Variational Quantum Eigensolver

In this notebook, my aim is to walk you through each aspect of the VQE implementation in Qiskit. First, we will analyze key aspects of code. Then, we'll run the algorithm and retrieve the results.

In the analysis section below, only the essential chunks of code are shown. Irrelevant details are omitted. Please do not run this code as it is not complete. It is only for explanation purposes.

Defining Molecule

In this step, we will first initialize the PySCF driver with our molecule of interest LiH and our desired basis set. The driver will serve as a reservoir of information from which we can pull out the hamiltonian's subterms (electron integrals), number of particles (electrons), number of spin orbitals, etc.

Note that the arguments passed into PySCFDriver class includes the variable 'distance'. This denotes the intermolecular distance between lithium and hydrogen molecule. We can calculate energies for different intermolecular distances using this variable.

In [ ]:
```
#Defining Molecule
driver = PySCFDriver(atom='Li .0 .0 .0; H .0 .0 ' + str(distance), unit=UnitsType.ANGSTROM,
                     charge=0, spin=0, basis='sto3g')
molecule = driver.run()
#Mining info out of the reservoir
h1 = molecule.one_body_integrals
h2 = molecule.two_body_integrals
nuclear_repulsion_energy = molecule.nuclear_repulsion_energy
num_particles = molecule.num_alpha + molecule.num_beta
#For every one molecular orbital, there are 2 spin orbitals since there are 2 electrons of opposite
#spins within each molecular orbital.
num_spin_orbitals = molecule.num_orbitals * 2
```

Preparation of Qubit Hamiltonian

Preparation of Qubit Hamiltonian is done through a series of approximations. Before proceeding, a note on indexing orbitals.

In the driver, molecular orbitals are stored in arrays and thus have indices.The index values which range from 0 to n-1 where n is the total number of molecular orbitals of the molecule. Negative idx denote the upper orbitals so -1 corresponds to the highest orbital, etc. We will later convert negative idx to positive idx for computation purposes.

Freezing and Removing Orbitals

To decrease the size of our problem, we freeze some orbitals and remove others. This results in significant cost cutting in term of our computation.

Freeze Core Approximation

Freeze Core Approximation: In a molecule, the core electrons or the electrons have a small probability of being excited to higher orbitals. So core electrons don't contribute significantly to the correlation energy. Thus, a good approximation is to freeze them in place in their current orbitals and disregard their contributions to correlation entirely.

So, in our example, LiH has 6 molecular orbitals (indexed 0, 1, 2, 3, 4, 5) and the core electrons are the in the 0'th molecular orbital. Note that each molecular orbital holds 2 electrons with opposite spins. However spin orbitals hold only 1 electron. Since LiH has 6 molecular orbitals, it has 12 spin orbitals. Now how are those spin orbitals indexed? Indexes 0-5 correspond to the spin up electrons and 6-11 correspond to the spin down electrons. To illustrate the relation between spin orbitals and molecular orbitals, we note that the 0th MO's 2 electrons belong to spin orbitals indexed 0 and 6. Since we denoted 0th MO's electrons as core electrons, we have to add 0 and 6 as spin orbital indices into our freeze list.

Removing Orbitals

While it is possible that electrons can be excited to higher MO's, some MO's are more likely to be excited to than others. An example of this is shown in this qiskit tutorial using BeH2 molecule.

Let's say we are given the information that removing the fourth and the fifth orbital (indexed 3 and 4 respectively) still gives us a good approximation to the exact energy curve as a function of intermolecular distance. Simply, MO's indexed 3 and 4 are not important. Thus, we can remove them. Note these are MO's not spin orbitals. Converting to spin electron orbital indexes, the spin up electrons of MO's 3 and 4 go to SO's 3 and 4. And, the spin down electrons of MO's 3 and 4 go to SO's 8 and 9.

```
In [ ]: #prearing the idx of the MO's that need to be removed or frozen
        freeze_list = [0]
        remove_list = [-3, -2] # negative number denotes the reverse order
        # convert all negative idx to positive
        remove_list = [x % molecule.num_orbitals for x in remove_list]
        freeze_list = [x % molecule.num_orbitals for x in freeze_list]
```

