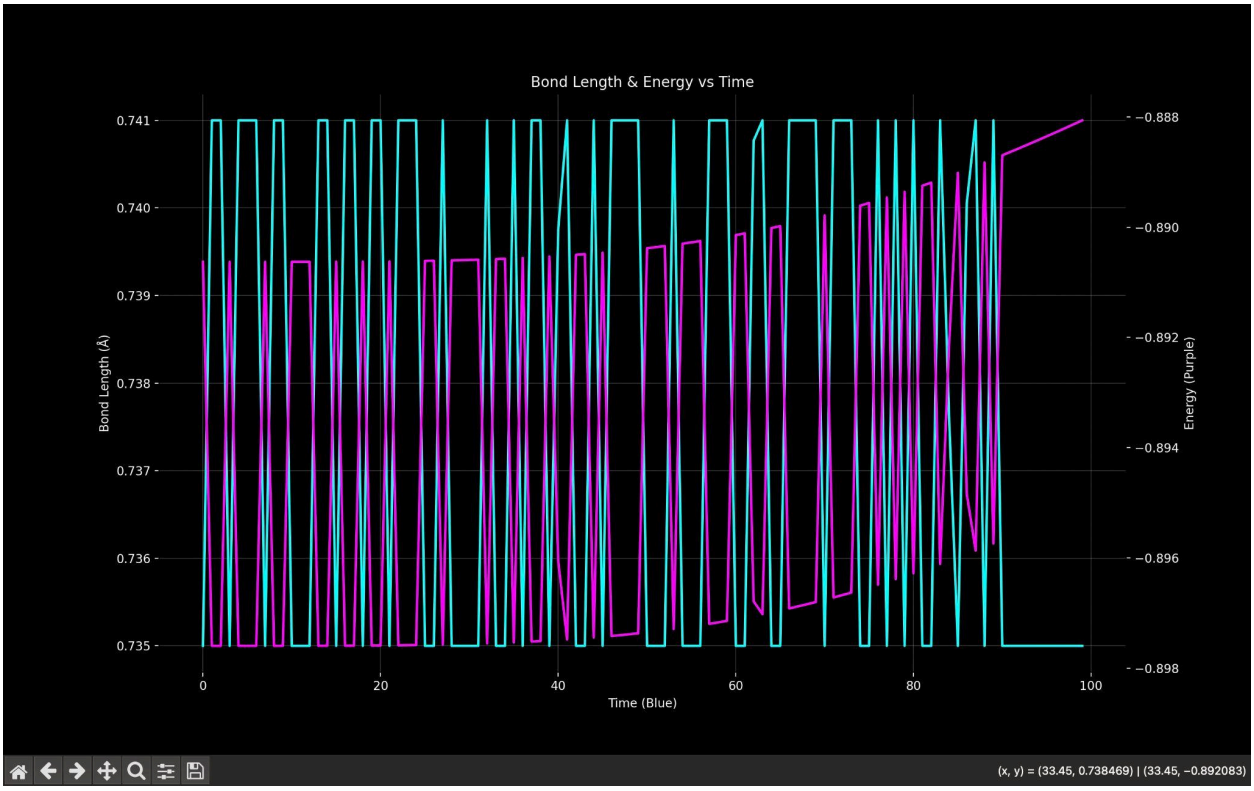
Running the Two Parts Together

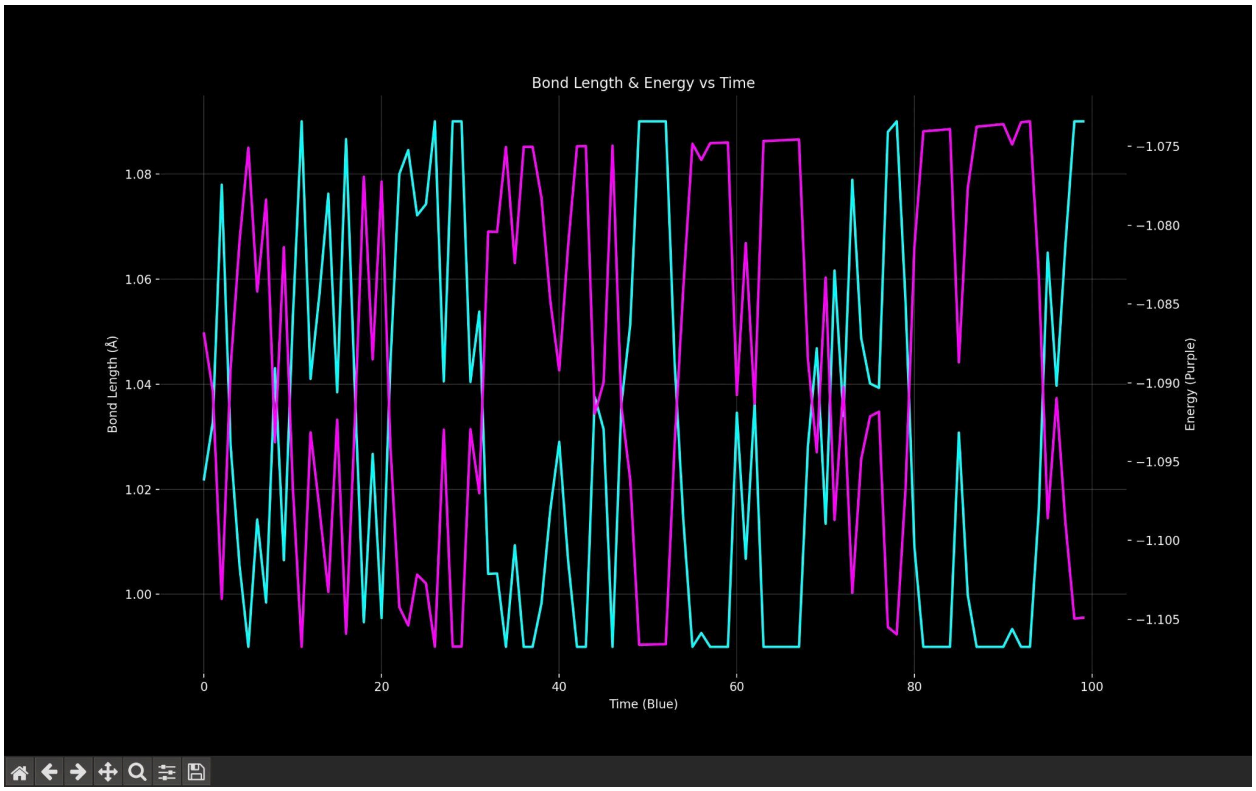
Now that I have built up our code for the simulations, I also wanted to visualize the data to make it easier on the viewers eyes. The initial thought I had was to set up plots to visualize the data, but