Updating the Array

Inside the fermionic operator object, there are arrays that store the spin orbitals. Let's choose one of those arrays and call it A. Initially, A contains 12 boxes, one for each spin orbital, indexed 0 to 11. When we freeze or remove orbitals, we are getting rid of those orbitals and shrinking the size of array. So after freezing the 0th MO or the 0th and 6th spin orbitals, A noe contains 10 boxes indexed 0 to 9. With this change, all the indices of the remaining orbitals change. For example, what was initially the 5th orbital becomes now the 4th orbital. Following this pattern, the 3rd, 4th, 8th and 9th spin orbitals we were trying to remove earlier now become the 2nd, 3rd, 7th and 8th spin orbitals.

After removing these orbitals, A now shrinks in size to 6 boxes. Since LiH initially contained 4 electrons and we froze 2 of them as core electrons, we have to now measure the correlation of the 2 remaining electrons relative to 6 spin orbitals. This is a dramatic simplification from 4 electrons and 12 boxes to 2 electrons and 6 boxes. And such simplification facillitates the rest of the quantum computation.

```
In [ ]: # Updating the indices in the remove list after core molecular orbital frozen
        remove_list = [x - len(freeze_list) for x in remove_list]
        # Converting from MOs to SOs indices
        remove_list += [x + molecule.num_orbitals - len(freeze_list)  for x in remove_list]
        freeze_list += [x + molecule.num_orbitals for x in freeze_list]

        #Feeding these simplifications to the operator
        ferOp = FermionicOperator(h1=h1, h2=h2)
        if len(freeze_list) > 0:
            ferOp, energy_shift = ferOp.fermion_mode_freezing(freeze_list)
            num_spin_orbitals -= len(freeze_list)
            num_particles -= len(freeze_list)
        if len(remove_list) > 0:
            ferOp = ferOp.fermion_mode_elimination(remove_list)
            num_spin_orbitals -= len(remove_list)
```

Mapping to Pauli Operators

Here we utilize Z2 symmetries and 2 qubit reduction to further simplify computation. Then we convert the molecular hamiltonian to its qubit representation, in terms of pauli matrices. With this, we move onto setting up the quantum circuit for VQE.

```
In [ ]: qubitOp = ferOp.mapping(map_type=map_type, threshold=0.00000001)
        qubitOp = Z2Symmetries.two_qubit_reduction(qubitOp, num_particles) if qubit_reduction else qubitOp
        qubitOp.chop(10**-10)
```

If one were to print the Qubit Hamiltonian 'qubit0p', it would look something like this:

IIII (-0.2076593350197075+0j)

XXZX (0.008650156860618772+0j)

YYIX (0.008650156860618772+0j)

ZIZI (-0.11344680300366691+0j)

Each string represents a set of pauli operators acting on the 4 qubits. The number of right denotes their weight of each term in the expansion of Hamiltonian.

Initializing Quantum Circuit

In this section we will prepare the initial state, set up the classical optimizer and run the VQE algorithm.

Initial State and Variational Form

Our initial state will be hartree fock state as it has a good overlap with the exact ground state of lithium hydride, hence making convergence by VQE easier. When that is done, we will prepare the UCCSD or the Unitary Coupled Cluster Singles and Doubles state by acting on the initial state with a couple of quantum gates (rotation operators)

In [ ]:
```
# setup HartreeFock state
HF_state = HartreeFock(qubitOp.num_qubits, num_spin_orbitals, num_particles, map_type,
                       qubit_reduction)

# setup UCCSD variational form
var_form = UCCSD(qubitOp.num_qubits, depth=1,
                   num_orbitals=num_spin_orbitals, num_particles=num_particles,
                   active_occupied=[0], active_unoccupied=[0, 1],
                   initial_state=HF_state, qubit_mapping=map_type,
                   two_qubit_reduction=qubit_reduction, num_time_slices=1)
```

Classical Optimizer

Qiskit supports a variety of optimization subroutines. In this example we will use Constrained Optimization BY Linear Approximation or COYBLA optimizer.

One has to also specify the number of times one wishes to run the VQE algorithm. The more iterations, the slower the computation but the better the result

In [ ]:
```
# setup COBYLA optimizer
max_eval = 200
cobyla = COBYLA(maxiter=max_eval)
```

Running VQE

While we can use an actual quantum computer, for the purposes of this tutorial, we will use a statevector simulator. Running VQE then reduces to a single command

In [ ]:
```
backend = Aer.get_backend('statevector_simulator')
vqe = VQE(qubitOp, var_form, cobyla)
quantum_instance = QuantumInstance(backend=backend)
```

Retrieving Results

In [ ]:
```
results = vqe.run(quantum_instance)
```

Real Time Implementation

In [ ]:
```
%matplotlib inline
# Importing standard Qiskit libraries and configuring account
from qiskit import QuantumCircuit, execute, Aer, IBMQ
from qiskit.compiler import transpile, assemble
from qiskit.tools.jupyter import *
from qiskit.visualization import *
# Loading your IBM Q account(s)
provider = IBMQ.load_account()
```

In [ ]:
```
#Libraries
from qiskit.aqua.algorithms import VQE, ExactEigensolver
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
from qiskit.chemistry.components.variational_forms import UCCSD
from qiskit.aqua.components.variational_forms import RYRZ
from qiskit.chemistry.components.initial_states import HartreeFock
from qiskit.aqua.components.optimizers import COBYLA, SPSA, SLSQP
from qiskit import IBMQ, BasicAer, Aer
from qiskit.chemistry.drivers import PySCFDriver, UnitsType
from qiskit.chemistry import FermionicOperator
from qiskit import IBMQ
from qiskit.providers.aer import noise
from qiskit.aqua import QuantumInstance
from qiskit.ignis.mitigation.measurement import CompleteMeasFitter
from qiskit.aqua.operators import Z2Symmetries
```

In [ ]:
```
def get_qubit_ops(dist):
    # Defining Molecule
    driver = PySCFDriver(atom='Li .0 .0 .0; H .0 .0 ' + str(dist), unit=UnitsType.ANGSTROM,
                         charge=0, spin=0, basis='sto3g')
    molecule = driver.run()
    # Mapping to Qubit Hamiltonian
    freeze_list = [0]
    remove_list = [-3, -2] # negative number denotes the reverse order
    map_type = 'parity'

    h1 = molecule.one_body_integrals
    h2 = molecule.two_body_integrals
    nuclear_repulsion_energy = molecule.nuclear_repulsion_energy
    repulsion_energy = molecule.nuclear_repulsion_energy
    num_particles = molecule.num_alpha + molecule.num_beta
    num_spin_orbitals = molecule.num_orbitals * 2
    print("HF energy: {}".format(molecule.hf_energy - molecule.nuclear_repulsion_energy))
    print("# of electrons: {}".format(num_particles))
    print("# of spin orbitals: {}".format(num_spin_orbitals))
    # prepare full idx of freeze_list and remove_list
    # convert all negative idx to positive
    remove_list = [x % molecule.num_orbitals for x in remove_list]
    freeze_list = [x % molecule.num_orbitals for x in freeze_list]
    # update the idx in remove_list of the idx after frozen, since the idx of orbitals are changed after freezing
    remove_list = [x - len(freeze_list) for x in remove_list]
    remove_list += [x + molecule.num_orbitals - len(freeze_list)  for x in remove_list]
    freeze_list += [x + molecule.num_orbitals for x in freeze_list]

    # prepare fermionic hamiltonian with orbital freezing and eliminating, and then map to qubit hamiltonian
    # and if PARITY mapping is selected, reduction qubits
    energy_shift = 0.0
    qubit_reduction = True if map_type == 'parity' else False

    ferOp = FermionicOperator(h1=h1, h2=h2)
    if len(freeze_list) > 0:
        ferOp, energy_shift = ferOp.fermion_mode_freezing(freeze_list)
        num_spin_orbitals -= len(freeze_list)
        num_particles -= len(freeze_list)
    if len(remove_list) > 0:
        ferOp = ferOp.fermion_mode_elimination(remove_list)
        num_spin_orbitals -= len(remove_list)

    qubitOp = ferOp.mapping(map_type=map_type, threshold=0.00000001)
    qubitOp = Z2Symmetries.two_qubit_reduction(qubitOp, num_particles) if qubit_reduction else qubitOp
    qubitOp.chop(10**-10)
    shift = energy_shift + repulsion_energy
    return qubitOp, num_particles, num_spin_orbitals, shift
```