that quickly became a little too complicated, and so I decided to not only have a simplified plot, but to also include animated three dimensional figures to show the data in real time as well.

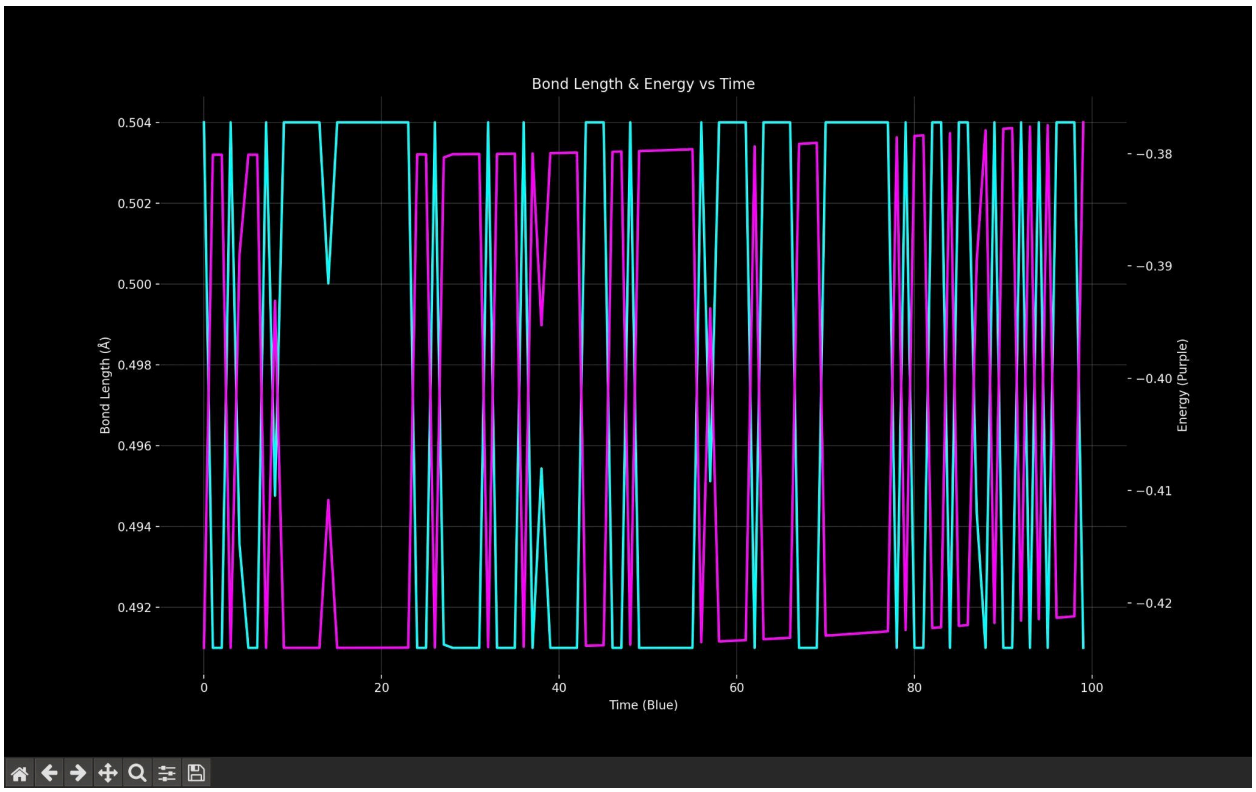
First, let's look at the plots, and explain what exactly they are showing:



This first plot shows the bond lengths (blue), and the energy (purple) over a given period of time. For these simulations I decided to use 100 time steps for each bond. The time steps are just arbitrary values and don't have a 'real world' representation in seconds. One thing to note about this graph is that the energy levels and the bond lengths are almost polar opposites, in the sense that when the bond length increases, the energy level decreases. This is what one would expect, as in real world results we have that an increased bond length will decrease the energy of the bond, since distance increases weakness. In terms of units, our bond lengths are based on atomic units of length, which are usually measured in nanometers. On the other hand we have our energy being measured in Hartrees, which are the base atomic unit for energy. Each of these are extremely small values. Hartrees are negative in this context because they are allowed to have a negative value when the energy is not enough to break the bond apart. The energy will obtain a positive value if we make the bond distance small enough though.

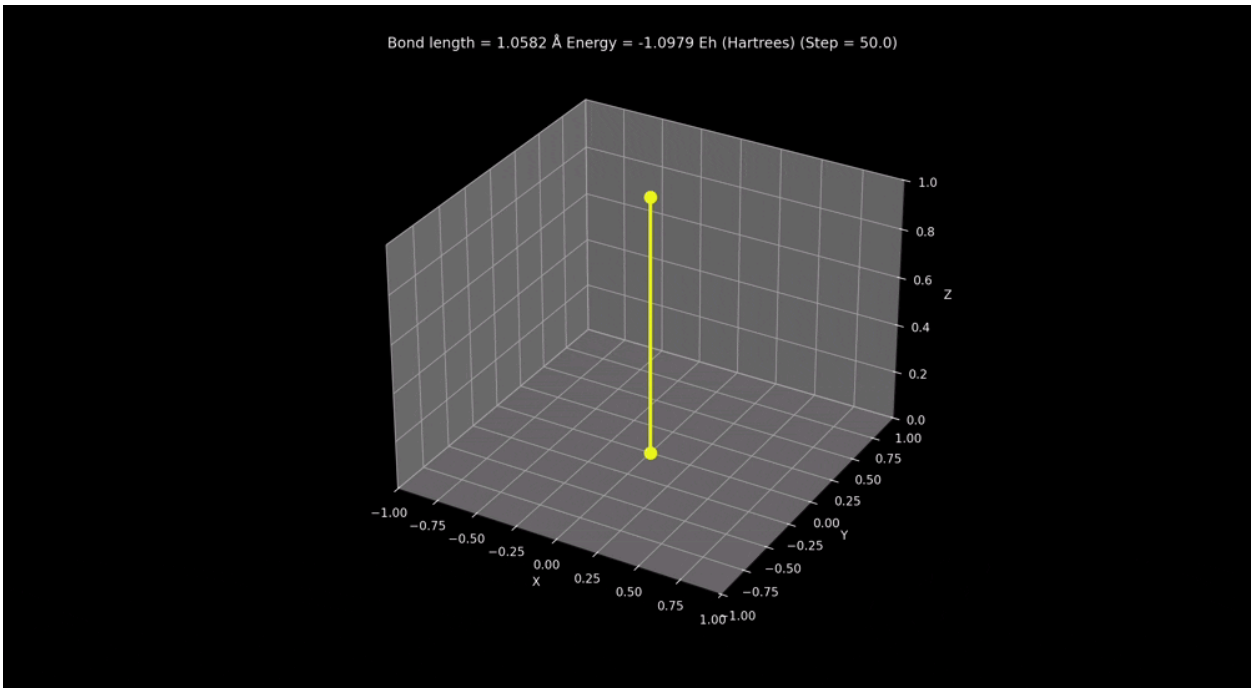


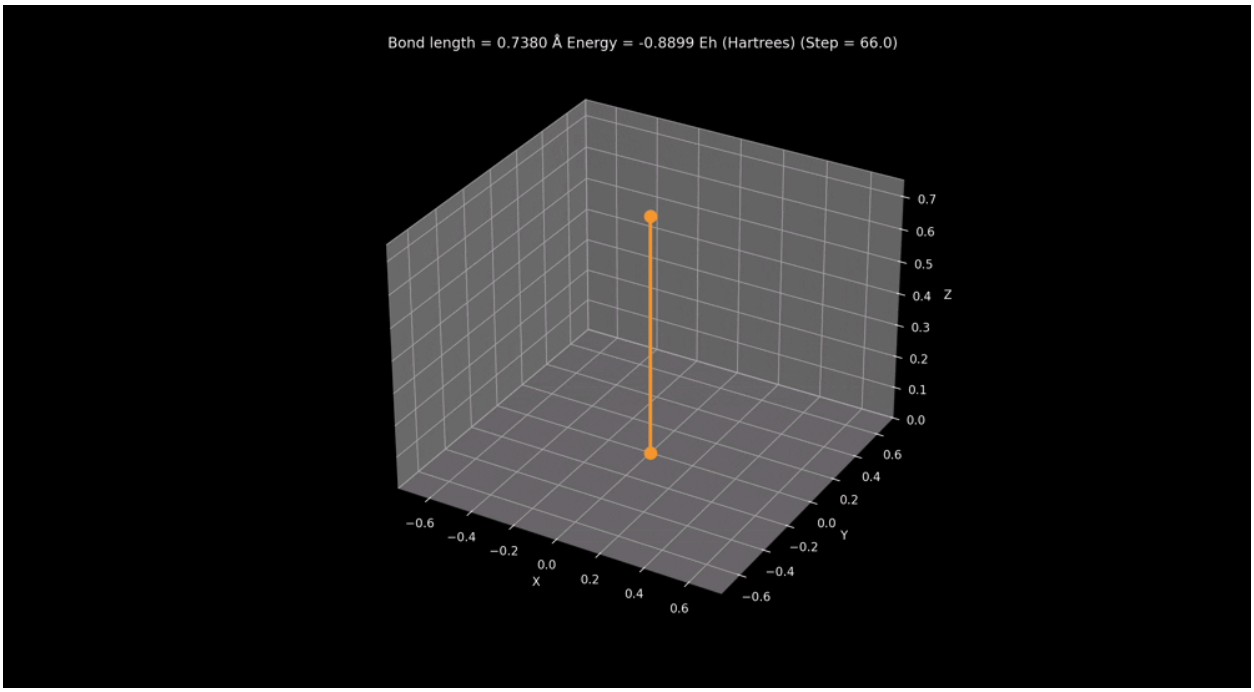

Bond Length & Energy vs Time
Bond Length (Å)
Energy (Purple)
Time (Blue)
1.08
1.06
1.04
1.02
1.00
−1.075
−1.080
−1.085
−1.090
−1.095
−1.100
−1.105
0
20
40
60
80
100

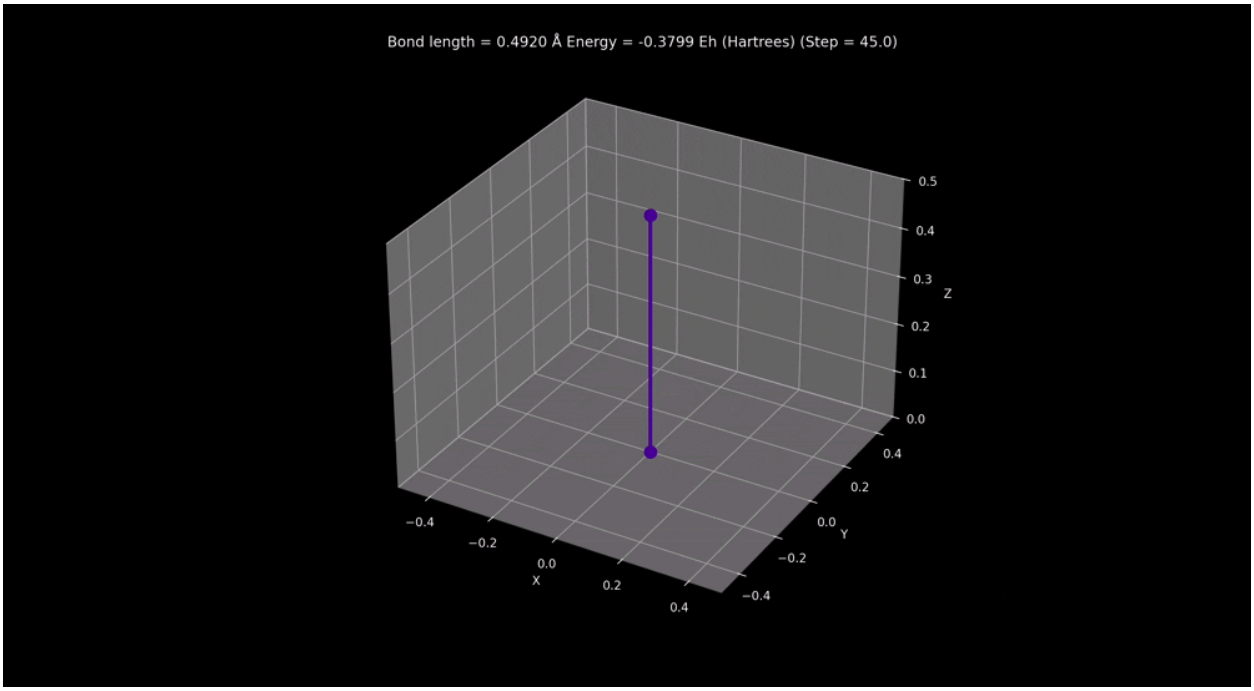

Bond Length & Energy vs Time
Bond Length (Å)
Energy (Purple)
Time (Blue)
0.504
0.502
0.500
0.498
0.496
0.494
0.492
−0.38
−0.39
−0.40
−0.41
−0.42
0
20
40
60
80
100

Looking at these other plots, we see a defining factor happening in all of them. It seems as time goes on, we have a general upwards curve in energy. The reason behind this is that we are simulating these hydrogen bonds in a vacuum, and given there are no quantum laws of physics being applied to these functions, it is expected that the energy in the system will rise overall, since no energy is lost in these simulations.

Now we can look at the animations, which also give some values at each time step of the energies and bond lengths:







Here we have three examples of Hydrogen bonds, where we have animated the process and show how the bonds change in length, in three dimensional space. While the Hydrogen bond is one dimensional, it is still interesting to see this change in our three dimensional world. On this last figure, one may also notice a slight change in color on the bond. The idea here was to show the change in energy over time using color change. This did not work too well, since the change in energy levels was usually much too small to have a noticeable effect color-wise. The reason the last figure is slightly noticeable at best is because of the larger change in energy level, which one can observe numerically above the figure as well. While the color variation was not too successful, we can notice the color variation among the different bonds is extremely noticeable, and this is because of the large variation in initial bond lengths. To explain the color-energy relation simply, the cooler colors like blue and purple represent higher energies, while hotter colors like red and orange represent lower energies.

Conclusion

Overall, with the results of the simulations, I think it is fair to say that Quantum Chemistry is a very reasonable and safe way to simulate molecules. While I did not simulate many aspects of the molecules, I definitely learned a lot and feel that given more time, I would have been able to

simulate and measure many more properties of molecules. With the measurements I did take, I learned that at the atomic level, there is a lot of randomness that occurs and therefore a lot of fluctuation of energy as well. At this level we can almost see the wave-like properties of particles in play, and the random motion of molecules shows that everything is always vibrating like a string, no matter how small it may be.

References

ChatGPT. (2025). *ChatGPT - Code Development Helper*. ChatGPT.

https://chatgpt.com/

Flow, T. (2025). *Hamiltonian simulation: Pennylane Quantum Topics*. PennyLane.

https://pennylane.ai/topics/hamiltonian-simulation

Matteo, O. D. (2021, April 23). *QHack QML challenge walkthrough: Variational quantum eigensolver: Pennylane Blog*. PennyLane.

https://pennylane.ai/blog/2021/04/qhack-qml-challenge-walkthrough-variational-quantum-eigensolver/

Modeling realistic chemistry with quantum computing. IBM Quantum Computing. (n.d.).

https://www.ibm.com/quantum/case-studies/modeling-realistic-chemistry