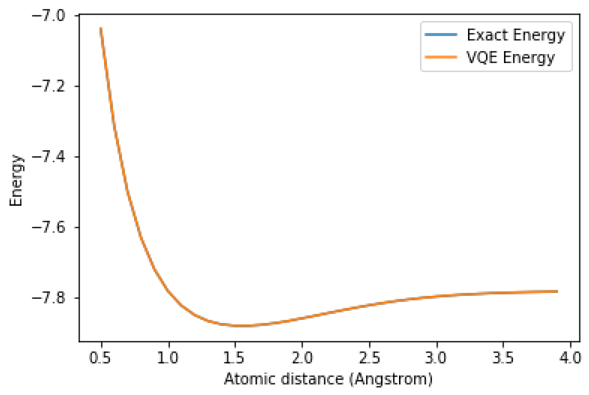
```
In [ ]: # Specifying whether running code on a simulator or an actual quantum device
        backend = BasicAer.get_backend("statevector_simulator")
        # creating a list of distances to run VQE on
        distances = np.arange(0.5, 4.0, 0.1)
        exact_energies = []
        vqe_energies = []
        optimizer = SLSQP(maxiter=5)
        for dist in distances:
            qubitOp, num_particles, num_spin_orbitals, shift = get_qubit_ops(dist)
            # Finding exact energies to show how off VQE was from exact energy curve
            result = ExactEigensolver(qubitOp).run()
            exact_energies.append(result['energy'] + shift)
            # Initial STate
            initial_state = HartreeFock(
                qubitOp.num_qubits,
                num_spin_orbitals,
                num_particles,
                'parity'
            )
            # UCCSD Variational Form
            var_form = UCCSD(
                qubitOp.num_qubits,
                depth=1,
                num_orbitals=num_spin_orbitals,
                num_particles=num_particles,
                initial_state=initial_state,
                qubit_mapping='parity'
            )
            # Running VQE using a simulator, not actual quantum computer
            vqe = VQE(qubitOp, var_form, optimizer)
            results = vqe.run(backend)['energy'] + shift
            vqe_energies.append(results)
            print("Interatomic Distance:", np.round(dist, 2), "VQE Result:", results, "Exact Energy:", exact_energies[-1])

        print("All energies have been calculated")
```

```
HF energy: -10.203473360061937 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 0.5 VQE Result: -7.039710213523066 Exact Energy: -7.039732521635205 HF energy: -9.945427105942237 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 0.6 VQE Result: -7.3133443017893285 Exact Energy: -7.313345828761002 HF energy: -9.753847203061564 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 0.7 VQE Result: -7.500921095834627 Exact Energy: -7.500922090905936 HF energy: -9.600184702182101 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 0.8 VQE Result: -7.630976915070163 Exact Energy: -7.630978249333206 HF energy: -9.469677376536117 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 0.9 VQE Result: -7.7208107946930715 Exact Energy: -7.720812412134783 HF energy: -9.354893768508559 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 1.0 VQE Result: -7.782240655600787 Exact Energy: -7.782242402637008 HF energy: -9.25195375172978 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 1.1 VQE Result: -7.823597493272537 Exact Energy: -7.823599276362812 HF energy: -9.15855885285692 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 1.2 VQE Result: -7.850696622489987 Exact Energy: -7.850698377596027 HF energy: -9.07313203700239 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 1.3 VQE Result: -7.867561602231774 Exact Energy: -7.867563290110052 HF energy: -8.994489827277832 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 1.4 VQE Result: -7.87699987634333 Exact Energy: -7.8770014918183655 HF energy: -8.921712043375127 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 1.5 VQE Result: -7.8810141739625 Exact Energy: -7.881015715646999 HF energy: -8.854072040283643 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 1.6 VQE Result: -7.881070663071925 Exact Energy: -7.881072044030919 HF energy: -8.790987097121475 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 1.7 VQE Result: -7.87826716198872 Exact Energy: -7.878268167584997 HF energy: -8.731980715366813 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 1.8 VQE Result: -7.873440111965323 Exact Energy: -7.873440293132832 HF energy: -8.676655005381926 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 1.9 VQE Result: -7.8672336666725595 Exact Energy: -7.867233964816032 HF energy: -8.624671401017078 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 2.0 VQE Result: -7.860152328096388 Exact Energy: -7.860153207378775 HF energy: -8.575737701949146 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 2.1 VQE Result: -7.852595105514179 Exact Energy: -7.852595827876733 HF energy: -8.529599656891161 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 2.2 VQE Result: -7.844878726358128 Exact Energy: -7.844879093009717 HF energy: -8.486035628495898 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 2.3 VQE Result: -7.837257439573816 Exact Energy: -7.837257967615507 HF energy: -8.444853140805389 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 2.4 VQE Result: -7.829935044649883 Exact Energy: -7.829937002623398 HF energy: -8.405886322325905 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 2.5 VQE Result: -7.82307019119758 Exact Energy:
```

```
-7.82307664213409 HF energy: -8.368993488572691 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 2.6 VQE Result: -7.816782591606897 Exact Energy: -7.816795150472934 HF energy: -8.334054386160982 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 2.7 VQE Result: -7.811153438400034 Exact Energy: -7.811168284803367 HF energy: -8.30096692329537 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 2.8 VQE Result: -7.806218297643893 Exact Energy: -7.806229560089848 HF energy: -8.269643484101085 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 2.9 VQE Result: -7.801962398275158 Exact Energy: -7.801973602332551 HF energy: -8.24000711113722 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 3.0 VQE Result: -7.798352410421447 Exact Energy: -7.798363430915127 HF energy: -8.21198791109717 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 3.1 VQE Result: -7.795326815336425 Exact Energy: -7.7953404516375295 HF energy: -8.18551999853749 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 3.2 VQE Result: -7.792800697755741 Exact Energy: -7.792834806738608 HF energy: -8.1605391772555 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 3.3 VQE Result: -7.790603798121621 Exact Energy: -7.790774009971015 HF energy: -8.136981417970148 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 3.4 VQE Result: -7.788715348124798 Exact Energy: -7.789088897991485 HF energy: -8.114782068018084 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 3.5 VQE Result: -7.787215779047779 Exact Energy: -7.787716973466144 HF energy: -8.093875649366343 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 3.6 VQE Result: -7.786080383693413 Exact Energy: -7.786603763673838 HF energy: -8.074196070261742 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 3.7 VQE Result: -7.785203494307274 Exact Energy: -7.785702912499895 HF energy: -8.055677083373519 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 3.8 VQE Result: -7.784479540134329 Exact Energy: -7.784975591698664 HF energy: -8.038252853722296 # of electrons: 4 # of spin orbitals: 12 Interatomic Distance: 3.9 VQE Result: -7.783853366126989 Exact Energy: -7.784389611671852 All energies have been calculated
```

In [ ]:

```python
# plotting the data
plt.plot(distances, exact_energies, label="Exact Energy")
plt.plot(distances, vqe_energies, label="VQE Energy")
plt.xlabel('Atomic distance (Angstrom)')
plt.ylabel('Energy')
plt.legend()
plt.show()
```



VQE did very well since we can't even see the exact energy curve. This tells us that VQE energies were very close to the actual energies.

In [ ]: